THE EXPLICIT INCORPORATION OF VARIANCE IN THE PERFORMANCE MODELING OF SCHEDULING ALGORITHMS IN DISTRIBUTED AND SOFT REAL-TIME SYSTEMS

By

Nathan Hamm

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2011

Nashville, Tennessee

Approved:

Professor Larry Dowdy

Professor J. Michael Fitzpatrick

Professor Aniruddha Gokhale

Professor Mark McDonald

Professor Jeremy Spinrad

To my beloved grandmother, Jewell, always in my heart

ACKNOWLEDGMENTS

I want to thank my mother, Sarah, for her love, encouragement, and understanding throughout my dissertation research. Her emotional and financial support has been essential.

I especially want to thank my loving wife, Deana, for her continuous encouragement and understanding, for the many personal sacrifices she made, and for being infinitely supportive during difficult times in both of our lives.

I sincerely thank my mentor, Dr. Larry Dowdy, for his continuous motivation, encouragement, insight, and fortitude, all of which have been instrumental in the completion of my dissertation. Dr. Dowdy is wise far beyond his years and he has taught me as much about life and people, as academics and research. I am honored to have studied under the guidance and leadership of such an exceptional character. I am forever grateful for his guidance, inspiration, patience, and support.

I thank Dr. Michael Fitzpatrick for our many enlightening and entertaining conversations over the years, as well as his interest and support in allowing me to help instruct his classes and teach summer courses.

I thank my committee members, Larry Dowdy, Michael Fitzpatrick, Andy Gokhale, Mark McDonald, and Jerry Spinrad, for their help, insight, and support. My dissertation has improved as a direct result of their valuable feedback.

Also, this work would not have been possible without the generous financial support of Vanderbilt University. A portion of this work was funded by a grant from Acxiom Corporation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

The performance of computer systems is of great interest to performance analysts, capacity planners, system administrators, and others involved in system design, evaluation, and maintenance. Due to the wide range of hardware and software found in today's environments, the performance of a system can vary greatly as its environmental conditions change. Hardware failures can introduce variability into a system by affecting the availability of devices and the effectiveness of related software components (e.g., load balancers). The workload a system is subjected to can also have adverse effects on system performance because it can exhibit variability that leads to undesirable and unpredictable behavior.

In traditional performance evaluation studies, the impact of variability is often minimized or overlooked in order to simplify the analysis and evaluation of the underlying system performance models. Therefore, it is important to better understand the effects of variability in order to maximize system performance and minimize the negative effects of variability inherently found in most systems. By explicitly incorporating knowledge of variability performance effects into performance evaluation models, the accuracy and effectiveness of these models can be improved.

1.2 The Importance of Variance

To illustrate the importance of variance, consider Figure 1, which shows histograms of the number of items purchased at two busy supermarkets. Both supermarkets exhibit the same mean (i.e., average) number of items per customer, 15. However, the number of items purchased at Supermarket A is always close to the mean value (i.e., about 15 items), while the quantity purchased at Supermarket B tends to vary more widely. Customers at A seem satisfied that they are treated fairly, while at B, customers with smaller purchases frequently complain about having to wait behind customers with larger purchases. In response, Supermarket B decides to install express checkout lanes to be used exclusively by customers purchasing 10 items or less.

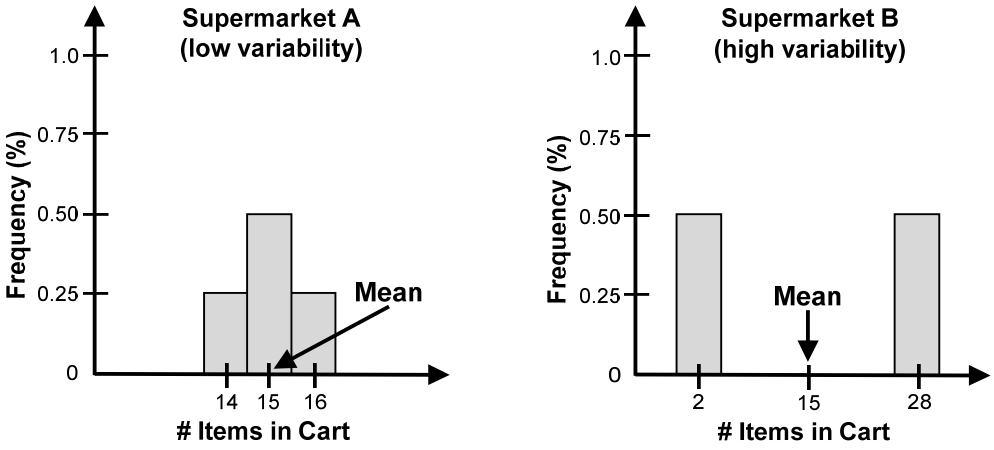

Figure 1: Example histograms of number of items purchased at two supermarkets

The shopping experience at Supermarket A is more predictable because the distribution of the number of items purchased exhibits low variability (i.e., high regularity). Without adding the new checkout lanes, the shopping experience at Supermarket B is less predictable because of the high variability (i.e., low regularity)

within the distribution of the number of items purchased. Customers with small purchases become more irritated because they often get delayed by customers ahead of them in line with larger purchases. However, with the new checkout lanes, all customers are better served. In fact, customers with small purchases will tend to shift their shopping preferences from Supermarket A to Supermarket B. This example illustrates how variance can affect scheduling (and management) decisions related to system performance and emphasizes the need for further study involving the effects of variability.

1.3 Motivation for Research

Although the impact of variability spans a wide range and type of environments, we focus on distributed and real-time systems in our research. In these systems, a key factor affecting performance is the scheduling algorithm used to allocate resources and schedule jobs[1]. Many traditional scheduling algorithms do not consider the variance of performance parameters when making decisions regarding how and when jobs should be allowed to execute. Therefore, when the environmental conditions of a system change due to variability (e.g., an irregular service time), these algorithms can cause a decrease in system performance that could otherwise be minimized or avoided. In this dissertation, the effects of variance in distributed and real-time systems is studied using formal modeling techniques, such as matching the second moment (or higher[2]) of performance parameters, the method of stages, discrete-event simulation, and state-space analysis. The emphasis in this study is placed on comparing and contrasting the

[1] The term *task* or *task stream* is typically used in the discussion of real-time environments.
[2] Higher moment typically has the connotation of any moment above the fourth moment. In this dissertation, higher moment loosely refers to the second moment, and in general refers to moments other than the first moment (i.e., not necessarily moments above the fourth moment).

performance of traditional scheduling algorithms to that of new and experimental algorithms.

The use of tools such as queuing network models and state-space diagrams is helpful in examining the performance and behavior of scheduling algorithms. However, it is often difficult, if not impossible, to apply these or other analytical techniques to realistic, large-scale environments, due to pitfalls such as state-space explosion. In these situations, simulation techniques serve as powerful modeling tools that overcome many of these shortcomings. Further, the use of simulation can provide comprehensive data that helps verify performance results not able to be verified analytically.

Due to the unique conditions found in many real-time environments, special-purpose modeling tools are often required to conduct detailed or focused experiments. We have developed a simulation tool called MOSS (Method Of Stages Simulator) that provides a way of systematically modeling the variance found in the workloads of real-time environments. In addition, we have created a Matlab State-space Analysis Tool (MSAT) that can be used to analytically solve small state-space models describing real-time environments.

1.4 Thesis Statement

The variance of parameters in performance models can have both positive and negative effects on system performance, particularly in real-time systems where scheduling algorithms often do not incorporate variance in their decisions. By studying the effects of systematically changing the variance of parameters in performance models, scheduling routines that take advantage of this information can outperform traditional scheduling algorithms. The explicit incorporation of variance in the performance modeling of

scheduling algorithms improves the design, efficiency, and performance of distributed and soft real-time systems.

1.5 Research Overview

As distributed and real-time systems become more pervasive, there is growing interest in the reliability and performance of these environments. More specifically, there is a need to better understand how tasks executing in these systems can be scheduled more efficiently and effectively. Research involving the performance analysis and capacity planning of real-time systems has been conducted for decades. A significant portion of this work has focused on comparing the average performance of scheduling routines and analyzing their theoretical behavior. However, most of this work has focused on the analysis of existing methods, rather than discovering new ones. Further, these studies have typically considered only the first moment of performance parameters. Therefore, much less is known about the performance effects of higher moments, such as variance.

As part of this dissertation research, we develop a performance model of a large enterprise (distributed) environment using Colored Petri Nets and use it to answer several performance prediction questions in capacity-planning scenarios. In this work, difficulty arises in developing a robust model that accurately captures the behavior of the measured system workload. In particular, the distributions of the inter-arrival and service times more closely resemble those of heavy-tailed or hyperexponential distributions. Through a rigorous refinement process, the overall mean and variance, as well as the mean and variance of each job type, are matched for the inter-arrival and service times. During this refinement process, we observe that by simply changing the variance of a single workload parameter (e.g., service time), the system performance is significantly affected.

This demonstrates experimentally that accurately modeling system variance is an important topic in performance evaluation.

The difficulty encountered in this study motivates further work that leads to a more unified and structured modeling approach that makes it easier to generate accurate workloads. This technique also provides a basis for analyzing the effects of higher moments, such as variance. The method of stages modeling technique is used to achieve a two-moment match for each performance parameter of interest. It also lends itself well to performance modeling of variability because the number of stages used to model a particular parameter is directly related to its variance. Therefore, conducting sensitivity analysis experiments on the variance of performance parameters is both easy and standardized using this approach. The method of stages is also well suited to analytical approaches, making the verification of results possible.

One caveat, however, is that applying such techniques to many practical systems is not feasible due to state-space explosion. Therefore, a simulation approach is employed that utilizes this powerful modeling technique, while not being susceptible to the pitfalls of analytical methods. Due to the additional constraints placed on distributed and real-time systems, specialized modeling and analysis tools are required. Existing tools often target only hard real-time systems, where even one missed deadline is unacceptable, or are too narrow in scope to be used effectively for detailed sensitivity analysis. Further, these tools are often based on ad-hoc and informal methods, making analysis and validation of results difficult. In this dissertation, we rectify some of these shortcomings by examining soft real-time systems, applying formal modeling techniques, developing specialized performance modeling tools, and analytically validating the results.

1.6 Research Contributions

The main contributions of our work are as follows:

1. Case Study of an Enterprise Grid Environment

   In cooperation with Acxiom Corporation, we develop a performance model of an enterprise grid environment that captures the important characteristics of the infrastructure, including job scheduling routines. During the model calibration process, it is observed that changing the variance of a single workload parameter significantly affects the estimated system performance. After a rigorous refinement process and model validation, several capacity-planning scenarios are examined, and output from the model is used to help guide decisions regarding future expansion.

2. Method of Stages Simulator

   We develop a simulation tool called MOSS (Method Of Stages Simulator) that uses the method of stages to help analyze the effects of variance in real-time scheduling. MOSS provides an intuitive graphical interface that makes it easy to conduct higher-moment sensitivity analysis experiments on practical workloads in real-time systems. The MOSS engine uses the method of stages as the theoretical basis for its simulations and the user interface provides a direct link between the method of stages theory and its practical application. By changing a single parameter, the number of stages, the user can model various distributions and simulate conditions ranging from soft to hard real-time environments. Using MOSS to study the effects of variance on existing scheduling algorithms, as well as to discover new state-based algorithms, has produced new and interesting results.

3. Uniform Sensitivity Analysis Experiments

We use the method of stages modeling technique to match the first two moments of performance parameters and systematically investigate the effects of variance in a uniform manner. We make several interesting observations and develop heuristics that summarize many of the important findings from our work. The basic framework of the method of stages, where real-world processes are modeled by a series of discrete stages, can be applied to many other areas of research outside of distributed or real-time systems.

4. The TLAX Algorithm

   Using MOSS, we develop the XLAX scheduling technique and compare its performance to that of the traditional scheduling algorithms. We then propose a simpler and easier to implement algorithm, named TLAX (Threshold LAXity), and show experimentally that it is robust and can outperform the traditional scheduling algorithms, particularly under heavy load conditions. These results help demonstrate the importance of explicitly considering the variance in the development of performance models.

5. Matlab State-Space Analysis Tool

   In order to help analytically validate results obtained from MOSS, we develop the Matlab State-space Analysis Tool (MSAT) that constructs and solves small state-space models describing real-time environments. Specifically, the tool can be used to construct a complete state diagram for any model based on the method of stages technique, where a pair of tasks is used. A number of different scheduling algorithms can be evaluated using this tool, and the models are solved analytically to obtain exact (theoretical) performance metrics.

6. Analytical Validation of Results

   Using MSAT, we analytically validate the results obtained from MOSS and demonstrate the feasibility and accuracy of using such modeling tools. The analytical validation improves the reliability of MOSS and encourages the use of this and similar tools to investigate the effects of variability in the future.

1.7 Organization

Related work is presented in Chapter II. This chapter includes the necessary background information related to performance modeling using Colored Petri Nets, performance modeling and task scheduling in real-time systems, and the method of stages modeling technique.

A case study of an enterprise grid environment is presented in Chapter III. This chapter includes the workload characterization and important aspects of the performance model, such as parameterization, calibration, and validation. Capacity-planning scenarios are used to illustrate the effectiveness and usefulness of the developed model.

The MOSS simulation tool is presented in Chapter IV. This chapter includes a motivating example to demonstrate the usefulness of the simulator. A detailed tutorial discusses the user interface and describes all the steps necessary to run simulations and output performance metrics.

A more focused discussion of MOSS is presented in Chapter V. This chapter includes a specific workload example, illustrating how MOSS can be used to study the effects of variability. Some interesting observations are used to help guide the remaining sensitivity analysis experiments.

Detailed sensitivity analysis experiments are presented in Chapter VI. This chapter discusses how MOSS is used to compare the performance of traditional scheduling algorithms. The development and performance of a new, generic scheduling algorithm (XLAX) is also presented.

A more specialized scheduling algorithm, named TLAX, is discussed and evaluated using MOSS in Chapter VII. This algorithm outperforms the traditional algorithms and proves to be a robust choice for scheduling tasks in workloads with high variability.

MSAT is used to study analytical results of TLAX in Chapter VIII. In this chapter, the experimental results obtained from MOSS are validated and some insights are gained into the TLAX algorithm by examining state-space information.

Concluding remarks and future work are discussed in Chapter IX.

CHAPTER II

RELATED WORK

This chapter presents research related to modeling and analyzing the performance of scheduling algorithms in distributed and real-time systems. A case study involving an enterprise grid environment is described in Chapter III that uses a performance model constructed from Colored Petri Nets (CPNs). Therefore, a brief history of Petri nets is provided along with a more detailed discussion of Petri net extensions and some specific examples of their use in performance modeling. A simulation tool called MOSS (Method Of Stages Simulator) is presented in Chapter IV that allows a user to analyze the effects of variance on workloads in real-time environments. To better explain the theoretical basis for MOSS, a fundamental survey of real-time scheduling algorithms is provided along with a detailed discussion of stage-type distributions, which MOSS uses to achieve higher-moment matches for performance parameters. In Chapter VIII, an analytical state-space tool developed using Matlab is presented that provides mathematical validation of MOSS results. Therefore, analytical techniques applied to state-space models are also discussed. The purpose of this chapter is to provide the reader with a concise understanding of the topics to be discussed in the remainder of this dissertation.

2.1 Performance Modeling Using Petri Nets

Petri nets are graphical and mathematical modeling tools that can be used in a wide variety of applications. The notion of Petri nets was first introduced by Carl Adam Petri in his 1962 dissertation [70]. A number of technical reports and papers followed, and

later the first textbook focusing on Petri nets was published in 1981 [69]. Since then, Petri nets have been used to model many different types of systems and applications. A myriad of research papers, textbooks, and tutorials have been written, and Petri himself co-authored a revised introduction to Petri nets in 2008 [71].

Due to their descriptive power, Petri nets are also used as visual aids to assist in graphically representing how complex systems interact and behave, similar to how flow charts or UML diagrams are used to describe complex processes. The illustrative power of Petri nets makes them relatively easy for novices to understand underlying system complexities. They also provide a powerful modeling framework that allows experts to design detailed simulators and performance analysis tools. However, even with the help of the latest modeling and simulation tools, it can be difficult for inexperienced users to create or apply Petri nets to designing and modeling complex systems.

2.1.1 Petri Net Definition

A Petri net (PN) is a directed bipartite graph consisting of two types of nodes: places and transitions. In the graphical representation of a PN, places are drawn as circles and transitions are drawn as thin rectangles. Weighted arcs are directed either from a place to a transition, or from a transition to a place. Arcs are labeled with positive integer weights that correspond to a number of tokens, and unit weights (i.e., weight values of 1) are typically omitted from illustrations. A marking is denoted by an m-vector M, where m is the total number of places in a PN. The p^{th} component in M denotes the number of tokens in place p. Therefore, a marking assigns a number of tokens to each place and an initial marking M_0 specifies the initial quantity of tokens in each place before any transitions fire. A place p is marked with k tokens if a marking assigns k tokens to p.

This is illustrated graphically by k dots drawn inside of p. A formal Petri net definition, based on [69], is provided in Table 1.

Table 1: Formal Petri net definition

A basic Petri net is a 5-tuple, $PN = (P, T, A, W, M_0)$ where:

$P = \{ p_1, p_2, \ldots, p_m \}$ is a non-empty, finite set of places,
$T = \{ t_1, t_2, \ldots, t_n \}$ is a non-empty, finite set of transitions,
$A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs,
$W : A \rightarrow \{ 1, 2, 3, \ldots \}$ is a weight function, and
$M_0 : P \rightarrow \{ 0, 1, 2, 3, \ldots \}$ is the initial marking.

Any place p_i with an arc directed from p_i to a transition t_i is called an input place of t_i, whereas any place p_j with an arc directed from t_j to p_j is called an output place of t_j. A transition t is enabled when each input place p of t is marked with at least $W(p, t)$ tokens, where $W(p, t)$ is the weight of the arc from p to t. An enabled transition can fire at any time, and there is no rule governing the order in which multiple-enabled transitions should fire. When a transition t fires, it removes $W(p, t)$ tokens from each input place p of t, and it adds $W(t, p)$ tokens to each output place p of t, where $W(t, p)$ is the weight of the arc from t to p. Therefore, the weight function W specifies the number of tokens that travel across a given arc.

As an example, consider Figure 2, which illustrates a Petri net with places p_1, p_2, and p_3, and transitions t_1 and t_2. The formal definition of the net shown in Figure 2 is given in Table 2. For simplicity, all arc weights have a value of 1 and are omitted from the diagram. Place p_1 contains three tokens, p_2 contains two tokens, and p_3 contains zero tokens (i.e., p_3 is empty). In Figure 2, t_1 is enabled because each of its input places (p_1 and p_2) contains at least one token. However, t_2 is not enabled because p_3 is empty. In

the current state, because t_1 is the only enabled transition, the only possible event is the firing of t_1. When t_1 fires, it will consume (remove) one token from p_1 and one token from p_2, and produce a single token into p_3. Since all tokens are identical, any one of the tokens in p_1 and any one of the tokens in p_2 may be consumed by t_1—all of the tokens in a particular place are indistinguishable. Figure 3 illustrates the Petri net immediately after t_1 fires, where t_2 is now enabled because p_3 contains a token. Strictly speaking, because t_1 and t_2 are both enabled, either of these transitions can fire but they cannot both fire simultaneously. Transitions t_1 and t_2 are said to be immediate firing transitions, meaning they fire whenever they are enabled (with no additional firing constraints). In the case of multiple enabled transitions, the firing occurs in random order. This non-deterministic nature of Petri nets is one reason they are effective in modeling the concurrent, non-deterministic nature of distributed systems such as grid environments [8].

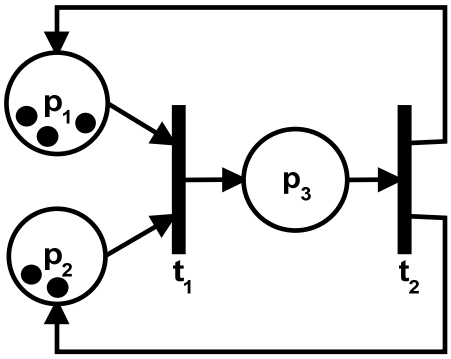

Figure 2: Simple Petri net

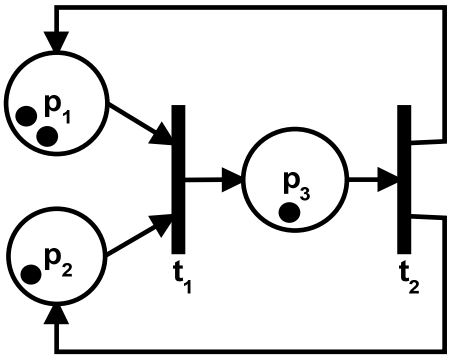

Figure 3: Petri net after t_1 fires

Table 2: Formal definition of the PN shown in Figure 2

$PN = (\ P, T, A, W, M_0\ )$ where:

$$P = \{\ p_1, p_2, p_3\ \}$$
$$T = \{\ t_1, t_2\ \}$$
$$A = \{(p_1 \rightarrow t_1),(p_2 \rightarrow t_1),(p_3 \rightarrow t_2),(t_1 \rightarrow p_3),(t_2 \rightarrow p_1),(t_2 \rightarrow p_2)\}$$
$$W = \{\ 1, 1, 1, 1, 1, 1\ \}$$
$$M_0 = \{\ 3, 2, 0\}$$

2.1.2 Basic Modeling Principles

Many of the constraints and requirements of real-world systems can be modeled using basic PN principles. For example, a series of two events can be sequenced as shown in Figure 4, where the token in p_1 progresses to p_2 and then to p_3. Transition t_1 must fire and produce a token in p_2 before t_2 is enabled, thus forcing the sequence of events.

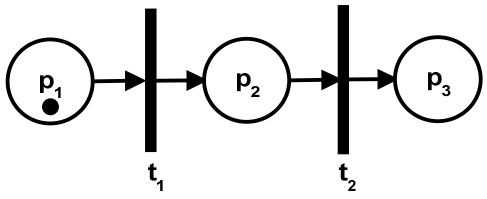

Figure 4: PN sequencing

Figure 5 indicates how decisions, concurrency, and synchronization can be modeled. In (a), the token in p_4 enables transitions t_3, t_4, and t_5 but when one of the transitions fires, the token is removed, leaving the remaining two transitions disabled. Therefore, a non-deterministic decision has to be made as to which one (and only one) of these three transitions fires. In (b), transition t_6 requires a token from places p_5, p_6, and p_7 in order to fire. Thus, the firing of t_6 represents the synchronization of the three places. In (c),

places p_9, p_{10}, and p_{11} represent concurrent actions, each of which may independently finish at any time, triggering the firing of the next step in the process.

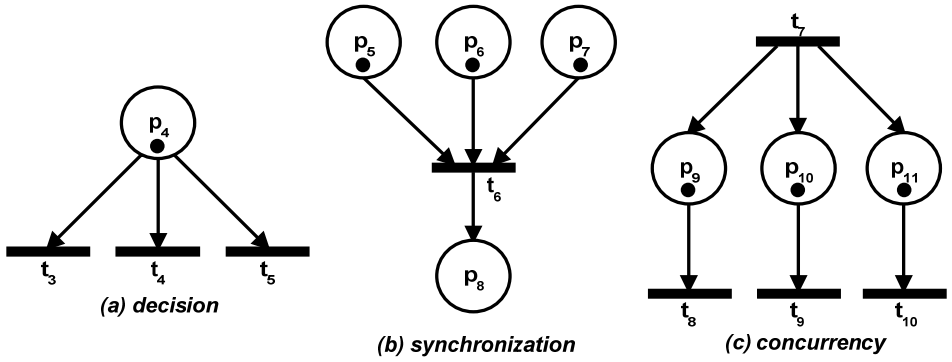

Figure 5: PN decision, synchronization, and concurrency

Mutual exclusion can be modeled as shown in Figure 6. Both transitions t_{11} and t_{13} require a token from p_{14} but after either transition fires, the remaining one is left disabled. Therefore, p_{13} and p_{16} can never simultaneously contain a token. When either t_{12} or t_{14} fires, a token is placed back into p_{14}, re-enabling both t_{11} and t_{13}. In this way, the token in p_{14} serves as an access token or key, and the presence of a token in either p_{13} or p_{16} implies possession of this key.

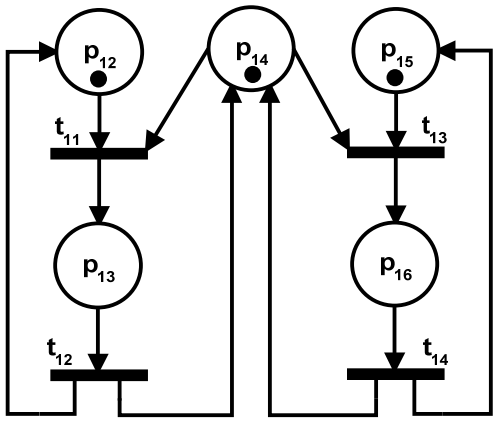

Figure 6: PN mutual exclusion

2.1.3 Petri Net Extensions

Using PNs to model distributed systems leads to a natural representation of components in real-world environments. For example, jobs are naturally represented as tokens and transitions correspond to various events in the job flow description, such as job arrivals and resource acquisition. However, when jobs are not all identical, a PN Model (PNM) can grow quite complex due to the overhead involved in adding the places and transitions necessary to track job information and events. For example, data clustering techniques result in multiple heterogeneous job classes and, therefore, a PNM that relies on cluster-based input parameters must accurately model distinct and often very different job types. Modeling such a system using basic PNs can be quite difficult because tokens are indistinguishable and therefore, different job types must be explicitly modeled in the PN structure. Modeling other job behavior can be difficult, if not impossible, as well. For example, it is well-known that basic PNs are not suited for modeling prioritized jobs [5].

Basic Petri nets are limited in their modeling power for other reasons as well. Because there is no notion of time or delays in basic PNs, all transitions are immediate and therefore, modeling specific temporal behavior (e.g., the service delay at a disk) is not possible. Further, because all tokens are identical, there is no method of representing more than a single flow of information through a basic PN without introducing duplicate copies of net pieces, which makes the PN more difficult to understand and analyze. Therefore, PNs have been continuously extended in various ways to add more flexibility and expressive power.

One of the earliest PN extensions is called Stochastic Petri Nets (SPNs), which adds a number of functions, as well as probabilistic firing of transitions and other features. For

example, transitions may also be "timed," which means that, once enabled, the transition fires after a specified non-deterministic time (e.g., exponential) delay. Timed transitions are illustrated graphically by unfilled rectangles. Input functions can be used to define additional operations or conditions that must be carried out or satisfied before a transition fires. Similarly, output functions can be used to specify additional operations to be carried out after a transition fires. Thus, SPNs are powerful tools for graphically expressing time-based systems, but their effectiveness rapidly decreases as the system they are modeling grows in complexity and size.

Generalized Stochastic Petri Nets (GSPNs) combine the features of basic PNs and SPNs and allow both immediate and timed transitions. When there are multiple enabled immediate transitions, the transition selected to fire is no longer chosen at random but is instead selected from firing weights (probabilities) associated with each transition. In the case of mixed enabled transitions (immediate and timed), the firing of immediate transitions has priority over timed transitions. The concept of inhibitor arcs is also introduced, which allow a transition to fire only when a corresponding place is empty. In Figure 6, for example, place p_{14} and its associated arcs can be removed and replaced with inhibitor arcs as shown in Figure 7. An inhibitor arc drawn from p_{13} to t_{13} prevents t_{13} from firing whenever a token is present in p_{13}. Similarly, the inhibitor arc drawn from p_{16} to t_{11} allows t_{11} to fire only if p_{16} is empty. These two inhibitor arcs achieve the same mutual exclusion effect as shown in Figure 6 and are more of a convenience in representation rather than a new formalism.

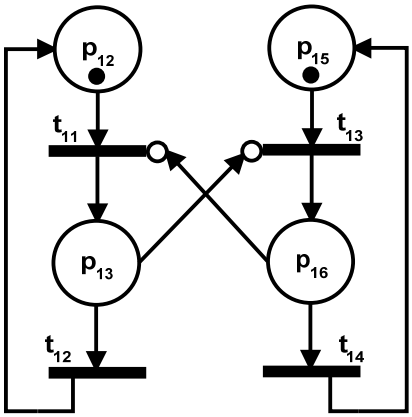

Figure 7: PN mutual exclusion using inhibitor arcs

Although GSPNs represent a small step in the PN lineage, their increased modeling power is significant. In [60], GSPNs are used to create a model of a multi-processor architecture in which processors cooperate by exchanging messages through a set of common memories connected through a bus network. Figure 8 depicts this architecture, where processors P_i use their local buses LB_i to access private memory PM_i and they use global buses GB_i to share common memories CM_i. Figure 9 shows a portion of the GSPN model that corresponds to the selection and accessing of one common memory. The shaded places p_1, p_2, and p_3 represent active processors, available buses, and waiting processors, respectively. Each processor may issue an access request at rate λ so that timed transition t_1 fires at rate $|p_1| \lambda$, where $|p_1|$ denotes the number of tokens in p_1 (i.e., the number of active processors). A token in p_3 represents a processor that needs to select a memory to perform an external access. If no bus is available (indicated by p_2 being empty) the processor waits, but if a bus is available the processor selects a memory according to a uniform distribution.

Because of the symmetry of the GSPN model, its behavior can be explained easier by focusing on the subnet that describes the access to the first common memory. Assume

that one bus is available and a processor in p_3 decides to access the first common memory. The selection of the first common memory is represented by the immediate firing of t_2, which removes a token from p_3 and places it into p_4. At the same time, a token is also removed from p_2 to indicate that a bus is now in use. Therefore, the tokens in p_4 represent processes that want to access a common memory and have an assigned bus. These tokens can be consumed by either transition t_5 or t_6. Transition t_5 is enabled only if no processor is accessing the selected memory (indicated by p_7 being empty). If t_5 fires, the processor keeps the bus and a token is placed into p_7 to indicate that the processor is accessing or queued for common memory. On the other hand, if another processor is already using a bus to access the same common memory, transition t_6 is enabled. If t_6 fires, the bus is released, indicated by a token returned to p_2, and the new accessing processor will be waiting for the same memory and the same bus that is already in use by a processor in p_7 (i.e., note the weight of 2 on the arc from t_6 to p_7). Therefore, the tokens in p_7 correspond to processors accessing or queued for common memory using a single bus. When a token is available in p_7, timed transition t_{11} fires probabilistically at rate μ, where $1/\mu$ is the average access time. The firing of t_{15} represents the bus being assigned to the next waiting processor. If no other processor is waiting to access the common memory (i.e., p_7 is empty), then t_{14} instead fires and removes the token from p_7. In this case, a token is returned to p_2 to indicate the availability of the bus.

The GSPN models of this architecture are continuously refined to produce more efficient model representations. By exploiting properties of the GSPN model and the corresponding state-space (e.g., Embedded Markov Chains (EMCs) and the existence of tangible and vanishing states) an efficient GSPN model is used to obtain upper and lower

bound estimates of various performance metrics [60]. It is also demonstrated how knowledge of the existing system can be used to create very different but equally powerful GSPN models of the same environment. For example, the models initially created are well suited for multi-processor systems containing a small number of memories or processors. However, in practice it is typically more common for such systems to contain a large number of processors and memories but relatively few buses. The GSPN model is, therefore, modified so that its complexity grows linearly with the number of buses, instead of memories or processors, and thus becomes a much more realistic and scalable model. In general, GSPNs are more powerful and scale better than SPNs, but they still become overly complex as the corresponding target environment's size increases. Although GSPNs provide increased flexibility, their modeling power is limited because tokens are indistinguishable, as with SPNs.

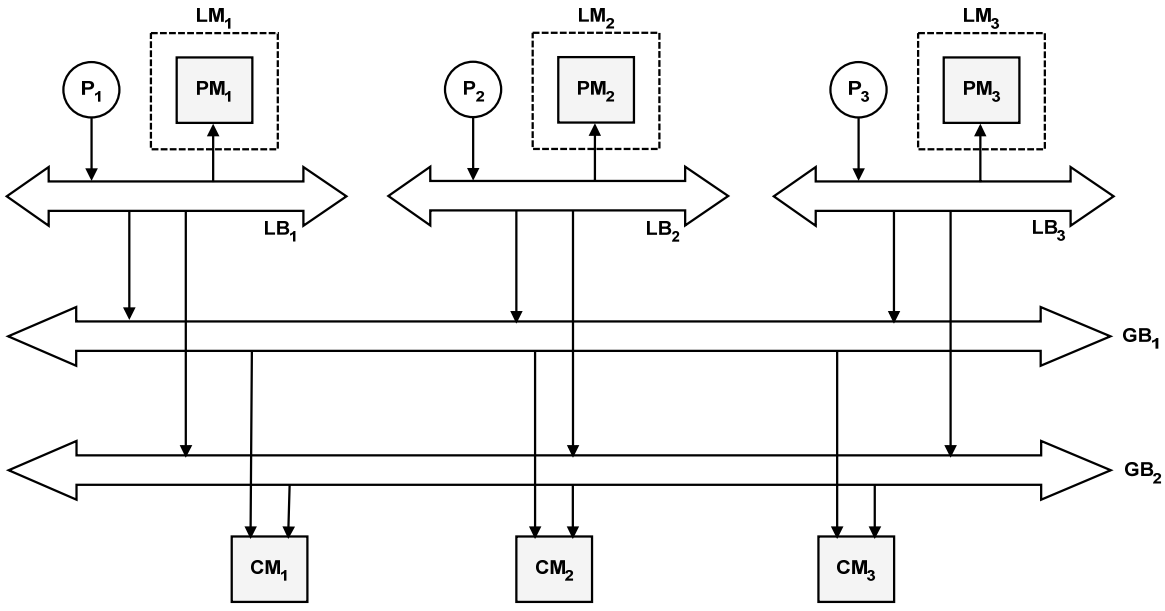

Figure 8: Example microprocessor architecture

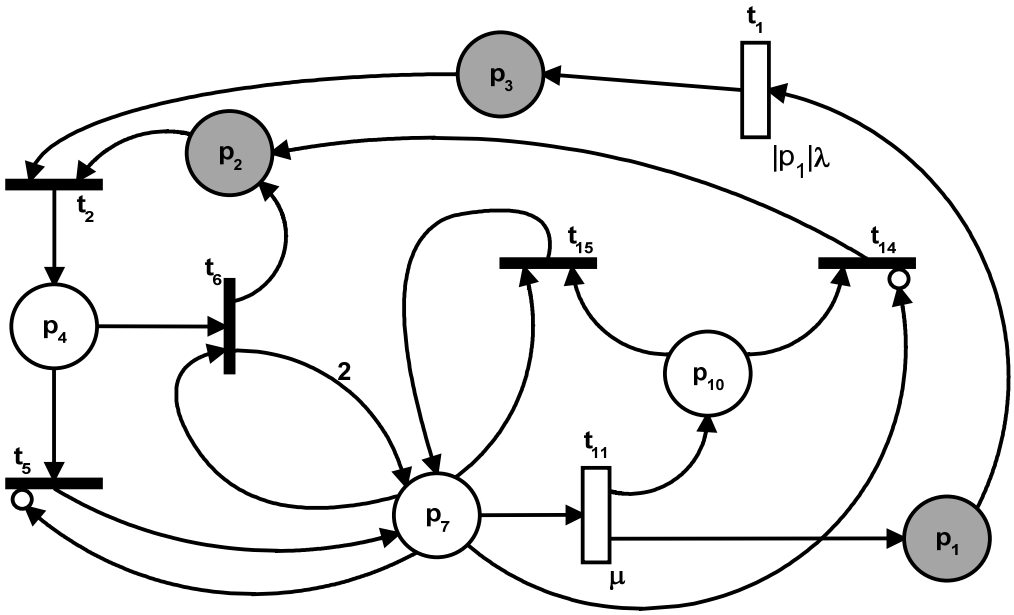

Figure 9: GSPN model of the microprocessor architecture

Queuing Petri Nets (QPNs) are one of the latest advancements in the PN lineage and are part of a larger group of PN extensions collectively referred to as high-level Petri nets. The idea behind QPNs is to combine the expressive power of queuing networks and the modeling power of GSPNs. Queuing networks work well for modeling resource contention and scheduling strategies, but are not as well suited for modeling process blocking and synchronization, which PNs in general are [51]. Therefore, QPNs significantly improve model expressiveness, allowing much more detailed system aspects to be modeled by integrating queuing stations directly into QPN places. This allows the modeler to easily experiment with different queuing strategies while maintaining the flexibility of PN-based models.

Figure 10 shows a QPN model of a central server environment where a job (launched from a terminal) must wait for available memory before entering the system [10]. After memory is allocated (indicated by t_2 firing), a job receives service at the CPU. After

completion at the CPU (indicated by t_3 firing), the job moves to p_4, where it either exits (via the firing of t_4) the system and restarts, or proceeds to receive service at a disk (via t_6 or t_7). When a job exits, its previously allocated memory is returned via t_1, as well as a terminal token. Alternatively, a job can receive service at either of two disks and then return to again receive service at the CPU. The places p_2, p_3, p_5, and p_6 are queuing places, where each one contains its own queuing station. In a queuing place, a discipline is specified that determines the order in which tokens are removed. For example, the order in which jobs are removed from p_2 to start executing at the CPU is *FIFO*, which is indicated in the figure by the label in curly braces for p_2. Therefore, jobs can be started in FIFO order, LIFO, or any other desired manner simply by changing the discipline built into p_2. The discipline associated with the CPU (i.e., p_3) is *Processor Sharing*, which means anytime multiple jobs are present at the CPU, each one is allocated an equal portion of the processor. Similarly, the discipline associated with both Disk 1 and Disk 2 is *Shortest Job Next* and causes shorter jobs to be processed first. Finally, the solid transitions are immediate and fire immediately once tokens are available in each of their respective input places. The remaining transitions (i.e., t_1, *t3*, *t6*, and *t7*) are timed and each one fires at a rate according to the μ_i value below each transition.

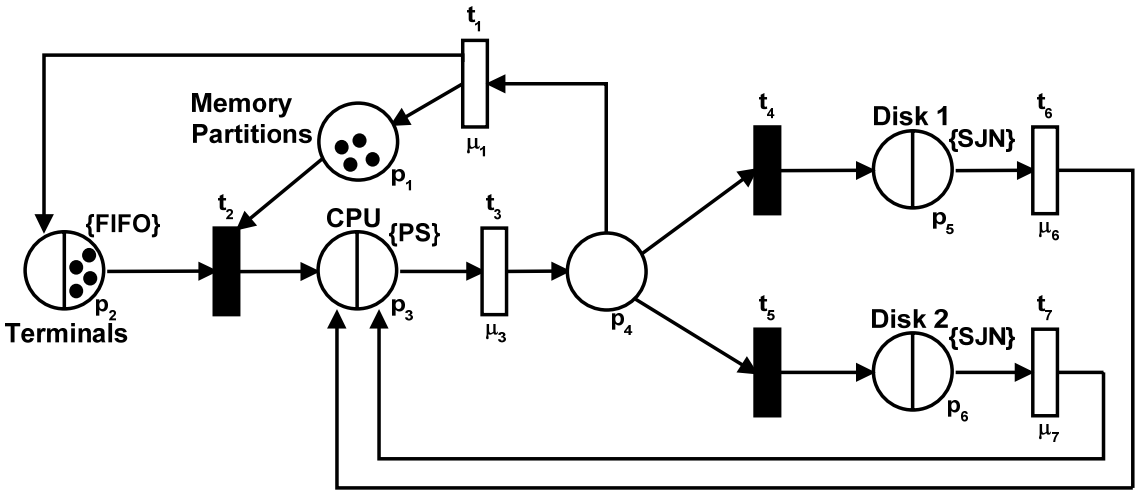

Figure 10: Example QPN model of central server system

The use of QPN-based modeling has produced promising results in the area of performance modeling and prediction. For example, a modified form of QPNs is used in [50] to analyze the performance of a realistic e-business application based on the SPECjAppServer2001 application, which is designed to produce realistic workloads based on large distributed applications complex enough to represent a real-world e-business system [1]. A similar study is done in [48] where the performance of distributed component-based systems is analyzed using QPNs.

2.1.4 Colored Petri Nets

Another type of high-level Petri net is Colored Petri Nets (CPNs), which extend GSPNs by allowing tokens to be different colors [35]. Different colored tokens can be used to represent different job types/classes. Transitions are also allowed to fire in different modes, depending on what types (colors) of tokens they consume. The use of CPNs in performance modeling has produced many results. For example, in [13] CPNs are used in the CAPLAN project, where a tool called Design/CPN is used in the capacity planning of web servers. The server environment studied consists of multiple layers (e.g.,

structural, application, and resource) and a corresponding CPN model is constructed to mimic the operation and interaction of these layers. Figure 11 shows the portion of the CPN used to model the behavior of each server thread that processes a client GET request. Each server thread executes a loop that consists of opening a connection, handling a request, and closing the connection. These operations are modeled by the transitions TCP_OpenConn, HTTP_GetURL, and TCP_CloseConn, respectively. Each of the bold places and transitions shown in Figure 11 corresponds to a CPN subnet that models similar or related behavior. For example, after a connection is opened, a token is produced into the *Opened* place and then passes through a related subnet before being consumed by HTTP_GetURL. Therefore, the place *Opened* corresponds to a subnet that performs all actions required after a connection is opened but before a GET request is processed. This ability to nest CPNs inside of places and transitions demonstrates the hierarchical nature of CPN performance models. Using the complete CPN model in [13], the server response time, number of delayed requests, and resource utilization can be examined for different simulated workloads. The CPN model is validated by comparing output metrics against the actual system measurements. It is determined, for example, that if the server's workload increases to approximately 100 requests/sec, incoming requests begin to stall and response time increases dramatically. It is found that a maximum workload of 75 requests/sec can be processed while still maintaining an acceptable Service Level Agreement (SLA) that specifies a maximum wait time for incoming requests.

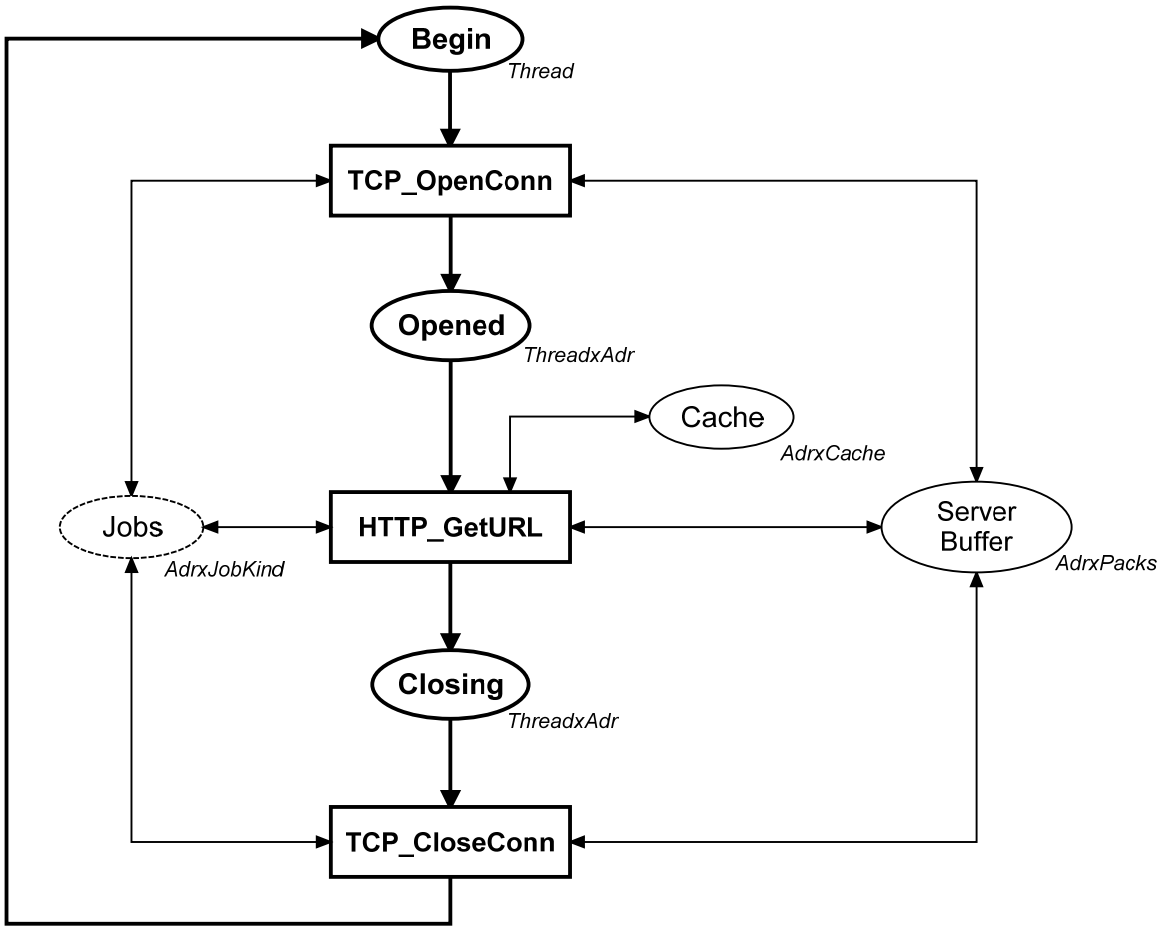

Figure 11: Portion of CPN model used in CAPLAN project

The hierarchical nature of CPNs is typically a restriction of the particular modeling environment, rather than the CPN definition itself. This hierarchical structure is an important feature, especially when constructing large, complex models. In fact, a hierarchical modeling technique is discussed in [80] and it suggests that just as distributed applications typically consist of a hierarchy of layers, any model describing such a system should itself be hierarchical in nature. It is argued that hierarchical models consisting of several layers provide a degree of accuracy that cannot be achieved with single layer performance models. The hierarchical representation of a distributed system within the proposed modeling framework inherently provides great flexibility in

assessing computation, communication, and hardware related issues [80]. A hierarchical model is, therefore, an important consideration to keep in mind when high accuracy is desired in performance predictions. Because they represent a higher level of abstraction than basic SPNs, CPNs provide an excellent compromise between readability and expressive power. With CPNs, token types (colors) can be defined that directly correspond to job classes, allowing tokens to store and carry with them all necessary job parameters. New token colors can be created to capture the behavior of common system changes, such as new job types, without requiring physical changes to the net structure.

Despite the many PN extensions developed over the years, CPNs remain one of the most commonly used and cited modeling tools in the PN lineage. A number of characteristics contribute to their popularity, including their hierarchical design structure, ability to be generically timed, and a large number of formal analysis methods [36]. Another significant contributing factor to the success of CPNs is the existence of CPN Tools, which is a graphical simulation tool designed specifically for constructing and analyzing CPN-based models. A detailed treatment of the basic concepts, analysis methods, and practical use of CPNs can be found in [33], [34], and [37]. CPNs are used in Chapter III to model the performance of an enterprise grid environment.

2.1.5 CPN Tools

One of the most commonly referenced simulation tools for modeling the performance of Colored Petri Nets is CPN Tools, which is discussed in [75]. CPN Tools provides a development environment that allows the creation, specification, modification, simulation, and analysis of CPNs. Figure 12 shows a screenshot of the CPN Tools development environment with an example CPN model displayed. The model shown is

simple and represents a job workflow description where jobs wait in a queue to be serviced one at a time until they finish and finally, restart. The left pane of the window displays custom token colors, functions, and other parameters created by the user. Token types, or colors, may contain complex data structures such as arrays, lists, vectors, and records. Many built-in functions supporting mathematical (and other) operations can be used to define custom functions, or may be accessed directly in the PN model.

In Figure 12, a custom token type named *JOB* has been created and defined as a *UNIT* that is timed. A *UNIT* is a predefined token color (based on the primitive data type *unit*) in CPN Tools and it behaves exactly as a token in basic PNs, except that *UNITs* may also possess a time value. Other color types can be defined using any number or combination of primitive types, or as lists/vectors of other color types. Two variables named *job* and *reset_job* have been defined, both of color type *JOB*. Two user functions named *expDelay* and *numJobsExecuting* have been defined that return a random value from an exponential distribution and the number of jobs currently executing, respectively.

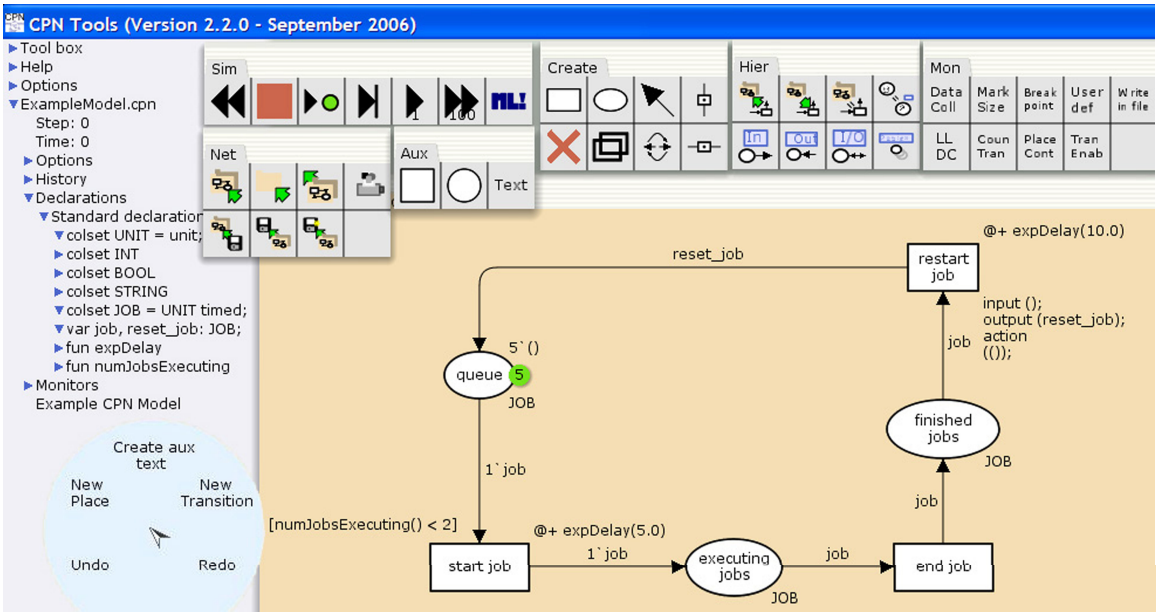

Figure 12: CPN Tools development environment

To construct a model, Petri net components are selected from the floating menus along the top and added to the drawing window. Components such as arcs can be drawn by first selecting a place or transition and then dragging a line to the target item. Overall, a model is drawn and created in a manner similar to that used in most graphical design tools. However, one distinct feature of CPN Tools is the use of circular menus that are accessed by right clicking an area of interest. The menu options displayed are context-specific and an option is selected by moving the mouse in a circular motion and then clicking. An example context menu is shown in Figure 12 on the lower left.

Places, transitions, and arcs resemble those in basic PN diagrams with a few key differences. Each place in a CPN model has a name, color type, and initial marking. The name is a label that uniquely identifies the place, while the color type identifies the type of tokens that may appear in the place at any given time. An initial marking specifies the number and type of tokens that are initially present before any simulation or analysis begins. In Figure 12, the place named *queue* may contain only *JOB* type tokens and

initially there are five such tokens present. All places shown in the example model may contain only *JOB* tokens, and no places other than *queue* contain initial tokens.

Each transition is drawn as a box or rectangle and has an associated name that uniquely identifies it. A transition may also have an associated guard condition, timing delay, or code segment. A guard condition specifies additional constraints that must be satisfied in order to enable the transition. In Figure 12, the *start job* transition has a guard condition that calls the *numJobsExecuting* function to verify there is at most one job currently executing. The delay inscription, attached to the upper right of the transition, specifies that a produced token will be time-stamped with a delay equal to the current time plus a delay value returned from the *expDelay* function. A code segment is used to specify additional operations that should be performed when a transition fires, as well as which values should be assigned/bound to output variables. An input function[3] specifies which input variables, if any, appear in the defined actions, and output functions specify variable bindings. Actions can invoke built-in or user-defined functions or perform other operations before ultimately creating the values to be used by the output function.

An important feature of CPN Tools concerns the timing and delay behavior of transitions. With SPNs, for example, a transition delay prevents a token in its input place from being consumed and defers firing until the delay period has elapsed. In CPN Tools, all transitions fire immediately, provided that any associated guard conditions are all true and there are sufficient input tokens available. When a transition fires, any produced tokens are time-stamped with the appropriate delay, forcing them to remain in their respective output places until the corresponding delay has passed. Therefore, a transition delay does not affect the firing of the transition, but rather the time value of a produced

[3] For simple expressions, binding is automatic and nothing must be specified in the input function.

token. This feature is merely a design decision on the part of the developers and not a restriction on the CPN framework, but it does affect the design of CPN models and it is important to keep in mind during model development.

Arcs in CPN models may also contain labels that specify the number and type of tokens that travel across them. These labels are called arc inscriptions and they typically specify variable names that the code segment of each transition assigns values to upon transition firing. For simple binding situations, such as when a single token flows along each arc, bindings are done automatically. In Figure 12, the *restart job* transition contains a code segment where the action creates a new *UNIT* (specified by the set of parentheses) and the output function binds this value to the variable *reset_job*. Code segments are common in most CPN models, although they are often not required. For example, due to the simple binding requirements, the code segment for *restart job* can be eliminated.

In the provided example, jobs must wait in the queue before being started. If a job is already executing, the remaining jobs in the queue must wait until the executing job finishes. When a job starts, the token is stamped with a delay representing its service time that will prevent it from being consumed by the *end job* transition until the delay has elapsed. After this service delay, *end job* will consume the token in *executing jobs* and place a new token in *finished jobs*. Notice that the newly created token will not be stamped with a delay because there is no delay associated with *end job*. Therefore, tokens added to *finished jobs* may be consumed at any time by *restart job*. When *restart job* fires, the produced token is stamped with a delay and placed back into *queue*.

Note that on average, jobs will execute for five time units and have to wait ten time units in the queue before they are available for consumption. Thus, it will likely be common to have jobs waiting in the queue even though no job is executing. The *restart job* transition in this example could also represent a simple job generator where the attached transition delay represents an average inter-arrival time. There are many other features of CPN Tools not discussed that provide built-in functionality for interactive simulations, constructing state-space diagrams, gathering various statistics, and using external scripts [35] [75] [86].

2.1.6 Analyzing Petri Nets

The analysis of PNs is closely related to the analysis of concurrent systems, primarily because PNs are often used to model concurrency. Therefore, solutions for many of the classical problems found in concurrent systems are applied directly to the analysis of PNs, making properties such as boundedness, reachability, and liveness fundamental to PN analysis. These and other properties can be used to analyze the behavior and correctness of PN models, although many analytical questions concerning PN models are still open problems [59].

Two main categories of PN properties are those that depend on the initial marking, and those that do not. Properties that are dependent upon the initial marking are called behavioral or marking-dependent properties, while those that are independent of the initial marking are referred to as structural properties [65]. Many behavioral properties of PNs are closely related to each other and provide insight into the expected or desired behavior of a PN model. For example, the boundedness property of a PN is concerned with whether or not its set of all possible markings is finite. The graph that results from

constructing this set of markings is called a coverability tree and algorithms for determining boundedness based on coverability trees are well known [42, 72, 76]. A similar property, k-boundedness, involves determining if any place in a PN may ever contain more than k tokens. Determining if a PN is bounded or k-bounded is useful in identifying bottlenecks, overflow situations, and under-run conditions.

The reachability property of a PN is concerned with whether or not a given marking M_i is reachable from the initial marking M_0. This property is useful in determining if a system could possibly reach a desired/undesired state of execution, such as when stop criteria are met or a deadlock situation arises. The problem of determining the reachability of a PN is decidable[4] [47, 61], but it requires exponential time and space to verify in general [57]. Due to this significant shortcoming, reachability analysis of many practical systems is often limited to a subset of a PN model.

The liveness property of a PN determines for each reachable marking if there exists at least one enabled transition. The liveness property ensures, among other things, that a PN will never reach a marking that results in a deadlock condition. The problem of determining the liveness of a PN is recursively equivalent to the reachability problem, and requires exponential time to determine [22]. The computational requirement of determining liveness remains open and this restriction is impractical for applications on most systems. Therefore, a number of relaxed liveness properties have been defined so that portions of PN models can be more effectively analyzed.

Much of the pioneering work regarding the analysis of PNs is based on the previous results of vector addition systems [42], which are known to be logically equivalent to PNs [21]. Thus, vector addition systems play a key role in the incorporation of other

[4] A problem is decidable if there exists an algorithm that gives the correct answer for every input instance.

properties into PN analysis. The concepts of deadlock freedom, home spaces, promptness, fairness, persistence, and semi-linearity have also been incorporated into the PN analysis framework. Collectively, there are many known techniques, methods, and results for the analysis of PNs but much of this work is not applicable to real-world systems due to space/time requirements or other constraints imposed on the PN structure.

A number of other properties and results exist and collectively the general analysis techniques for PNs can be grouped into three main categories: a coverability/reachability method, a matrix-equation approach, and a reduction or decomposition technique [65]. The coverability method involves examining a coverability tree by recursively checking each enabled transition and recording the new marking created when the transition fires. In most cases, various properties regarding the PN behavior can be discovered during construction of the coverability tree. However, this tree grows infinitely large in an unbounded PN and therefore, is of little use. By modifying the tree construction method so that only reachable states are added, a reachability tree can be constructed instead. The reachability tree contains only reachable markings and is therefore, more useful in the application of PN analysis properties. Similarly, various operational laws (e.g., Little's Law, Forced Flow Law, Service Demand Law, and Utilization Law) can be used along with state-space information to compute performance metrics of interest [63]. However, due to the size and complexity of most distributed systems, the state-space of PN-based models quickly becomes intractable due to the common state-space explosion problem. Further, many distributed systems of interest are open systems, in which customers enter, spend some time in the system, and then exit. In describing such a

system, the corresponding state-space is infinite and cannot be enumerated, leaving only approximate solutions [62].

The remaining two methods (i.e., matrix-equation solving and reduction/decomposition techniques) are powerful but are often limited to specialized subclasses of Petri nets. For example, the matrix-equation approach involves transforming the PN model into a corresponding system of linear equations. Because such systems can be solved using matrix operations, this method is efficient but generally only provides necessary or sufficient information for either inferring desired properties, or ruling out dangerous conditions [12]. Similarly, reduction or decomposition techniques can be applied to PNs in order to create a simpler model while preserving the system properties to be analyzed. These techniques typically use transformations to convert PNs into smaller, simpler models but often their range of application is quite narrow.

Many transformation and synthesis methods exist to transform an abstracted model back into a more refined one [32, 87]. However, the problems of state-space explosion and the limited application of many PN analysis techniques and properties are significant shortcomings. Therefore, modeling tools typically also provide simulation capabilities that allow otherwise intractable models to be studied. These simulation tools lend themselves well to performance analysis of PN-based models, in particular because they are not hindered by the state-space explosion issue.

2.1.7 Modeling Example

As an example, consider the queuing network shown in Figure 13 that depicts a queuing network model of a closed system containing three jobs (i.e., the multiprogramming

level, MPL, is three). Jobs wait for a *scheduler* to route them to one of two servers, whichever has the fewest jobs. In the situation where *server1* and *server2* have the same number jobs, the tie is broken by sending the job to *server1*. A job is scheduled at an average rate of μ_s and then proceeds to either *server1* to be serviced at rate μ_1, or to *server2* to be serviced at rate μ_2. Each of the two servers can service only a single job at a time and after completing its service, a job returns to the *scheduler*.

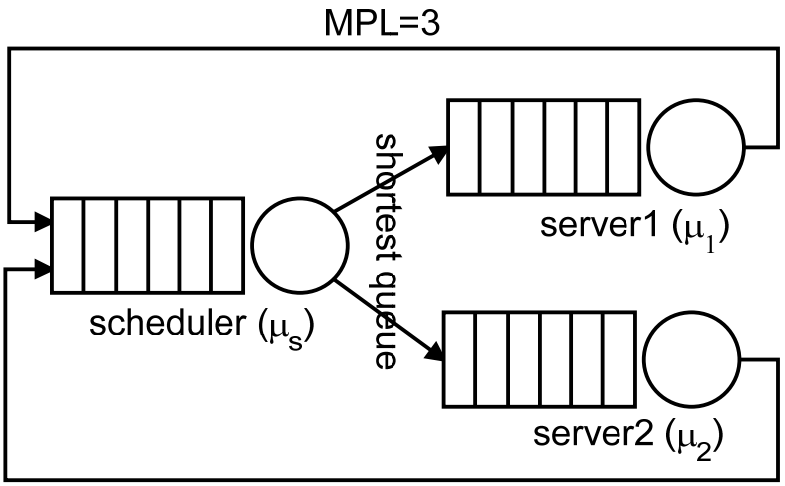

Figure 13: Example queuing network model

An equivalent PN model[5] of this example network is shown in Figure 14. Places p_1, p_4, and p_6 serve as wait queues where jobs must wait before being serviced. In the figure, two jobs are waiting on the *scheduler*, while the third job is currently being scheduled. When a job is processed by the *scheduler*, t_2 selectively places a token into p_3. The transitions t_3 and t_6 fire immediately if their respective guard conditions are satisfied. Transition t_3 fires when the number of jobs at *server1* is less than or equal to the number executing at *server2*. Similarly, when there are fewer jobs at *server2*, transition t_6 fires

[5] The model shown uses notation from multiple types of PNs to make the illustration more concise.

and routes the job to *server2*. An inhibitor arc drawn from p_2 to t_1 prevents a job from beginning the scheduling process if one is already being scheduled at p_2. Similar inhibitor arcs ensure no more than one job is ever executing at places p_5 or p_7. The timed transitions t_2, t_5, and t_8 correspond to completion events for the *scheduler*, *server1*, and *server2*, respectively.

Note that an explicit job queue is modeled for the *scheduler*, but if statistics involving job queuing at the *scheduler* are not important to the modeler, the inhibitor arc from p_2 to t_1 can be removed and places p_1 and p_2 can be combined. The inhibitor arc from p_5 to t_4 can be removed and places p_4 and p_5 can be combined if detailed information (e.g., the order of job completions) is not desired for *server1*. A similar operation can be performed for *server2* by removing the inhibitor arc from p_7 to t_7 and combining places p_6 and p_7.

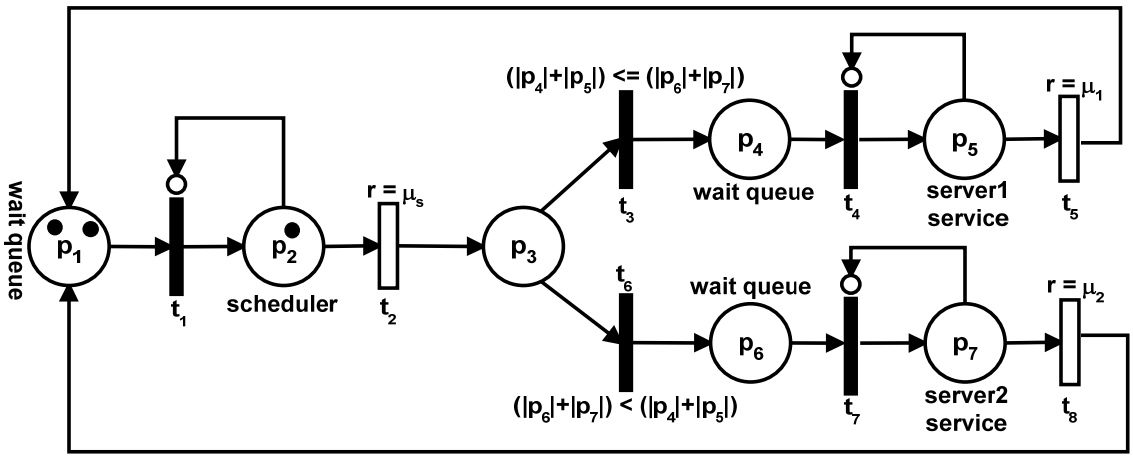

Figure 14: Equivalent PN diagram

From this PNM, a corresponding state-space diagram can be constructed and is shown in Figure 15. Each state descriptor indicates the number of jobs present in each

place in the PN diagram in Figure 14. For example, in state 110100, the number of jobs/tokens present in places *p1*, *p2*, *p3*, *p4*, *p5*, *p6*, and *p7* are one, one, zero, one, zero, and zero, respectively. In this state, two possible things can happen: a job can finish the scheduling process at the *scheduler*, or it can complete service at *server1*. A newly scheduled job proceeds to *server2* (i.e., state 010101) because *server2* has fewer jobs. If a job finishes at *server1*, it returns to the *scheduler* (i.e., state 210000). Notice that states such as 110001 are possible due to jobs completing service at a server. Due to the shortest-queue scheduling, such a state cannot be entered as a result of a job completing service at the *scheduler*. Note that because transitions t_6 and t_7 in Figure 14 are immediate, there are never any tokens for any length of time in P_3.

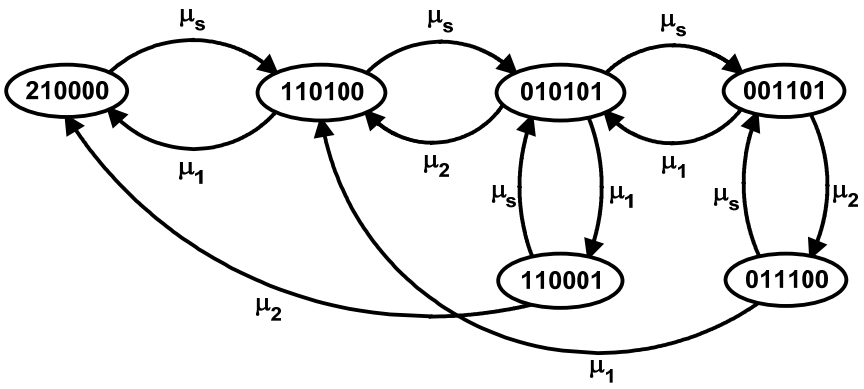

Figure 15: Equivalent state-space diagram

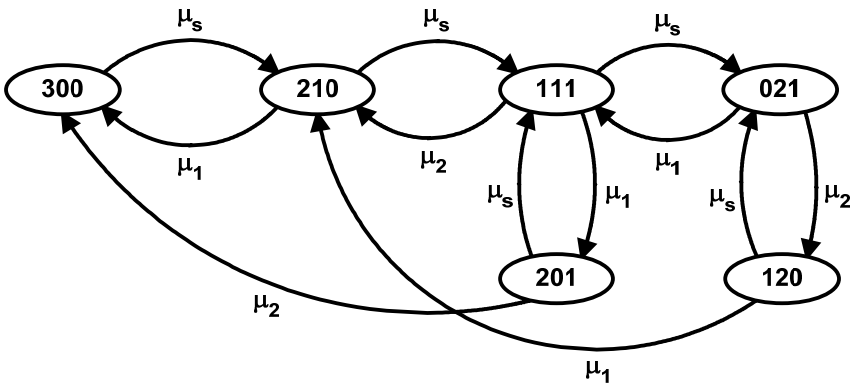

Figure 16: Equivalent state-space diagram with simplified descriptors

Due to the inhibitor arcs present in the PN diagram in Figure 14, there is never more than one job executing at a device at any given time. Any jobs arriving to a busy device must wait in the *wait queue* before receiving service. Therefore, the state descriptors used in the state diagram can be simplified by combining the number of waiting jobs and the number of executing jobs at each device. For example, in state 110100, there are two jobs present at the *scheduler* (i.e., one job waiting and one being scheduled), one job at *server1* (receiving service), and no jobs at *server2*. From the total number of jobs at a device, it is known how many jobs are waiting. Therefore, state 110100 can be described simply as state 210. The remaining state descriptors can be described and simplified in a similar manner. Figure 16 illustrates the state diagram that uses the shorter state descriptors. Both the short and long descriptors capture the same information, but shorter descriptors are sometimes easier to work with.

The state-space diagram shown in Figure 16 (or Figure 15) can be solved using a number of techniques. For example, according to the forced flow law, in steady-state, the flow for any state must be balanced such that incoming flow to the state equals the outgoing flow from the state [63]. Therefore, a balance equation can be written for each

state and the set of all of these equations can be solved to obtain the steady-state probability for each state. These balance equations are shown in Table 3, where P_s denotes the steady-state probability of being in state s. The last equation specifies that the sum of all the steady-state probabilities is one for an n-state diagram.

Table 3: Global balance equations for steady-state diagram

State	Balance Equation
300	$\mu_1 P_{210} + \mu_2 P_{201} = \mu_s P_{300}$
210	$\mu_s P_{300} + \mu_2 P_{111} + \mu_1 P_{120} = (\mu_s + \mu_1) P_{210}$
111	$\mu_s P_{210} + \mu_s P_{201} + \mu_1 P_{021} = (\mu_s + \mu_1 + \mu_2) P_{111}$
021	$\mu_s P_{111} + \mu_s P_{120} = (\mu_1 + \mu_2) P_{021}$
201	$\mu_1 P_{111} = (\mu_s + \mu_2) P_{201}$
120	$\mu_2 P_{021} = (\mu_s + \mu_1) P_{120}$
	$\sum_{i=1}^{n} P_i = 1$, for an n-state diagram

After obtaining the steady state probabilities, a number of performance metrics can be calculated. For example, the utilization of the *scheduler* can be calculated by noting the states in which there is at least one job present at the *scheduler*. Referring to Figure 16, there is at least one job present at the *scheduler* in all states except state 021. The utilization at the *scheduler* is, therefore, the sum of the probabilities of being in any state where at least one job is present at the *scheduler*. Put another way, the utilization at the *scheduler* is the portion of time the *scheduler* is idle, subtracted from 1, which is $1.0 - P_{021}$. As with all state-space analysis, analyzing most practical systems in this manner becomes intractable due to state-space explosion. Therefore, many approximation and simulation techniques exist for estimating these performance metrics.

It should be noted that once a PNM has been constructed, it can easily be modified to incorporate various changes in architecture. For example, the previous example assumes

there is one job type, and all jobs are identical. Suppose the test environment is modified so that multiple job types are introduced, *type1* and *type2*. Suppose *type1* jobs can access only *server1* and *type2* jobs can be serviced at either server. Suppose *server1* can only service a *type2* job at half the rate of a *type1* job. Although these modifications represent a significant change to the model description, only slight changes are required to the PNM to incorporate these new features. An additional token color can be added in order to identify and distinguish the two job types. To limit access for *type1* jobs to only *server1*, a constraint can be added to p_4 specifying that only *type1* tokens are allowed. To account for *type2* jobs being serviced at half the rate at *server1*, the firing rate of t_5 can be defined as a function of the job type that is currently executing.

2.1.8 Summary of Petri Net Modeling

Petri nets (PNs) are often exploited for their descriptive power and used as visual aids to help illustrate the intricacies of complex systems. PNs and their numerous extensions have been used for performance modeling in a wide variety of applications. Colored Petri Nets (CPNs) in particular have received a significant amount of attention because they exhibit an excellent compromise between readability and expressiveness. CPNs provide great flexibility in model development and with the help of CPN Tools, their resulting performance models can be analyzed both analytically and via simulation. Other classical analysis tools such as queuing networks and state diagrams continue to play an important role in modern performance analysis as they often help gain insights and identify important trends in system behavior.

2.2 Performance Modeling in Real-Time Systems

A real-time system is one in which explicit timing requirements place constraints on the scheduling and execution of tasks. Such constraints are typically defined in the form of deadlines, where a deadline represents the latest time for which a task should have completed its execution. The goal of real-time systems is often to miss as few deadlines as possible, and preferably to avoid them altogether. Therefore, performance modeling of real-time systems typically involves evaluating and comparing the performance of different scheduling algorithms.

Real-time systems are typically divided into two main categories: hard and soft real-time systems. In hard real-time systems, a missed deadline is unacceptable because it can lead to total system failure or other catastrophic events. Such systems are commonly found in airplane control systems, automobile electronics, and medical equipment. In soft real-time systems, a missed deadline is undesirable, rather than intolerable. Therefore, in these systems, the value or utility of a task completion typically decreases after its deadline elapses. Systems involving weather forecasting, displaying flight-plan information, and streaming media are examples of soft-real time systems.

Figure 17 graphically illustrates a deadline in both hard and soft real-time systems. For a hard deadline (a), an operation results in maximum value if it is completed anytime before the deadline occurs, but a completion after the deadline results in zero or negative value. Similarly, with a soft deadline (b), an operation results in maximum value if it is completed before its deadline time, but afterwards, the resulting value decreases according to some value function $V(t)$. (Here, the function $V(t)$ is defined only for values of t that occur after the deadline.) In many systems, there are also specialized deadline

semantics adopted. For example, deadlines in streaming media applications are sometimes termed isochronal or just-in-time (c), where a completion before or after the deadline time results in little or no value. That is, positive value is obtained only if an operation is completed within a short time window around the deadline. Advanced algorithms or buffer-and-hold techniques are sometimes used to treat these systems as specialized hard or soft real-time systems [43]. The exact meaning of hard and soft deadlines, as well as the function *V(t)*, is often application specific.

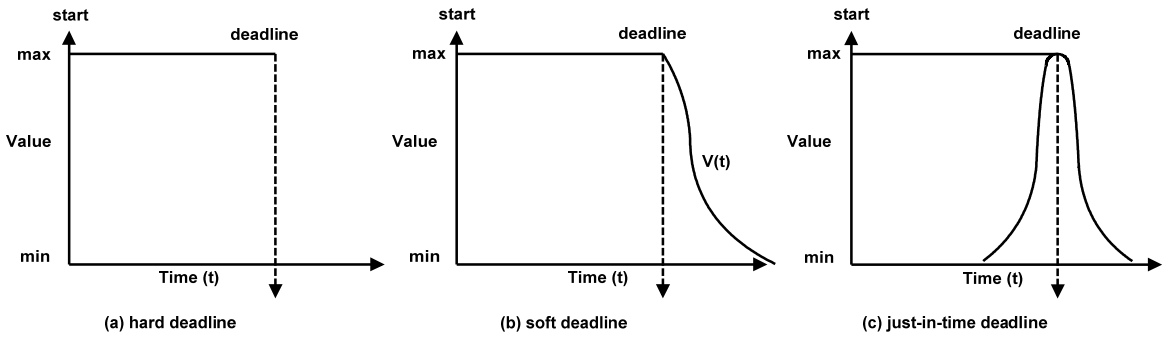

Figure 17: Hard, soft, and just-in-time deadlines

In this dissertation, soft real-time systems in which *V(t)=0* are studied. That is, some missed deadlines are acceptable, but there is no value in the service completion of a task after its deadline has elapsed. This type of system essentially applies the notion of hard deadlines to a real-time environment in which task (arrival, service, and deadline) behavior is aperiodic, and missed deadlines are acceptable. However, unlike in hard real-time systems, no attempt is made to guarantee that all the deadlines are met. Instead, the performance goal is to maximize the overall percentage of met deadlines in light of unpredictable task behavior. In this case, the occurrence of a deadline is only statistically known and is, therefore, *fuzzy*. The term *fuzzy real-time system* is not widely used in the

literature, but it typically refers to the use of fuzzy logic in a real-time environment. Therefore, the term *soft real-time system* is used loosely in the remainder of this dissertation.

2.2.1 Real-Time Scheduling

Some of the earliest results concerning scheduling in real-time systems date back to the early 1970's, when the study of priority scheduling strategies first became popular. With priority scheduling, each task is assigned a priority based on a policy or classification technique. When multiple tasks are ready to execute and there is contention for a resource (e.g., processor), the conflict is resolved by allocating the resource to the task with the highest priority.

Two main categories of priority scheduling algorithms are static priority and dynamic priority [12]. With static priorities, tasks are assigned priorities ahead of time and therefore, a priori knowledge is required to determine a task schedule. After task execution begins, the priorities of tasks and the schedule do not change. By contrast, with dynamic priorities, the schedule is determined at runtime by updating task priorities based on some well-defined policy. In general, static priority algorithms require less overhead and are easier to implement in practice, but provide less flexibility than dynamic priority algorithms because the latter can make adjustments on the fly in light of unexpected events. Therefore, dynamic algorithms provide greater potential for fewer missed deadlines.

2.2.2 Static Priority Scheduling

In the pioneering work of Liu and Layland [58], tasks are assumed to be periodic, independent (e.g., no blocking), and have constant service times, as well as deadlines that

coincide with their periods. For simplicity, only a single processor environment is considered. Their work introduced a static priority scheduling algorithm, known as Rate Monotonic (RM), that can always schedule a set of periodic tasks of any size such that no deadlines are missed, provided that the total system utilization remains less than 0.693 [58]. The general result for such a group of *n* periodic tasks is given by the expression in equation 1.

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \leq n\,(2^{\frac{1}{n}} - 1) \qquad (1)$$

As shown, for a task *i*, C_i denotes its computation time and R_i denotes its request time (i.e., period). That is, $\frac{C_i}{R_i}$ is the required processor utilization for task *i*. This expression defines an upper bound for the combined utilization of a set of *n* tasks, such that a feasible[6] schedule is not guaranteed for the set of tasks if their combined utilization exceeds this upper bound. For two tasks (i.e., *n=2*), the utilization bound is *ln*(2), which is approximately 0.828. Therefore, a pair of tasks is guaranteed to be feasible (i.e., schedulable) if their combined utilization does not exceed 0.828. As *n* approaches infinity, the utilization bound approaches the value 0.693 and the above result guarantees the existence of a feasible schedule only if the combined task utilization does not exceed 0.693. It should be noted that this is only a sufficient, and not, necessary condition. That is, there are sets of tasks having a combined utilization greater than 0.693 that can still be scheduled feasibly. Liu and Layland's work used only worst-case analysis. Lehoczky,

[6] A feasible schedule is one where all tasks meet their deadlines.

Sha, and Ding later showed that the average actual utilization is about 0.88 in most practical situations [52].

The progression of static-priority scheduling theory is a direct result of extending Liu and Layland's work in a number of ways. Some studies present results containing stronger feasibility conditions than Liu and Layland, such as Lehoczky, who examines tasks with deadlines greater than their periods [53]. Other work focuses on examining the worst-case behavior of metrics other than deadline times, such as response time [28]. The common goal of most of this research is to develop more widely applicable analysis methods for priority-based algorithms that are not restricted by the assumptions of Liu and Layland. Additional research focuses on topics such as non-periodic tasks [27, 55, 81], interdependent tasks [74, 78], and tasks operating in distributed environments [73-74]. It was only after most of this research that Rate Monotonic scheduling theory gained popularity and software developers took serious interest, thanks to renewed interest from researchers at IBM [18].

2.2.3 Dynamic Priority Scheduling

With dynamic priority algorithms, new issues arise that must be addressed. For example, because priorities are computed at runtime, an executing task must be stopped if its priority decreases to less than that of some other waiting task. This formerly executing task will be blocked until its priority again becomes the highest. There is also the increased overhead that comes with making scheduling decisions at runtime. Still, the importance of more flexible scheduling algorithms is recognized and this motivates research of dynamic priority algorithms.

A deadline-driven algorithm with dynamic priorities was introduced by Liu and Layland, commonly referred to as Earliest Deadline First (EDF). With EDF, priorities are dynamically assigned to tasks according to their approaching deadlines such that the task with the nearest deadline is assigned the highest priority. This algorithm is known to be optimal, in that, if a task set has a feasible schedule via any priority assignment algorithm, the EDF algorithm will also produce a feasible schedule, even if the system utilization approaches 1 [58]. It is also known that EDF is optimal among any preemptive scheduling algorithm [16]. Liu and Layland's result, listed in equation 2, is necessary, as well as sufficient. The EDF algorithm requires more overhead than RM due to the dynamic nature of priority updates, despite its apparently more appealing feasibility condition. Also, RM has been shown to be quite versatile and adaptable in practice [52].

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \leq 1 \tag{2}$$

Another dynamic priority algorithm, Least Laxity First (LLF), was introduced by Mok and is also known to be optimal [64]. With LLF, the laxity[7] of each task is recomputed each time there is a system change, such as a task arrival or completion. Priorities are assigned based on laxity, such that that the task with the least laxity is assigned the highest priority. The feasibility constraint for LLF is the same as shown in equation 2. While a task is executing, there may be other tasks with lower priorities waiting for their turn at the processor. Even though these tasks are blocked, their

[7] The laxity of a task is defined as the deadline time minus the expected remaining service time.

deadlines are still approaching and, therefore, their laxities and priorities must be updated continuously. When the priority of a blocked task becomes higher than that of an executing task, the executing task must be preempted, and the waiting task must be started. However, the same situation could occur again immediately. This process of priority toggling can continue in a repetitive manner and eventually lead to thrashing, where the processor spends more time performing context switches than it does performing meaningful processing. However, if the overhead due to context switching is ignored, LLF is known to be optimal [17].

To help alleviate or eliminate the priority toggling problem, a number of variant techniques have been introduced, such as the Stack Resource Policy (SRP) introduced in [9]. With SRP, the key observation is that jobs with long relative deadlines can delay, but not preempt, jobs with shorter relative deadlines. Therefore, the focus is placed on jobs with longer relative deadlines that block jobs with shorter relative deadlines. In this way, preemption levels and priority levels are defined separately so that a job's preemption level is inversely proportional to its relative deadline [9]. Under SRP, priorities are assigned based on absolute deadlines, as well as static preemption levels, and a job is not allowed to start executing unless its preemption level is high enough.

Both EDF and LLF have implementation overhead due to the context switching required when task priorities are updated. With EDF, deadline times have to be updated regularly in order to adjust task priorities correctly. With LLF, the deadline time as well as the remaining service time must be updated continuously in order to compute the laxity of each task. Due to the extra overhead incurred by LLF, much of the research that followed tends to focus on EDF scheduling. Another reason for this trend stems from the

desire to remove or strengthen the assumptions imposed on deadline driven scheduling, and extend its applicability. EDF is easier to analyze, making its generalization and application to a wider range of situations seem more feasible.

2.2.4 Other Scheduling Techniques

Many other scheduling techniques, both static and dynamic priority, have been introduced over the years. Much of this work is an extension or modification of earlier work done by Liu and Layland that resulted from removing or relaxing some of their assumptions. In the same way their RM results guided the progression of static priority scheduling, so too did their EDF algorithm guide studies in dynamic priority scheduling. For example, Spuri investigated using EDF to schedule non-periodic tasks [82-83]. Mok showed that EDF remains optimal when there is a combination of periodic and non-periodic tasks, provided a lower bound can be placed on the inter-arrival times of non-periodic tasks [64]. Further relaxations led to a number of hybrid scheduling techniques and allocation protocols. In [46], for example, Koren and Shasha discuss the optimality of EDF and RM variants that allow tasks to be occasionally skipped. This work provided a glimpse of future soft real-time scheduling topics that would later become popular.

Overall, research topics have progressed from the scheduling of single processor, periodic, independent tasks to that of present-day systems, where both periodic and non-periodic tasks can use multiple processors, may have dependencies, and can experience processor overload and network failures. Thus, this expansion of target environments has led to a wide range of focused scheduling topics. Protection against system overload, for example, is addressed by Abeni and Buttazzo with their Constant Bandwidth Server technique [2-3] and the need for mutual exclusion of resources is addressed by Baker et

al. with a number of access control policies [9, 41, 84]. Despite such advancement and expansion of scheduling topics, much of the research still focuses on hard deadlines and assumes worst-case behavior to support analysis methods. The worst-case assumption makes it easier to place bounds on deadline analysis but this also leads to pessimistic bounds for system utilization. Indeed, in systems where the expected utilization is significantly less than the worst-case behavior, the system performance can be greatly improved.

2.2.5 Scheduling in Soft Real-Time Systems

In soft real-time systems, a few missed deadlines are often acceptable. For example, in streaming media applications where frames of image data are sent across a network, an occasional dropped frame may be acceptable (e.g., when viewing a video). Assuming worst-case behavior in such systems leads to low system utilization due to the imposed pessimistic constraints. That is, assuming every frame of data will require the maximum possible utilization assures all frames of data are eventually delivered, but this comes at the cost of extended wait times. On the other hand, assuming that all frames will require only the average utilization will lead to about half of the frames being delivered on time—the half that requires the average utilization. However, the other half of these frames requires (on average) more than the average utilization and, therefore, these frames are dropped. In such systems, it is often a tradeoff between the expected number of missed deadlines and the expected system utilization.

Due to the past emphasis on hard deadlines, previous work related to scheduling in soft real-time systems is scarce by comparison. Algorithms such as Stochastic Rate Monotonic Scheduling (SRMS) [6] and the development of Real-Time Queuing Theory

[54] were developed with soft real-time systems in mind and other studies have followed. However, research dealing with soft deadlines has not been as rapid to follow as its hard deadline counterpart has and this commonly leads to the modification of existing methods to meet new deadline requirements. Accordingly, many present-day schedulers utilize RM, EDF, or LLF scheduling algorithms, or close variants. Much of the latest research in the area of soft real-time scheduling focuses on the development of hybrid scheduling techniques, where traditional algorithms are being adapted for use in modern application environments. For example, in [68], the Quality of Service (QoS) guarantees in a streaming media application are improved by using elastic priorities along with RM scheduling to reduce task starvation. In [56], the success ratio of EDF is improved by forming dynamic groups of tasks, where each group of tasks is scheduled using a secondary algorithm. Another example from Abeni considers EDF scheduling along with task skipping to improve performance in multimedia applications [2]. In [29], the deadline miss ratio of periodic tasks scheduled using LLF is reduced by using fuzzy inference tables to determine task priorities. Scheduling in soft real-time systems remains an active and open area of research.

2.2.6 Summary of Real-Time Scheduling

Performance modeling of scheduling algorithms in real-time systems has advanced significantly since the first static and dynamic priority-based algorithms were introduced in the 1970's. Classic algorithms such as RM, EDF, and LLF have been the focus of numerous studies involving periodic, independent tasks. These scheduling algorithms are known to be optimal, under certain conditions, in hard real-time environments and are the basis for many of the current real-time scheduling algorithms in use today. Despite the

advancement and expansion of scheduling topics, much of the latest real-time scheduling research still focuses on hard deadlines and assumes worst-case behavior. Accordingly, many present-day schedulers still utilize the RM, EDF, or LLF scheduling algorithms, or close variants. Therefore, it is beneficial to improve the performance of these traditional algorithms as they often serve as the framework for new, hybrid routines. In this dissertation, the effects of variability on the performance of traditional algorithms are studied and used to develop a new hybrid algorithm with improved performance.

2.3 The Method of Stages

The exponential distribution has been used to model a wide range of aspects of the real-world, from satellite constellations to woodpecker attacks on power poles [4]. In performance analysis, the exponential distribution is often used because of its ease of use and memoryless property. Using an exponential distribution simplifies the analysis techniques by eliminating the need to explicitly model time due to its appealing Markovian properties. For example, non-constant service times are often modeled by exponential distributions because the knowledge of a service start-time does not affect when the service will end. However, in reality this is often not a practical assumption for many realistic workloads, especially those of distributed environments which sometimes demonstrate sporadic or heavy-tail behavior [26]. Thus, it is sometimes desirable and useful to use distributions other than the exponential distribution, and yet retain some of its appealing analysis properties. This can be done using a general phase-type distribution, which is also called a stage-type distribution. The technique of using stage-type distributions to model performance parameters is loosely referred to as the method

of stages. Thus, the method of stages can be used in a number of situations when exponential distributions are not appropriate.

2.3.1 Stage-Type Distributions[8]

Stage-type distributions get their name from the fact that they can be represented as a combination of exponential stages. The exponential distribution itself can be thought of as a stage-type distribution consisting of only one stage. The exponential distribution has a probability density function (pdf) of $f_X(x)=\mu e^{-\mu x}$ for $x \geq 0$, expected mean of $1/\mu$, and variance of $1/\mu^2$. Consider a random variable *A* that represents the service time of a job at a disk, where *A* is exponentially distributed with parameter $\mu>0$. Figure 18 illustrates this graphically, where a single stage is represented by a circle containing the exponential parameter, μ. A job will arrive at the disk, receive service for an amount of time sampled from *A*, and then exit. Because the values of *A* are exponentially distributed, the expected mean of the service time is $1/\mu$ and its variance is $1/\mu^2$.

Now suppose the disk instead services jobs in a sequence of two identical stages as shown in Figure 19. Here, the service time is sampled from a random variable *B*, where *B* is exponentially distributed with parameter 2μ. In Figure 19, an arriving job receives an amount of service in stage 1 that is sampled from *B*, followed by a second, independent amount of service sampled from *B* in stage 2. Note that in order to match the mean value ($1/\mu$) from the single stage exponential distribution (for variable *A*), each of the two stages shown in Figure 19 (for variable *B*) must use a parameter of 2μ. The disk operates in one stage or the other, but not both. Therefore, a job spends an amount of time randomly chosen from a pdf $f_B(b)$, followed by another independent amount of

[8] The summary given in this section is based on the discussion provided in [85].

time chosen from $f_B(b)$. The expression for the total amount of service received by the job is the sum of the two service amounts received in each stage, which is the sum of two independent and identically distributed exponential random variables. Let B be an exponentially distributed random variable with parameter 2μ and let $T=B+B$, where T represents the total service time across both stages.

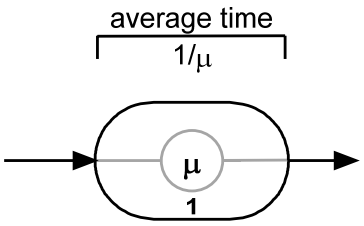

Figure 18: A single exponential stage

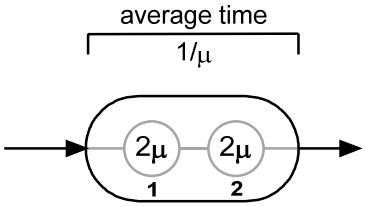

Figure 19: Two exponential stages in sequence

To find the pdf for T, first consider the general case of two continuous independent random variables X and Y, and let $S=X+Y$. Assuming that X and Y are each exponentially distributed with parameter λ, the pdf for X is $f_X(x)=\lambda e^{-\lambda x}$ and the pdf for Y is $f_Y(y)=\lambda e^{-\lambda y}$. Using the convolution formula for independent (continuous) random variables and the fact that $y=s-x$, the pdf for S is given by [85]:

$$\begin{aligned} f_S(s) &= \int_{-\infty}^{\infty} f_X(x)\, f_Y(s-x)\, dx \\ &= \int_{0}^{s} f_X(x)\, f_Y(s-x)\, dx, \text{ for nonnegative x and y.} \end{aligned}$$

In the more specific case, when $S=X+X$, and the samples taken from X are nonnegative, the pdf for S is given by:

$$\begin{aligned} f_S(s) &= \int_{0}^{s} f_X(x)\, f_X(s-x)\, dx \\ &= \int_{0}^{s} [\lambda e^{-\lambda x}]\, [\lambda e^{-\lambda(s-x)}]\, dx \end{aligned}$$

$$= \lambda^2 \int_0^s e^{-\lambda x - \lambda s + \lambda x}\, dx$$

$$= \lambda^2 e^{-\lambda x} \int_0^s dx = \lambda^2 x e^{-\lambda x}, \text{ for } x \geq 0.$$

For the sum $S=X+X$, the rate parameter is λ and the resulting pdf for S is given by $\lambda^2 x e^{-\lambda x}$. Therefore, the pdf for the previous expression for $T=B+B$ can be found by replacing λ (in the expression for the pdf of S) with 2μ, resulting in $(2\mu)^2 x e^{-(2\mu)x} = 4\mu^2 x e^{-2\mu x}$. Thus, the pdf for a distribution consisting of two consecutive exponential stages is $4\mu^2 x e^{-2\mu x}$ and the mean and higher moments can be found using Laplace transforms [85], resulting in a mean of 1/μ and a variance of $1/(2\mu^2)$. Therefore, the mean of the distribution resulting from splitting it into two stages does not change, but the variance is reduced by 50%.

2.3.2 Common Stage-Type Distributions

The technique of using a sequence of exponential stages can easily be generalized to the case where there are a succession of k identical, but independent, exponential stages with parameter $k\mu$ as shown in Figure 20. The resulting distribution is typically referred to as a k-stage Erlang distribution (denoted Erlang-k) [67]. A job receiving service via this type of distribution must spend k consecutive intervals of time, each selected from an exponential distribution with parameter $k\mu$, before its service is completed. During this time, no other job can receive service from the same server, and the job cannot leave until all k stages have been completed. The resulting mean of the Erlang-k distribution is 1/μ and the variance is $(1/k)(1/\mu^2)$, where k represents the number of stages. Table 4

summarizes the properties of an Erlang-k distribution. From the table, it is seen that the CV[9] of an Erlang-k distribution is always < 1 (for k>1).

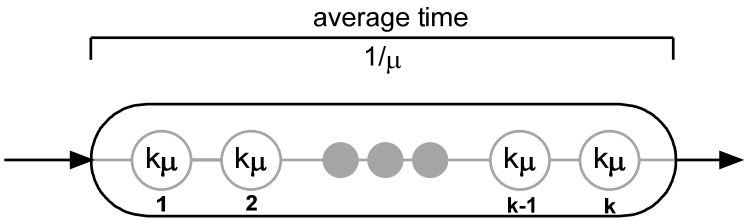

Figure 20: The Erlang-k distribution

Table 4: Properties of the Erlang-k distribution

pdf	Mean	Variance	CV
$\frac{(k\mu)^k}{(k-1)!}x^{k-1}e^{-k\mu x}$	$\frac{1}{\mu}$	$\frac{1}{k}\left(\frac{1}{\mu}\right)^2$	$\frac{1}{\sqrt{k}}$

Using this approach, it is possible to use a sequence of exponential stages to model performance parameters that have less variability than that of the exponential distribution, and yet maintain its desired mathematical properties (e.g., memoryless property). However, using a single sequence of stages, the choice of variance is limited to a discrete set because only multiples of 1/k for integer k are possible (where the multiplier is $1/\mu^2$). One way to overcome this problem is to use a mixture of Erlang-(k-1) and Erlang-k distributions, resulting in distributions with variances ranging from $\frac{1}{k-1}\left(\frac{1}{\mu}\right)^2$ to $\frac{1}{k}\left(\frac{1}{\mu}\right)^2$. Another technique is to relax the constraint on the parameter kμ of each stage, allowing each stage s_i to have its own parameter μ_i. This leads to the hypoexponential distribution, as illustrated in Figure 21.

[9] The coefficient of variation (CV) is the ratio of the standard deviation to the mean.

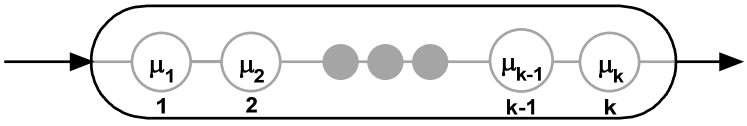

Figure 21: The hypoexponential distribution

Table 5: Properties of the hypoexponential distribution

pdf	Mean	Variance	CV
$\sum_{i=1}^{k} a_i \mu_i e^{-\mu_i x}$, where $a_i = \prod_{j=1, j\neq i}^{k} \frac{\mu_j}{\mu_j - \mu_i}$	$\sum_{i=1}^{k} \frac{1}{\mu_i}$	$\sum_{i=1}^{k} \frac{1}{\mu_i^2}$	$\frac{\sqrt{\sum_{i=1}^{k} \frac{1}{\mu_i^2}}}{\sum_{i=1}^{k} \frac{1}{\mu_i}}$

The properties of the hypoexponential distribution are summarized in Table 5. The pdf results from the convolution of k exponential pdf's, each of which has its own rate parameter, μ_i. The expected mean of the hypoexponential distribution is given by $\sum_{i=1}^{k} \frac{1}{\mu_i}$, while its variance is equal to $\sum_{i=1}^{k} \frac{1}{\mu_i^2}$. From Table 5, it is seen that the denominator in the expression for the CV must be greater than or equal to the numerator and therefore, the coefficient of variation of the hypoexponential distribution is ≤ 1.0. Using this distribution provides greater flexibility in matching the real-world, observed variance of performance parameters.

The previous techniques can be used to match the first two moments of performance parameters having CV's less than 1.0. However, to model parameters with an observed CV greater than 1.0, a new but similar approach must be taken. To accomplish this, branching probabilities can be introduced so that either of two exponential stages can be

visited, but not both. Figure 22 shows this configuration, where with probability α_1 a job will visit the top stage and receive service at rate μ_1, or with probability $1-\alpha_1$ it will visit the lower stage and receive service at rate μ_2. Only one stage is active at any given time, and after completing service at one of these stages, a job exits the system. In a manner similar to the previous discussion, Laplace transforms are used to obtain the expressions for the mean and variance, which are given by $\frac{\alpha_1}{\mu_1}+\frac{1-\alpha_1}{\mu_2}$ and $\frac{2\alpha_1}{\mu_1^2}+\frac{2(1-\alpha_1)}{\mu_2^2}-\left(\frac{\alpha_1}{\mu_1}+\frac{1-\alpha_1}{\mu_2}\right)^2$, respectively.

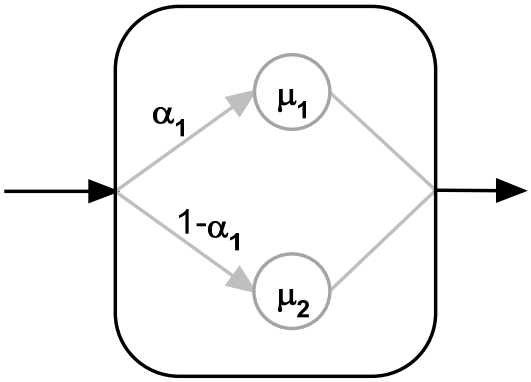

Figure 22: Two exponential stages in parallel

This stage-type distribution can be generalized by increasing the number of stages, resulting in the hyperexponential distribution shown in Figure 23. A starting job will take exactly one of the available paths determined by the branching probabilities. The mean of the hyperexponential distribution is given by $\sum_{i=1}^{k}\frac{\alpha_i}{\mu_i}$ and its variance is given by $\left(2\sum_{i=1}^{k}\frac{\alpha_i}{\mu_i^2}\right)-\left(\sum_{i=1}^{k}\frac{\alpha_i}{\mu_i}\right)^2$. The properties of the hyperexponential distribution are

summarized in Table 6. The Cauchy-Schwartz inequality can be used to show that the CV is ≥ 1.0 [85].

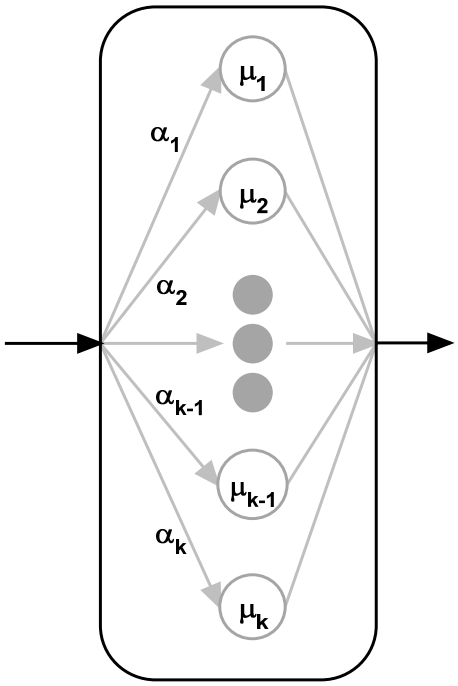

Figure 23: The hyperexponential distribution

Table 6: Properties of the hyperexponential distribution

pdf	Mean	Variance	CV
$\sum_{i=1}^{k} \alpha_i \mu_i e^{-\mu_i x}$	$\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i}$	$\left(2\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i^2}\right) - \left(\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i}\right)^2$	$\frac{\sqrt{\left(2\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i^2}\right) - \left(\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i}\right)^2}}{\sum_{i=1}^{k} \frac{\alpha_i}{\mu_i}}$

2.3.3 The Coxian Distribution

It is also possible to combine hypoexponential and hyperexponential distributions to obtain representations that are more complex. Cox shows how any distribution with a rational Laplace transform can be represented as a sequence of exponential stages [15].

This series of stages can be represented using branching probabilities that permit a job to exit the system after any given stage. Such a distribution is commonly referred to as the Coxian distribution and is illustrated in Figure 24. After receiving service in the first stage, a job will continue to the next stage with probability α_1 or with probability $1-\alpha_1$ it will exit the system, bypassing the remaining stages.

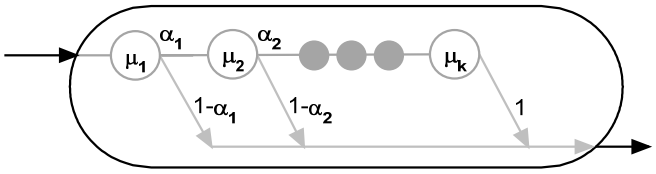

Figure 24: The Coxian distribution

From Figure 24, it can be deduced that with probability $\alpha_1(1 - \alpha_2)$, a job will receive service in the first two stages and then exit the system. Continuing this reasoning, the probability p_j that only the first j stages will be completed before a job exits is given by the expression $p_j = (1-\alpha_j)\prod_{i=1}^{j-1}\alpha_i$. Therefore, the Coxian distribution can also be represented as a probabilistic choice from among k hypoexponential distributions as shown in Figure 25. This distribution is typically referred to as the extended Erlang distribution. Overall, a Coxian distribution can be used to match any CV of any distribution by choosing the appropriate number of stages, branching probabilities, and rate parameters [85].

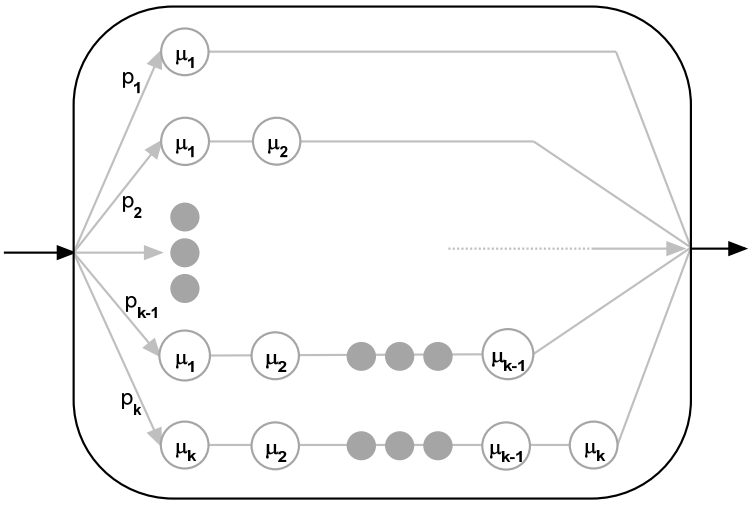

Figure 25: The extended Erlang distribution

2.3.4 Advantage of Stage-Type Distributions

The use of stage-type distributions provides a convenient method of matching the observed characteristics of performance parameters while maintaining the appealing analysis properties of the exponential distribution. The key advantage of using stage-type distributions is that the concept of time is represented discretely by a number of stages, rather than in a continuous manner. By focusing on individual stages, rather than on an entire distribution, the notion of time is effectively captured by simply noting which stage of a given distribution is currently active. Because each stage has the desirable memoryless property, the overall arrangement of stages has the memoryless property as well. This leads to discrete, rather than continuous, state diagrams and simplifies the resulting discussion and analysis methods.

The method of stages can be applied to many situations where a single exponential distribution is not appropriate. Using stage-type distributions provides a direct method of modeling the higher moments of performance parameters while holding the mean

constant. This allows focused performance analysis studies to be conducted in a uniform manner that is applicable to both analytical and simulation techniques. State diagrams can be constructed for small or simple models and analyzed in order to gain insights into the behavior of a system. When the analysis of state diagrams becomes intractable, simulation techniques can be applied that also utilize the method of stages modeling approach.

2.3.5 Matching Higher Moments

The work presented in this dissertation focuses on the study of the second moment (i.e., variance) and its effects in performance modeling. However, the general technique of using stage-type distributions to model the behavior of performance parameters can be used to match third, fourth, and higher moments as well. For example, analytical expressions can be derived for Erlang distributions that match the third moment of any distribution [38]. The first three moments of a distribution can also be matched using only Coxian distributions [39].

The generalized family of distributions, commonly known as a Johnson distribution, is based on a transform of the normal distribution and includes normal, lognormal, bounded, and unbounded forms. This fitting technique provides great flexibility in the choice of fitting parameters and it can be used to match the first four moments of virtually any distribution [40]. A significant drawback to this approach, however, is that the distribution forms are not memoryless, and therefore, time must be modeled in a continuous manner. This prevents the use of discrete Markov analysis and therefore, restricts the application of analytical techniques (e.g., discrete state-space diagrams).

In [31], an iterative technique is presented that uses acyclic stage-type distributions to match an arbitrary number of moments. When using stage-type distributions such as these, the memoryless property of the exponential distribution can be taken advantage of to represent time discretely. The common problem with these and other techniques is often the computational complexity involved in determining the various parameters for the modeled distributions. In general, the numerical complexity of the solution increases as the number of desired moments increases and, therefore, applying such techniques to real-world systems can be challenging.

2.3.6 Modeling Technique

The method of stages is used to effectively approximate the characteristics of task parameters such as inter-arrival times, service times, and deadline times. This is done by using three separate processes to represent each task. An arrival process is used to model the arrival characteristics of the task, while the service and deadline processes are used to model the execution and deadline behaviors, respectively. In effect, each process corresponds to the value of a performance parameter that is divided into a fixed number of stages, where each stage is modeled using an exponential distribution. This is achieved by using stage-type distributions, where each process is modeled by a k-stage Erlang distribution [85]. The arrival, service, and deadline processes operate independently based on their defined distribution specifications.

Within the arrival process, each stage is statistically identical and the amount of time spent in any given stage is exponentially distributed. Recall Table 4 which shows the probability density function (pdf), mean, variance, and coefficient of variation (CV) for a k-stage Erlang distribution, where k is a positive integer. For k=1, the pdf reduces to that

of an exponential distribution having a CV of 1.0. As the value of k increases, the variance of the distribution decreases. As k approaches infinity, the variance approaches zero which characterizes a completely deterministic process. Therefore, by adjusting the value of k, a wide range of distribution behaviors can be captured and modeled, from one that is completely deterministic with low variance (i.e., CV=0) to one that is exponential with relatively high variance (i.e., CV=1.0). For distributions that are more complex, Coxian distributions can be used to approximate the actual distribution to any desired level of accuracy. The tradeoff is between the accuracy of the model and its complexity.

The explicit modeling of the variance in this manner comes at a cost of increased overhead due to extra computation and storage requirements of the underlying detailed state information. Because exact modeling of a completely deterministic distribution would require an infinite number of stages, such a distribution can instead be approximated using a sufficiently large (e.g., 50) number of stages. More importantly, the behavior of many realistic distributions with low variance can be approximated using only a small number of stages. Using this technique, the mean and variance of the inter-arrival, service, and deadline time distributions for a given task stream can be accurately modeled.

The method of stages can be used to completely describe a system's workload and lends itself well to Markovian analysis. At any given time, each task can be described by specifying the current stage of each of its arrival, service, and deadline processes. A combined description of all tasks in the system provides a complete representation of the current system state. Therefore, a list of all possible states can be enumerated and a state-space diagram can be constructed. As discussed previously, a number of analysis

methods can be applied to state-space diagrams to determine various performance metrics [63]. However, due to the state-space explosion problem, simulation methods are often used to approximate these values. A special-purpose simulation tool has been developed that uses the method of stages for its simulation engine [23].

2.3.7 Workload Representation

A workload in a real-time system is represented by a group of task streams, each of which corresponds to a collection of tasks that are all statistically identical. Within each stream, tasks are characterized by an arrival, service, and deadline process. Each of these three processes is composed of a number of stages that defines the progress of each corresponding process. The number of stages within each process is indirectly proportional to the variance of the process. For a given task, these processes compete with each other and progress through each of their stages until one of them reaches completion. Similarly, processes belonging to different task streams also compete with each other. For example, the service processes of task streams compete against each other for scarce resources (e.g., processors).

The stages of the arrival process are used to model the task inter-arrival time. When the arrival process completes its last stage, a new (actual) arrival occurs. When this happens, the service and deadline processes are both started, and the arrival process restarts, indicating the progress of the next arrival. The service stages model the task service/execution time, where an actual service completion (i.e., met deadline) occurs when the final service stage is completed before the final deadline stage. Thus, the deadline stages model the progression of a deadline process that represents an approaching deadline that should be met. When a deadline process completes its last

stage, the deadline time has elapsed and if the corresponding service process has not already finished its last stage, the task misses its deadline.

Figure 26 illustrates the representation of a single example task stream consisting of an arrival, service, and deadline process. In this example, there are four arrival stages, two service stages, and three deadline stages. For the arrival process, ¼ of the total inter-arrival time is spent in each stage, whereas ½ of the total service time is spent in each of the two service stages. For the deadline process, ⅓ of the total deadline time is spent in each of the three deadline stages. Therefore, in terms of rates, each arrival stage must operate at four times the overall arrival rate, whereas each service stage must operate at twice the overall service rate. Similarly, each deadline stage operates at three times the overall task deadline rate.

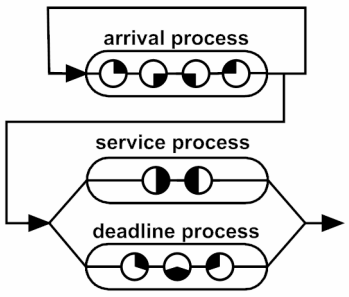

Figure 26: Example task stream representation

In this example, the deadline process is explicitly represented, meaning it directly competes against the arrival and service processes. However, sometimes the deadline process is implicitly represented by the arrival process. For example, the Method Of Stages Simulator (MOSS) (see Chapter IV) allows the linking between the arrival process and the deadline process [23]. When such linking is specified, the deadline time is implicitly modeled in the arrival process, where a new arrival corresponds to the deadline

of the previous arrival. In this case, the deadline process *is* the arrival process. Note the distinction between this situation and one where a deadline process has identical distribution characteristics as the arrival process. In the latter situation, it is possible that the deadline process finishes before the arrival process. In the former, this is not possible. The situation where the deadline process *is* the arrival process is representative of many practical soft real-time systems, such as a weather satellite, where the data is periodically collected. This data should be processed and reported before the next (more current) data arrives from the satellite. Thus, the arrival of new data corresponds to the deadline of the previous data.

2.3.8 State-Space Representation

Consider a system with two task streams, S1 and S2. Suppose that S1 and S2 are characterized by their workload parameters shown in Table 7. Assume that the variance of the arrival rate for stream Si is $\frac{1}{2}\left(\frac{1}{\lambda_i}\right)^2$. Thus, the arrival process of each stream can actually be modeled by an Erlang-4 process. Similarly, the variance of the service rates for stream S1 and S2 is $\frac{1}{4}\left(\frac{1}{\mu_1}\right)^2$ and $\frac{1}{3}\left(\frac{1}{\mu_2}\right)^2$, respectively. Therefore, the service process for each stream can be modeled using an Erlang-4 and Erlang-3 process, respectively. The deadline process for each stream uses four stages and can be modeled similarly. Note that in practice the measured or estimated variance of the performance parameters would be used to choose an appropriate value for the number of stages.

Because the mean inter-arrival time for S1 is $1/\lambda_1$, each of the two stages of the arrival process for S1 operates for a mean time of $1/2\lambda_1$. That is, each stage in the arrival

process of S1 operates at rate $2\lambda_1$. The arrival process for S2 is identical, except that its arrival rate, λ_2, is used. The service time for S1 is $1/\mu_1$ and therefore, each of the four stages of its service process operates for a mean time of $1/4\mu_1$, where each stage operates at rate $4\mu_1$. Each stage of the service process for S2 operates for a mean time of $1/3\mu_1$, or a rate of $3\mu_1$. The formulation of the deadline processes is similar.

Table 7: Example task stream workload parameters

Task Stream	Arrival Rate	Service Rate	Deadline Rate
S1	λ_1	μ_1	δ_1
S2	λ_2	μ_2	δ_2

Figure 27 illustrates an example state descriptor that captures one specific state of the two task streams S1 and S2. This illustration is a simplified version of that used in Figure 26. The state descriptor illustrates the number of stages of each process for each task. The top half of the state shows information for S1, while the bottom half shows information for S2. A filled circle in a process indicates the current/active stage and the absence of a filled circle in a process indicates that process has not yet started. The state shown in Figure 27 represents the initial state (State 1), when the arrival process of each stream is in its first stage, and the service and deadline processes of each stream have not yet started.

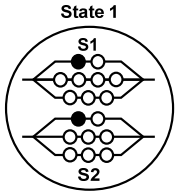

Figure 27: Example state descriptor

Using these state descriptors, a complete state diagram can be constructed, from which the overall system status is easily identified. Figure 28 shows a portion of the state diagram that illustrates the possible state transitions from the initial state (State 1). From State 1, the arrival process of S1 can progress to its next stage at rate $2\lambda_1$, causing the system to move to State 2. Alternatively, the arrival process of S2 can progress to its next stage at rate $2\lambda_2$, resulting in the system changing to State 3. From State 2, the system can move to State 4 if the arrival process of S1 completes its second stage, or to State 5 if the arrival process of S2 completes its first stage. In State 4, the arrival process of S1 has just completed both of its stages. This represents the arrival of a new S1 task to the system. Consequently, the service and deadline processes of S1 become active and the arrival process is restarted at stage 1.

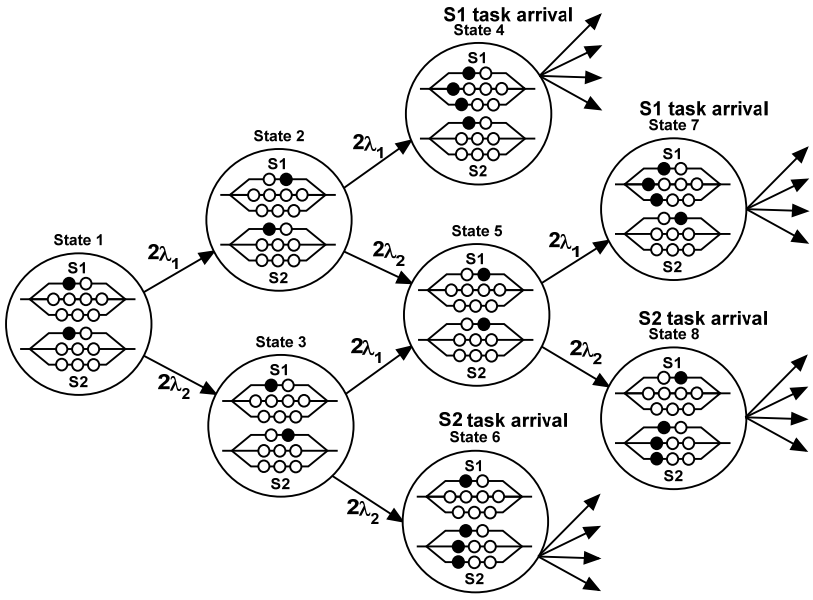

Figure 28: Example task arrivals

After an arrival for a stream has occurred, the service and deadline process for this stream begin executing in their first stage. Figure 29 shows a portion of a state diagram where State 1 indicates the service process of each stream is in its second stage. Note that the single (bottom) incoming arc for State 1 corresponds to an S2 event (i.e., the service process for S2 moving to its second stage). The diagram illustrates the possible state transitions that can lead to a service completion (State 13) or deadline expiration (State 11) for S1. Consider EDF scheduling, where the task with the earliest expected deadline is allowed to execute whenever multiple service processes are able to execute. Assume that the mean execution time for each stage of each process is identical, which for the deadline processes, is equivalent to assuming $3\delta_1=3\delta_2$. This assumption simplifies the discussion by allowing the reader to determine the earliest expected deadline for a task by simply counting its remaining number of deadline stages. In State 1 of Figure 29, S1 has two remaining deadline stages to execute and S2 has three remaining deadline stages to execute. Because the execution time of each stage is assumed identical, the remaining time until its respective deadline is less for S1. Therefore, the service process for S1 is allowed to execute in State 1, resulting in the system state described in State 3. However, S2 is not allowed to execute and therefore, there is no transition leaving from State 1 that corresponds to the progression of the service process of S2. The arrival and deadline processes for S2 continue because their progression is not dependent upon the service process execution. In this figure, the two transitions corresponding to the progression of arrival and deadline processes for S2 are shown as small arcs leaving the bottom of some states.

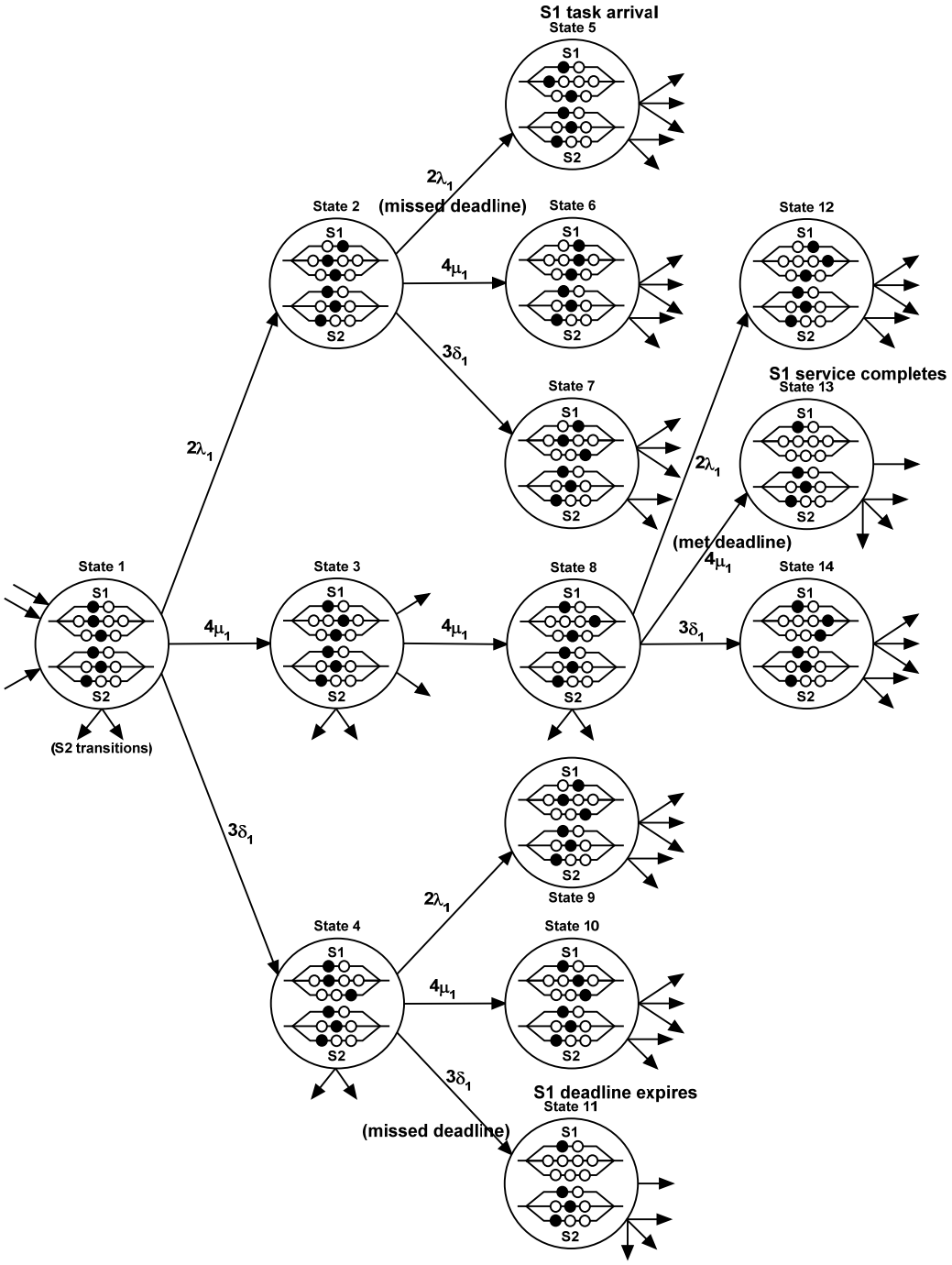

Figure 29: Example service completions and missed deadlines for EDF

In State 1, the arrival, service, and deadline processes of S1 are in stages 1, 2, and 2, respectively. The arrival process can finish a stage at rate $2\lambda_1$, leading to State 2. The

service process can finish a stage at rate $4\mu_1$, leading to State 3. The deadline process can finish a stage at rate $3\delta_1$, leading to State 4. In State 2, if the arrival process completes its current stage (indicated by the top solid circle), the arrival process completes. This indicates the arrival of a new S1 task. Thus, in State 5, the arrival, service, and deadline processes of S1 are reset to their first stages. This represents the event that the previously executing S1 task missed its deadline due a subsequent S1 arrival.

In State 8, the service process of S1 is executing in its final stage. If the service process completes the stage, the task successfully completes execution. This models the event that the task's service requirements are satisfied before its deadline expiration, indicating the task met its deadline (State 13). When a task meets its deadline, its service and deadline processes are terminated. In State 4, the deadline process for S1 is in its final stage. If the deadline process finishes its current stage, the deadline for S1 expires, resulting in a missed deadline (State 11). When a task misses its deadline, any remaining service stages are terminated.

When the service processes of more than one task stream are ready to execute, a scheduling algorithm is used to determine which task should be given priority and assigned the processor. Figure 30 shows the partial state diagram when the RM scheduling algorithm is used, assuming that $\lambda_1 > \lambda_2$. With RM scheduling, the processor is always given to the task with the largest arrival rate (i.e., smallest inter-arrival time). Therefore, in this example, priority is given to S1 whenever the service processes of both S1 and S2 are ready to execute. Therefore, in any given state, the service process of only one stream can be executing, either at rate $4\mu_1$ if an S1 task is present, or at rate $3\mu_2$ if only an S2 task is present.

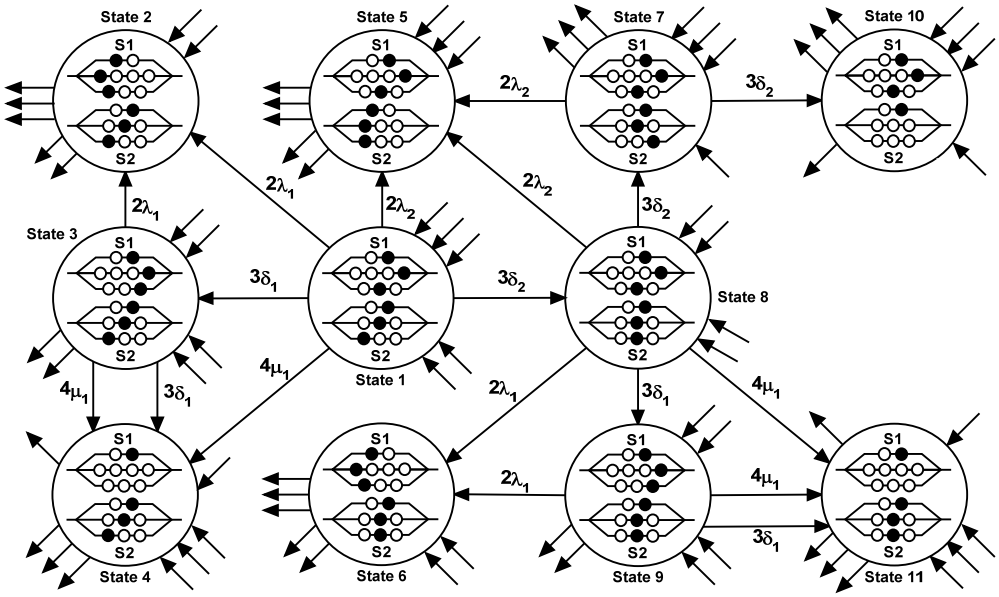

Figure 30: Partial state diagram assuming RM scheduling

In State 1, tasks from both S1 and S2 are present. Therefore, the processor is given to the S1 task. The arrival process for S1 is executing in its last stage at rate $2\lambda_1$. If it finishes its current stage, it will terminate, indicating a new task arrival for S1 (State 2). In this case, the newly arriving task preempts the currently executing task, causing it to miss its deadline. The arrival, service, and deadline process of the new task begin at their first stage.

Figure 31 shows a partial state diagram if the EDF scheduling algorithm is used. Again, to keep the discussion simple, the assumption is made that the mean execution time for each stage of each deadline process is identical. Therefore, the stream with the earliest expected deadline can be determined in any state by simply counting the number of remaining deadline stages. In State 1, the deadline process for S1 is executing in its second stage, while the deadline process for S2 is executing in its first stage. Therefore, there are two deadline stages remaining for S1 and three deadline stages remaining for

S2. In this state, the S1 task is given the processor because it has the earliest expected deadline. For S1, the arrival of a new task occurs at rate $2\lambda_1$, causing the currently executing task to miss its deadline. The new task preempts the old task, and the arrival, service, and deadline processes of the new task restart in stage 1 (State 2). The service process of S1 completes a stage at rate $4\mu_1$, resulting in State 4. Because the service process of S1 is executing in its last stage in State 1, the transition from State 1 to State 4 at rate $4\mu_1$ corresponds to a met deadline for S1 (State 4). Therefore, in State 4, the service and deadline processes of S1 are deactivated due to the met deadline.

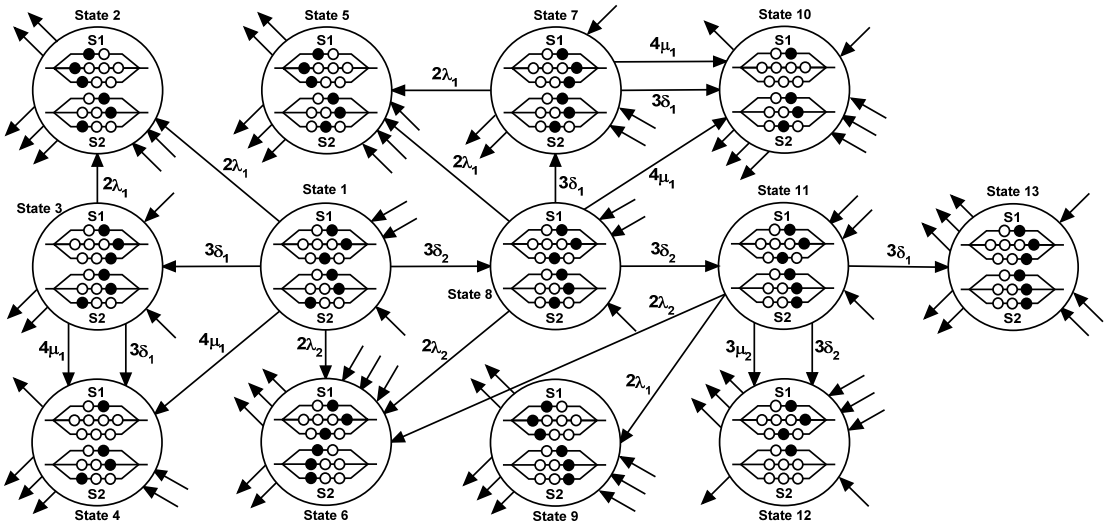

Figure 31: Partial state diagram assuming EDF scheduling

Similarly, from State 1, the deadline process of S1 can complete its current stage at rate $3\delta_1$, resulting in State 3. In State 1, the processor is not allocated to S2 and the service process of S2 cannot progress to its next stage. However, an arrival for S2 can occur at rate $2\lambda_2$, resulting in State 6 and causing the S2 task to miss its deadline due to the subsequent S2 task arrival. Finally, in State 1, the deadline process for S2 can

complete its current stage at rate $3\delta_2$, resulting in State 8. Notice that in State 8, both S1 and S2 have the same number of deadline stages remaining. Therefore, under the assumption that $3\delta_1=3\delta_2$, both S1 and S2 should be allowed to execute. The decision of what to do in such a tie-breaking situation is often left to the system designer. For this discussion, S1 is given priority over S2 as a tie-breaking rule, as indicated by the arc from State 8 to State 10 in Figure 30.

State diagrams can be constructed for other scheduling algorithms as well, including new hybrid scheduling algorithms. Constructing and analyzing these diagrams is helpful in describing and determining the performance characteristics of different scheduling algorithms, but for practical systems, the problem of state-space explosion again arises. Because MOSS uses the method of stages as its modeling engine, the overall system state at any given time during a simulation corresponds to a state in an algorithm-specific state diagram. Therefore, the simulation capabilities of MOSS are crucial in analyzing system behavior based on the method of stages, without suffering from the pitfalls of state-space explosion.

2.3.9 Summary of the Method of Stages

Stage-type distributions get their name from the fact that they can be represented as a sequence of exponential stages. Using such distributions preserves the appealing properties (e.g., memorylessness) of the exponential distribution while adding the increased flexibility associated with modeling time using discrete stages. Having discrete stages allows the moments (e.g., variance) of a performance parameter to be easily matched. By changing the number of stages, the variance of the workload parameters can be systematically adjusted and studied in detail. A mixture of Erlang, hypoexponential,

and hyperexponential distributions can be used to model workload parameters that have variances different from that of the exponential distribution. Ultimately, the Coxian distribution can be used to mimic the behavior of any distribution having a rational Laplace transform, and to any desired degree of accuracy. Using the method of stages modeling approach, a separate process is used to model the behavior of each important workload parameter, allowing its sensitivity to variance to be studied in detail.

2.4 Chapter Summary

In this chapter, the three main topics discussed are performance modeling using Petri nets, performance modeling in real-time systems, and the method of stages modeling technique. Each of these topics provides the fundamental information necessary to understand the remaining work in this dissertation.

Petri nets (PNs) are mathematical modeling tools that are often exploited for their descriptive power and ability to help illustrate and explain complex system behavior. Colored Petri Nets (CPNs) provide an excellent compromise between readability and expressiveness and, therefore, provide great flexibility in the development of performance models. CPN Tools allows CPN-based models to be analyzed both analytically and via simulation. Other classical analysis tools such as queuing networks and state diagrams help gain insights and identify important trends in system behavior.

Performance modeling of scheduling algorithms in real-time systems has advanced significantly since the first static and dynamic priority-based algorithms were introduced in the 1970's. Classic algorithms such as RM, EDF, and LLF have been the focus of numerous studies involving periodic, independent tasks. Much of the past work involving the study of real-time algorithms focuses on hard deadlines and assumes worst-

case behavior. However, many present-day schedulers still utilize the RM, EDF, or LLF scheduling algorithms, or close variants. Thus, it is beneficial to improve the performance of these traditional algorithms as they often serve as the framework for new, hybrid routines.

The method of stages technique uses stage-type distributions that preserve the appealing properties (e.g., memorylessness) of the exponential distribution while adding the increased flexibility associated with modeling time using discrete stages. Having discrete stages allows the moments (e.g., variance) of a performance parameter to be easily matched, and the variance of workload parameters can be easily studied simply by changing the number of stages. A mixture of Erlang, hypoexponential, and hyperexponential distributions can be used to model workload parameters that have variances different from that of the exponential distribution. Ultimately, Coxian distributions can be used to mimic the behavior of any observed distribution to any desired degree of accuracy.

In the next chapter, a case study of the enterprise grid environment is discussed.

2.5 Research Contributions

The contributions presented in this chapter include:

- A discussion of the fundamentals of performance modeling with PNs
- A brief overview of the relevant historical aspects of real-time scheduling and a primer of task scheduling in soft real-time environments
- A concise development and discussion of the method of stages technique and its flexibility in modeling and analyzing variance

CHAPTER III

PERFORMANCE MODELING OF AN ENTERPRISE GRID ENVIRONMENT USING COLORED PETRI NETS[10]

3.1 Introduction

Performance modeling of distributed systems involves developing accurate models that capture critical aspects of the operating environment, such as workload characterization, hardware constraints, and the interaction between jobs and distributed software components. To accurately characterize the workload, measurements of the actual system are used to develop a concise job description that effectively captures the overall load placed on the system due to job input. Any physical system limitations such as hardware constraints can then be used to guide the model development, using the workload characterization results for model parameterization. The requirements of jobs and their interaction with the operating environment is an especially important aspect of the model development. This is particularly true for grid systems, where jobs compete for resources (e.g., processors) and wait for the necessary shared services to become available before beginning execution. However, it is often difficult to produce an accurate performance model in these environments due the variability found within the system workload.

Various types of grid systems are in use today, many of which form the backbone of large businesses. These businesses rely on Quality of Service (QoS) standards and

[10] Reprinted with the Permission of Acxiom Corporation.

Service Level Agreements (SLAs) in order to maintain customer satisfaction. A client experiencing repeated job delays or other problems will quickly become dissatisfied. Therefore, capacity planning is especially important in these commercial environments, where an unexpected decrease in system performance may have a negative impact on the company's profitability. It is also essential for planning purposes to identify client resource needs, both before a grid system is deployed and after it is in use. The latter requires constant system reevaluation to ensure client expectations are met as the system demand evolves.

This chapter presents a case study of a real-world enterprise grid system. The workload this system is subjected to is studied and characterized to produce a group of job classes whose behavior accurately mimics that of the real workload. It is shown that the variability found within the job parameters plays an important role in the behavior of the system and is difficult to incorporate directly into a performance model. The explicit incorporation of variance in the performance model developed in this study proves to be an essential step in developing an accurate prediction model of such enterprise systems.

Figure 32 shows a simplified view of a grid environment. Users connect through client software and use batch scheduling systems to submit jobs to a job scheduler. The job scheduler is responsible for allocating computers or processors (loosely called "nodes") to waiting jobs. Jobs execute applications on their allocated nodes and utilize various data services (e.g., sorting routines, mathematical calculations, and data cleansing techniques). These data services operate on dedicated computers and communicate with nodes in the node pool. Jobs execute on nodes and repeatedly invoke the data services until they complete their execution. While executing, a job locks nodes exclusively until

it completes, at which time the nodes are released back to the node pool for use by other competing jobs.

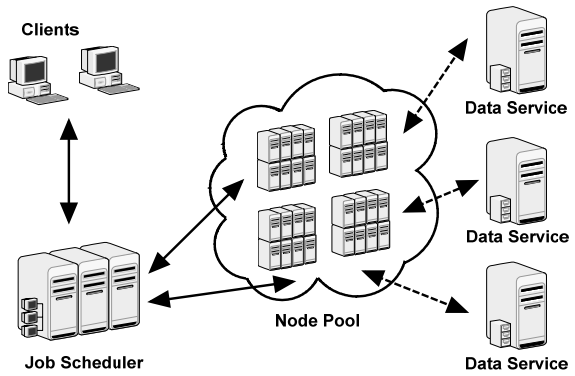

Figure 32: Simplified view of grid environment

When a job arrives, it is placed into a queue, where it must wait until it acquires locks on a set of resources (e.g., hardware and software). In the case study described in this chapter, only a single type of resource is considered: the processing nodes. Once the necessary nodes are available, the job locks the nodes and begins executing. During execution, it performs computations and utilizes various other resources, such as database servers and shared service routines. Nodes locked by an executing job are not accessible by other jobs. Once execution is complete, the job releases its locks on the processing nodes and exits the system.

The Petri Net Model (PNM) described in this chapter predicts various performance metrics for the grid environment it models, including node pool utilizations, job queue times, and system throughputs. The role of the model is to abstract and capture the primary characteristics of the system to help assess whether the expected workload queue times will likely satisfy a particular SLA (e.g., queue times less than 15 minutes). Alternative design scenarios created by varying any combination of the system attributes

can be evaluated and compared on the basis of the output from the model analysis. Because the required objective of the system (e.g., fifteen-minute SLA for application queue time) can be specified, the model results can be used for business value analysis. For example, the model can be used to assess whether or not a new client's workload can be handled with existing hardware without violating the SLAs of current clients. The case study described in this chapter is based on performance data provided by Acxiom Corporation of an enterprise grid system used in the large-scale data processing industry.

3.2 System Overview and Data Analysis

The queuing discipline used in the test environment is based on a fair-share policy that uses custom rules for balancing system usage among different user groups. For the case study, First Come First Serve with fill in (FCFS-FI) is used as the test environment's scheduling policy. FCFS-FI is FCFS, but if the job at the head of the queue cannot start because it requires more processing nodes than are currently available, the queue is searched for the first job that can start. If such a job is found, it is allowed to start even though it is not at the head of the queue.

The collection of processing nodes, called the node pool, consists of 128 nodes and represents all the available nodes from which each job must obtain locks on its specific number of required nodes. These nodes are removed from the pool, allocated to a job when it starts, and returned when the job terminates. In this study, a month of complete job data is obtained in raw format, a portion of which is shown in Figure 33. There are 26,557 jobs in this data set and for each job, there are a number of values (i.e., features) characterizing the job. For example, Column *A* (the *Job ID* feature) contains unique job identifiers and Column *B* (the *Job Type* feature) lists the job type or category. Column *C*

indicates the number of processing nodes required for allocation before the job can start executing. Column *D* indicates the number of database records required during job execution. Columns *E*, *F*, and *G* list the day and time each job arrives, starts, and stops, respectively. Column *H* lists the number of splits (i.e., threads) generated when the job starts.

Each job can be completely described by its feature vector (i.e., its row in the job data file). The goal is to obtain a set of representative feature values accurately describing the workload data that can be used as input parameters to the PNM. Ideally, the entire dataset would be used as the input stream but this is not efficient or practical. However, reducing the dataset too much can cause important data trends to be inaccurately modeled or altogether overlooked. A compromise is needed between a large, accurate dataset, and a small, unrepresentative dataset.

There are important workload and performance measurements of interest available from the raw data presented in Figure 33. For example, queue time is computed by taking the difference between job submit-time and job start-time. Execution/service time is obtained by taking the difference between start time and stop time. The inter-arrival time is the time that passes between two consecutive job arrivals. Another metric is the average number of nodes in use at any given time, which is a measure of how busy the system is. From the raw data set, the average number of nodes in use is found to be 76.54, or 59.80%. Due to the large variability within the data set, using the mean values for each feature vector would produce a parameter set that is not representative of the entire data set [63]. This would reduce the entire data set to the mean values, ignoring many of the trends contained in the raw workload data. Incorporating the variance of the

performance parameters into the model development is particularly important in order to generate an accurate workload to use as model input. The initial step in determining the workload variability is developing an appropriately sized and descriptive data set, which is the goal of workload characterization.

	A	B	C	D	E	F	G	H
1	Job ID	Job Type	Nodes	Records	Submit DayTime	Start DayTime	Stop DayTime	Splits
2	O1719395	A	1	20293	1 0:02	1 0:02	1 5:40	1
3	O1722831	A	1	17	1 0:02	1 0:02	1 0:03	1
4	O1722838	A	1	15	1 0:04	1 0:04	1 0:04	1
5	O1722787	A	16	196	1 0:05	1 0:05	1 0:08	16
6	O1722839	A	16	194	1 0:05	1 0:05	1 0:08	16
7	O1722842	A	4	149	1 0:06	1 0:06	1 0:09	4
8	O1722845	A	2	76	1 0:07	1 0:07	1 0:08	4
9	O1722788	A	1	20	1 0:08	1 0:09	1 0:09	1
10	O1722840	A	1	21	1 0:08	1 0:09	1 0:09	1
11	O1722848	A	1	461	1 0:09	1 0:09	1 0:17	1
12	O1722851	A	2	2403	1 0:09	1 0:09	1 0:49	4
13	O1722852	B	2	123	1 0:11	1 0:53	1 0:55	2
14	O1722849	A	16	177	1 0:18	1 0:52	1 0:55	16
15	O1722853	A	1	19	1 0:20	1 0:46	1 0:46	1
16	O1722861	A	1	20	1 0:35	1 0:53	1 0:53	1
17	O1722182	B	10	7594	1 0:42	1 0:46	1 2:53	20
18	O1722868	A	1	63	1 0:43	1 0:46	1 0:47	1
19	O1722869	A	1	26	1 0:46	1 0:46	1 0:47	1
20	O1722865	A	16	241	1 0:47	1 0:48	1 0:52	16

Figure 33: Portion of raw job data

3.3 Workload Characterization

Clustering is a technique used to identify homogenous clusters, or groups, of jobs within a large workload. Clustering techniques are often used in large, grid computing environments to identify similar classes of jobs in order to reduce the size and complexity of the workload being studied. With any clustering technique, the goal is to identify groups of jobs that are similar to all jobs within the same cluster, but as different as possible from jobs in other clusters. Each cluster has a centroid, which is a vector that

contains the mean value of each feature for the jobs within the cluster. The resulting clusters correspond to different job classes of a multi-class performance model [63].

Although there are a number of clustering techniques (e.g., k-means and spanning-tree), hierarchical clustering is selected for this study. In this clustering technique, a hierarchy of clusters is constructed from individual features by repeatedly merging clusters. For this study, the hierarchical clustering is done based on the number of nodes required, because this parameter is used in the real-world scheduling policy. The number of nodes required[11] ranges from 1 to 30 so initially 30 clusters are created where each job is placed into its corresponding cluster based entirely on its number of required nodes. Clusters are then combined based on their percentage of the workload and observed performance characteristics. The notation *Cx* is used to refer to job clusters, where *x* denotes the number of nodes required by a job in that class. For example, the *C8* cluster is created by combining all jobs requiring five, six, or eight nodes because these jobs require a similar number of nodes and account for only a small percentage of the total workload. The *C12* and *C16* classes are created in a similar manner. Using this refined method of hierarchical clustering requires detailed analysis of the workload and is more of an art, rather than a science.

The results of grouping jobs into six job clusters/classes are shown in Table 8. Consider class *C4*, which consists of all jobs requiring four processing nodes. All *C4* jobs require four processing nodes and account for 12.66% of the entire job workload. Classes *C8*, *C12*, and *C16* consist of jobs requiring different numbers of nodes, as mentioned previously. For modeling purposes, all *C8* jobs are assumed to require eight

[11] Some values for the number of nodes required are not listed because jobs do not request this specific number of nodes (e.g., no jobs require exactly 3, 7, 9, 13-14, 17-19, 21-29, or greater than 30 nodes).

nodes and account for 2.19% of the entire workload. Using clustering, the 26,557 jobs are reduced to six job types, where the centroid of each cluster represents the average job characteristics within that cluster.

Table 8: Six job classes resulting from clustering

Class	# Nodes Required	# Jobs	% Jobs
C1	1	9,862	37.13
C2	2	4,598	17.31
C4	4	3,361	12.66
C8	5, 6, 8	581	2.19
C12	10, 11, 12	3,216	12.11
C16	15, 16, 20, 30	4,939	18.60
	Total	26,557	100.00

Table 9 shows the per-class mean (μ), standard deviation (σ), and coefficient of variation (CV) for each feature of interest. All of the values listed in Table 9 are calculated from the raw dataset. The CV, defined as the ratio σ/μ, is a measure of variability within a data set; larger CV values denote larger variability within the job class. Note that overall measurements are obtained by placing all jobs into a single large cluster, and not by averaging the six job cluster measurements. For example, in Table 9 the average overall inter-arrival time is 1.68 minutes. This value is obtained by measuring the average delay between each consecutive job arrival, regardless of class. Naturally, the average inter-arrival time within each class is larger.

3.4 Metrics of Interest

The inter-arrival time and service time measurements for each job class are used as inputs to parameterize the PNM. Since the inter-arrival time distribution characterizes the input data stream, it is essential to mimic this distribution accurately in the PNM. Likewise, service times indicate the amount of time jobs are in execution and are used to

parameterize the service time distributions. One of the outputs of the PNM is queue times, which are used in the model validation step. Queue time is a key output metric of interest because Quality of Service (QoS) measurements and Service Level Agreements (SLAs) are specified based on the amount of time a job should have to wait before being allowed to execute.

Correctly characterizing the job arrival process is an important step in developing an accurate and useful model. In modeling, it is often assumed that arrival and service processes are exponentially distributed with a CV equal to 1.0. However, as is often the case in practice, the data set in Table 9 shows that the coefficients of variation for inter-arrival and service time distributions are all greater than 1.0, suggesting the corresponding distributions are more accurately modeled using a heavy-tailed or a hyperexponential distribution. This reiterates the importance of not only matching the mean of performance parameters, but also their variance. A performance model that fails to capture the second moment of the workload (i.e., the variance) will not produce accurate and representative input, leaving the predictions based on such models, of limited value.

Table 9: Workload characterization results

	Inter-arrival Time (min)			Service Time (min)			Queue Time (min)		
Class	μ	σ	CV	μ	σ	CV	μ	σ	CV
C1	4.53	21.04	4.64	5.84	44.37	7.59	1.53	6.99	4.58
C2	9.71	27.57	2.84	4.63	25.81	5.58	1.74	8.38	4.82
C4	13.27	44.72	3.37	15.36	143.80	9.36	4.60	25.47	5.54
C8	74.79	248.00	3.32	14.25	75.88	5.32	42.43	126.40	2.98
C12	13.81	41.99	3.04	17.83	128.40	7.21	13.12	45.69	3.48
C16	9.02	25.31	2.81	29.65	119.30	4.02	28.51	84.18	2.95
Overall	1.68	6.50	3.87	12.90	91.25	7.07	9.28	46.54	5.01

3.5 Petri Net Model

The Petri net structure is constructed from the system architecture description, job flow description, and workload characterization. Jobs are represented by tokens and places are used to simulate the resource pool, execution states, and other workflow components. Job tokens contain all the data necessary to propagate them through the net. Transitions are used to time and sequence the token propagation, as well as ensure necessary operating conditions and constraints are fulfilled. Transitions specify input conditions by referencing variables contained inside of job tokens waiting to be selected for consumption.

The scheduling algorithm is implemented by attaching a sequence number to each job token as it enters the job queue, where jobs arriving later receive larger sequence numbers. This sequence number identifies the order in which jobs enter the queue and it is used to provide additional constraints on the consumption of the corresponding job tokens. Therefore, the order in which a job starts depends on its sequence number, as well as its number of required nodes. In this way, FCFS-FI is implemented by searching the sequence numbers from smallest to largest whenever the job at the head of the queue requires more nodes than are currently available. Additional scheduling strategies can be explored by adding or removing similar constraints on the way in which tokens are removed from the job queue. For example, strict FCFS can be implemented by forcing the consumption of job tokens to follow their sequence numbers, regardless of whether or not their required nodes are available. The job with the current sequence number waits at the head of the queue until its required number of nodes becomes available and forces all jobs with larger sequence numbers to wait behind it.

Figure 34 shows a simplified diagram of the PNM, where initialization functions, timing variables, and other details are removed. A more complete PNM diagram [24] is shown in Figure 35. The notation *i`v* is used to denote *i* identical tokens, one of which is bound to the variable *v* when a transition fires. When the job arrival transition fires, a job token *t* is removed from *job token* and sent to *job queue*. At the same time, one new job token *nt* is generated and placed into *job token*, where it must wait an appropriate delay before it is removed and sent to *job queue*. In this manner, the job generator mimics a job arrival stream by generating jobs from the six class types, and delays are generated from hyperexponential distribution functions to match the observed job parameters found in Table 9.

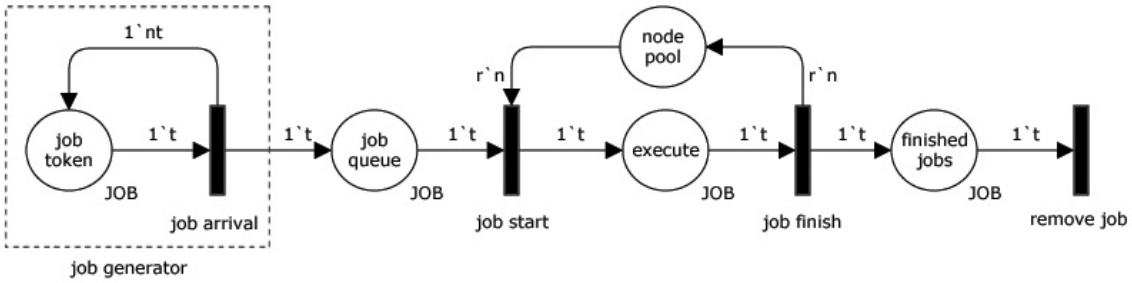

Figure 34: Simplified view of the PNM

Jobs in *job queue* must wait until the required number of nodes (i.e., *r`n*) become available, at which time, token *t* and its required number of nodes *r* are removed from *job queue* and *node pool*, respectively. Next, the job begins executing where it remains in place *execute* an amount of time determined by its average service time, also determined by a per-class hyperexponential distribution. Finally, the job finishes and is sent to place *finished jobs* where it waits to be removed (i.e., deleted) from the net. When a job finishes, the nodes acquired when the job started are released and returned to *node pool*.

All job information collected while the token traverses the net is examined and written to output files for later analysis.

The complete PNM shown in Figure 35 includes the configuration parameters and details omitted from the simplified view. In the figure, the lower left portion of the model represents the job generator responsible for accurately mimicking the workflow observed in the real-world system. The input, output, and action functions shown beside transitions perform actions necessary for the correct operation of the PNM. Before a transition fires, any required input variables (e.g., tokens) are initialized by the input functions. When a transition fires, any declared action variables (specified by the actions section) are bound (i.e., assigned) to the appropriate output variables by the output functions. When there is a one-to-one relationship between input and output variables, or when the semantics are well understood, the input, output, and action functions are omitted.

In the complete PNM, implementation details omitted from the general discussion can be seen. For example, a job starts after it acquires its required number of processing nodes, but it must then wait to acquire locks on all (if any) required database records. The job is then classified as either a *Type I* or *Type II* job, where the latter type requires the use of shared service routines (e.g., database queries) and the former does not. For *Type I* jobs, the acquired processing nodes and database records are sufficient for the job to execute until completion. In Figure 35, a job token in place *job wait classify* continues along the top branch in the net if it does not require the use of any shared services. This criterion is indicated in the figure by the guard condition (enclosed in brackets) on the transition *job type I check* asserting that the number of required shared service certificates

(i.e., permissions) of token *t* must equal zero. Similarly, any job that requires shared services requests them using the appropriate number of certificates. Therefore, a job token *t* is classified as a *Type II* job if it requires greater than zero certificates and proceeds along the bottom branch shown in the figure. The remainder of the operation of the PNM is similar to that provided in the previous discussion.

3.5.1 Parameterization

Several parameters and metrics obtained from the workload characterization are used to parameterize the PNM. These values provide a starting point for assigning values to various rates and timing variables. For example, the average inter-arrival time IT_i is used to regulate the rate of the job generator so that a new job is created and placed into the queue every IT_i simulation time units, where *i* specifies the job class. The average service time ST_i is used to timestamp tokens so that they remain in the execution place ST_i time units before being allowed to continue through the net. The average number of nodes required by a particular job class is used to specify the value of *r* in Figure 34.

The goal is to parameterize and calibrate the PNM by matching the first two moments (μ and σ) of the arrival and service processes so that they closely match the raw, real-world measurements. To do this, parameters related to the job generation are varied so that the inter-arrival time distribution of the PNM closely matches that of the actual measurements. Similarly, numbers created and used as service time delays in the PNM are varied until the service time distribution statistics closely match those from the real data. The queue time measurements, obtained as output from the PNM, are compared to the measured values for validation purposes.

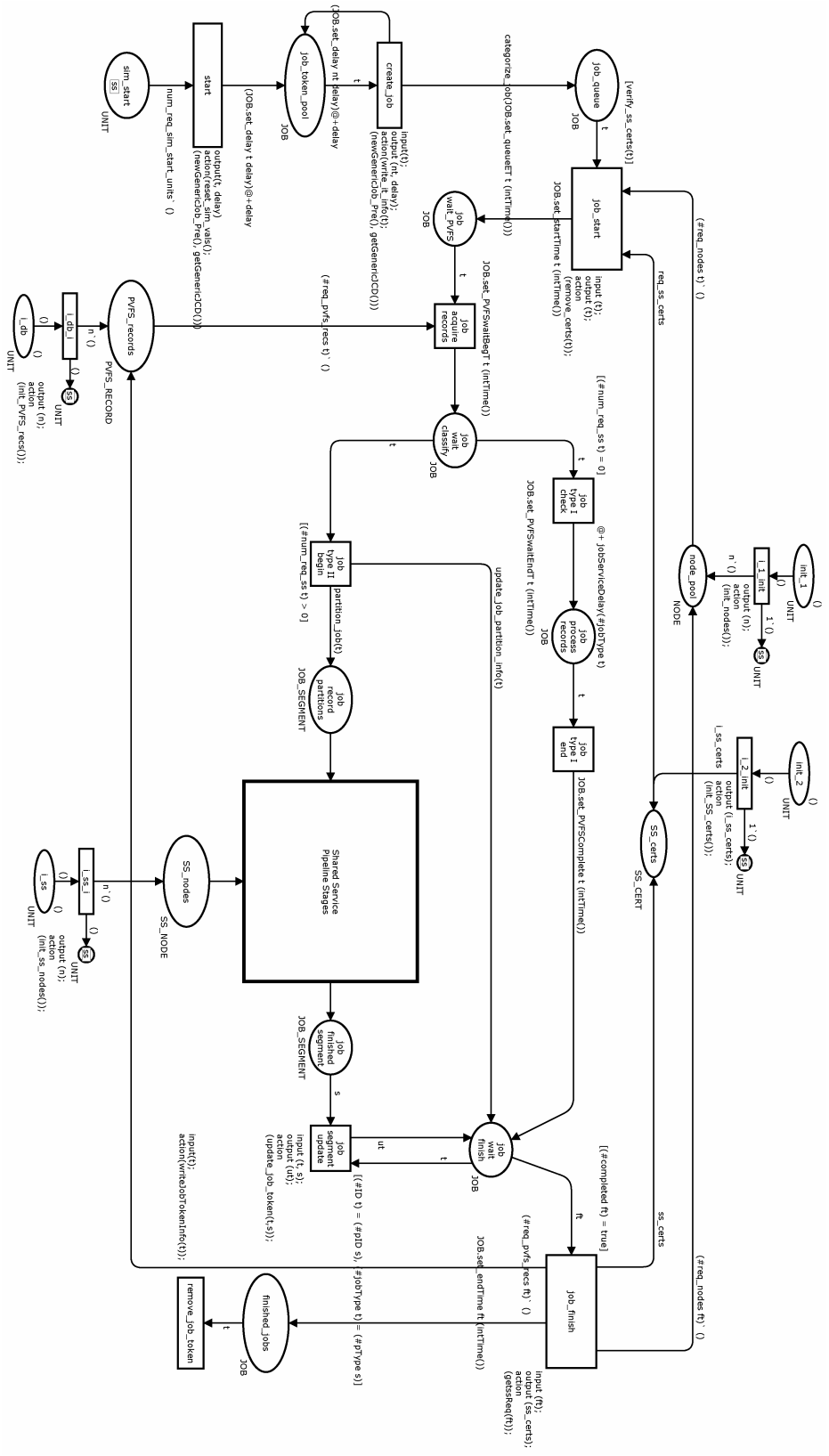

Figure 35: Complete CPN model

3.5.2 Calibration

Finding a job generator that accurately mimics the first and second moments of each of the job classes, as well as the overall job arrival stream, is particularly difficult. Initially, a different job generator is used for each job class to create six independent job streams, all of which supply job tokens to the job queue. A number of hyperexponential distributions are tested and a two-moment match for inter-arrival times (ITs) for each job class can be achieved. Matching the mean and standard deviation of a distribution is a well-studied technique [44] [77]. Applying this technique to the independent job streams assures that the first two moments of each job class' arrival stream match the measured values. However, only the first moment of the overall average can be matched—the second moment (σ) of the overall IT cannot be matched using this technique.

The PNM is thus modified to use a two-step job generator, with the first step being used to match the first two moments of the overall job arrival stream, and the second step being used to match the correct percentage of jobs within each job class. This is done by using a single distribution function in the first step that matches the overall measured values (μ and CV) for the ITs in order to decide when a new job should be generated. The new job is then tagged in the second step with a particular job class based on the job percentages from the measured workload (see Table 8). Using this adaptation, the first two moments for the overall IT values are matched. Using the measured job percentages to tag jobs assures the first moment for each class is also matched. However, the per-class second moment cannot be matched using this method.

The problem with using a job generator in the first step and having only one input stream to the job queue is that large ITs from one class are not correlated with large ITs

from other job classes. Daily cycles are present in the system being studied that result in job arrival correlations during the peak and non-peak hours. A large IT tends to indicate non-peak hours and this behavior is seen across all of the different job classes. Large gaps in IT values tend to occur at night, for each of the job classes. To further calibrate the model, a method is needed that simulates these observed daily cycles. Several techniques are possible but the method presented here involves using burst factors.

The final technique selected is a three-step job generator, as shown in Figure 36. Note that the job path for class *C2* jobs is emphasized. In step one, the job class percentages (see Table 8), p_i*'s*, are used to select which job class *Ci* to use. In step two, a burst factor b_i is used to determine how many consecutive jobs of class *Ci* should be generated. In step three, a branching probability α is used to determine which one of two mean values m_1 and m_2 should be used as input for a two-stage hyperexponential distribution. That is, when a new job event is triggered, the measured job percentages are used in the first step to determine which job class should be used. In the second step, b_i jobs are created, all of which are assigned the same job class, *Ci*. Each of the b_i jobs is assigned an inter-arrival time based on its job class in the third step. Each of these jobs is then sent to the job queue after its inter-arrival delay has passed. After the last job output from the third step has been sent to the job queue, the three-step process is repeated. This technique is used to synthetically mimic the day and night cycles observed in the real-world system. In effect, a large burst factor increases the CV of a particular job class' IT.

To determine optimal burst factors, experiments are run to vary the burst factor from 1 to 20 for each job class. The output produces a table allowing a burst factor to be chosen for each job class that results in an approximate two-moment match for each job

class' IT values. The optimal burst factors are found to be 15, 4, 6, 6, 5, and 4 for job classes *C1*, *C2*, *C4*, *C8*, *C12*, and *C16*, respectively. For instance, when a new job event is triggered and class *C2* is chosen in step one, four consecutive *C2* jobs will be generated in step two, each of which is stamped with a different inter-arrival delay in step three.

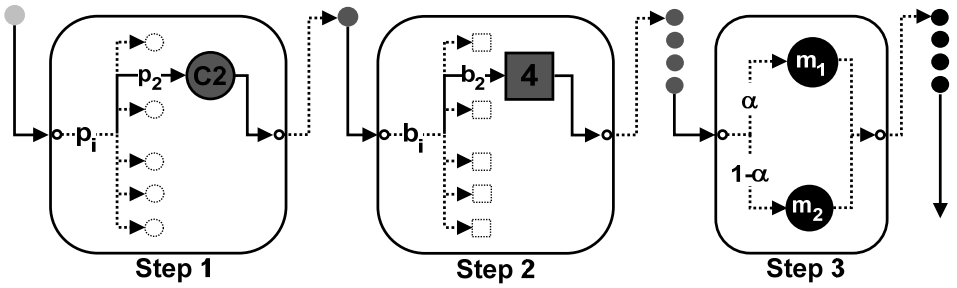

Figure 36: The three steps of job generation

Having matched the first two moments for the per-class and overall IT distributions, the service times (STs) are next matched by using a method similar to that used for the ITs. A hyperexponential distribution is created for each job class in order to create a two-moment match for each job class' ST distribution. No daily cycles are observed in the ST distributions (i.e., a *C4* job runs approximately just as long whether submitted during peak hours or non-peak hours). That is, unlike ITs, a large ST event for a job of one job class is, in general, not correlated to a large ST event of a job in a different job class.

3.5.3 Validation

Having created a two-moment match for IT and ST distributions, the queue time (QT) values output from the PNM are next examined to determine the validity of the model. The goal is to find a baseline model that mimics the observed measurement data. The metrics for the baseline model are shown in Table 10. By comparing the baseline model

results in Table 10 against the actual measurement data in Table 9, the overall model is validated (in general), with a few noted exceptions.

Table 10: Metrics for baseline model

	Inter-arrival Time (min)			Service Time (min)			Queue Time (min)		
Class	μ	σ	CV	μ	σ	CV	μ	σ	CV
C1	4.35	18.68	4.29	5.81	44.12	7.59	0.27	1.61	5.93
C2	9.48	28.05	2.96	4.66	26.42	5.67	0.55	2.26	4.11
C4	12.92	43.69	3.38	14.83	130.63	8.81	1.85	5.56	3.01
C8	71.88	236.08	3.29	14.21	70.69	4.97	5.34	13.26	2.48
C12	13.55	42.45	3.13	17.89	130.79	7.31	12.63	36.79	2.91
C16	8.85	26.62	3.01	28.89	115.73	4.01	31.13	89.26	2.87
Overall	1.63	6.48	3.97	12.65	89.10	7.04	7.82	42.33	5.42

Some discrepancies and uncharacteristic values can be seen when comparing the measured values in Table 9 to the baseline metrics in Table 10. For example, class *C8* has a measured IT of 74.79 minutes, but the baseline model approximates the *C8* IT only moderately well, with a value of 71.88 minutes. Also of interest are the queue times, particularly the large *C8* measured queue time of 42.43 minutes and standard deviation of 126.4 minutes. The baseline model approximates the *C8* queue time average and standard deviation as 5.34 minutes and 13.26 minutes, respectively, which is a significant difference. However, this unusual behavior of class *C8* jobs is not a major concern because *C8* jobs account for only 2.19% of the entire workload. By comparing the actual measurement statistics in Table 9 to those used by the baseline PNM in Table 10, the other per-class parameters (e.g., the first two moments of IT_i and ST_i) match well. Such a validated PNM can be used to predict various capacity-planning scenarios.

3.6 Capacity-Planning Scenarios

In the following scenarios, only the specified parameters are varied in each case—all other model parameters are held constant, at the baseline values. A 15-minute SLA for queue time is used in these scenarios.

Scenario 1: What is the effect of varying the number of processing nodes in the node pool? Here, the initial size of the node pool is varied from 100 to 200 nodes, in increments of 10. The effect on queue time is shown in Figure 37.

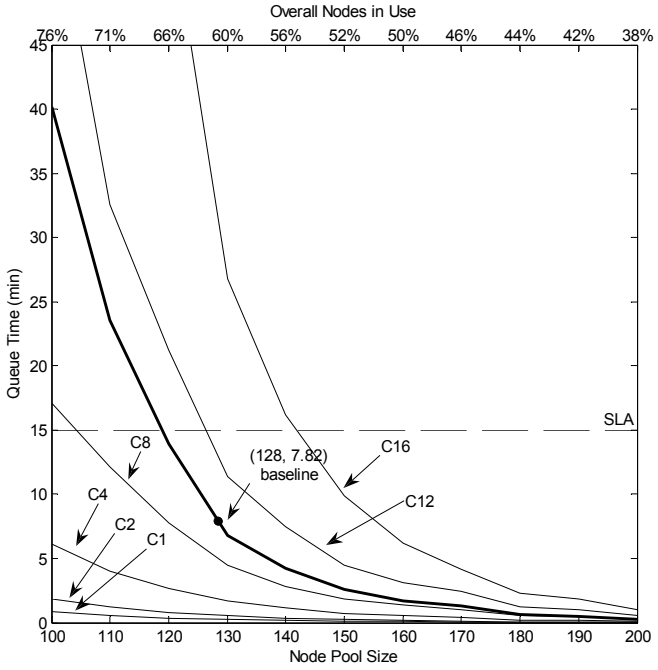

Figure 37: Effect of node pool size on queue time

The bold line indicates the overall average queue time and the black dot on this line at 128 nodes marks the baseline average queue time of 7.82 minutes. As the number of processing nodes (node pool size) increases, the average job queue time decreases. An overall queue time of approximately 40 minutes is expected when the node pool size is

100 nodes and a queue time of nearly zero is expected as the node pool size approaches 200.

From Figure 37, it is seen that it is necessary to maintain a node pool size of approximately 120 nodes, in order to maintain an SLA of 15 minutes for overall queue time. Therefore, the node pool size can be reduced by approximately eight nodes, to 120, and still meet a 15-minute queue time SLA for the overall average. Notice that at baseline, a 15-minute SLA is not met for the class *C16* jobs. The node pool must be increased to approximately 140 nodes for all job class queue times to meet such an SLA.

Along the top of Figure 37, the overall percentage of nodes in use is shown for each node pool size. At baseline, the node pool usage is approximately 60%. If the node pool were reduced by the eight nodes suggested previously, the node pool usage would increase to approximately 66%.

Scenario 2: What is the effect of varying the job arrival rate? Here, the overall average job arrival rate is varied by an intensity factor ranging from 0.7 to 1.3, in increments of 0.1. An intensity factor of 1.3 indicates a 30% increase in the job arrival rates for all job classes. The effect on queue time is shown in Figure 38.

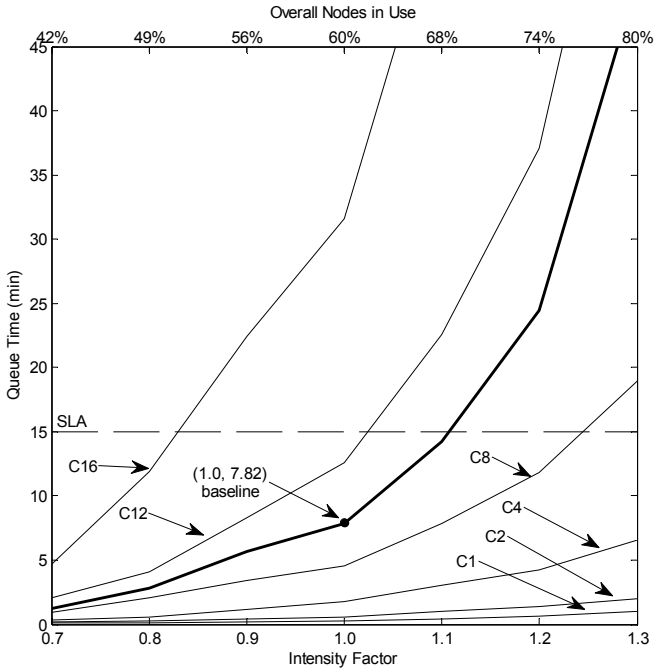

Figure 38: Effect of job arrival rate on queue time

An intensity factor of 1.0 corresponds to the baseline model. As the intensity factor increases, the average job queue time also increases. A queue time near zero is expected when the baseline arrival rate decreases by 30% (to intensity factor 0.7), and a queue time of more than 45 minutes is expected if the baseline arrival rates increase by 30% (to intensity factor 1.3).

Notice from Figure 38 that if the (baseline) average job arrival rates increase by more than 10%, a 15-minute SLA for overall queue time can no longer be met. In order for all job classes to meet the 15-minute SLA, the workload must be reduced by approximately 20% because of the *C16* average queue time. If this were done, the node pool usage would be reduced from 60% to 49%. Also, notice that even though a 20% increase in job arrival rates from the baseline (to intensity factor 1.2) only increases the node pool usage by about 14 percent, the *C16*, *C12*, and overall queue times increase rapidly.

Scenario 3: What is the effect of changing the queuing discipline from FCFS-FI to strict FCFS? This scenario involves changing the manner in which jobs are removed from the job queue. The effect on queue time is shown in Figure 39.

In the baseline model, FCFS-FI is used. If the queuing discipline is changed to strict FCFS, the overall average job queue time increases from 7.82 minutes (baseline) to 25.39 minutes, as shown in Figure 39. With strict FCFS, job class has no effect on the order in which jobs are removed from the queue, which tends to produce a balancing effect on all of the per-class job queue times. Notice that in the baseline model (FCFS-FI), average queue times increase left to right across job classes, as do the number of required nodes. With strict FCFS, this trend is not seen, as each of the per-class queue times is within 10% of the overall queue time. This is due to the fact that FCFS-FI discriminates against larger-node jobs by allowing smaller-node jobs to "fill-in" and bypass the larger-node jobs that arrived earlier. Strict FCFS does not allow such bypassing and treats all jobs uniformly.

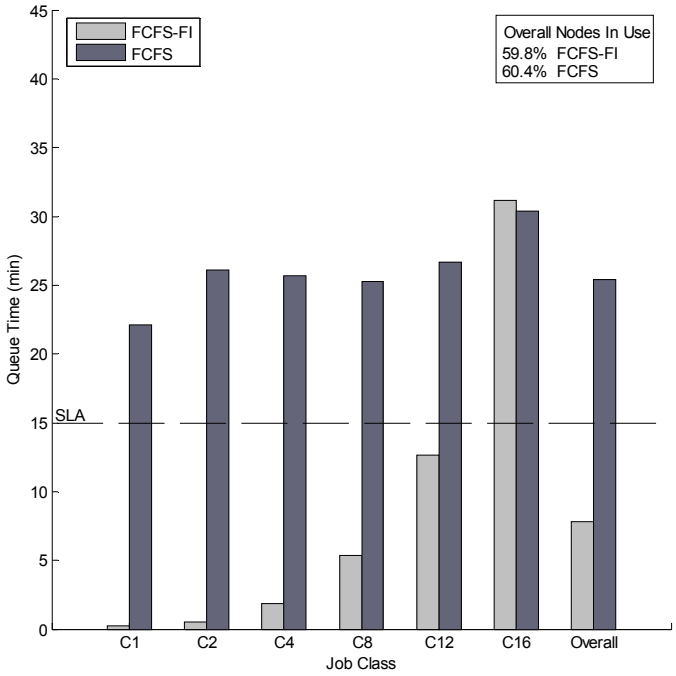

Figure 39: Effect of queuing discipline on queue time

In the baseline model, only *C16* jobs do not meet a 15-minute SLA, exceeding the SLA by approximately 15 minutes. The SLA must be doubled to 30 minutes in order for the baseline model to meet this requirement for all job classes. However, with strict FCFS, no job class meets the 15-minute SLA and the only job class slightly benefiting from a change to strict FCFS is *C16*. All of the remaining job classes suffer a significant performance hit because of the change. The overall node pool usage increases by 1% (1.28 nodes) as a result of changing the queuing discipline. Because the number of nodes in use is a relative measure of how busy the system is, this suggests there is no significant effect on the overall system utilization. However, with FCFS-FI, there are likely more smaller node jobs in the system at any given time because they are given preference by the scheduling algorithm. Therefore, the individual utilizations of the smaller-node job classes are expected to increase, while the per-class utilizations of the larger-node jobs are expected to decrease. These results are confirmed experimentally.

3.7 Chapter Summary

A case study is presented that describes how the workload of a large enterprise grid environment is characterized and used to parameterize a high-level Petri Net Model (PNM) for performance prediction. By studying the workload characterization results, it is demonstrated that job parameters in this system are best approximated by using hyperexponential distributions. However, using such distributions to achieve a two-moment match for the inter-arrival and service times, as well as developing a job generator that accurately mimics the behavior of the entire workload, proved to be particularly difficult. A formal modeling technique, such as the method of stages, is needed in order to develop performance models that consistently and efficiently achieve higher-moment matches for performance parameters. In addition, the relative importance of these higher-moment matches and the effect of variance on performance metrics require further study. In the next chapter, a simulation tool called MOSS is presented that provides a systematic manner of studying the effects of variability in real-time systems.

3.8 Research Contributions

The contributions presented in this chapter include:

- A thorough discussion and analysis of a case study of a real-world enterprise grid environment
- A novel three-step job generation technique that accurately reproduces the workload observed in the real-world environment and matches the first two moments of performance parameters

- An empirical demonstration that incorporating variance into performance models is important, and sometimes necessary, for model calibration
- A demonstration, through the use of capacity-planning scenarios, that the validated performance model is a good tool for predicting the effects of changes to the real-world system

CHAPTER IV

A SIMULATION TOOL FOR MODELING VARIANCE IN SOFT REAL-TIME SYSTEMS[12]

4.1 Introduction

In an attempt to better quantify the importance of achieving higher-moment matches for performance parameters, a technique known as the method of stages can be used. The method of stages is a formal modeling technique that provides an intuitive method for achieving higher-moment matches for performance parameters. In addition, the method of stages provides a direct way of manipulating the variance of performance parameters through the number of stages used. This modeling technique is used as the basis for a new simulation tool called MOSS (Method Of Stages Simulator) that provides an intuitive interface for conducting sensitivity analysis experiments involving the effects of variance on performance parameters. MOSS can be used to describe, simulate, and analyze the workload and performance characteristics of real-time environments, as well as the effects of variance on performance metrics of interest. The effects of variance on parameters such as inter-arrival, service, and deadline times in real-time environments can be easily studied using MOSS. This chapter focuses on the analysis of soft real-time systems, and in particular, how variance can affect the performance of scheduling algorithms in these environments.

[12] Reprinted with the Permission of the Society for Modeling and Simulation International (SCS).

Soft real-time systems are able to tolerate a limited number of deadline misses and still be useful, provided these misses are bounded by a threshold. For example, an airline information system must be able to keep departure times updated in order to satisfy customers. However, due to issues such as weather conditions, unexpected maintenance, a larger than normal number of flights, and/or unanticipated layovers, it is difficult to keep passengers accurately informed. In these situations, sporadic data input causes unreliable service times and results in an unpredictable operating environment. When passengers miss their flights, alternate flights are typically substituted. Therefore, a common goal in soft real-time systems is to keep deadline misses at a minimum, while maximizing system throughput.

A number of scheduling algorithms attempt to achieve performance goals by modeling system variability using worst-case time analysis in order to predict actual system behavior. Other algorithms ignore variance altogether, even though it can significantly affect their performance. Ignoring variance can lead to overly optimistic predictions, while assuming worst-case behavior is pessimistic because the worst case may rarely happen in practice. Interestingly, the first moment (i.e., the mean) of performance parameters typically receives most of the focus of study and less attention is paid to the second moment (i.e., the variance). However, accurately modeling the second moment can be more important than matching the first moment exactly. That is, system performance can be more sensitive to the variance than the mean.

A more strategic approach is to incorporate the information regarding the system variability directly into a performance model that can then use this information to analyze system behavior and make better scheduling decisions. For example, if a system is under

light load, a simple EDF (Earliest Deadline First) scheduler may perform quite well, regardless of the workload variability. However, as the system load increases, and particularly as the system utilization approaches 100%, a scheduler that does not adjust its scheduling decisions based on the variance of the workload can cause the system to miss more deadlines than necessary. A number of existing tools and simulators can be adapted to analyze soft real-time systems to determine ways of improving system performance. Such tools include QPME, GreatSPN, Möbius, JMT, and CPN Tools [7, 11, 20, 49, 75]. However, these tools focus on providing a generic framework for constructing many different types of models for analyzing and/or simulating a wide range of target systems, rather than on a specific aspect of performance.

The primary goal of MOSS is to allow a user to investigate and analyze the effects of variance on different performance parameters in soft real-time systems. A key advantage of using MOSS is that it is based on the well-established technique of method of stages, which it uses to capture and model the variance within task parameters. This allows the variance of inter-arrival times, service times, and deadline times to be matched to measured values. In addition, the effect of variance on parameters and performance metrics can be studied in a uniform and systematic manner.

4.2 Motivating Example

As an example, consider a pharmacy where prescriptions are received by a technician, who then fills the orders and prepares them for customer pick-up. Suppose there are two possible types of prescriptions: drop-offs and phone-ins. Drop-off orders are sporadic with an average inter-arrival time of six minutes between submitted orders. Phone-in orders on average also arrive every six minutes. However, these orders are much more

regular because they tend to originate from scheduled doctor's office visits and exhibit less variance in their inter-arrival times than drop-offs. The service time is the amount of time it takes the pharmacist to mix the needed drugs together to fill the order. Deadlines occur when the customer arrives to pick up their order. Both the service and deadline time behaviors are assumed identical for the drop-off and phone-in prescriptions. Therefore, the only difference between the two prescription types in terms of arrival, service, and deadline characteristics is the variance of their inter-arrival times.

When either type of order arrives, a technician/receptionist records the prescription details and submits the order to a pharmacist who later fills it. The pharmacist requires an average of two minutes to mix the needed drugs together to fill an order of either type. In most cases, this is also a practical assumption because prescriptions of either type are not expected to exhibit any significant differences. Because two orders cannot be filled at the same time, after an order is started by the pharmacist, it must be completed before the next order can be processed. However, given a set of outstanding orders to fill, the pharmacist can decide which one to fill next, to maximize the likelihood of completing the order before the next customer pickup (i.e., the order's deadline). The emphasis in this example is placed on the variance of the arrival process, rather than on the filling of prescriptions or customer pickups. The performance goal is to maximize the overall percentage of met deadlines (i.e., to successfully complete as many prescriptions as possible before customer pick-up). The free variable is which order the pharmacist should work on next (i.e., the scheduling algorithm), a drop-off order or a phone-in order, which are identical in every way except for their inter-arrival time variances.

MOSS is used to run four different groups of simulations for this pharmacy scenario. Two groups are run under light system load (i.e., when the mean inter-arrival time between orders is large), while two are run under heavy system load (i.e., when the mean inter-arrival time between orders is small). Under light load, one group is used to estimate the overall percentage of met deadlines if drop-off prescriptions are given priority, while the second group is used to estimate the same metric assuming phone-in prescriptions are given priority. An average is taken across all the simulations in a given group in order to obtain an estimate of the overall percentage of met deadlines. This process is repeated assuming heavy system load. The results from the simulations are summarized in Table 11. (see Section 4.8 for validation of these results)

Table 11: Simulation results from pharmacy scenario

System Load	Priority Given To	Overall % Met Deadlines
Light	Drop-offs	92.66%
	Phone-ins	91.06%
Heavy	Drop-offs	46.23%
	Phone-ins	51.67%

As Table 11 indicates, giving priority to drop-off prescriptions maximizes the overall percentage of met deadlines when the system is under light load. However, under heavy system load, priority should instead be given to phone-in prescriptions to maximize the percentage of met deadlines. Recall that the only statistical difference between the drop-off and phone-in prescriptions is the variance within their inter-arrival time patterns. This example illustrates that the workload variance influences the best scheduling decision. This effect of variance has been noted in other research as well. In [45], for example, the arrival patterns of network traffic data are studied and a second-moment characterization

of traffic streams is introduced. It is shown how simple computations based on the variances of arrival patterns can be used to determine accurate QoS estimates for performance parameters. These estimates are used in new scheduling routines that result in higher system utilizations.

4.3 Method of Stages Simulator

The Method Of Stages Simulator (MOSS) is a graphical Windows program that allows a user to describe a workload from a soft real-time system and to model its variance by discrete event simulation. MOSS is written in Visual C++ and is comprised of approximately 40,000 lines of code. The core of the simulation engine uses the method of stages to achieve a two-moment match for inter-arrival, service, and deadline times of each task stream input to the simulator. The layout and look of the program are intended to be simple and easy to use. The MOSS clock-icon shown in Figure 40 reflects the fact that MOSS is a simulator with an internal floating-point clock, and the colored circles represent arrival, service, and deadline process stages. Standard buttons and controls are used to operate the simulator, similar to those found in most Windows applications.

Figure 40: The MOSS icon

The information describing an input workload is composed of a number of task streams whose description and behavior are based on the method of stages technique. A

task scheduler is used to determine how competing tasks should share resources. MOSS currently supports a single resource (i.e., processor) and a number of traditional scheduling algorithms (e.g., Earliest Deadline First (EDF), Rate Monotonic (RM), Least Laxity First (LLF), and Shortest Remaining Service Time First (SRSTF)) [14]. Other scheduling algorithms are currently under development. The scheduling algorithm and additional simulator parameters can be configured to further customize each simulation.

A series of dialog windows assist the user in entering workload, scheduler, and simulator parameters. These parameters can be loaded from a configuration file, or entered directly into fields provided by the user interface. Error checking is performed and the user must enter valid values in all required fields before progressing to the next step. After the required configuration information has been loaded from a specified file or entered manually, a simulation can be started, which executes to completion unless it is paused by the user. A number of performance statistics can be viewed in real-time while the simulation is running. Once complete, various output files can be examined to observe the performance metrics of interest.

An installer program, sample configuration files, and help files are provided to aid in getting started. The installer consists of a standard installation wizard similar to the installation programs that accompany most current Windows applications. The installer prompts for an installation directory and then installs the necessary files and components. The file type MCF (MOSS Configuration File) is registered with the operating system to associate files with a .mcf extension with MOSS. This associates the MOSS program icon with the MCF file type, and allows the user to click on any MCF file and open it

directly in MOSS. MOSS is currently only supported on the Windows platform and may be downloaded at [23].

4.4 Configuration Information

A MOSS configuration describes a system workload and consists of a number of parameters that specify the configuration of the task streams, scheduler, and simulator. This information can be loaded from an existing configuration file, which is an ordinary text file containing formatted blocks of data similar to those found in most scripting languages. Therefore, MOSS configuration files can be opened and edited by any program capable of editing ASCII text files. A configuration file contains all the information needed to precisely reproduce a given simulation. These files provide a convenient mechanism for loading, saving, and modifying different simulation configurations. MOSS also provides its own basic file editor that allows editing of configuration files and provides features such as syntax highlighting, multi-line indenting, and error checking. (The MOSS file editor is shown in Figure 42). The user may alternatively specify the configuration parameters manually and allow MOSS to generate a new configuration file that is later saved. Note that some screenshots of MOSS, such as the one shown in Figure 42, show only partial dialog windows in order to conserve space.

The first dialog window of MOSS is shown in Figure 41. This window asks the user to either load a configuration file or create a new configuration file manually. If a file is loaded, the user can click the *Edit File* button to open the built-in editor and edit the configuration information. Any changes made to the configuration information can be saved or discarded, but MOSS will generate an error if the modified information cannot

be successfully parsed. This prevents a user from inadvertently saving a faulty configuration file that would later fail to load. The user can also click the *Go to Simulation* button provided at the top of the window in order to bypass the configuration screens and go directly to the *Run Simulation* window. Alternatively, the user can progress through a series of dialog windows and manually enter or edit parameters. If a file is loaded, the various fields in each window are pre-populated with information from the configuration file. At the bottom of each dialog window, there is a *Back*, *Next*, and *Exit* button to go back, move forward, or exit the application, respectively. Throughout the manual configuration process, the user may move forward and backward through the configuration steps as needed. However, if a configuration file is not loaded, or if the user makes changes at any step, the *Next* button is disabled until all required information is successfully entered. After loading a file successfully or selecting the manual configuration option, the user progresses to the next dialog window by clicking the *Next* button.

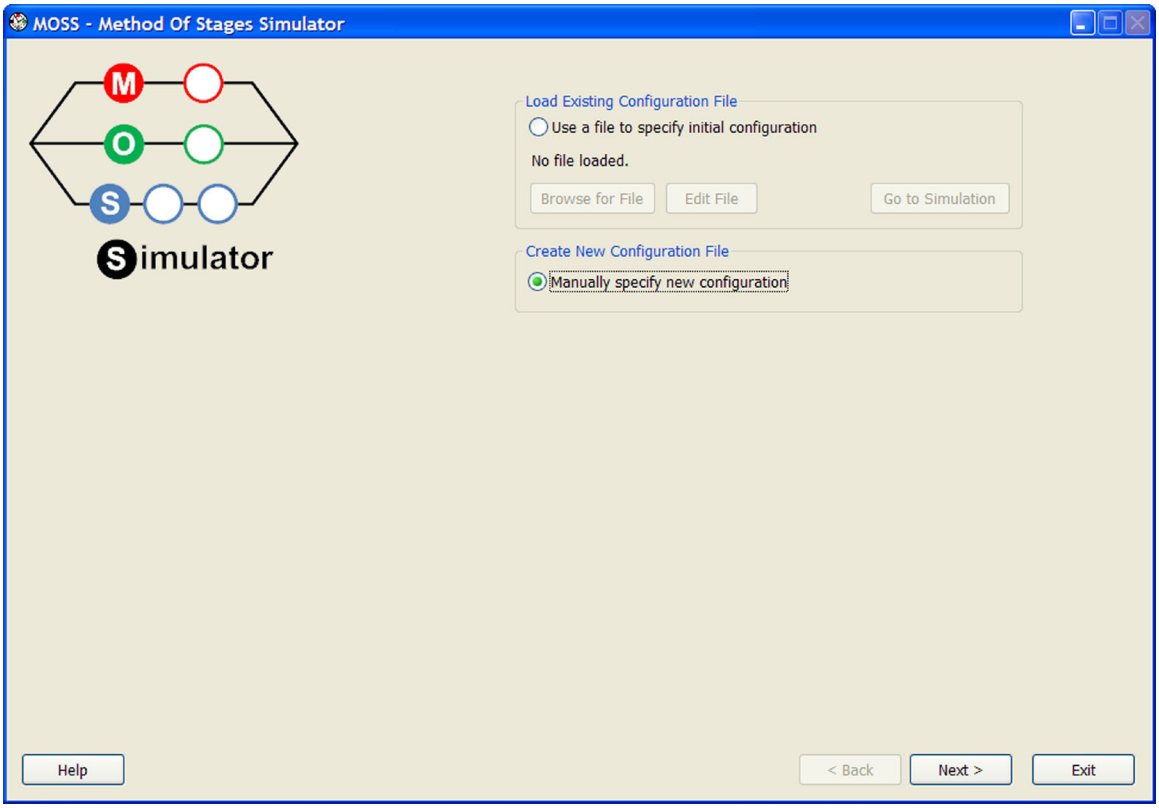

Figure 41: Main dialog window

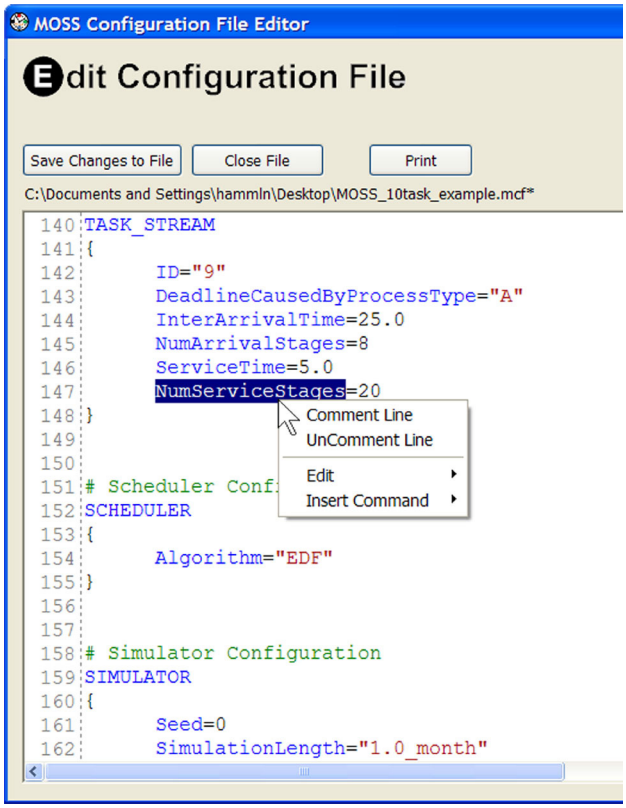

Figure 42: Configuration file editor

4.4.1 Task Stream Configuration

For each task stream, the user specifies values that define the characteristics of each stream's arrival, service, and deadline processes. The information for each task stream is displayed on a separate tab. Tabs can be added or removed by clicking the appropriate buttons. MOSS currently limits the number of task streams to ten in order to reduce simulation overhead and maximize screen readability. Figure 43 shows a screenshot of the task stream configuration window for a sample configuration consisting of ten task streams.

MOSS internally labels task streams with the integer identifiers zero through nine, but the user can enter a custom text string to more appropriately identify each stream. The user can delete the currently selected task stream by clicking the *Remove Stream* button.

At least one task stream is required and, therefore, a user cannot delete the last remaining stream. In this example, the *Add Task Stream* button is disabled because the maximum number of allowed streams (i.e., ten) has been reached.

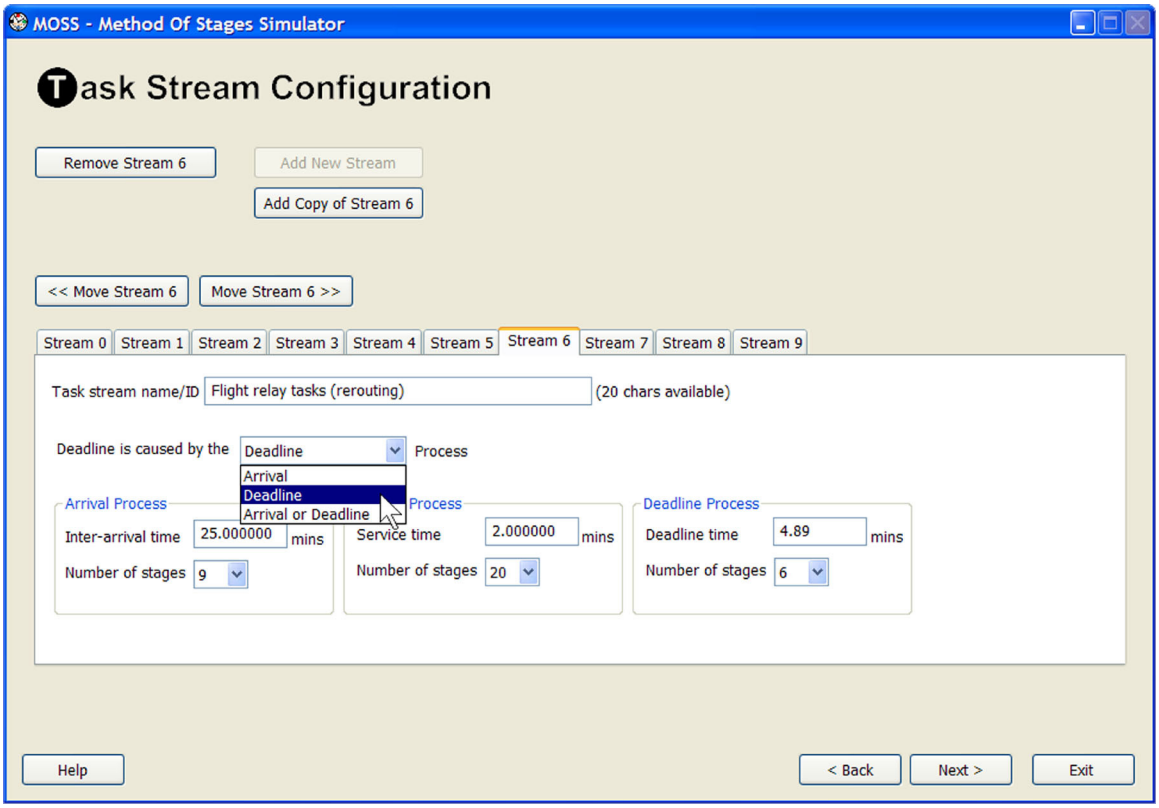

Figure 43: Task stream configuration window

A dropdown list provides three options for the deadline cause, as shown in Figure 43. Recall that the deadline process is sometimes defined as (i.e., linked to) the arrival process. In other words, when the deadline process is specified as the arrival process, the next succeeding arrival of a task also represents the deadline for the immediately preceding task in that stream. In MOSS, when *Arrival* is selected as the cause for the deadline in the dropdown list, the deadline process information is not needed and is hidden. If the deadline cause is changed to *Deadline or Arrival* or *Deadline*, the deadline

process information is required and reappears in the window. For the arrival, service, and deadline processes, the user specifies the amount of time that passes between successive events, as well as the number of stages for each process. Because the number of stages for a process determines its variability (i.e., second moment, variance), MOSS allows the user to effectively specify the first two moments of the arrival, service, and deadline processes.

All time values must be entered in minutes, but decimal values are accepted, which allows task streams with time values of seconds or non-integral minute values to be configured. The input boxes for time values do not allow invalid characters to be entered and the number of stages must be selected from a dropdown list. This helps prevent the user from entering invalid or undesired values. Once all task streams have been configured, the user is asked to enter scheduler configuration information.

4.4.2 Scheduler Configuration

The scheduler configuration currently consists of a single parameter that specifies the scheduling algorithm to be used. The algorithm is selected from a dropdown list that provides several popular scheduling algorithms. Figure 44 shows a portion of the scheduler configuration window with the algorithm selection list expanded. Because additional scheduling algorithms are continually being implemented, the user is prompted to proceed at their own risk if they select an algorithm that is still under development. The list of scheduling algorithms contains short acronyms but a more detailed description is displayed next to the listing to help the user select the best choice. This detailed description is updated as different algorithms are selected. In Figure 44, the *Rate Monotonic (RM)* scheduling algorithm is currently selected and its description is

displayed beside the algorithm selection list. The other acronyms are EDF (Earliest Deadline First), FCFS (First Come First Serve), FS (Fair Share), LLF (Least Laxity First), PS (Processor Sharing), and SRSTF (Shortest Remaining Service Time First).

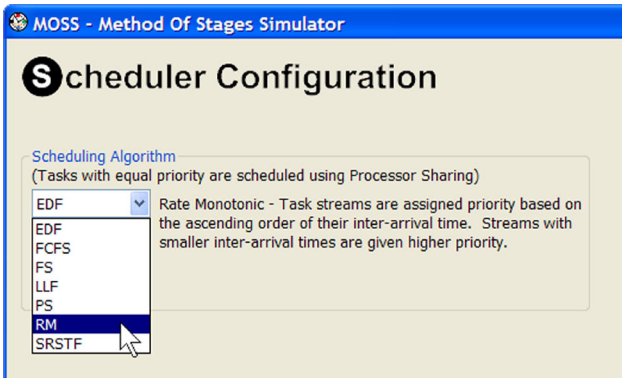

Figure 44: Portion of scheduler configuration window

One issue that arises with scheduling algorithms is what to do in the case of a tie, where multiple tasks have the same priority and all of them should be allowed to execute. For example, when using the Rate Monotonic algorithm, if two tasks have the same inter-arrival time, they have the same priority. The decision of selecting what do in this case is typically arbitrary and often left up to the implementer. In MOSS, priority is determined by using a tolerance value to test each value in question. If multiple tasks have nearly (i.e., within a small tolerance) the same determining value (e.g., deadline time in the case of EDF), then the tasks are allowed to share the processor equally. The tolerance value MOSS uses to break ties is currently 0.00001. For example, when using RM scheduling, the value that determines priority is the inter-arrival time of each task stream. Therefore, using RM scheduling, if a workload has three task streams S1, S2, and S3 with inter-arrival times of 5.000001, 5.000002, and 4.999 minutes, respectively, then streams S1

and S2 will be assigned the same priority, *P*, while S3 will be assigned a priority higher than P.

4.4.3 Simulator Configuration

The simulator configuration window allows the user to specify a seed for the internal random number generator or the user can choose to let MOSS generate a seed randomly. The seed is used to initialize the random number generator and set its start state. The seed used during any given simulation is recorded in the configuration file. This value is saved, which allows any simulation to be reliably repeated later. The simulation length is specified by a decimal value time unit. Figure 45 shows a portion of the simulator configuration window with the time unit list expanded. While the unit of time for the simulation length is being entered, a corresponding maximum value is displayed to help the user specify a valid numeric value. To reduce overhead, MOSS currently limits the simulation length to one year. The remaining option in the simulator configuration allows the user to open a browse window and select the desired output directory for resulting simulation data.

When the *Next* button is clicked, a dialog window presents the user with a summary of all the current configuration information, displayed in a hierarchical tree view as shown in Figure 46. Displaying all of the configuration information in this form helps the user to notice any undesired values or errors in their configuration settings. The settings-tree can be printed or saved in a text file.

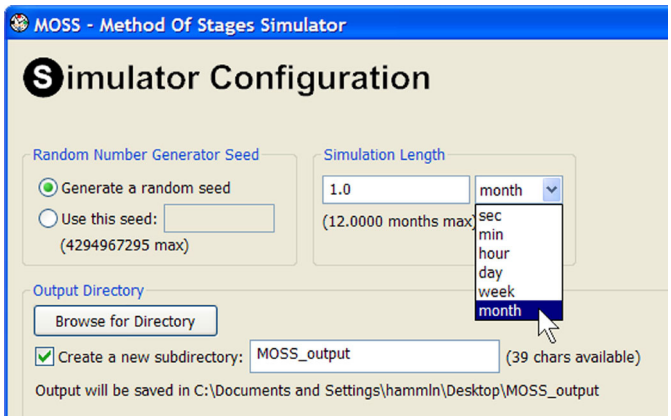

Figure 45: Portion of simulator configuration window

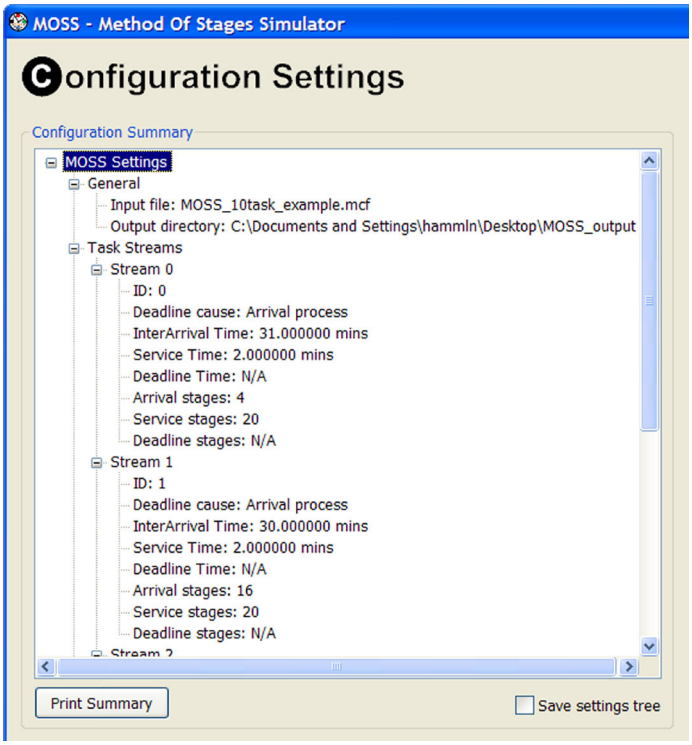

Figure 46: Portion of configuration summary window

4.5 Running a Simulation

The final dialog window allows the user to start the simulation by clicking the *Start* button. Once a simulation is started, the *Start* button label changes to *Pause*. A repeated click of this button pauses or resumes the simulation. Figure 47 shows a screenshot of a

simulation that is currently paused. While a simulation is running, a progress bar indicates the rate at which the simulation is progressing and the number of simulated minutes is displayed as well. A status message indicates the current state of the simulation (e.g., paused, or resumed) as well as any other important information, such as warnings about possible error conditions. A *Terminate* button is provided in case the user wants to terminate a simulation early. For example, if a simulation takes longer than expected, the user can terminate the simulation gracefully without waiting for it to complete. If the *Terminate* button is pressed, a message box is displayed asking the user to confirm the termination, before the simulation is actually terminated.

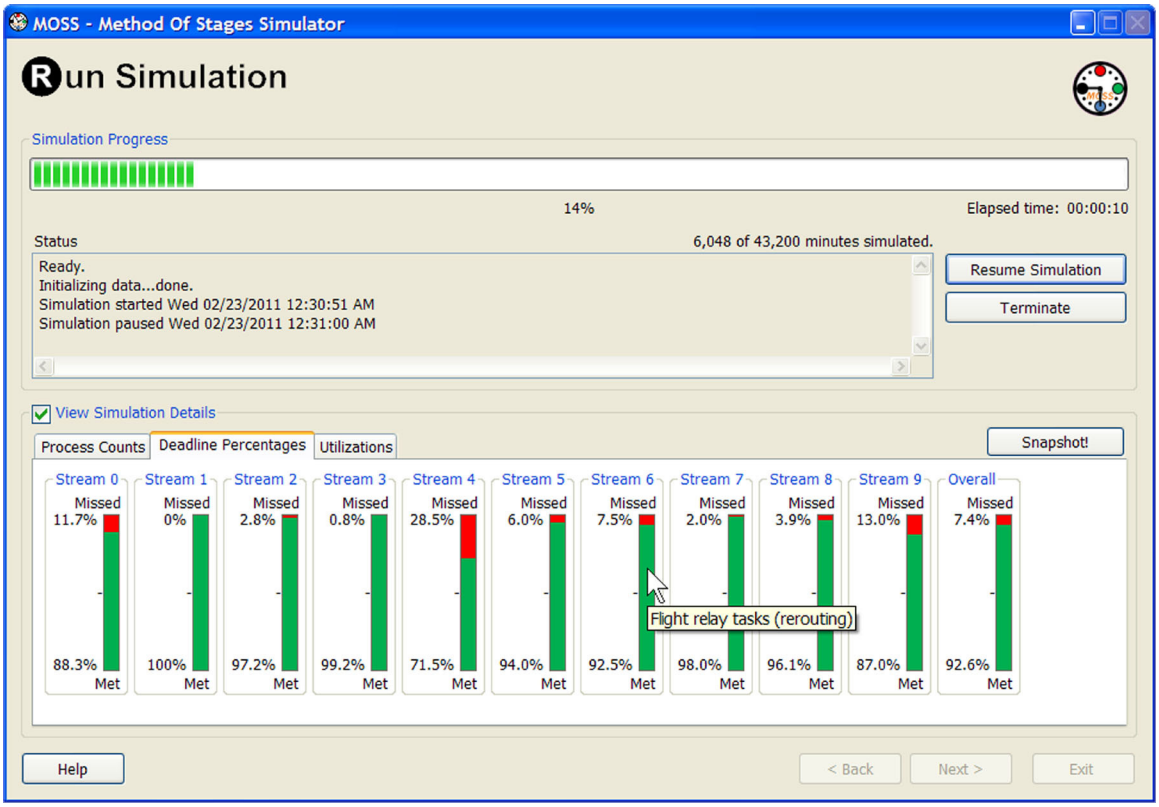

Figure 47: Run simulation dialog window

On the left side of the window is a checkbox that allows real-time simulation statistics to be viewed while the simulation is executing. By displaying these statistics, it extends the user-perceived run time of the simulation, due to the additional overhead. Therefore, a refresh interval can be selected that controls how often the statistics are updated—the smaller the interval, the more overhead is incurred and the longer the perceived run time is extended. To change the refresh interval, the user must uncheck the *View Simulation Details* option. Selecting this option displays tabs that group the statistics into categories. MOSS currently displays three categories of statistics: method of stages process counts, deadline met/miss percentages, and system utilizations.

The *Process Counts* tab displays the number of processes created, completed, and aborted for the arrival, service, and deadline processes of each task stream. The *Deadline Percentages* tab provides a graphical display that indicates the percentages of met and missed deadlines for each of the task streams. Because these percentages are an important metric, one advantage of MOSS is that it allows the user to intuitively view this information in real-time as the simulation is executing, rather than having to wait until the simulation completes. For each task stream, a single bar indicates the percentage of met and missed deadlines by displaying a green and red portion, respectively. The *Utilizations* tab displays the utilizations of each task, as well as overall, using blue bars in a manner similar to the deadline percentages.

The screenshot in Figure 47 shows a simulation that is currently paused and the *Deadline Percentages* tab is selected. The mouse pointer is hovering over task stream 6 and a small yellow label displays the custom task stream name supplied by the user during the configuration setup. When a simulation finishes, a finished message is added

to the status box and all buttons are disabled except for the *Exit* button. This message indicates that the simulation has completed and all output data has been saved successfully. MOSS does not terminate immediately so that the user can finish viewing the graphical statistics before exiting. When the *Exit* button is clicked, MOSS closes.

4.6 Examining Output Files

Several output files are created for each simulation and each file can be examined to observe desired statistics or performance metrics. Figure 48 shows a list of files that are output from one of the simulations from the pharmacy example. All of the simulation settings are saved in a MOSS configuration file that can be reloaded later in order to precisely repeat a given simulation. A text file containing the same settings in a tree format is also saved if the user selects this option. An XML file is saved that contains a summary of all the simulation results and a portion of an example summary file is shown in Figure 49. Note that the deadline time information is empty because no deadline process was configured for the given simulation. The summary file uses a spreadsheet scheme that can be opened using Microsoft Excel and other spreadsheet applications capable of opening XML documents.

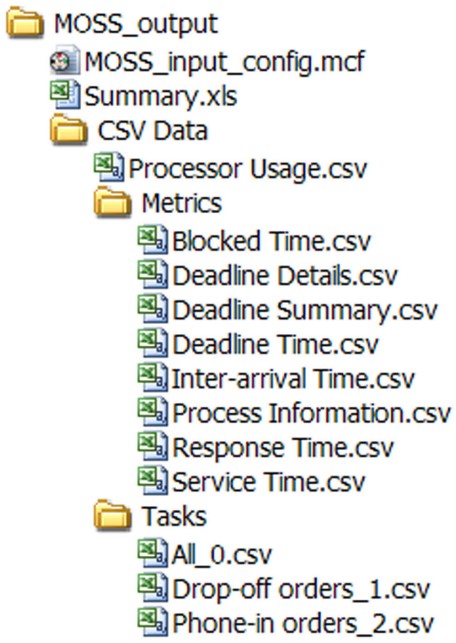

Figure 48: List of output files generated by MOSS

Summary.xls

	A	B	C	D	E	F	G	H	I	J
28		SERVICE	13340	26779.82	2.0075	0.5398	0.2914	0.2689		
29		DEADLINE	0	0	0	0	0	0		
30		Overall	28872	65879.95	2.2818	0.7897	0.6236	0.3461		
31										
32										
33		Service Time								
34		Stage Type Completed	Count	Sum	AVG	STD	VAR	CV		
35		ARRIVAL	15532	14603.4	0.9402	0.6448	0.4157	0.6858		
36		SERVICE	13340	25050.81	1.8779	0.4171	0.174	0.2221		
37		DEADLINE	0	0	0	0	0	0		
38		Overall	28872	39654.21	1.3734	0.7229	0.5226	0.5263		
39										
40										
41		Deadline Time								
42		Stage Type Completed	Count	Sum	AVG	STD	VAR	CV		
43		ARRIVAL	0	0	0	0	0	0		
44		SERVICE	0	0	0	0	0	0		
45		DEADLINE	0	0	0	0	0	0		
46		Overall	0	0	0	0	0	0		
47										
48										
49		Deadline Summary								
50			#Met	%Met	#Missed	%Missed				
51		Overall	13340	46.2	15532	53.79				
52		Per Minute	0.31		0.36					
53		Per Hour	18.53		21.57					
54										
55										
56		Deadline Details								
57		Group	1		2		3		4	
58			#	%	#	%	#	%	#	%
59		Deadlines Missed	3881	24.99	3599	23.17	4155	26.75	3897	25.09
60		Aborted by Arrival	3881	100	3599	100	4155	100	3897	100
61		Aborted by Service	0	0	0	0	0	0	0	0
62		Aborted by Deadline	0	0	0	0	0	0	0	0

All / Drop-off orders / Phone-in orders /

Figure 49: Portion of example summary file

A number of text files containing detailed statistics for each task stream are saved in CSV format in two directories, *Tasks* and *Metrics*. These two directories reflect how each group of files is organized. The files in the *Tasks* directory are grouped by task and allow the user to view all the recorded statistics for a particular task stream. The filename of each task stream file indicates the user-specified text string as well as the stream number. Similarly, the files in the Metrics directory are grouped by metric, and the user can view a particular metric for all task streams at once. In this way, a user can quickly view all the metrics for a given task, or view all the task data for a given metric,

without having to scan across multiple files. The files in the Tasks directory include the count, sum, mean, standard deviation, variance, and CV for performance metrics such as inter-arrival, service, deadline, response, and wait times. Deadline statistics include the number and percentage of met/missed deadlines for each stage of the arrival, service, and deadline processes. Utilizations, overall percentage of met/miss deadlines, and process statistics (e.g., number of service processes aborted and their causes) are output as well. Each file in the Metrics directory lists the same information, only in a different format.

4.7 Discussion

It has been demonstrated that variance does affect performance metrics and that it should be taken into account when making scheduling decisions. MOSS can easily assist in exposing such issues. Because the variance of the input parameters is related to the number of stages in a given process type (e.g., an arrival, service, or deadline process) by the formula shown in Table 12, various sensitivity analysis experiments can easily be performed. As an example, using the performance parameters from the pharmacy scenario, simulations are run where the only parameter that changes in each simulation is the number of stages in each of the arrival processes. In each simulation, the number of stages in the arrival process is the same for both task streams, but across simulations, the number of stages is varied from 1 to 10. As the number of stages in the arrival processes increases, the variance within the inter-arrival times decreases. This decrease in variance causes the arrival patterns of both task streams to become more regular, improving performance.

Figure 50 shows the results from running a series of simulations using the Shortest Remaining Service Time First (SRSTF), Earliest Deadline First (EDF), and Least Laxity

First (LLF) scheduling algorithms. The graph shows that as the variance decreases, the overall percentage of met deadlines increases, resulting in better performance. As the number of stages increases from 1 to 10, each of the three algorithms results in approximately the same overall performance boost—about a 50% increase over the starting value. This is somewhat surprising because only the second moment (i.e., variance) is changing. The mean (i.e., first moment) of a distribution is more often the focus of such sensitivity analysis experiments when comparing the performance of different scheduling algorithms. In this example, the means are held constant and yet, the trend lines for each algorithm are not flat, indicating the importance of accurately modeling the variance. This example demonstrates that variance plays an important role in the performance of scheduling algorithms. Decreasing the CV by half (i.e., from 1.00 to 0.50) results in an improved performance change of about 35% – 45%.

Table 12: Properties of an Erlang-k distribution (revisited)

pdf	Mean	Variance	CV
$\frac{(k\mu)^k}{(k-1)!}x^{k-1}e^{-k\mu x}$	$\frac{1}{\mu}$	$\frac{1}{k}\left(\frac{1}{\mu}\right)^2$	$\frac{1}{\sqrt{k}}$

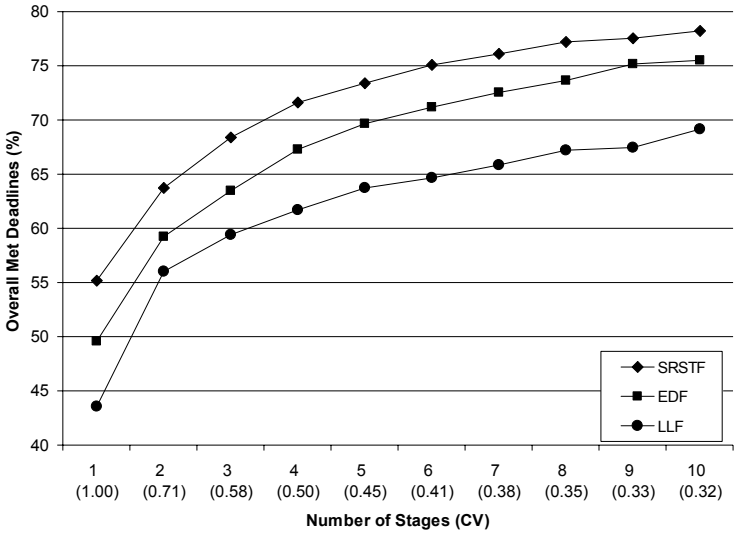

Figure 50: Algorithm sensitivity with respect to variance

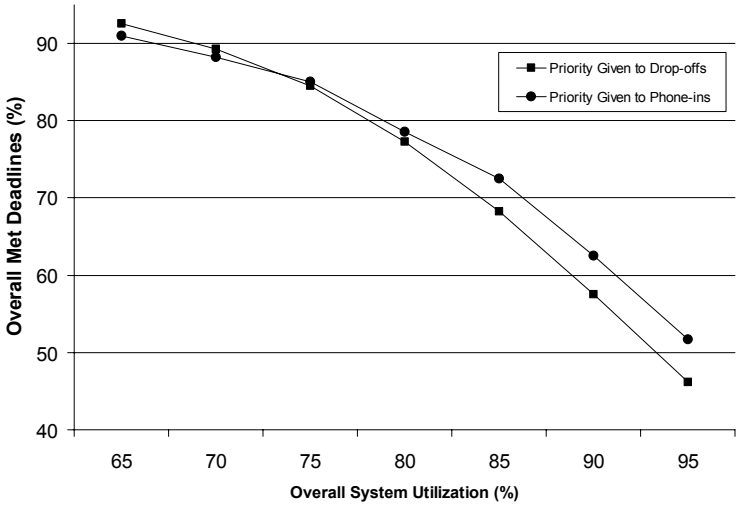

Figure 51: Algorithm sensitivity with respect to system load

Two sets of simulations were previously presented that demonstrate the effect of giving priority to one task type over the other, for both light and heavy system loads. Under light load (i.e., approximately 65% utilization) it is better to give priority to drop-off prescriptions (i.e., streams with higher variance), while under heavy load (i.e., approximately 90% utilization) it is better to give priority to phone-in prescriptions (i.e.,

streams with lower variance). One method of varying the system load is to change the number of task streams in the system because more tasks in the system will lead to higher utilization. Another method of varying the system load (i.e., the one used here), is to change the inter-arrival times. Decreasing the inter-arrival times proportionally across all task streams causes tasks of each type to arrive more frequently, thereby increasing the system load.

To span the gap between light and heavy system loads, additional MOSS simulations are run in a similar manner. The resulting graph is shown in Figure 51. As expected, as the overall system utilization increases from 65% to 95%, the percentage of met deadlines decreases significantly, from more than 90% to about 50%. When the system is about 75% utilized, it makes little difference which prescription type is given priority. Giving priority to either type results in the same percentage of met deadlines. Therefore, an effective scheduling strategy in this system would be to give priority to the task stream that will result in the highest percentage of met deadlines as a dynamic function of the system load. In this example, if the system utilization is less than 75%, priority should be given to drop-offs, and otherwise the priority should be given to phone-ins.

A similar analysis could be conducted in order to compare the performance of several different scheduling algorithms operating a under varying system loads. For any given utilization, the algorithm that produces the highest percentage of overall met deadlines can easily be identified. In particular, all of the various crossover points (i.e., utilizations where the choice of the best algorithm changes) can be identified and used in a hybrid scheduling algorithm that bases its scheduling decisions on the current system utilization. In separate simulations run using an experimental version of MOSS, the results appear to

support the claim that using such a hybrid scheduling algorithm will not only match the performance of traditional algorithms such as RM, EDF, and LLF, but also outperform them, sometimes significantly. Searching for and testing such hybrid scheduling algorithms appears to be a promising direction towards developing generic state-based scheduling algorithms. Such hybrids will necessarily include the higher moments in the workload parameters as well as system performance metrics such as the overall system load/utilization. MOSS enables sensitivity analysis experiments to help guide such research efforts.

4.8 Simulator Validation

Recall from the motivating example that the priority between two task stream types (i.e., drop-offs and phone-ins) is toggled under both light and heavy load conditions. MOSS is used to obtain estimates of the percentage of met deadlines in each case. The differences between these metrics in each case are used to demonstrate how variability alone can affect system performance. To validate these results, state-space models are constructed for each scenario and solved analytically to obtain the correct/exact values for the performance metric (i.e., percentage of met deadlines). Table 13 lists the analytical (exact) percentage of met deadlines, the MOSS estimates of this same metric, and the MOSS confidence intervals. The table indicates that with 99% certainty, the MOSS estimates are accurate to within one-tenth of one percent. This validation of the MOSS values demonstrates that they are both accurate and significant, supporting the argument that changing only the variability affects the performance considerably.

Table 13: Summary of MOSS performance estimates with confidence intervals

		Overall % Met Deadlines		MOSS Confidence Intervals	
System Load	Priority Given To	Analytical Values	MOSS Estimate	95th	99th
Light	Drop-offs	92.71%	92.66%	± 0.040	± 0.062
	Phone-ins	91.10%	91.06%	± 0.071	± 0.109
Heavy	Drop-offs	46.28%	46.23%	± 0.058	± 0.090
	Phone-ins	51.62%	51.67%	± 0.048	± 0.074

4.9 Chapter Summary

Many soft real-time systems, such as those involving airline flight information, weather forecasting, and highway traffic analysis, operate in uncertain and unpredictable operating environments. This causes tasks in such systems to exhibit unpredictable and variable inter-arrival, service, and deadline behavior. It has been demonstrated that modeling this variability is important and it can have a significant effect on the performance of scheduling algorithms. This chapter describes MOSS, a simulation tool that uses the method of stages technique to model the variability in the workload of soft real-time systems in a uniform and understandable manner.

MOSS has been used to conduct sensitivity analysis on input parameters such as inter-arrival. It demonstrates that variance can significantly impact the overall system performance. This variability impacts the performance of various scheduling algorithms by as much as 50%. This topic will be explored in more detail in Chapters VI and VII. Prior to this, Chapter V provides a more detailed discussion of modeling and simulation of variance in order to provide an overview of the basic structure of our remaining sensitivity analysis experiments.

4.10 Research Contributions

The contributions presented in this chapter include:

- The introduction, explanation, and demonstration of the Method Of Stages Simulator (MOSS) through live screenshots and examples
- Discussion of advanced features of MOSS, including an installation and setup program, graphical user interface, flexible configuration file format, built-in file editor, and extensive simulation output
- Demonstration of the usefulness of MOSS through a specific workload example, illustrating the ease in which the effects of variance on system performance can be studied in detail

CHAPTER V

MODELING AND SIMULATION OF VARIANCE

5.1 Introduction

Much work has been done involving the performance analysis of scheduling algorithms but little is known regarding the impact of variance and higher moments on scheduling decisions. The following work involves conducting in-depth sensitivity analysis experiments, developing variability-based guidelines for improving system performance, and developing improved state-based scheduling algorithms by incorporating these results into hybrid scheduling strategies. This chapter discusses the reasons for the chosen modeling technique (i.e., method of stages) and describes the general techniques and procedures used to conduct sensitivity analysis experiments in the remainder of this work. Added emphasis is placed on task laxity because it is an important concept that is used in the development of the resulting hybrid algorithms. A large number (approximately 75,000) of simulations are run using MOSS to gather the data for the results presented in this chapter. In light of this work, several improvements can be made to the MOSS tool to improve its usability and functionality. The last section of this chapter provides a discussion of a new prototype of MOSS based on this feedback.

5.2 Choosing a Modeling Technique

Variability can be introduced into a system from various sources. In order to model and study the effects of this variability, a number of different approaches can be taken. To decide upon the most appropriate modeling technique, an important consideration is the

source of the variability that is to be studied. Two types of variability that arise in performance modeling are workload variability and system variability. In real-time systems, the workload consists of task streams characterized by unique arrival, service, and deadline behavior. Workload variability, therefore, originates from the variable behavior of the arrival, service, and deadline processes of task streams. System variability results from unpredictable behavior found within the environment to which the workload is subjected. It is, therefore, a result of issues such as resource contention, device and communication failures, bottlenecks, poor scheduling policies, and overload conditions.

Other issues regarding how variability is to be modeled must be considered as well. For example, a method of representing the variability information must be selected, one that is compatible with the desired modeling framework [79]. That is, an abstraction must be defined that links or correlates the real-world variability to a mechanism in the modeling framework. Next, a modeling tool that supports and captures the necessary information must be identified and implemented. Finally, the issue of verification must be addressed, where the results obtained from the modeling tools are verified. Fortunately, a number of verification techniques are known and are applicable to a wide range of real-time modeling environments [66].

Our goal is to study the effects that workload variability has on the overall system performance in a soft real-time environment. Therefore, a technique is required that both generates accurate workloads containing realistic and variable behavior, and allows this variability to be systematically changed and studied. The method of stages technique meets these requirements and lends itself well to analytical, as well as simulation,

modeling techniques. Because the scheduling algorithm is directly affected by the variable nature of task streams, the performance differences resulting from scheduling decisions serve as good indicators as to the effects of changes in workload variability. Therefore, the scheduling algorithm is selected as the main criterion to evaluate the overall effects on system performance.

In this work, the variability to be studied is explicitly represented in the behavior of tasks that make up the workload. That is, there is a direct correlation between the variability in the real-world task behavior and that of the task representation used in the modeling framework. Using the method of stages technique, stage-type distributions are used to generate accurate task streams that result in realistic workloads. The variability of these streams can be systematically modified and the resulting performance effects on different scheduling algorithms can be studied. A custom simulation tool, MOSS, utilizes this method of stages framework to analyze the effects of variance within the workload, and to study the resulting performance of scheduling algorithms. Analytical techniques, such as state-space analysis, are used to validate the results from MOSS.

5.3 Modeling and Simulation Framework

MOSS can be easily used to conduct a wide range of experiments to determine the effects of variance on system performance. To help guide the discussion of the general modeling framework, a simple system composed of two task streams, S0 and S1, is presented. These two streams are characterized by the performance parameters shown in Table 14. Both task streams have the same mean inter-arrival time but slightly different service times. S0 has more regularity in both its arrival and service behavior, while S1 exhibits more variability in these parameters. For simplicity, the deadline of each stream

is assumed to coincide with its next arrival (i.e., the deadline process *is* the arrival process).

Table 14: Task stream parameters for the baseline system

Task Stream	Inter-arrival Time (min)	Number of Arrival Stages	Service Time (min)	Number of Service Stages
S0	6.0	10	2.85	10
S1	6.0	2	2.95	2

5.3.1 Sensitivity Analysis Experiments

Using MOSS, a number of simulations[13] are run to generate the data used to create sensitivity graphs and comparison results. Figure 52 shows a sensitivity graph that compares the performance of the EDF, LLF, and SRSTF scheduling algorithms for an intensity factor[14] (IF) of 1.0 as the number of stages (i.e., variance) of the task workload changes. This graph corresponds to the baseline system, except that the number of stages for the arrival and service processes of S0 is varied; all other parameters for both task streams are held constant. Because the inter-arrival times of each stream are roughly twice the value of their service times, this graph corresponds to a lightly loaded (i.e., approximately 46%-80% utilized) system.

[13] The simulation length used in these experiments is 30 days.
[14] An intensity factor is a multiplier for the arrival rate such that an intensity factor of 2.0 corresponds to an arrival rate that is twice that of the baseline system.

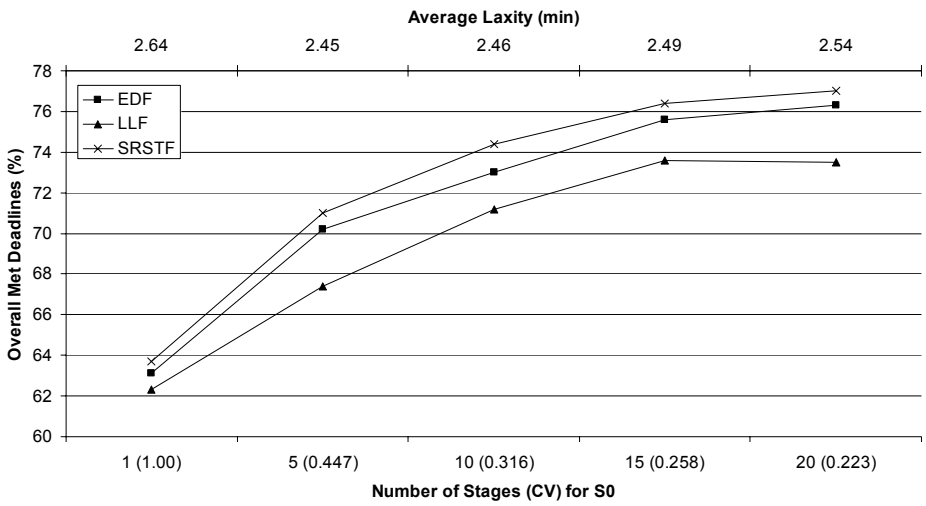

Figure 52: Algorithm comparison for IF=1.0

As Figure 52 shows, increased regularity in this under-utilized system boosts the overall performance by increasing the percentage of met deadlines. Thus, increased variability leads to a decrease in system performance and the system under light load performs best when the workload is very regular. Comparing the performance of each algorithm[15] when there is one stage (relatively high variability, CV of 1.0) to that when there are 20 stages (relatively low variability, CV of 0.223), the impact on performance is roughly the same—about a 15% decrease in the overall number of met deadlines. For a given number of stages, the average laxity for each algorithm is approximately the same and the average values of these laxities are shown along the top of the graph. Notice that under light load, the average laxity is positive.

The effects of an intensity factor of 1.5 are shown in Figure 53. With an intensity factor of 1.5, the system is moderately loaded (i.e., approximately 75% – 92% utilized) and the performance of each algorithm across the number of stages is not as widespread

[15] The RM scheduling algorithm exhibits nearly identical performance compared to LLF and is not shown in this, or the remaining graphs in this chapter.

as it is for light load. However, the performance difference across the three algorithms is more widespread when compared to the system having an intensity factor of 1.0. In this moderately loaded system, increased variability still causes a decrease in system performance, only not as much. The average laxity values have decreased but they are all still positive.

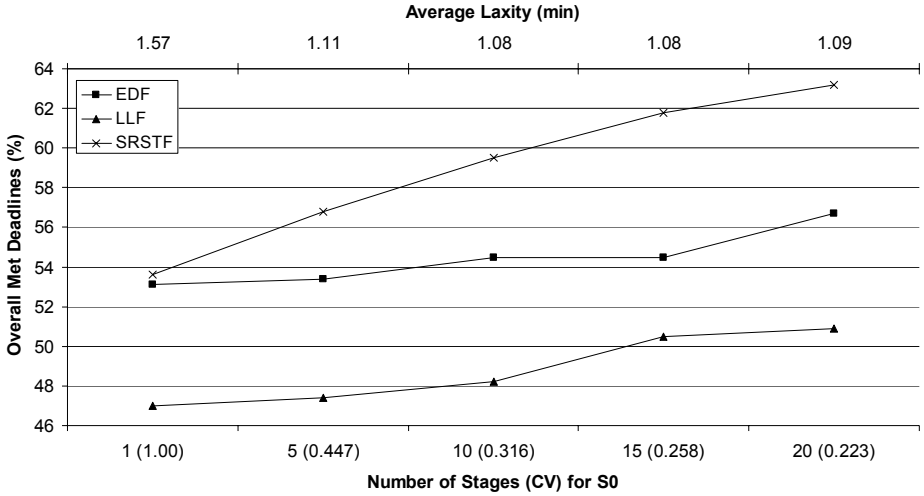

Figure 53: Algorithm comparison for IF=1.5

Figure 54 shows the performance comparison with an intensity factor of 2.0, which corresponds to a heavily loaded (i.e., approximately 87% – 99% utilized) system. Contrary to the performance of the light or moderate system loads, increased variance boosts system performance in the heavily loaded environment. The average laxity values have decreased to the point that they have become negative and this characteristic is exacerbated as the variance decreases. A negative laxity value indicates that a job currently requires an average amount of service time that is greater than its expected deadline time. Therefore, if some of the jobs expected to miss to their deadlines (identified by their negative laxity values) are aborted, there is increased opportunity for

performance improvement under heavy system load. As the workload becomes more regular, the laxity values become more negative. Therefore, the optimal conditions for laxity-based performance improvement occur under heavy, regular system load.

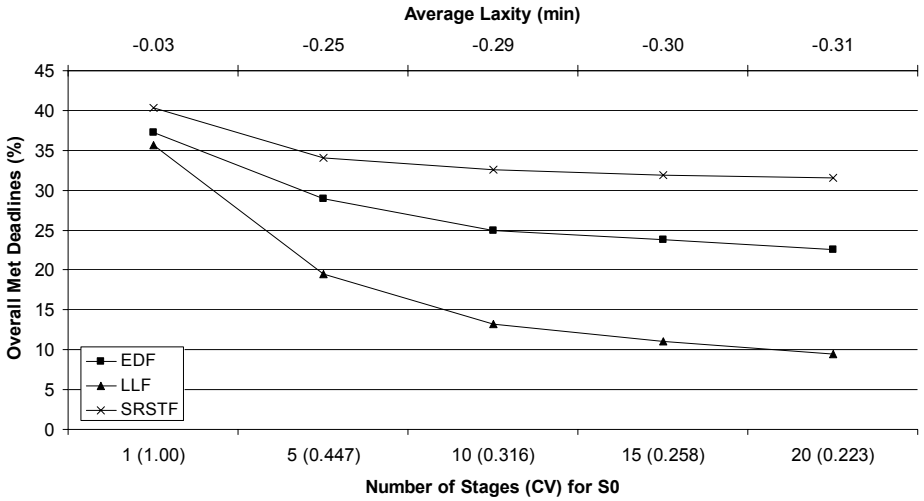

Figure 54: Algorithm comparison for IF=2.0

The results from the previous three graphs indicate that as the intensity factor increases, the percentage of met deadlines may increase or decrease as the variance of the workload changes, depending on the system load. However, the trend of the overall system utilization is consistent in that, as the intensity factor increases, the system becomes more utilized and therefore, the overall system utilization increases. Figure 55 summarizes this result where each utilization value is the average of all the individual utilizations of each algorithm. As the workload becomes more regular, the system reaches its maximum utilization, suggesting that as the variance decreases, the system utilization increases in a concave manner. In future work, we hope to prove this result analytically.

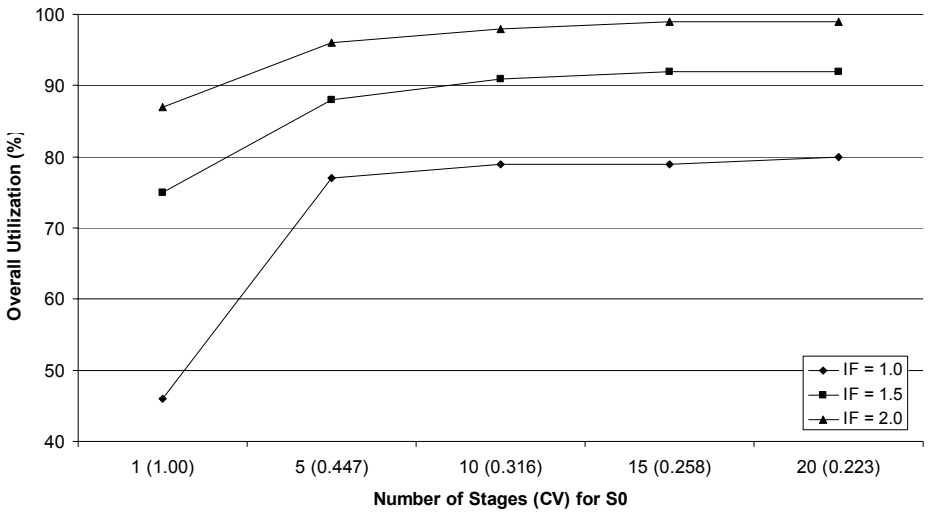

Figure 55: Overall system utilization for various scheduling algorithms

Under heavy system load, negative laxity values identify tasks that are not expected to meet their deadlines. Thus, it might be better to sacrifice these tasks, not allocate any further resources to them, and instead allocate resources to tasks with positive laxity. This research indicates that tasks with negative (and maybe even some tasks with positive laxities, depending on the system load) should be terminated unless they are characterized by high variability in their service times. A highly regular task with a negative laxity will most likely miss its deadline and therefore, it should be aborted as to not waste any further processing time. On the other hand, a highly variable task with a negative laxity is more likely to meet its deadline because its irregular nature could lead to it finishing early. The difficult decision is determining exactly when a task should be aborted.

The LLF algorithm makes scheduling decisions based on laxity, but negative laxity values are often used to abort tasks without giving them any further chance to finish. This is because the variance of task parameters is assumed equal to zero, implying that a

task with a negative laxity will definitely miss its deadline. With some scheduling algorithms, negative laxity values are taken advantage of in an attempt to boost performance, but the system variance is rarely taken into account. Therefore, because many existing scheduling algorithms typically do not incorporate variance into their scheduling decisions, their performance can be significantly affected, particularly under heavy load. Considering these negative laxity values and the effects of variance can be utilized and taken advantage of to develop new scheduling guidelines based on the variability of the workload. This issue is investigated further in order to determine how task variability can be used to schedule tasks most effectively and abort (i.e., give up on) those that are expected to miss their deadlines. Examining different ways of using task laxity to improve performance is of particular concern.

5.3.2 Variability-Based Guidelines for Scheduling

Although the sensitivity analysis presented in the previous section is relatively simple, closer inspection reveals some interesting observations. For example, as the arrival rate intensity factor increases from 1.5 to 2.0, the overall effect of variance on system performance is inverted. Under light and moderate load, increased variability hurts system performance, but under heavy load, the increased variance results in a higher percentage of met deadlines. Therefore, there exists an intensity factor between 1.5 and 2.0 that results in an overall balancing effect of variance on the system performance. That is, there must be some intensity factor such that the overall percentage of met deadlines remains approximately constant as the number of stages is varied from 1 to 20.

A relative measure of change in the percentage of met deadlines can be obtained for a given algorithm (e.g., SRSTF) by computing the difference between the met-deadline

percentage using one stage and the met-deadline percentage using 20 stages. This provides a relative measure of comparison between a system with high variability and one with high regularity. Although the percentage of met deadlines is commonly used for performance comparisons, sometimes the raw number of met deadlines can be more useful or relevant than the percentages. For example, under light load, the percentage of met deadlines is relatively high but there are fewer jobs in the system. In other words, the gross number of jobs is less in a lightly loaded system, which indicates excess system capacity. Under heavy load, the percentage of met deadlines is significantly smaller, but there are many more jobs in the system. Therefore, a system under heavier load can often outperform a more lightly loaded system in terms of raw system throughput.

Figure 56 illustrates the change in the percentage of met deadlines for SRSTF as the system load increases from moderate (IF=1.5) to heavy (IF=2.0). The dotted line indicates the point at which the overall change in the met deadline percentage is approximately zero. Therefore, for an intensity factor of approximately 1.68, the overall system performance is expected to be nearly constant, regardless of the system variance. Using MOSS to simulate a workload having an intensity factor of 1.68, the resulting change in the percentage of met deadlines is found to be 0.1%. The corresponding system utilization ranges from 81.6% to 95.8% and the average laxity ranges from 0.233 minutes (13.98 seconds) to 0.375 minutes (22.5 seconds). Therefore, if the SRSTF algorithm is used, a target utilization of around 88% is expected to minimize the effects of variance on the overall percentage of met deadlines.

Similarly, Figure 57 shows that an intensity factor of approximately 1.71 minimizes the effects of variance on the overall number of met deadlines. Using SRSTF, a target

utilization of approximately 90% is expected to minimize the effects of variance on the overall number of met deadlines. Similar results are observed for EDF and LLF. Thus, for each scheduling algorithm, there exists a preferred target utilization range that minimizes the effects of variance and maximizes the overall system performance.

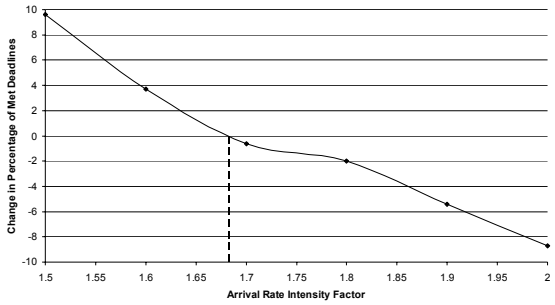

Figure 56: Change in percentage of met deadlines for SRSTF

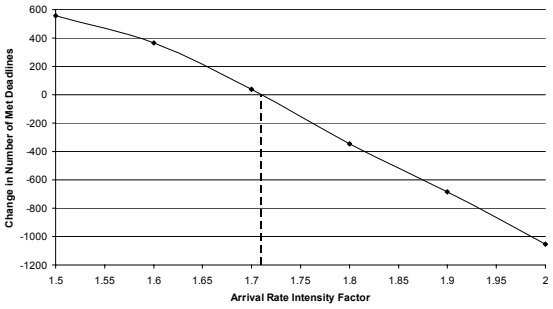

Figure 57: Change in number of met deadlines for SRSTF

An important use of MOSS is analyzing results from sensitivity analysis experiments to determine a target utilization range for a given scheduling algorithm. Adapting an existing algorithm in this manner essentially renders it unaffected by variability in the system workload. As shown, this can lead to a significant performance increase, particularly under heavy system load. These results provide the first important step of analyzing the effects of variance on scheduling decisions. Using MOSS to systematically test the impact of variability on task scheduling has proven fundamental in identifying important trends that warrant future research.

Overall, MOSS can be used to vary a number of parameters for each task stream, as well as the scheduling algorithm used to schedule or abort tasks. By conducting more detailed sensitivity experiments using the previous findings, the parameters most sensitive to variance can be further studied under different workloads. For small

examples, state diagrams can also be constructed and analyzed to identify states in which these conditions exist. This state-space information, as well as the results from MOSS sensitivity analysis experiments, can be incorporated directly into scheduling strategies to develop hybrid scheduling algorithms. The performance of these hybrid algorithms and their sensitivity to changes in variability can be compared to that of traditional scheduling algorithms. Using these techniques and results, MOSS can be used to develop improved state-based algorithms that outperform existing traditional scheduling algorithms.

5.3.3 Improved State-Based Scheduling Algorithms

It might be tempting to assume that a scheduling algorithm that maximizes system throughput, such as SRSTF, should ideally maintain an average laxity of zero. The idea is that tasks should barely meet their deadlines with no wasted execution time. However, the variance of the workload can significantly affect the system performance of any scheduling algorithm, particularly under heavy system load. Increased workload variability can cause tasks to take longer than expected to complete their execution and ultimately miss their deadline. Therefore, unless the variability and intensity of a workload can be guaranteed to stay within a predetermined range, the previous results show an average laxity of approximately 18 seconds (out of a total average execution time of 3 minutes) is ideal in order to minimize the effects of variance.

Figure 58 shows a comparison of the number of met deadlines for SRSTF as the workload intensity increases, where the average system utilizations for each throughput curve (and intensity factor) are shown along the top of the figure. Each throughput curve corresponds to different variability within the workload, ranging from relatively high variability (n=1, CV=1.00) to relatively high regularity (n=20, CV=0.223), where the

dotted circles indicate the maximum point of each curve. Recall that an intensity factor (IF) of approximately 1.7 was previously identified as optimal in order to minimize the effects of variance on the system performance. An IF of 1.7 in the graph corresponds to the point at which all of the curves are closest to each other according to the sum of their differences. To the left of this point, increased regularity improves system performance but to the right, increased variability boosts performance.

Under moderate to heavy system load, the increased performance due to high variability is significant. At IF=2.0 for example, an additional 800 (approximately 18%) deadlines are met for the n=1 curve, compared to the next best performance of the n=5 curve. The number of met deadlines for IF=2.0 ranges from 4,770 (n=20 curve) to 5,825 (n=1 curve), which is a difference of 1,055 deadlines (22% relative performance difference). For light to moderate loads, the system performance is maximized by maintaining as much regularity in the workload as possible. Under light load (i.e., IF=1.0), the increased performance due to high regularity leads to an additional 210 (approximately 4%) deadlines met when comparing the baseline system (i.e., n=10 curve) to the more regular system (n=20 curve). For IF=1.0, the number of met deadlines ranges from 4,629 (n=1 curve) to 5,546 (n=20 curve), which is a difference of 917 deadlines (20% relative performance difference). Under moderate load (i.e., IF of approximately 1.4), the increased performance due to high regularity leads to an additional 393 (approximately 7%) deadlines met when comparing the baseline system to the more regular system. Therefore, the performance gain of the more regular system (i.e., n=20) over the baseline system (i.e., n=10) is nearly doubled under moderate load, when compared to light load.

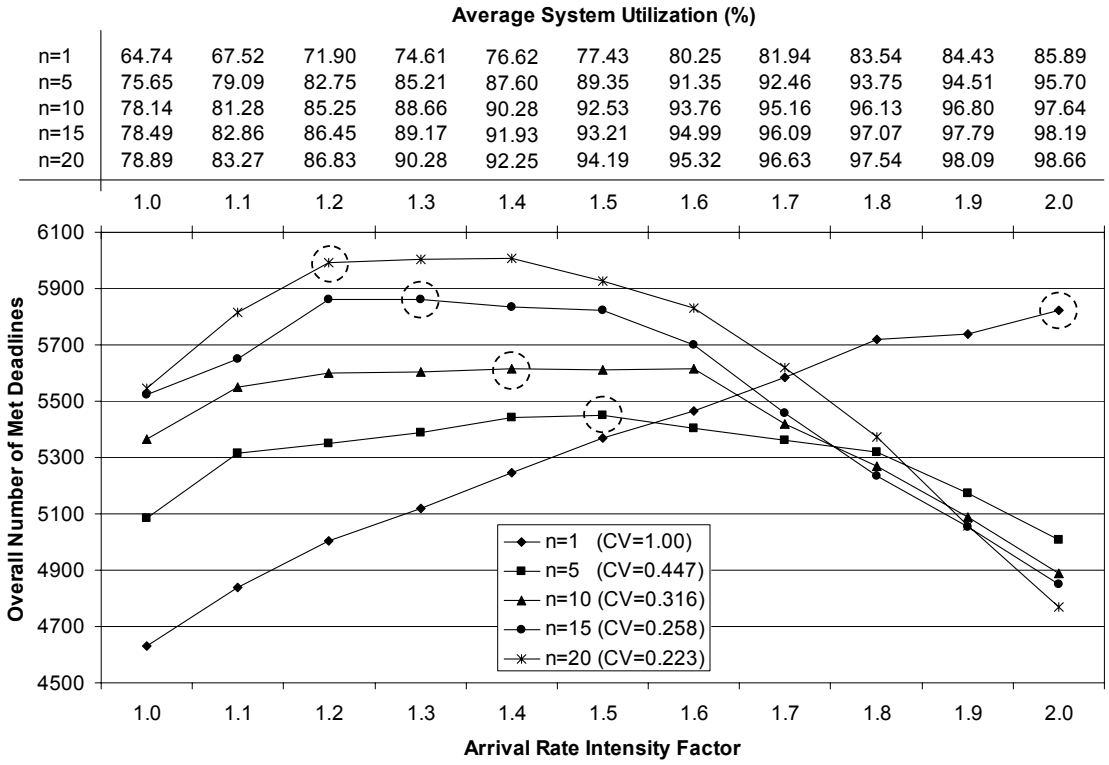

Figure 58: Throughput comparison and system utilizations for SRSTF

If knowledge of the workload variability is known, the information in Figure 58 can be used to determine a target range for system utilizations. Recall that the baseline workload variability corresponds to the n=10 curve. Therefore, the ideal target workload IF for this system is approximately 1.4, which will result in the maximum number of met deadlines. In practice, the workload intensity is often not known in advance, or it may not remain bounded. Thus, it is beneficial to develop scheduling strategies that dynamically adjust their behavior based on the current workload variability.

Alternatively, maintaining a target system utilization range may be more practical. As the workload intensity increases, the system utilization will also increase due to the additional load. Therefore, a workload with an intensity factor of 1.4 can become less intense and decrease towards an intensity factor of 1.3, or it can become more intense and

increase towards an intensity factor of 1.5. If either of these intensity factors is reached, the number of met deadlines will be less than optimal. Therefore, a target range for the system utilization can be determined a priori using MOSS. Then, during execution, the system can take self-adjusting actions (e.g., admitting fewer or more tasks) to maintain its target utilization range. Using MOSS, the target utilization range for a workload with intensity factor 1.4 is about 89% to 92%. MOSS is used to create the list of system utilizations shown along the top of Figure 58, which allow target utilization ranges to be identified for different workload variability and various load conditions. Maintaining such a target utilization range maximizes the percentage of met deadlines and at the same time, minimizes the effects of changes in workload variability.

A number of the previous results can be incorporated into an experimental hybrid scheduling algorithm that takes variance into account. Such an algorithm can be designed to minimize the effects of variance regardless of system load, improving its overall performance and applicability to real-world systems. In addition, the explicit incorporation of variance in scheduling decisions can help guide the development of state-based scheduling algorithms that either take advantage of the known system variance, or are independent of it. Examining different strategies for incorporating the task laxity into scheduling decisions is also important.

5.4 Prototype of the Next Version of MOSS

Using MOSS, a number of experimental techniques for studying variance have been presented in the previous sections. In light of the large volume of data gathered, some suggested improvements can be made to the MOSS tool. This section provides a discussion of some of these improvements and includes screenshots of a prototype of the

next version of MOSS. Although the user interface of MOSS has already been updated and is reflected in the screenshots in the following sections, the underlying implementation details are left for future work.

The main improvement to MOSS is the support for automatically running batches of related (but independent) simulations. With the current version of MOSS, a user starts the MOSS application, configures the desired options, and waits for the simulation to complete and output data. The user then repeats this process the desired number of times and calculates the average value of target performance metrics. In this manner, the user does tedious work by repeatedly running the simulator to gather output data and computing the averages of performance metrics. With the MOSS prototype, the simulator automatically runs batches of simulations that are related according to the criteria specified by the user. The average, standard deviation, variance, coefficient of variation, and (90%, 95%, and 99%) confidence intervals are automatically computed for each performance metric. This new functionality greatly improves the user "friendliness" of MOSS and makes it applicable to a wider range of usage.

5.4.1 Main Dialog Window

The main dialog window of the MOSS prototype is shown in Figure 59. This window has been updated to make it easier for the user to run simulations by providing additional help when a manual configuration option is selected. There are four types of manual configuration options provided. Each one serves as a template that populates the subsequent fields and windows of MOSS with default values.

A *Basic* configuration type is used to run a single simulation any desired number of repetitions. This type of configuration is best suited for estimating performance metrics

for a single scheduling algorithm. The *Advanced* configuration type is used to run a group of independent simulations any number of times. This option is best suited for repeatedly evaluating a performance metric across a range of values. The *Algorithmic* configuration type is designed for conducting sensitivity analysis experiments on scheduling algorithms. With this simulation type, the user specifies criteria (in a subsequent dialog window) that control which algorithms are evaluated and how the parameters (if any) of each one are modified across simulations. The last type, *Workload*, is used to conduct sensitivity analysis on workloads by systematically modifying the workload parameters specified by the user. Each of these options enhances the usability of MOSS and reduces the time required in running groups of simulations.

Support is also added for a recent files list so that a list of recently used configuration files is displayed for convenience. Figure 60 illustrates the main dialog window with the configuration file options enabled (as opposed to the manual configuration options emphasized in Figure 59). A user can browse for a configuration file not in the recent list, or select a recent filename from the list and load the file immediately. An attribute named *Workload Description* has been added to the configuration file format that allows a detailed description of the workload to be saved inside the corresponding configuration file. This feature makes it easier for a user to distinguish among similar workloads, particularly when conducting lengthy sensitivity analysis experiments involving multiple configuration files.

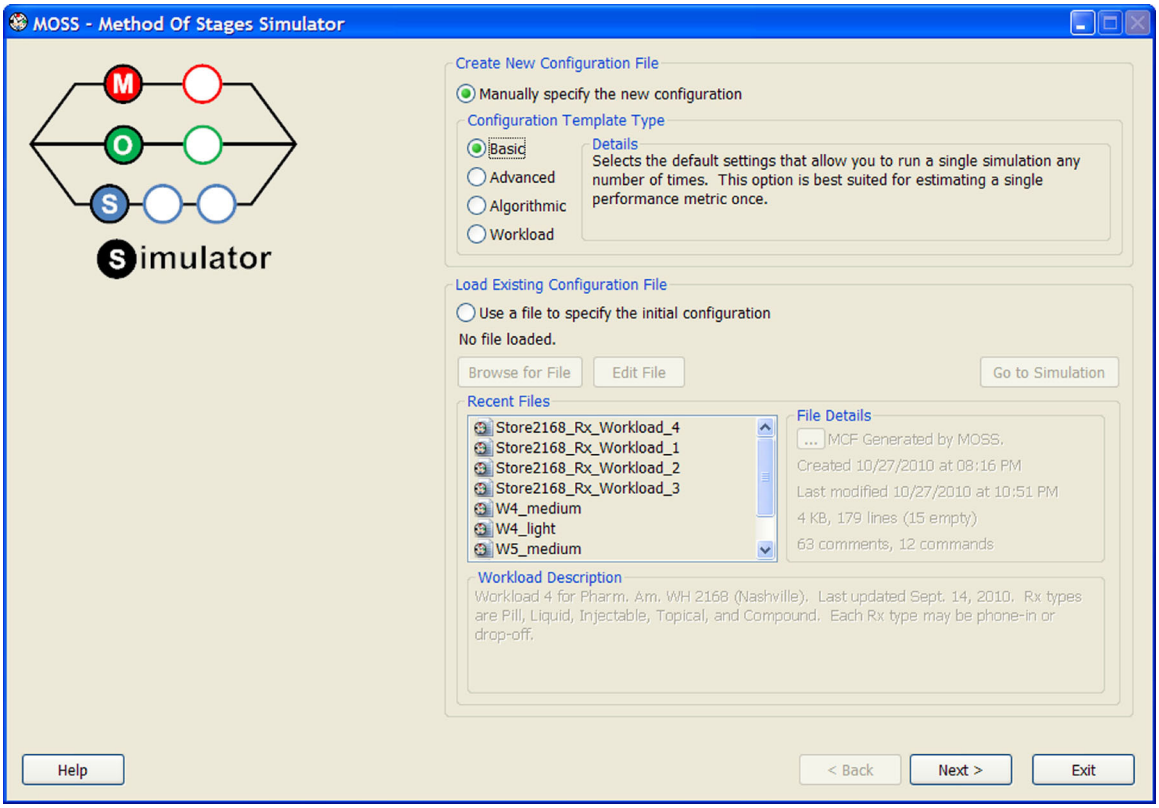

Figure 59: Prototype of main dialog window emphasizing manual configuration options

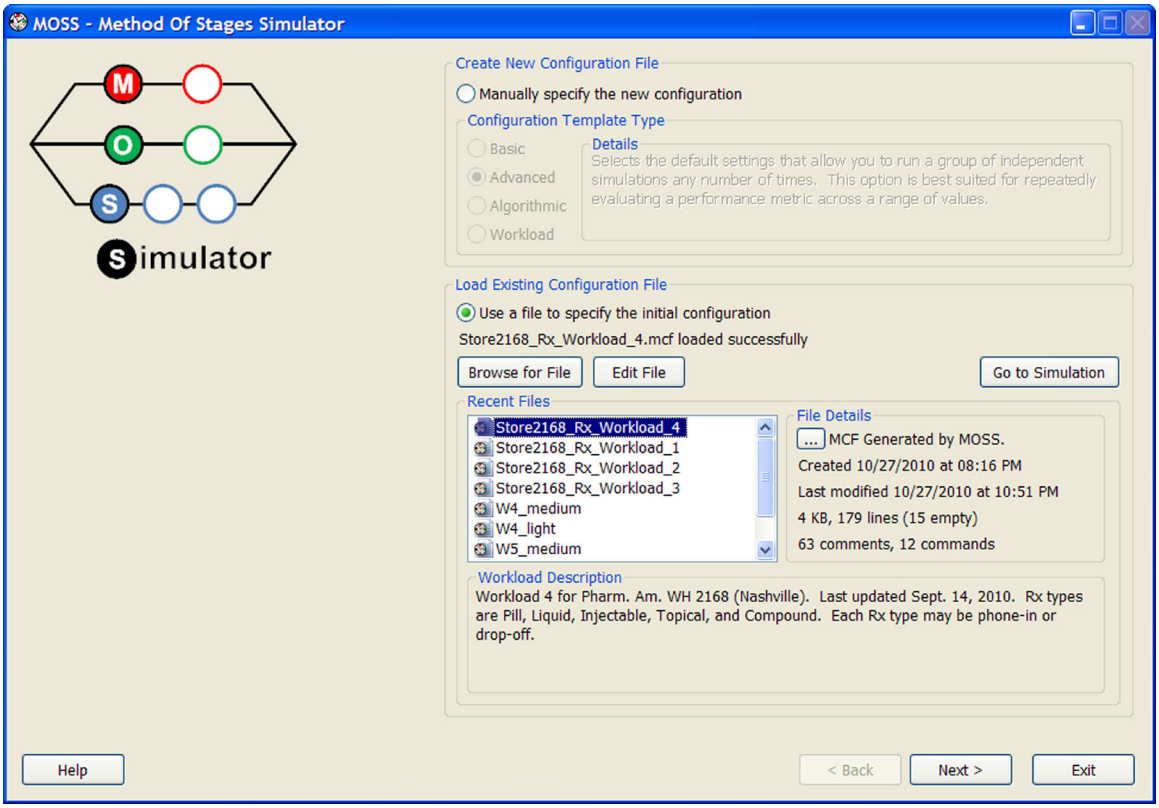

Figure 60: Prototype of main dialog window emphasizing the configuration file option

5.4.2 Configuration File Editor

Some small changes have been made to the interface of the configuration file editor in the form of larger buttons and additional support for inserting commonly used file commands (e.g., task stream, scheduler) quickly. Figure 61 shows a screenshot of the updated configuration file editor with an example configuration displayed. Notice that at the top of the configuration information, the *workload description* (as discussed in the previous section) is specified. As with all parameters, the user can edit configuration information either in the file editor (if a file is loaded) or via the dialog windows provided by MOSS.

In addition to parsing the current configuration, the user can check for errors (via the *Check for Errors* button) by using the interactive help tool. When an error is found in the

current configuration, a help wizard suggests possible causes for the error and provides suggestions for making corrections. This feature makes it easier for novice users to edit configuration files, which encourages a more thorough understanding of using and editing MOSS configuration files.

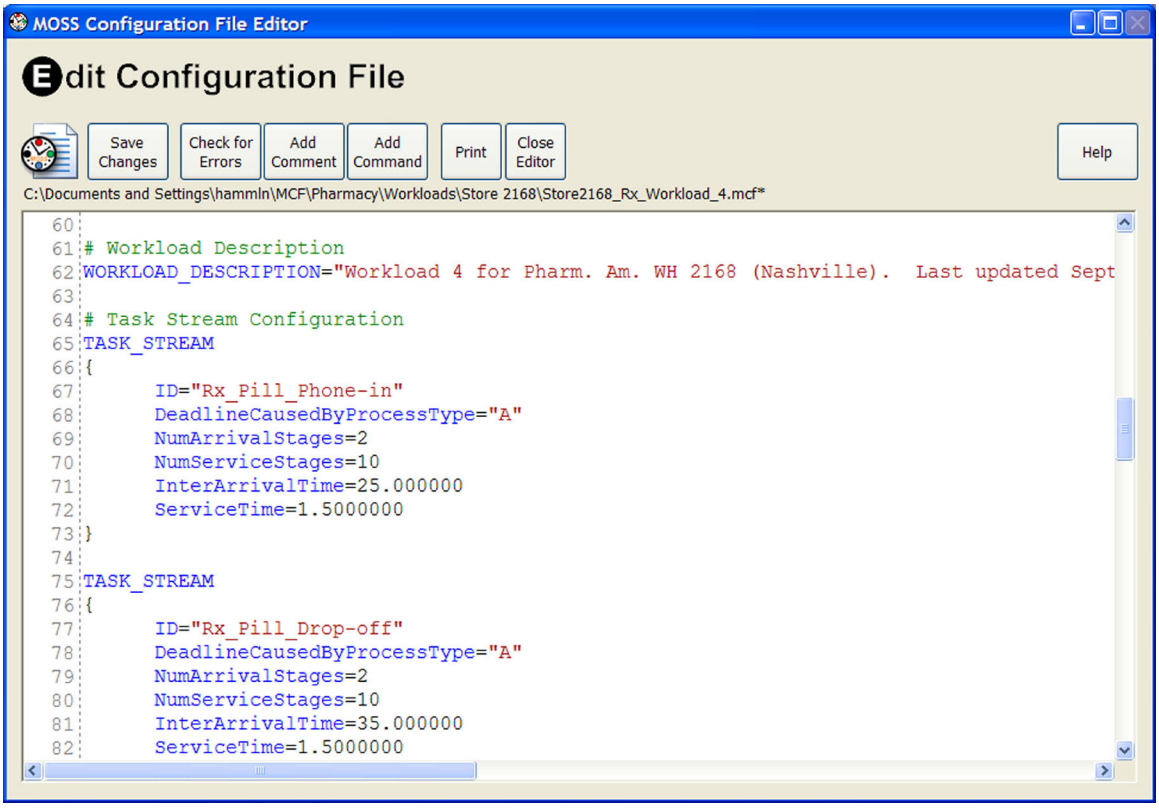

Figure 61: Prototype of configuration file editor

5.4.3 Task Stream Configuration

Figure 62 illustrates the updated task stream configuration window. At the top of the window, a field allows the user to modify the workload description. The fields providing the configuration options for the arrival, service, and deadline process of each task stream have been redesigned to make it easier and more intuitive to adjust these parameters. For each process type, time adjustment controls allow the user to easily increase or decrease

the appropriate time parameter by clicking the increase/decrease (i.e., + or -) buttons. A step value controls the increment by which the time value is adjusted and is changed by clicking the left/right (i.e., < and >) buttons. The equivalent number of seconds and hours is updated immediately when the time is adjusted.

For each type of process, the distribution type and number of stages are selected from dropdown lists. If the selected distribution type has additional configuration parameters, the *Configure* button is enabled. Clicking this button opens another configuration window that allows the user to specify additional settings for the given distribution. For example, in Figure 62, the distribution type for the service process of *Stream 8* is currently set to a hyperexponential distribution with four stages. Clicking the *Configure* button allows the user to specify the rate parameter and branching probability for each of the hyperexponential distribution's four stages. (When the distribution type is changed in the dropdown list, all configurable parameters for the distribution are set to default values.) The variance and CV resulting from the distribution settings are displayed below the distribution type. The user can increase/decrease the number of stages by clicking the plus/minus buttons beside the number of stages. This allows the user to immediately see the effect on the variance and CV of changing the number of stages.

In the upper right of the window, a summary pane provides a graphical representation of various task stream details including relative arrival, service, and deadline behavior of tasks, as well as the relative task loads imposed on the system. In Figure 62, the task details pane currently depicts the relative tasks loads using orange bars, where larger bars indicate greater relative load. For example, task *Stream 2* (*S2*) of the current

configuration imposes the greatest relative load on the system, followed by streams *S3* and *S4*, respectively. A grey bar on the far right of the pane indicates the average load.

The user can change the view of the details pane by clicking the left/right buttons on the upper right. The update button (i.e., *!*) allows the user to toggle updating of the details pane on and off. (When the user switches task streams or performs other operations quickly, the application window can flicker due to constant updating.) Clicking the left/right buttons of the details pane steps through the available panes, where each option displays a different set of task stream details. For example, Figure 63 illustrates the detail panes that show the inter-arrival times and arrival rates. Similar detail panes (not shown) illustrate the relative nature of the service times and service rates. Figure 64 illustrates the detail panes that show the deadline times and deadline rates. Notice that the bar for *Stream 8* (*S8*) is red instead of blue, because *Stream 8* has an actual deadline process associated with it. The deadline of each the remaining task streams is the next arrival, and thus the arrival is the deadline (i.e., these streams do not have a distinct deadline process).

The purpose of the detail panes is to show the user, at a glance, the relative relationship among all of the task streams. When several task streams are present, it is useful to have a common representation of a particular parameter for all of the task streams. For example, if the user is adjusting the inter-arrival times of the task streams, the detail view can be switched to the inter-arrival times (or arrival rates) in order to immediately see the effect of any adjustments.

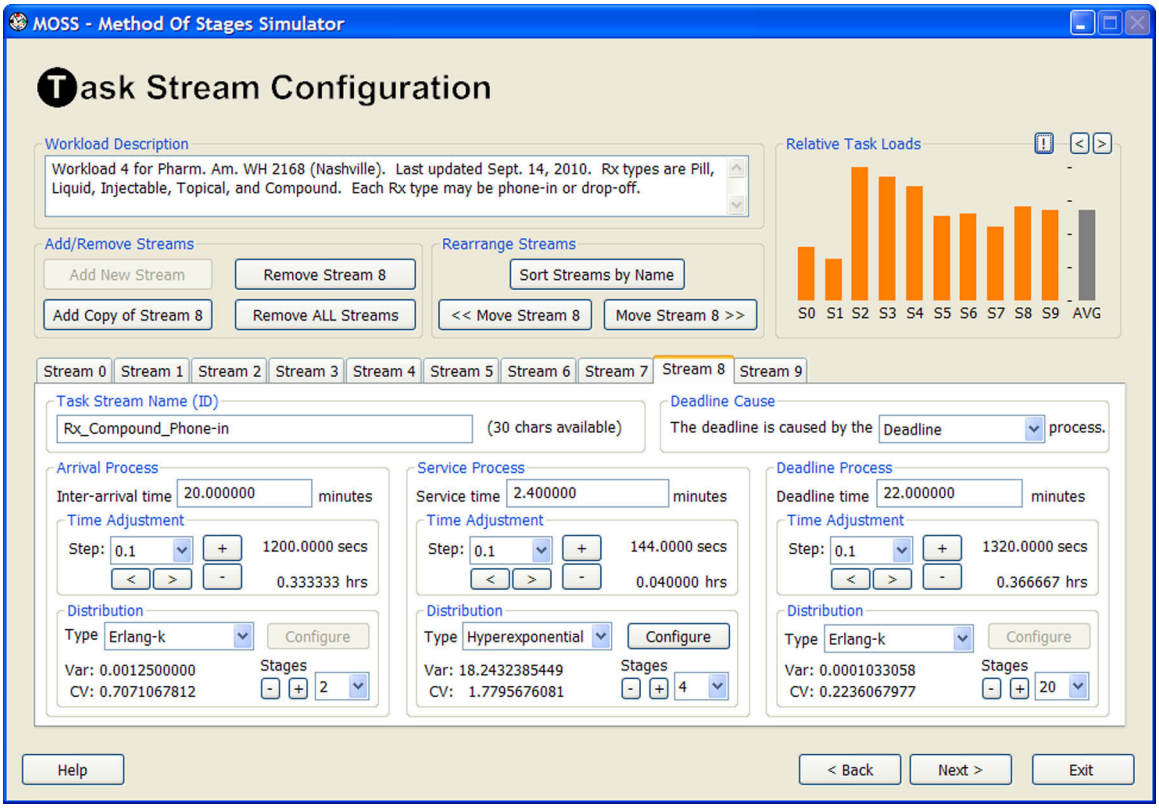

Figure 62: Prototype of task stream configuration window

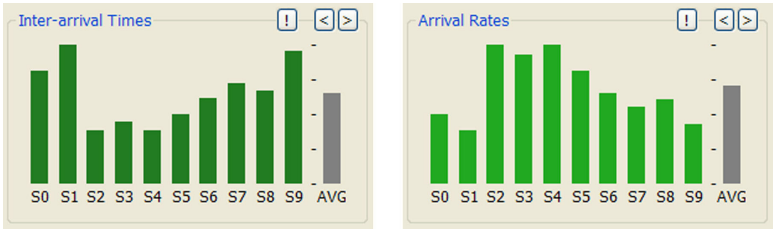

Figure 63: Portion of task details pane illustrating arrival behavior

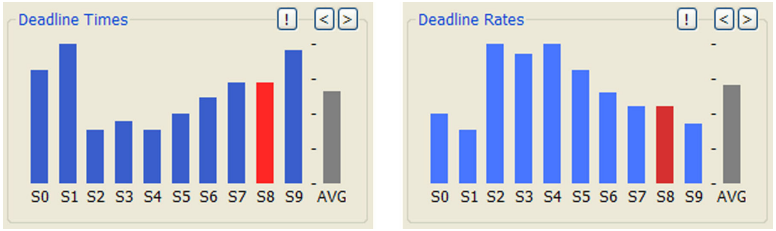

Figure 64: Portion of task details pane illustrating deadline behavior

5.4.4 Scheduler Configuration

A portion of the scheduler configuration window is shown in Figure 65. For some scheduling algorithms, configurable parameters have been added. For the TLAX algorithm, for example, the user can specify a threshold value other than the default (i.e., 0.50). New scheduling algorithms continue to be added and the options associated with each algorithm continue to increase.

At the bottom of the configuration window are options to run simulations based on a multi-processor environment. Although the default setting is a single processor, the user can specify up to ten processors. If desired, a different scheduling algorithm can be specified for each of the processors, which means the scheduler will use only the corresponding algorithm when assigning the given processor to a task. For example, for a two-processor configuration, EDF might be specified for Processor 1 while LLF is specified for Processor 2. In this case, all tasks assigned to Processor 1 are scheduled based on the EDF algorithm and all tasks assigned to Processor 2 are scheduled using the LLF algorithm. These options allow the user to investigate the effects of running different scheduling algorithms concurrently.

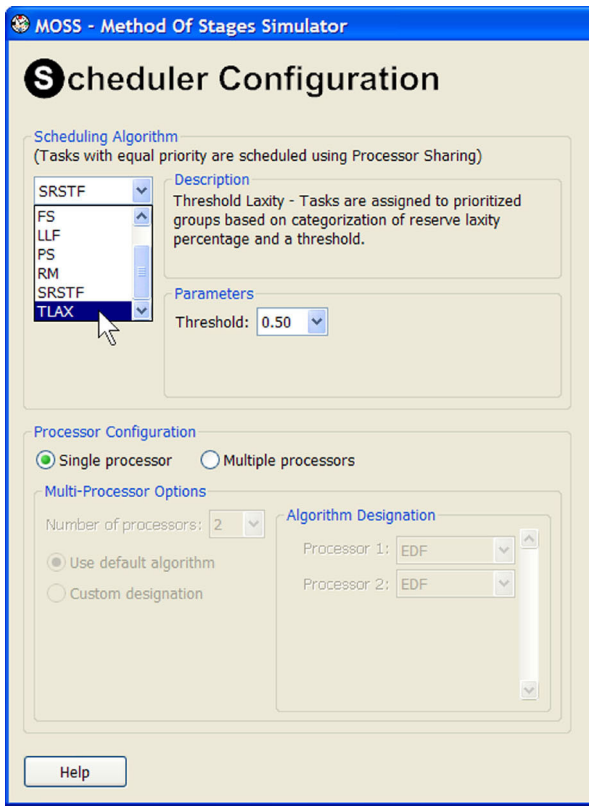

Figure 65: Prototype of scheduler configuration window

5.4.5 Simulator Configuration

The updated simulator configuration window is shown in Figure 66. As shown, a number of new features have been added that enhance the usefulness and effectiveness of MOSS. The simulation mode shown in the upper left corresponds to the manual configuration options shown in the main dialog window of MOSS (see Figure 59). In Figure 66, the simulation mode is set to *Workload analysis* and, therefore, the *Mode Settings* pane displays options for configuring simulations for sensitivity analysis of the workload.

When the simulation mode is changed, the *Mode Settings* pane is updated to display the available options for the selected simulation mode. For example, when the simulation mode is set to *Single simulation*, the mode settings are hidden because no additional options are available for running only a single simulation. For the *Multiple repetitions*

option, the only configurable parameter is the number of repetitions, which specifies the number of times the specified simulation should be run. In this case, average metrics are automatically computed and output when the simulations finish.

For the *Algorithm analysis* mode, the current configuration is used to run multiple simulations, where in each simulation, the only difference is the scheduling algorithm used. A simulation can be run for each available scheduling algorithm, or the user can select which algorithms to use in the *Mode Settings* pane. The user can also select the number of times to repeat each simulation by changing the number of repetitions. As the number of repetitions increases, so too do the accuracy and confidence of the average metrics that are later output.

For the *Workload analysis* mode, several settings are available in the *Mode Settings* pane, as shown in Figure 66. The user must select the type of parameter to systematically change, which is a time parameter (i.e., inter-arrival time, service time, or deadline time), the number of (arrival, service, or deadline) stages, or the variance. Note that for streams whose deadline process is the arrival process, changing the arrival process is equivalent to changing the deadline process. In Figure 66, the inter-arrival time is selected as the type of parameter to change. For the *Stages/Variance* option, the user is able to change the number of stages or the variance of any process modeled using an Erlang-k distribution. The number of stages (only) can be varied for hypoexponential and hyperexponential distributions. (Due to the complexity in estimating unknown distribution parameters, the options are limited to Erlang-k distributions.)

The task streams that are to be changed are selected from a list displaying all of the available task streams. Then the settings that specify the initial value and step parameters

are specified. The options for the initial value are either the current value (i.e., time value, number of stages, or variance) as specified in the task configuration parameters, or a fixed value entered by the user. This value either increases or decreases in each subsequent simulation, depending on the selection in the dropdown list. In either case, the user specifies a step value in the form of either a percentage or a numeric value. That is, if a time parameter or the variance is to be changed, the step is either a percentage or a decimal value. If the number of stages is to be changed, the step is either a percentage or an integer value. A stop value, specified as either a percentage or absolute value, specifies the maximum value for the parameter to be changed. One final option allows the user to specify all of the selected streams, or any of the selected streams, as a further constraint on the stop criteria. All of these options determine the number of simulations required, and each simulation is repeated the number of times specified by the number of repetitions.

Figure 67 shows an example of the *Mode Settings* pane after the parameter type is changed from a time value to the *Stages/Variance* option. The dropdown list for this option allows the user to systematically change the number of stages for the arrival, service, or deadline process of the available task streams. Alternatively, the user can choose to directly modify the variance of the arrival, service, or deadline process (modeled by an Erlang-k distribution) of any task stream. In Figure 67, the number of deadline stages is to be changed and, therefore, all task streams whose deadline process is not modeled by an Erlang-k distribution are disabled in the task stream list. In the figure, streams 8 and 9 are disabled because each of their service processes is modeled by a hyperexponential distribution (see Figure 62).

Table 15 provides a summary of the available combinations of configuration parameters and distribution types that can be automatically changed using MOSS. (Any possible combination can be specified for any task stream, but combinations other than those listed in the table require the user to edit configuration files.) In the table, a checkmark indicates an option with full automation support and MOSS can automatically run a range of sensitivity analysis experiments based on settings provided by the user. The *Dec* entries mean the parameter can automatically be decreased only. The remaining *CF* entries represent options that can be performed using MOSS, but require the use of additional configuration files.

As shown in Table 15, the number of stages for hypoexponential and hyperexponential distributions can be systematically decreased in experiments. MOSS achieves this by using the rate parameters already specified by the user (during task configuration) and distributing the remaining branching probability equally for each stage (this default weighting strategy can be changed by the user). For example, suppose the user specifies a four-stage hypoexponential distribution for the service process of a task stream and specifies branching probabilities of 0.1, 0.2, 0.4, and 0.3 during the task configuration step. During the simulation configuration step, the user can select options to decrease the number of service stages for the same task stream. Suppose the user chooses to decrease the number of service stages by 1 stage in each simulation. To determine the branching probabilities for the three-stage hypoexponential distribution, MOSS divides the remaining probability required (0.3 in this case) by the number of stages (i.e., 3), and therefore, uses branching probabilities of 0.2, 0.3, and 0.5 for stages 1, 2, and 3, respectively. That is, MOSS distributes the probability removed (by decreasing

the number of stages) equally across the branching probabilities of the remaining stages. Other options are provided, such as using equal branching probabilities for each stage, and the user can specify custom distribution settings using MOSS configuration files.

Increasing the number of stages for a given distribution type requires an additional rate parameter and branching probability for each stage added. Therefore, *automatically* increasing the number of stages in experiments using distributions other than Erlang-k distributions is not supported. Similarly, due to the more complex nature (e.g., a wide range of CV values) of experiments involving Coxian distributions, support for *automatically* changing the number of stages or variance of these distributions is not supported from the user interface. Lastly, systematic changing of the variance for distributions other than Erlang-k distributions is supported through configuration files only.

Table 15: Summary of available parameter and distribution combinations automatically changed by MOSS

		Distribution Type			
		Erlang-k	Hypo-exponential	Hyper-exponential	Coxian
Parameter Automatically Changed					
Category	Type				
Time	Inter-arrival	✓	✓	✓	✓
	Service	✓	✓	✓	✓
	Deadline	✓	✓	✓	✓
Stages	Arrival	✓	Dec	Dec	CF
	Service	✓	Dec	Dec	CF
	Deadline	✓	Dec	Dec	CF
Variance	Arrival	✓	CF	CF	CF
	Service	✓	CF	CF	CF
	Deadline	✓	CF	CF	CF

To examine other complex groups of simulations, the user can run each group separately. For example, the combination of an algorithm analysis and a workload analysis can be conducted by repeatedly running groups of *workload analysis* type simulations. In this way, a scheduling algorithm is selected and then the workload analysis is automatically performed by MOSS. After the simulations finish, the user can restart MOSS, select the next algorithm of interest, and repeat the same workload analysis. The user is also able to specify complex workload configurations by using Erlang-k, hypoexponential, hyperexponential, and Coxian distributions to model the behavior of each task stream. This is done from the task configuration window, or using the configuration file editor.

The final enhancements to the simulator configuration are the *Real-World Priority*, *Trace Options*, and *Output Mode*. The real-world priority is used to specify the priority of the MOSS application while it is running. The default setting is *Auto/Throttled*, which means the application automatically adjusts its priority to achieve the best performance based on criteria such as available memory and the number of programs running. The user can also set the priority level to *Normal*, *Medium*, or *High* priority. Increasing the priority causes MOSS to run faster (in real-time) but requires additional resources. The *Trace Options* affect which, if any, detailed log information is collected and later output. This information is most useful to those conducting advanced sensitivity analysis experiments using MOSS and wish to examine running statistics. Finally, the *Output Mode* can be set to either *Incremental* or *Bulk*, depending on if the user prefers for MOSS to output simulation data immediately as it becomes available, or wait and output all data after the last simulation finishes, respectively. The *Enable compression* option specifies

that MOSS should compress all the output data. This option conserves space and is useful when a large volume of information is generated.

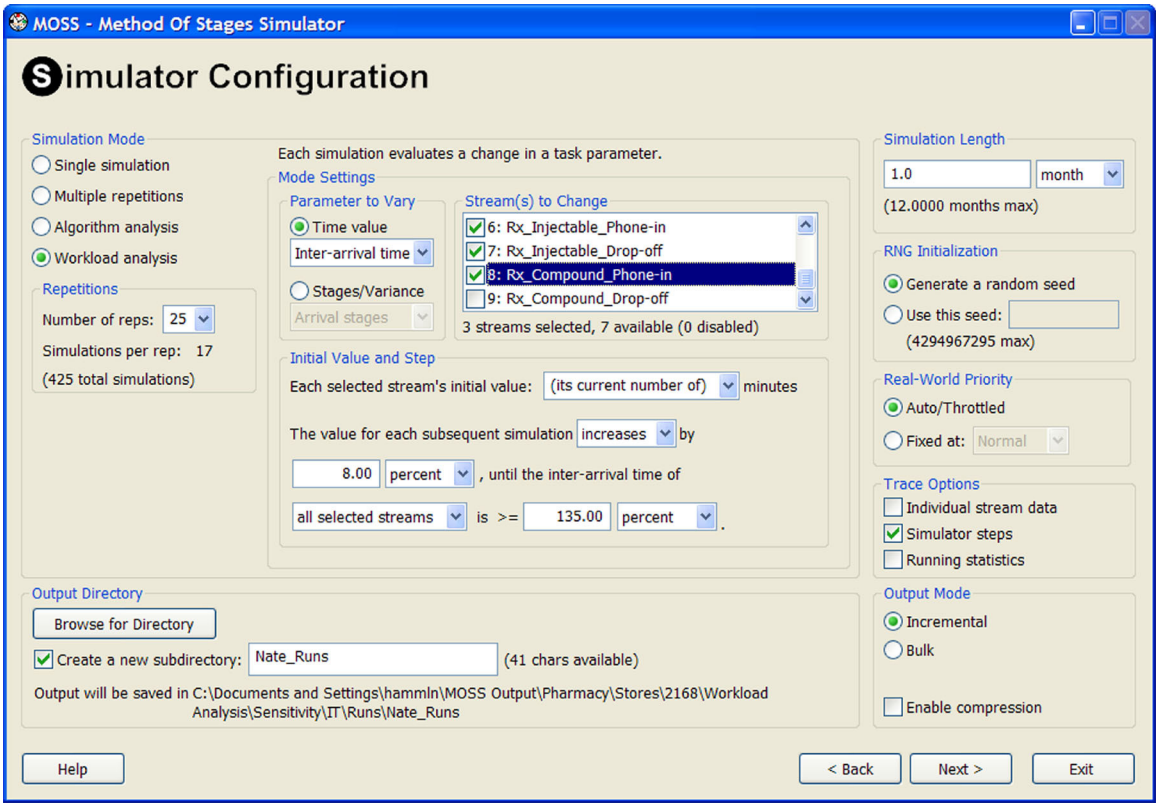

Figure 66: Prototype of simulator configuration window

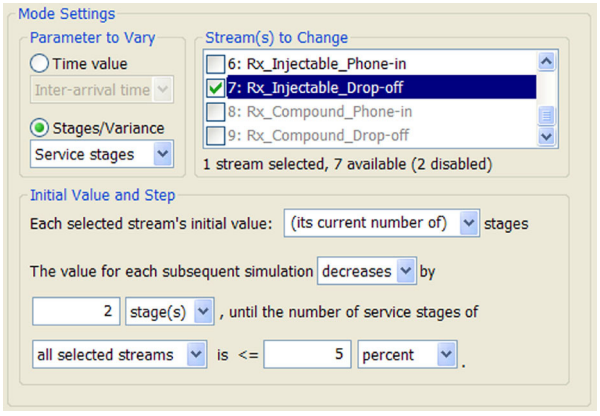

Figure 67: Mode settings pane illustrating advanced options

5.4.6 Configuration Settings and Options

The updates to the configuration settings window are shown in Figure 68. The enhancements are related to snapshots, which are images of the detailed statistics (i.e., process counts, deadline percentages, and utilizations). A snapshot combines all of these statistics for a given instant in (simulation) time and provides a summary of the entire system state at that instant. A timestamp can be included in each image if desired, and the mode of recording snapshots can be set to *Manual* or *Automatic*. In *Manual* mode, a snapshot is created and saved only when the user clicks the *Snapshot!* button from the *Run Simulation* window (see Figure 69). In *Automatic* mode, snapshots are automatically created in regular intervals that depend on the snapshot count and disk size restrictions specified by the user. Regardless of the snapshot record mode, the user can choose to output each snapshot as soon as it is created, or wait until all of the simulations have finished and output the snapshots in bulk. Lastly, the *File Format* options shown at the bottom of Figure 68 allow the user to choose which file type(s) (i.e., image format) to use for saved snapshots.

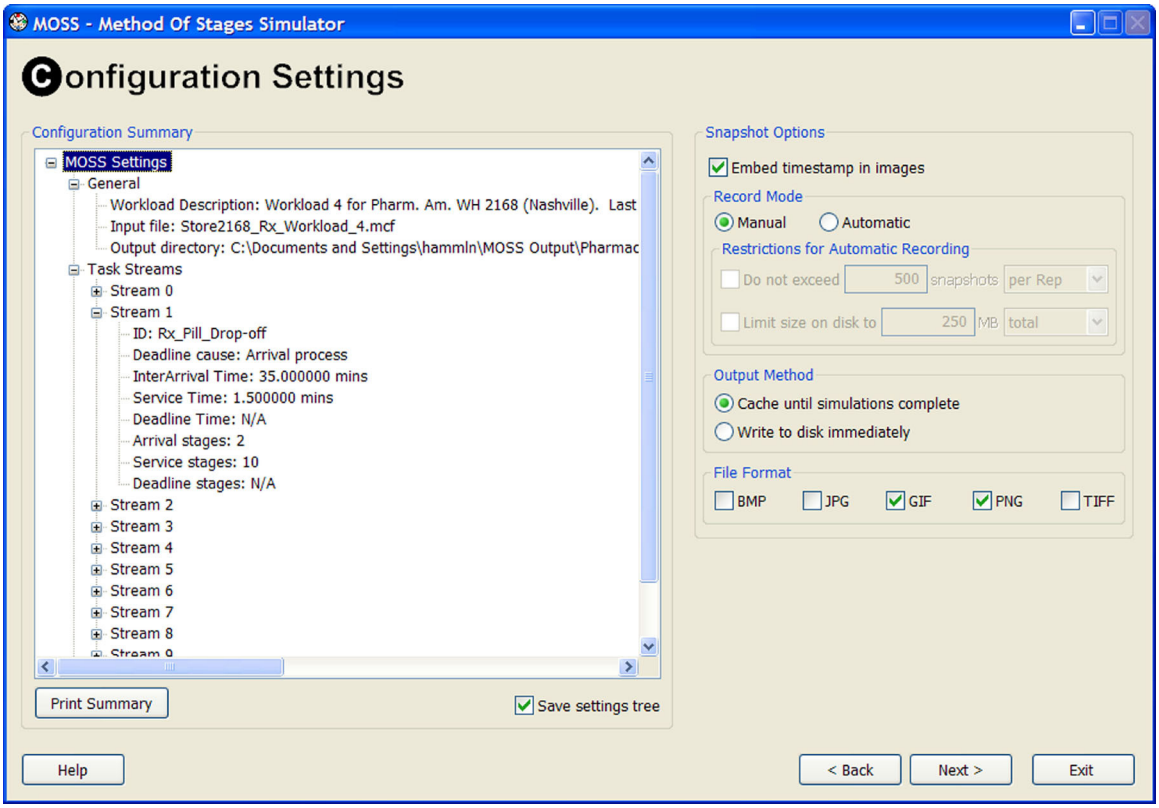

Figure 68: Prototype of configuration settings window

5.4.7 Running a Simulation

The only significant changes to the run simulation window (Figure 69) are the addition of a second progress indicator and a button to open the directory containing output files. Because MOSS will run multiple groups of possibly length simulations, the overall simulation progress, as well as the progress of the current simulation, is displayed to keep the user informed. During this time, the user can view statistics and estimates of performance metrics in real-time, via the *Simulation Details* windows. The user can immediately view any available output by clicking the *Open Output Directory* button. This button opens an operating system window that allows the user to view any of the output files or spreadsheets that are available.

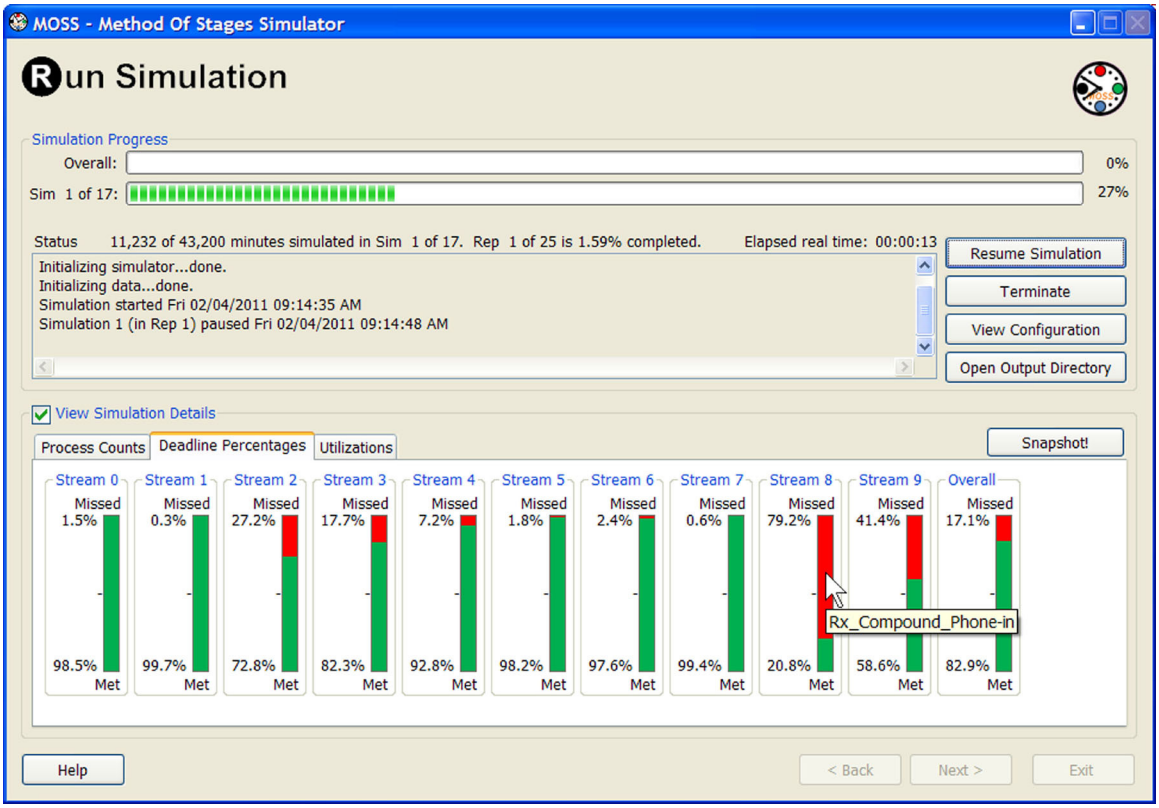

Figure 69: Prototype of run simulation dialog window

Collectively, the enhancements to MOSS provide great flexibility in conducting sensitivity analysis experiments and increase the practicality and usability of the tool. The next version of MOSS will automatically run batches of simulations and output summary metrics for each batch, allowing the user to focus on examining and interpreting results, rather than manually running simulations. The hope is that future versions of MOSS will lead to its more widespread use and help guide research efforts in the study of variance in real-time environments.

5.5 Chapter Summary

In the previous discussion, the example system is kept simple for ease of discussion. However, complex systems can be studied using MOSS just as easily. In particular, parameters such as mean inter-arrival, service, and deadline times, as well as number of

stages (i.e., variance) for each of these values, can be varied by holding all other parameters constant in order to generate sensitivity analysis results for comparison purposes. Such analysis provides the primary effect of the parameter being varied. Pairs of related (or unrelated) parameters can also be varied simultaneously in order to study secondary effects. That is, two parameters can be varied at the same time in order to determine the causal relationship that exists, if any. The myriad of possible secondary pairs of parameters prevents a full enumeration of all possibilities, but a representative sample can be tested and used to guide more focused studies in specific contexts. The MOSS prototype discussed in the previous section will improve the MOSS user interface and enhance its functionality, making it more practical and applicable for a wider range of users.

Using MOSS, many different types of sensitivity analysis experiments can be conducted. In the next chapter, MOSS is used to conduct sensitivity analysis on traditional scheduling algorithms to compare their performance to that of a new scheduling algorithm, XLAX, that considers variance and reserve laxity values when making scheduling decisions. Such hybrid scheduling algorithms take advantage of the variability within a workload and can outperform traditional algorithms.

The prototype of MOSS discussed in this chapter illustrates several changes to the MOSS user interface that enhance its usability. Editing workload configurations is made easier by the new configuration options and addition of a recent files list. The task stream configuration is also enhanced by new adjustment controls, additional distribution types, and a graphical display summarizing the relative relationships among task streams.

A number of additional upgrades, such as support for multiple processors, further enhance the power of the MOSS tool.

5.6 Research Contributions

The contributions presented in this chapter include:

- A discussion of the modeling framework and techniques used throughout various sensitivity analysis experiments using MOSS
- Observations reinforcing the argument that variability affects system performance, which include
    - Variance has about a 15% performance impact on the percentage of met deadlines under light load conditions
    - As the system utilization increases, the average laxity decreases, and increased workload variability results in better performance
    - For each scheduling algorithm, there exists a preferred target utilization range, corresponding to a workload intensity factor, that maximizes the overall system performance, as well as minimize the effects of changes in workload variability
- A presentation of the next version of MOSS and discussion of the enhanced interface and features via prototype screenshots

CHAPTER VI

SENSITIVITY ANALYSIS OF RM, EDF, LLF, AND XLAX

6.1 Introduction

In this chapter, MOSS is used to compare the performance of traditional scheduling algorithms (i.e., RM, EDF, and LLF) to that of a proposed algorithm, XLAX, that considers variance as well as a threshold-based laxity value, when making scheduling decisions. A simple but representative workload consisting of four task streams is used as the basis for sensitivity experiments. By examining a subset of the possible variability-based techniques and laxity thresholds, the relative performance of XLAX can be compared to that of the traditional scheduling algorithms RM, EDF, and LLF.

6.2 Workload Description

A workload consisting of four task streams is used to facilitate the discussion. For simplicity, the deadline for a given task is assumed to coincide with its next arrival. A summary of task parameters is given in Table 16.

Table 16: Task parameters for baseline system

Task Stream	Inter-arrival Time (min)	Number of Arrival Stages	Service Time (min)	Number of Service Stages
S0	30.0	10	4.0	10
S1	30.0	2	4.0	2
S2	20.0	10	8.0	10
S3	20.0	2	8.0	2

As shown in Table 16, S0 and S1 have the same average inter-arrival and service times but differ with respect to the variance of their arrival and service processes. S0 is characterized by arrival and service processes that have less variability than those of S1. Similarly, S2 and S3 have the same average inter-arrival and service times, but S2 contains less variability within its arrival and service behavior compared to that of S3. Notice that relative to S2 and S3, streams S0 and S1 impose lighter load on the system because tasks in these streams arrive less often and require less service time than streams S2 and S3.

The configuration presented in Table 16 serves as a moderately loaded baseline system. An arrival rate intensity factor (IF) of 1.0 is used to describe this system. Light and heavy system loads will also be discussed and correspond to intensity factors of 0.5 and 1.5, respectively. Under light load (IF=0.5), the arrival rates of all task streams are half of those found in the baseline system. Similarly, the tasks in the heavily loaded system (IF=1.5) have arrival rates that are 1.5 times those found in the baseline system. For the given workload, the light, medium, and heavy loads correspond to average system utilizations of approximately 52%, 88%, and 99%, respectively. Table 17 summarizes the workloads for light, moderate, and heavy system loads.

Table 17: Task parameters for light, medium, and heavy loads

System Load	Average System Utilization	Task Stream	Inter-arrival Time (min)	# of Arrival Stages	Service Time (min)	# of Service Stages
Light (IF=0.5)	52%	S0	60.0	10	4.0	10
		S1	60.0	2	4.0	2
		S2	40.0	10	8.0	10
		S3	40.0	2	8.0	2
Medium (IF=1.0)	88%	S0	30.0	10	4.0	10
		S1	30.0	2	4.0	2
		S2	20.0	10	8.0	10
		S3	20.0	2	8.0	2
Heavy (IF=1.5)	99%	S0	20.0	10	4.0	10
		S1	20.0	2	4.0	2
		S2	13.33	10	8.0	10
		S3	13.33	2	8.0	2

6.3 The XLAX Algorithm

Consider an experimental form of the LLF algorithm, named XLAX, that maintains x-percentage of laxity for each task at any given time. The current reserve laxity percentage r_i is computed separately for each task whenever a new scheduling decision is made. For a task t_i, its current reserve laxity percentage r_i is computed using the ratio of its remaining service time (S_i) and the amount of time remaining until its deadline (D_i), as shown in equation 1. This notation simplifies the discussion of XLAX and does not affect the connotation associated with the term laxity. For the remainder of the discussion, the term *reserve laxity* is used to loosely refer to a reserve laxity percentage. Table 18 provides a summary of the relationship between the absolute laxity[16] and the corresponding reserve laxity percentage (r_i) of a task t_i.

[16] Absolute laxity refers to the value of the remaining service time subtracted from the time until deadline, $D_i - S_i$, as opposed to the laxity ratio which is computed as the relative ratio $(D_i - S_i) / D_i$.

$$r_i = 1 - \frac{S_i}{D_i} \tag{1}$$

Table 18: Relationship between absolute laxity and laxity ratio

Relationship between S_i and D_i	Sign of Laxity Value	Value of r_i	Meaning
$S_i < D_i$	+	(0, 1]	t_i has positive reserve laxity
$S_i = D_i$	0	0	t_i has no reserve laxity
$S_i > D_i$	-	$(-\infty, 0)$	t_i has negative reserve laxity

From Table 18, a positive laxity value corresponds to a positive reserve laxity ratio of r_i. Under the proposed XLAX scheduling algorithm, if $r_i < x$ (and $r_i > 0$) for a given task t_i, then t_i does not have the desired percentage of reserve laxity and could be given priority over other tasks whose reserve laxities are greater than x. Variability can be taken into account so that, for example, the more variable tasks are given priority over tasks exhibiting more regularity. In Table 18, laxity values that are zero or negative have a direct correlation to their corresponding laxity ratios. That is, zero laxity corresponds to a reserve laxity of zero, and negative laxity corresponds to a negative reserve laxity. Because a task is expected to miss its deadline when it has negative laxity, priority can be given to the task with the largest variance. The reasoning is that due to its larger variance, it is possible such a task could finish executing early and meet its deadline. A task exhibiting low variance (i.e., high regularity) is more consistent and it will likely miss its deadline regardless if it is allocated the processor or not. Depending on the system state, algorithms such as XLAX can dynamically adjust their scheduling decisions based on the variability of performance parameters, the relationship of task laxities, and

the changing system load. To further investigate the XLAX algorithm, a sensitivity analysis is conducted in order to compare different variability methods and evaluate the importance of task laxities in determining their effect on scheduling decisions.

6.4 Parameters for the XLAX Algorithm

The key concept of XLAX is that the reserve laxity of a task eventually falls below a minimum value (i.e., threshold), distinguishing it from other tasks whose reserve laxities are still above the threshold value. After dividing tasks into groups based on their reserve laxities, variability can be taken into account to further determine the optimal scheduling decision. For example, tasks with less reserve laxity could be given priority over tasks having greater reserve laxity because the deadlines of the former are likely to occur sooner. A group of tasks that all have similar amounts of reserve laxity can be further classified based on the variability of their parameters. In this way, a positive threshold value is used to define a minimum percentage of reserve laxity that should be maintained for a given task. Note that a single threshold value is currently used for all task streams, but each stream could just as easily use a different threshold value if desired. At any instant, the reserve laxity of a task can be classified as either negative (N), below (B) the threshold, or above (A) the threshold. Therefore, this method provides a means of assigning a task to one of three possible categories: *N*, *B*, or *A*. Figure 70 illustrates this classification that uses the reserve laxity value of a task along with a threshold, *t*. For simplicity, threshold values are referred to as a percentage of the reserve laxity value.

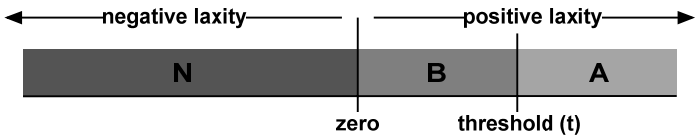

Figure 70: Categories of reserve laxity for a task

When the laxity of a task is negative, it is placed into the negative (N) group. Tasks in group N are expected to miss their deadline because they are estimated to require more service time than the amount of time remaining before their deadline. A task with positive reserve laxity is placed into either group A or B, depending on if its reserve laxity is above or below the given threshold value. A task with reserve laxity above the given threshold is considered safer in that it still has enough laxity in reserve to meet its deadline. Therefore, tasks with reserve laxities above the threshold are placed into group A. If the laxity of a task falls below the threshold value, but is still positive, the task is expected to barely meet its deadline and is placed into group B. Tasks in group B are presumed to be less safe than tasks in the group A because their deadlines are more quickly approaching.

Using this classification technique, a target group of tasks can be identified from which a single task is selected to execute. A number of techniques can be used to select a particular task from a given target group, but this dissertation focuses on three different methods. The first (LV) method involves selecting the task with the least variability in its arrival behavior. The second (HV) method involves selecting the task with the highest variability in its arrival behavior. Note that methods involving the variability of the service or deadline behavior of tasks can also be used. In our experiments, such methods yield similar results to those presented in this chapter and therefore, they are not discussed individually. The last (AL) method involves selecting the task with the least

absolute (reserve) laxity. For tasks in groups A or B, the AL method is equivalent to using the LLF algorithm within the group of tasks. If LLF is used within the N group, priority is given to the task with the *most* negative laxity because the least laxity corresponds to the largest negative laxity value. For tasks in the N group, the AL method instead selects the task having the least negative reserve laxity (i.e., the task with negative laxity closest to zero). When negative laxity values are involved, LLF selects the task with the largest negative laxity value to execute, even though this task is more likely to miss its deadline when compared to other tasks that also have negative laxities. The reasoning is that by selecting the task with the *least* negative laxity, the AL method increases the likelihood of selecting a task from the N group that meets its deadline.

6.5 Performance of Traditional Algorithms

For each of the system workloads, the performance of the traditional scheduling algorithms RM, EDF, and LLF is evaluated by using MOSS to conduct several groups of simulations. Within each group, independent simulations produce estimates of desired performance metrics (e.g., percentage of met deadlines). These metrics are then averaged across all simulations within a group in order to obtain approximate performance values of the RM, EDF, and LLF algorithms under different system loads. Table 19 provides a summary of the performance of the traditional scheduling algorithms. Under light load (i.e., IF=0.5), all three traditional algorithms perform similarly and only a small percentage of deadlines is missed. Under medium load (i.e., IF=1.0), both EDF and LLF outperform RM, and EDF provides about a 4 percent improvement over LLF. Under heavy load (i.e., IF=1.5), the performance difference is more significant and EDF (with 46.88%) outperforms the next best (i.e., RM with 38.20%) by more than 20 percent. The

performance of these traditional scheduling algorithms will be used to evaluate the relative performance of the XLAX algorithm. In particular, the performance of the EDF algorithm will serve as the primary comparison metric for evaluating the relative performance of the XLAX algorithm.

Table 19: Performance summary for traditional algorithms

		% Met Deadlines		
System Load	Intensity Factor	RM	EDF	LLF
Light	0.5	95.46	96.46	96.36
Medium	1.0	70.98	78.49	75.30
Heavy	1.5	38.20	46.88	36.21

6.6 Experimental Setup for Testing XLAX

To evaluate the XLAX algorithm, four threshold values are used in the experiments—25%, 50%, 75% and 90%. Using a 50% threshold means that any task (with positive reserve laxity) having at least 50% laxity in reserve is placed into the A group, whereas a task with less than 50% reserve laxity is placed into the B group. For each of these threshold values, the three XLAX variability methods (LV, HV, and AL) are tested to compare their relative performance under different system loads. The experiments consider light, medium, and heavy load conditions. Consider a lightly loaded system (i.e., IF=0.5) where tasks are expected to meet most of their deadlines. At any instant, the system contains tasks from a combination of each of the four task streams described in the workload (see Table 16). Because the deadline of a task coincides with its next arrival, a given task stream always has at most one task present in the system. In addition, due to task completions before their next arrival, tasks from some streams are not always present. Given that at least one task is available for scheduling, Table 20 lists

the possible situations (i.e., cases) that can arise based on the task groups described previously.

Table 20: Possible combinations of task groups

Case	Group N	Group B	Group A
0			✓
1		✓	
2	✓		
3		✓	✓
4	✓		✓
5	✓	✓	
6	✓	✓	✓

The period of time after a task meets or misses its deadline, but before the next arrival of its associated task stream occurs, corresponds to an unschedulable time period for that stream. Therefore, a task stream has a schedulable task present only if a task of that type is present (i.e., has arrived) but has not yet met or missed its deadline. Referring to Table 20, Case 0 corresponds to the situation where all schedulable tasks have reserve laxities greater than the current threshold value and are therefore, placed into group A. Case 1 is similar and corresponds to a situation where all schedulable tasks have positive reserve laxities that are below the threshold value. Case 2 represents the situation where all schedulable tasks have negative reserve laxities. Beginning with case 3, there are schedulable tasks present in more than one task group. For example, in case 3 all schedulable tasks are in either group B or group A. Cases 4 and 5 are similar and correspond to situations where tasks are classified into either groups N and A, or groups N and B, respectively. Finally, case 6 corresponds to the situation where at least one task is present from each of the three groups.

To systematically test the performance of the XLAX algorithm, simulations are first run assuming light load. Initially, scheduling decisions are varied only when the system state corresponds to case 0. For all other cases, a default scheduling algorithm of Processor Sharing (PS) is used. After the results from applying the various techniques in case 0 have been tested, the best variability method can be identified for each threshold value. These preferred methods are then fixed for case 0 and the testing progresses to case 1. When testing the performance of techniques in case 1, scheduling decisions are varied only in case 1. When a situation arises that corresponds to case 0, the best technique discovered in the previous step is used. When a situation arises that corresponds to any case 2 through 6, the default PS algorithm is used. Once the performance results are obtained for a given case, the best strategy for each threshold value is identified and fixed so that all subsequent decisions for that case are the best discovered thus far. This step-wise process continues until optimal scheduling methods have been identified for all six cases. As the study progresses through each of the cases, more information is discovered and incorporated into the overall scheduling algorithm. For example, when case 6 is tested, the best techniques for cases 0 through 5 have already been determined and fixed. This series of simulations is referred to as the first iteration. A second iteration of simulations is later conducted and will be discussed separately.

For cases 0 through 2, there is only one task group present and therefore, the choice of the task group is clear. However, in cases 3 through 6, multiple groups of tasks are concurrently present and a decision must be made to determine which one of the task groups to select a task from for scheduling. In the experiments, each group is tested individually to better evaluate the overall performance of the XLAX algorithm. In case

3, for example, task groups A and B are always present for any given threshold value. Therefore, before one of the three variability methods (i.e., LV, HV, or AL) can be applied, one of the groups must be designated to select tasks from for scheduling. In this case, a series of simulations is run in which tasks are always selected from group A. Then another series of simulations is run in which tasks are always selected from group B instead. In this way, one group is essentially given priority over another group throughout the duration of the series of simulations.

To obtain an estimate for a given performance metric, a set of simulations is run and the metric values from the set are averaged. Each individual simulation is independent, and any given group of simulations is independent of other groups. For a given workload type (i.e., light, medium, or heavy), some 15,000 simulations are run to obtain the comparison data for the first iteration of case testing. Within a given iteration, the testing begins with case 0 and proceeds through case 6.

6.7 Sensitivity Analysis Results from the First Iteration

In this section, the results from the first iteration of sensitivity analysis experiments are discussed. Results for each of the system workloads are discussed separately, followed by an overall summary for the entire iteration. An experimental version of MOSS is used to conduct these experiments.

6.7.1 Results for the System under Light Load

Recall from the previous discussion that under light load, the EDF algorithm performs best among the traditional algorithms, meeting approximately 96.46% of overall deadlines. Under light load, the system utilization is approximately 52% and the percentage of met deadlines is expected to be high. Table 21 provides a summary of the

performance of XLAX for the system under light load. As mentioned previously, priority is given to each task group separately in cases 3 through 6. The performance of each of the three variability methods is given and the best option for each threshold, task group, and case is provided in Table 21.

Table 21: First iteration results for light load

LIGHT LOAD IF=0.5 (~52% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	94.63	95.94	96.41	AL	96.41	✓
	50	A	94.52	95.71	96.28	AL	96.28	
	75	A	94.56	95.69	96.10	AL	96.10	
	90	A	95.30	95.36	95.28	HV	95.36	
1	25	B	96.42	96.20	96.40	LV	96.42	✓
	50	B	96.28	96.30	96.28	HV	96.30	
	75	B	95.88	96.15	95.92	HV	96.15	
	90	B	95.08	95.64	95.69	AL	95.69	
2	25	N	96.37	96.43	96.17	HV	96.43	
	50	N	96.45	96.30	96.24	LV	96.45	✓
	75	N	95.96	96.13	96.06	HV	96.13	
	90	N	95.64	95.58	95.63	LV	95.64	
3	25	B	96.48	96.36	96.20	LV	96.48	
		A	96.20	96.54	96.41	HV	96.54	✓
	50	B	96.22	96.37	96.24	HV	96.37	
		A	96.34	96.20	96.26	LV	96.34	
	75	B	96.23	96.38	96.10	HV	96.38	
		A	95.75	95.85	95.74	HV	95.85	
	90	B	95.76	95.95	96.06	AL	96.06	
		A	95.50	95.53	95.73	AL	95.73	
4	25	N	96.26	96.37	96.32	HV	96.37	
		A	96.40	96.32	96.32	LV	96.40	
	50	N	96.48	96.38	96.45	LV	96.48	✓
		A	96.31	96.30	96.33	AL	96.33	
	75	N	96.27	96.35	96.42	AL	96.42	
		A	96.16	96.37	96.40	AL	96.40	
	90	N	96.03	95.70	96.19	AL	96.19	
		A	95.97	96.00	96.00	AL	96.00	
5	25	N	96.35	96.37	96.47	AL	96.47	✓
		B	96.40	96.35	96.34	LV	96.40	
	50	N	96.40	96.30	96.45	AL	96.45	
		B	96.33	96.22	96.34	AL	96.34	
	75	N	96.32	96.28	96.33	AL	96.33	
		B	96.31	96.27	96.33	AL	96.33	
	90	N	96.12	96.01	95.87	LV	96.12	
		B	96.01	96.06	95.90	HV	96.06	
6	25	N	96.20	96.40	96.36	HV	96.40	
		B	96.40	96.51	96.48	HV	96.51	✓
		A	96.25	96.19	96.39	AL	96.39	
	50	N	96.35	96.35	96.10	HV	96.35	
		B	96.35	96.30	96.32	LV	96.35	
		A	96.20	96.26	96.38	AL	96.38	
	75	N	96.28	96.25	96.21	LV	96.28	
		B	96.32	96.37	96.36	HV	96.37	
		A	96.12	96.45	96.06	HV	96.45	
	90	N	96.09	95.83	96.24	AL	96.24	
		B	95.99	95.78	95.94	LV	95.99	
		A	95.97	96.12	96.01	HV	96.12	

From the data in Table 21, it is apparent that the performance remains comparable, regardless of the threshold value, variability method, or which group is given priority. That is, the percentage of met deadlines is approximately the same (i.e., around 96%) each time. For a given threshold value, there is no clear advantage to using any particular one of the three variability methods (i.e., LV, HV, and AL) over another. For all cases other than 2 and 4, a threshold value of 25% results in the best performance but further classification of the best approach is inconsistent from one case to another. In general, there is no obvious performance improvement as the optimal decisions from earlier cases is incorporated. Overall, it can be seen that under light load the performance of XLAX is perhaps slightly better, but at least comparable, to that of the traditional algorithms. Note that under such light load, there is typically only one task to schedule, making the scheduling decision trivial. Therefore, under light load, all scheduling algorithms are expected to be comparable.

6.7.2 Results for the System under Medium Load

Under medium load, the EDF algorithm again performs best among the traditional algorithms, meeting approximately 78.49% of overall deadlines. Under medium load, the system utilization is approximately 88% and the percentage of met deadlines is expected to be less than that under light load. Table 22 provides a summary of the performance of XLAX for the system under medium load. In the early cases (i.e., cases 0 through 3), the performance of XLAX is comparable to that of EDF but marginally lower by about 2%. The reason for this is that the default algorithm, PS, is sub-optimal and it takes cases 0 through 2 to eliminate PS from the analysis. However, after case 6 is tested, XLAX meets 79.03% of overall deadlines, which is slightly better than EDF.

Table 22: First iteration results for medium load

MEDIUM LOAD IF=1.0 (~88% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	74.94	72.34	76.74	AL	76.74	✓
	50	A	74.84	71.31	76.33	AL	76.33	
	75	A	73.39	72.79	73.59	AL	73.59	
	90	A	73.14	73.15	73.04	HV	73.15	
1	25	B	76.91	76.74	76.61	LV	76.91	✓
	50	B	76.42	76.80	76.24	HV	76.80	
	75	B	74.98	73.27	74.06	LV	74.98	
	90	B	76.06	72.51	75.14	LV	76.06	
2	25	N	76.70	76.92	76.63	HV	76.92	✓
	50	N	76.55	75.80	76.28	LV	76.55	
	75	N	74.98	75.37	75.16	HV	75.37	
	90	N	76.58	76.07	76.30	LV	76.58	
3	25	B	74.84	75.14	75.03	HV	75.14	
		A	77.70	78.00	78.44	AL	78.44	✓
	50	B	74.70	74.53	75.13	AL	75.13	
		A	76.83	77.02	77.89	AL	77.89	
	75	B	70.54	70.20	69.81	LV	70.54	
		A	77.83	77.66	77.98	AL	77.98	
	90	B	75.71	72.21	74.36	LV	75.71	
		A	76.86	76.87	77.13	AL	77.13	
4	25	N	78.41	78.49	78.74	AL	78.74	
		A	78.69	78.65	78.75	AL	78.75	✓
	50	N	76.86	76.82	76.77	LV	76.86	
		A	78.21	78.22	78.18	HV	78.22	
	75	N	77.70	77.84	77.85	AL	77.85	
		A	78.17	78.09	78.07	LV	78.17	
	90	N	76.72	77.01	76.97	HV	77.01	
		A	76.97	77.08	76.96	HV	77.08	
5	25	N	78.45	78.94	78.99	AL	78.99	✓
		B	78.91	78.70	78.91	LV	78.91	
	50	N	78.03	78.35	78.06	HV	78.35	
		B	78.14	78.18	77.95	HV	78.18	
	75	N	77.98	78.07	77.80	HV	78.07	
		B	77.97	78.14	78.25	AL	78.25	
	90	N	77.22	77.04	76.76	LV	77.22	
		B	76.88	77.07	77.28	AL	77.28	
6	25	N	78.83	78.37	78.97	AL	78.97	
		B	78.70	79.03	78.83	HV	79.03	✓
		A	78.35	78.51	78.69	AL	78.69	
	50	N	78.43	77.87	77.96	LV	78.43	
		B	77.86	78.62	78.35	HV	78.62	
		A	78.05	78.52	78.30	HV	78.52	
	75	N	78.01	77.59	77.97	LV	78.01	
		B	78.20	77.88	78.18	LV	78.20	
		A	78.28	78.33	78.12	HV	78.33	
	90	N	77.15	76.80	76.95	LV	77.15	
		B	77.01	76.83	76.96	LV	77.01	
		A	76.94	76.58	77.09	AL	77.09	

Under medium load, a 25% threshold appears to work best and except for case 6, the AL method appears to be preferable. However, the particular task group to give priority to varies from one case to another and it is difficult to make any further claims for medium load. Overall, the performance of XLAX is still comparable to that of EDF under medium load.

6.7.3 Results for the System under Heavy Load

Under heavy load, the EDF algorithm again performs best among the traditional algorithms, meeting approximately 46.88% of overall deadlines. Under heavy load, the system utilization is near 100% and the percentage of met deadlines is expected to be significantly less than that under medium load. Table 23 provides a summary of the performance of XLAX for the system under heavy load. Here, even in the early cases, XLAX already outperforms EDF by about 4%. After the maximum percentage of met deadlines is determined in case 6, XLAX is found to meet approximately 59% of overall deadlines, significantly outperforming EDF by about 26%.

Under heavy load, a 25% threshold works best for cases 0, 1, 2, and 4. For these cases, the LV method is the best choice, except for case 2, where the AL method results in better performance. For the cases 3, 5, and 6, a 50% threshold works best, but the choice of the best variability method is mixed because the AL, LV, and HV methods appear to be preferred for cases 3, 5, and 6, respectively. For cases 3 through 6, the particular task group to give priority to A in all cases, except for case 5, where priority should instead be given to group B. Thus, the results for heavy load are mixed but they provide good suggestions and motivation for the next iteration of testing.

Table 23: First iteration results for heavy load

HEAVY LOAD IF=1.5 (~99% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	48.55	45.12	45.08	LV	48.55	✓
	50	A	44.89	44.78	45.35	AL	45.35	
	75	A	43.64	43.16	43.59	LV	43.64	
	90	A	32.25	32.80	32.85	AL	32.85	
1	25	B	48.43	48.36	48.36	LV	48.43	✓
	50	B	45.31	45.67	45.47	HV	45.67	
	75	B	44.74	43.22	44.20	LV	44.74	
	90	B	40.44	33.45	36.23	LV	40.44	
2	25	N	48.28	48.40	48.54	AL	48.54	✓
	50	N	45.81	45.65	45.42	LV	45.81	
	75	N	44.37	44.51	44.59	AL	44.59	
	90	N	40.06	41.12	40.58	HV	41.12	
3	25	B	48.15	48.06	47.86	LV	48.15	
		A	49.25	48.67	48.88	LV	49.25	
	50	B	36.76	39.98	38.01	HV	39.98	
		A	49.50	49.12	49.99	AL	49.99	✓
	75	B	36.63	37.00	35.87	HV	37.00	
		A	48.75	48.19	48.72	LV	48.75	
	90	B	40.19	38.36	39.67	LV	40.19	
		A	41.12	41.26	41.40	AL	41.40	
4	25	N	41.53	41.78	41.71	HV	41.78	
		A	55.98	55.47	55.88	LV	55.98	✓
	50	N	46.03	47.16	46.80	HV	47.16	
		A	52.68	52.29	52.72	AL	52.72	
	75	N	47.26	47.25	47.42	AL	47.42	
		A	49.67	49.58	49.86	AL	49.86	
	90	N	41.45	41.61	41.59	HV	41.61	
		A	41.40	41.87	41.47	HV	41.87	
5	25	N	55.99	55.89	55.92	LV	55.99	
		B	55.83	56.30	55.95	HV	56.30	
	50	N	52.86	52.72	52.95	AL	52.95	
		B	56.46	56.29	56.39	LV	56.46	✓
	75	N	49.98	49.67	49.99	AL	49.99	
		B	54.89	54.82	54.98	AL	54.98	
	90	N	41.33	41.69	41.44	HV	41.69	
		B	54.09	54.64	54.59	HV	54.64	
6	25	N	55.95	55.47	55.91	LV	55.95	
		B	56.05	55.85	55.82	LV	56.05	
		A	56.02	56.32	56.54	AL	56.54	
	50	N	51.57	51.62	51.46	HV	51.62	
		B	57.57	57.55	57.84	AL	57.84	
		A	58.82	59.05	58.66	HV	59.05	✓
	75	N	49.69	49.60	49.73	AL	49.73	
		B	56.38	56.31	56.15	LV	56.38	
		A	57.42	57.45	57.40	HV	57.45	
	90	N	53.98	54.12	53.87	HV	54.12	
		B	55.34	55.21	55.33	LV	55.34	
		A	55.21	55.41	55.53	AL	55.53	

6.7.4 Summary of Results from the First Iteration

To provide a more concise summary of the results from the first iteration, the data for individual task groups is removed. Table 24 provides a summary of all the results from the first iteration of simulations. For cases 3 through 6, the notation *_X* suffix of the variability method denotes the task group to which priority is given. Overall, it can be seen that XLAX performs comparably to EDF for light and medium system load, but for heavy system load, there is a significant advantage in using the XLAX algorithm.

Table 24: Summary of results from first iteration

LIGHT LOAD				**MEDIUM LOAD**				**HEAVY LOAD**			
IF=0.5				IF=1.0				IF=1.5			
~52% Overall Utilization				~88% Overall Utilization				~99% Overall Utilization			
Best Traditional: EDF (96.46%)				Best Traditional: EDF (78.49%)				Best Traditional: EDF (46.88%)			
Case	Threshold	Best Method	% Met	Case	Threshold	Best Method	% Met	Case	Threshold	Best Method	% Met
0	25	AL	**96.41**	0	25	AL	**76.74**	0	25	LV	**48.54**
	50	AL	96.28		50	AL	76.33		50	AL	45.35
	75	AL	96.10		75	AL	73.59		75	LV	43.64
	90	HV	95.34		90	HV	73.15		90	AL	32.85
1	25	LV	**96.42**	1	25	LV	**76.91**	1	25	LV	**48.43**
	50	HV	96.30		50	HV	76.80		50	HV	45.67
	75	HV	96.15		75	LV	74.98		75	LV	44.74
	90	AL	95.70		90	LV	76.06		90	LV	40.44
2	25	HV	96.43	2	25	HV	**76.92**	2	25	AL	**48.54**
	50	LV	**96.45**		50	LV	76.55		50	LV	45.81
	75	HV	96.13		75	HV	75.37		75	AL	44.59
	90	LV	95.64		90	LV	76.58		90	HV	41.12
3	25	HV_A	**96.54**	3	25	AL_A	**78.44**	3	25	LV_A	49.25
	50	HV_B	96.37		50	AL_A	77.89		50	AL_A	**49.99**
	75	HV_B	96.40		75	AL_A	77.98		75	LV_A	48.75
	90	AL_B	96.06		90	AL_A	77.13		90	AL_A	41.40
4	25	LV_A	96.40	4	25	AL_A	**78.75**	4	25	LV_A	**55.98**
	50	LV_N	**96.48**		50	HV_A	78.22		50	AL_A	52.72
	75	AL_N	96.42		75	LV_A	78.17		75	AL_A	49.86
	90	AL_N	96.19		90	HV_A	77.08		90	HV_A	41.87
5	25	AL_N	**96.47**	5	25	AL_N	**78.99**	5	25	HV_B	56.30
	50	AL_N	96.45		50	HV_N	78.35		50	LV_B	**56.46**
	75	AL_N	96.33		75	AL_B	78.25		75	AL_B	54.98
	90	LV_N	96.12		90	AL_B	77.28		90	HV_B	54.64
6	25	HV_B	**96.51**	6	25	HV_B	**79.03**	6	25	AL_A	56.54
	50	AL_A	96.38		50	HV_B	78.62		50	HV_A	**59.04**
	75	HV_A	96.45		75	HV_A	78.33		75	HV_A	57.45
	90	AL_N	96.24		90	LV_N	77.15		90	AL_A	55.53

When the first iteration of testing begins, in all cases except 0, the default PS algorithm is used to make scheduling decisions. Therefore, the PS algorithm is not completely eliminated from the decision-making process until after the testing in case 6 has been completed. In some sense, only the metrics listed for case 6 provide a truly

accurate representation of the performance of XLAX. Therefore, a second iteration of testing is done in which the default PS algorithm is never used.

6.8 Sensitivity Analysis Results from the Second Iteration

During the first iteration, the testing began with case 0 and in all other cases, the default PS algorithm is used. In the second iteration, testing begins with case 0, just as with the first iteration. However, none of the remaining cases uses the PS algorithm this time, but instead use the best decisions discovered during the first iteration. The overall performance of XLAX is expected to continually improve as better information is discovered and incorporated into each case. In particular, the results are expected to stabilize and provide a consistent summary for the best choices to make when using the XLAX algorithm.

6.8.1 Results for the System under Light Load

Under light load, the results from the second iteration are comparable to those from the first iteration. Table 25 provides a summary of the performance results from the second iteration for the system under light load. As with the first iteration, there is no clear decision as to the optimal variability method or to which task group priority should be given. A threshold value of 25% appears to be the preferred choice overall, but cases 0, 1, and 4 suggest that a larger threshold (e.g., 50% – 75%) should be used.

Table 25: Second iteration results for light load

LIGHT LOAD IF=0.5 (~52% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	94.81	95.81	96.13	AL	96.13	
	50	A	94.80	95.78	96.33	AL	96.33	
	75	A	94.69	95.59	96.35	AL	96.35	✓
	90	A	95.83	96.05	95.97	HV	96.05	
1	25	B	96.40	96.33	96.18	LV	96.40	
	50	B	96.51	96.33	96.26	LV	96.51	✓
	75	B	96.26	96.42	96.24	HV	96.42	
	90	B	95.94	96.16	96.21	AL	96.21	
2	25	N	96.38	96.47	96.31	HV	96.47	✓
	50	N	96.40	96.35	96.29	LV	96.40	
	75	N	96.34	96.24	96.38	AL	96.38	
	90	N	96.12	96.04	96.05	LV	96.12	
3	25	B	96.33	96.47	96.30	HV	96.47	
		A	96.41	96.37	96.48	AL	96.48	✓
	50	B	96.26	96.34	96.19	HV	96.34	
		A	96.28	96.23	96.12	LV	96.28	
	75	B	96.20	96.27	96.28	AL	96.28	
		A	95.91	95.72	95.88	LV	95.91	
	90	B	95.73	96.10	96.00	HV	96.10	
		A	95.68	95.68	95.70	AL	95.70	
4	25	N	96.36	96.13	96.17	LV	96.36	
		A	96.33	96.28	96.32	LV	96.33	
	50	N	96.22	96.44	96.15	HV	96.44	✓
		A	96.21	96.35	96.31	HV	96.35	
	75	N	96.29	96.22	96.10	LV	96.29	
		A	96.27	96.28	96.14	HV	96.28	
	90	N	95.98	95.74	96.01	AL	96.01	
		A	95.91	95.87	95.56	LV	95.91	
5	25	N	96.35	96.38	96.35	HV	96.38	
		B	96.20	96.24	96.44	AL	96.44	✓
	50	N	96.33	96.30	96.06	LV	96.33	
		B	96.30	96.27	96.20	LV	96.30	
	75	N	96.34	96.27	96.33	LV	96.34	
		B	96.39	96.08	96.20	LV	96.39	
	90	N	95.80	95.83	95.72	HV	95.83	
		B	95.74	95.83	95.65	HV	95.83	
6	25	N	96.11	96.42	96.63	AL	96.63	✓
		B	96.08	96.50	96.28	HV	96.50	
		A	96.28	96.19	96.33	AL	96.33	
	50	N	96.35	96.24	96.46	AL	96.46	
		B	96.35	96.22	96.42	AL	96.42	
		A	96.35	96.46	96.24	HV	96.46	
	75	N	96.41	96.26	96.53	AL	96.53	
		B	96.21	96.41	96.18	HV	96.41	
		A	96.33	96.19	96.40	AL	96.40	
	90	N	95.85	95.93	95.73	HV	95.93	
		B	96.04	95.82	95.76	LV	96.04	
		A	95.75	95.81	95.77	HV	95.81	

6.8.2 Results for the System under Medium Load

Under medium load, the results from the second iteration are again comparable to those from the first iteration, in terms of overall performance. Table 26 provides a summary of the performance results from the second iteration for the system under medium load. Contrary to the first iteration results, in this second iteration, the unanimous decision for the best threshold value appears to be 25%. However, the choice of which variability method to use (i.e., LV, HV, or AL) is still not clear, as the preferred choices for each case vary. It is also still difficult to make a uniform decision regarding which task group (A, B, or N) should receive priority. In terms of overall performance, the percentage of met deadlines is comparable to the values found during the first iteration of testing.

Table 26: Second iteration results for medium load

MEDIUM LOAD IF=1.0 (~88% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	77.36	75.95	78.31	AL	78.31	✓
	50	A	76.90	75.77	78.21	AL	78.21	
	75	A	78.22	78.23	78.22	HV	78.23	
	90	A	76.79	76.94	76.84	HV	76.94	
1	25	B	78.82	78.61	78.58	LV	78.82	✓
	50	B	78.30	78.21	78.43	AL	78.43	
	75	B	78.06	76.53	77.08	LV	78.06	
	90	B	76.77	74.48	75.47	LV	76.77	
2	25	N	78.97	78.94	78.93	LV	78.97	✓
	50	N	78.11	78.68	78.18	HV	78.68	
	75	N	78.23	78.13	77.98	LV	78.23	
	90	N	76.94	76.95	76.92	HV	76.95	
3	25	B	75.30	75.35	75.25	HV	75.35	
		A	78.65	78.40	79.04	AL	79.04	✓
	50	B	74.56	74.39	74.92	AL	74.92	
		A	77.82	77.70	78.16	AL	78.16	
	75	B	66.65	68.27	67.19	HV	68.27	
		A	77.78	78.27	78.17	HV	78.27	
	90	B	75.72	71.11	73.87	LV	75.72	
		A	76.69	76.54	76.95	AL	76.95	
4	25	N	78.59	78.26	78.53	LV	78.59	
		A	78.65	78.86	78.76	HV	78.86	✓
	50	N	76.85	77.06	76.82	HV	77.06	
		A	78.04	78.38	78.26	HV	78.38	
	75	N	77.78	78.05	77.73	HV	78.05	
		A	78.24	77.88	77.71	LV	78.24	
	90	N	76.83	77.02	77.27	AL	77.27	
		A	76.76	77.03	77.22	AL	77.22	
5	25	N	78.65	78.28	78.75	AL	78.75	
		B	78.77	78.59	78.62	LV	78.77	✓
	50	N	78.32	78.36	77.96	HV	78.36	
		B	78.26	78.21	78.62	AL	78.62	
	75	N	78.01	77.59	77.99	LV	78.01	
		B	78.09	78.09	78.08	HV	78.09	
	90	N	76.82	77.48	76.93	HV	77.48	
		B	77.41	76.67	76.79	LV	77.41	
6	25	N	78.53	78.63	78.72	AL	78.72	
		B	78.62	78.31	78.81	AL	78.81	✓
		A	78.61	78.57	78.70	AL	78.70	
	50	N	77.94	77.93	78.21	AL	78.21	
		B	78.21	78.33	78.29	HV	78.33	
		A	78.46	78.10	78.12	LV	78.46	
	75	N	77.96	77.43	77.65	LV	77.96	
		B	77.68	78.18	78.04	HV	78.18	
		A	78.02	78.02	78.16	AL	78.16	
	90	N	77.06	77.27	76.99	HV	77.27	
		B	77.21	77.12	77.01	LV	77.21	
		A	76.86	76.69	76.96	AL	76.96	

6.8.3 Results for the System under Heavy Load

Under heavy load, the results from the second iteration show that the performance of XLAX is more stable throughout the case testing, which is to be expected. Table 27 provides a summary of the performance results from the second iteration for the system under heavy load. Contrary to the first iteration results, in this second iteration, the unanimous decision for the best threshold value is 50%. Looking at the second iteration results for light, medium, and heavy loads, it appears that as the system utilization increases, so too does the optimal threshold value. That is, as the system load increases, the amount of laxity held in reserve for each task should also be increased.

From Table 27, it is apparent that priority should always be given to the "rightmost" task group in terms of laxity. That is, task group A should always be given priority over task group B, whenever both groups are present. Similarly, task group B should always be given priority over task group N, whenever both groups are present. However, the choice of which variability method (i.e., LV, HV, or AL) to use overall is still unclear. Whenever group A is present, the best variability method to use is AL, except for case 0, where LV is the preferred method. There are only two cases (i.e., cases 1 and 5) where group B is present and group A is not. However, these two cases provide contradictory methods, with case 1 suggesting LV and case 5 suggesting HV. The choice of variability methods will be addressed in the next section.

Table 27: Second iteration results for heavy load

HEAVY LOAD IF=1.5 (~99% Utilization)								
Case	Threshold	Priority to Group	XLAX Method			Best Performance for Threshold		Best Performance for Case
			LV	HV	AL	Method	% Met	
0	25	A	56.36	56.32	54.80	LV	56.36	
	50	A	58.72	58.57	58.68	LV	58.72	✓
	75	A	57.58	57.55	57.65	AL	57.65	
	90	A	55.08	55.36	55.30	HV	55.36	
1	25	B	56.28	56.34	56.41	AL	56.41	
	50	B	59.34	58.87	59.32	LV	59.34	✓
	75	B	57.45	57.56	56.99	HV	57.56	
	90	B	55.26	54.61	51.50	LV	55.26	
2	25	N	56.10	56.16	56.52	AL	56.52	
	50	N	59.30	59.55	59.79	AL	59.79	✓
	75	N	56.95	57.23	57.39	AL	57.39	
	90	N	55.47	55.16	55.22	LV	55.47	
3	25	B	55.56	55.68	55.39	HV	55.68	
		A	56.51	56.25	56.19	LV	56.51	
	50	B	48.83	51.35	49.97	HV	51.35	
		A	59.34	59.24	59.60	AL	59.60	✓
	75	B	50.25	52.88	49.75	HV	52.88	
		A	57.50	57.51	57.27	HV	57.51	
	90	B	54.90	54.21	53.94	LV	54.90	
		A	55.47	55.21	55.41	LV	55.47	
4	25	N	43.44	42.51	43.21	LV	43.44	
		A	56.64	56.66	56.22	HV	56.66	
	50	N	53.52	53.82	54.00	AL	54.00	
		A	59.38	59.32	59.51	AL	59.51	✓
	75	N	53.94	54.77	54.60	HV	54.77	
		A	57.25	57.21	57.43	AL	57.43	
	90	N	55.27	54.82	55.04	LV	55.27	
		A	55.42	55.22	55.12	LV	55.42	
5	25	N	56.22	55.99	55.98	LV	56.22	
		B	56.21	55.97	56.48	AL	56.48	
	50	N	54.80	55.01	54.98	HV	55.01	
		B	59.45	59.72	59.65	HV	59.72	✓
	75	N	50.10	50.05	50.17	AL	50.17	
		B	57.76	57.34	57.46	LV	57.76	
	90	N	41.12	42.00	42.04	AL	42.04	
		B	55.34	55.46	55.12	HV	55.46	
6	25	N	55.45	55.49	55.52	AL	55.52	
		B	55.48	55.72	55.39	HV	55.72	
		A	56.24	56.51	56.37	HV	56.51	
	50	N	52.09	52.04	52.12	AL	52.12	
		B	58.16	58.16	58.26	AL	58.26	
		A	59.35	59.60	59.77	AL	59.77	✓
	75	N	48.19	48.13	47.89	LV	48.19	
		B	55.40	55.66	55.23	HV	55.66	
		A	57.51	57.63	57.58	HV	57.63	
	90	N	53.87	53.91	53.84	HV	53.91	
		B	55.30	54.97	54.95	LV	55.30	
		A	55.27	55.38	55.45	AL	55.45	

6.8.4 Summary of Results from the Second Iteration

Table 28 provides a summary of all the results from the second iteration of testing. As before, the data for individual task groups is not shown. Examining this summary table, it is easier to identify the trends mentioned previously. Under light or medium system load, there is little performance gain when using the XLAX algorithm. However, under heavy system load, XLAX provides a significant performance improvement (i.e., about 26%) over the best-performing traditional algorithm (i.e., EDF). Using a 50% threshold and giving priority to the rightmost task group results in the overall best performance under heavy load. This demonstrates that the optimal amount of reserve laxity is not zero, as might be expected.

Table 28: Summary of results from second iteration

LIGHT LOAD				**MEDIUM LOAD**				**HEAVY LOAD**			
IF=0.5				IF=1.0				IF=1.5			
~52% Overall Utilization				~88% Overall Utilization				~99% Overall Utilization			
Best Traditional: EDF (96.46%)				Best Traditional: EDF (78.49%)				Best Traditional: EDF (46.88%)			
Case	Threshold	Best Method	% Met	Case	Threshold	Best Method	% Met	Case	Threshold	Best Method	% Met
0	25	AL	96.13	0	25	AL	**78.31**	0	25	LV	56.36
	50	AL	96.33		50	AL	78.21		50	LV	**58.72**
	75	AL	**96.35**		75	HV	78.23		75	AL	57.65
	90	HV	96.05		90	HV	76.94		90	HV	55.36
1	25	LV	96.40	1	25	LV	**78.82**	1	25	AL	56.41
	50	LV	**96.51**		50	AL	78.43		50	LV	**59.34**
	75	HV	96.42		75	LV	78.06		75	HV	57.56
	90	AL	96.21		90	LV	76.77		90	LV	55.26
2	25	HV	**96.47**	2	25	LV	**78.97**	2	25	AL	56.52
	50	LV	96.40		50	HV	78.68		50	AL	**59.79**
	75	AL	96.38		75	LV	78.23		75	AL	57.39
	90	LV	96.12		90	HV	76.95		90	LV	55.47
3	25	AL_A	**96.48**	3	25	AL_A	**79.04**	3	25	LV_A	56.51
	50	HV_B	96.34		50	AL_A	78.16		50	AL_A	**59.60**
	75	AL_B	96.28		75	HV_A	78.27		75	HV_A	57.51
	90	HV_B	96.10		90	AL_A	76.95		90	LV_A	55.47
4	25	LV_N	96.36	4	25	HV_A	**78.86**	4	25	HV_A	56.64
	50	HV_N	**96.44**		50	HV_A	78.38		50	AL_A	**59.51**
	75	LV_N	96.29		75	LV_A	78.24		75	AL_A	57.43
	90	AL_N	96.01		90	AL_N	77.27		90	LV_A	55.42
5	25	AL_B	**96.44**	5	25	LV_B	**78.77**	5	25	AL_B	56.48
	50	LV_N	96.33		50	AL_B	78.62		50	HV_B	**59.72**
	75	LV_B	96.39		75	HV_B	78.09		75	LV_B	57.76
	90	HV_N	95.83		90	HV_N	77.41		90	HV_B	55.46
6	25	AL_N	**96.63**	6	25	AL_B	**78.81**	6	25	HV_B	55.72
	50	HV_A	96.46		50	LV_A	78.46		50	AL_A	**59.77**
	75	AL_N	96.53		75	HV_B	78.18		75	HV_A	57.63
	90	LV_B	96.04		90	HV_N	77.27		90	AL_A	55.45

Further inspection of the heavy-load results shows that priority should be given to tasks in group A, and that the preferred variability method is AL in this case. This means that the task with the least laxity in group A is selected to execute, which is also by definition, the task with reserve laxity closest to the threshold value. When no tasks in

group A are present, priority should be given to tasks in group B, and the best selection method is LV in case 1 and HV in case 5. Although the performance of any of the three variability methods is similar, it can be hypothesized that either LV or HV (but *not* AL), is the best choice. Under this hypothesis, it is best to avoid the AL method and, therefore, an easy method of doing this is by selecting the task with the *most* laxity. A similar argument can be given for the situation where priority is given to tasks in group N (i.e., whenever no tasks in groups A or B are present). This hypothesis is tested and further examined in the next section.

All tasks in group A have excess positive laxity and, therefore, selecting the task with the least amount in excess is logical because this task is likely to be the next one to move from group A to group B. Similarly, all tasks in group B are "critical" because they are running out of laxity. When one of these tasks is selected to execute, it is again logical to select the task that has the best chance of meeting its deadline. This corresponds to the task that has the most laxity to spare, which is the task with reserve laxity closest to the threshold value. All tasks in group N have run out of laxity and will likely not meet their deadlines. Of all the tasks in group N, the one most likely to meet its deadline is the task with the least amount of negative laxity. Therefore, in all cases it seems the best overall strategy is to select the task with reserve laxity closest to the threshold value. This provides the motivation and insight into the development of a new scheduling algorithm, TLAX, which is the subject of the next chapter.

6.8.5 Summary of Simulation Results

It has been demonstrated that system performance is improved by using the XLAX scheduling algorithm, particularly under heavy system load. Table 29 provides a

summary of the performance of the traditional algorithms, as well as XLAX, under the different system loads. Under light load, the performance of all four algorithms is approximately the same and there are few missed deadlines. Under medium load, the performance of XLAX is comparable to that of EDF and both of these algorithms provide a performance boost over RM and LLF. Under heavy load, XLAX significantly outperforms all of the traditional algorithms. Figure 71 illustrates this performance boost graphically.

Table 29: Performance summary of all algorithms

		% Met Deadlines			
System Load	Intensity Factor	RM	EDF	LLF	XLAX
Light	0.5	95.46	96.46	96.36	**96.63**
Medium	1.0	70.98	78.49	75.30	**78.81**
Heavy	1.5	38.20	46.88	36.21	**59.77**

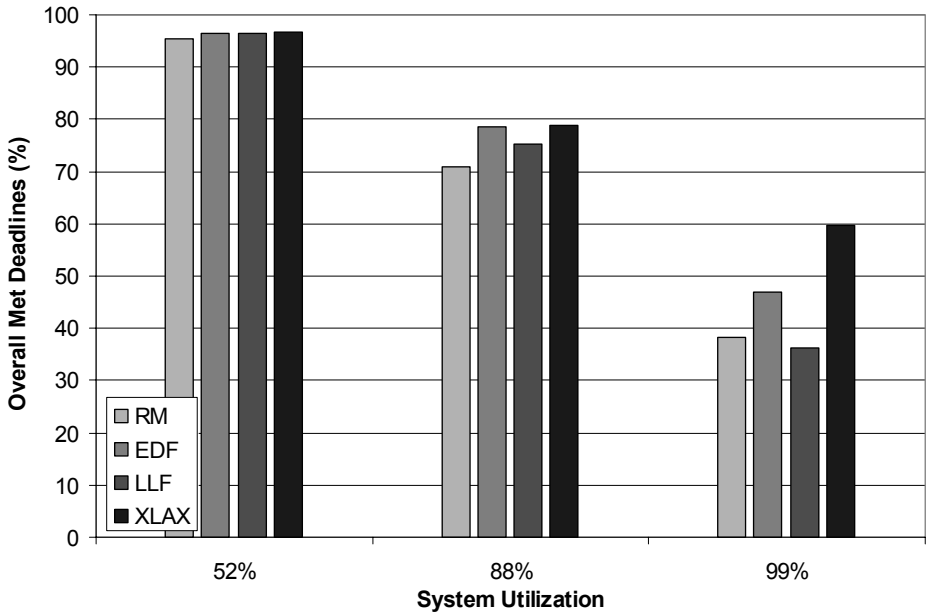

Figure 71: Graphical performance summary of all algorithms

Incorporating the information discovered in this sensitivity analysis about using XLAX under heavy load, a simple algorithm can be described that uses a threshold-based laxity. This algorithm, TLAX (Threshold LAXity), uses the same classification technique of task groups as XLAX. However, TLAX prioritizes the task groups by assigning the highest priority to tasks in group A, followed by tasks in group B. The lowest priority is assigned to tasks in group N. However, instead of using different variability-based techniques (i.e., LV, HV, or AL) to select a task from a given group, TLAX selects the task from within a target group that has the reserve laxity closest to the threshold value. To select a task from group A, TLAX identifies the task having the *least* reserve laxity. To select a task from group B, TLAX selects the task having the *most* reserve laxity because the laxity of this task is closest to the threshold value. Similarly, to select a task from group N, the task with the most laxity is again chosen. This simple strategy is easy to understand, easy to implement, and it summarizes the most important characteristics and decisions discovered from the sensitivity analysis presented in this chapter.

As with XLAX, TLAX uses a threshold value to place tasks into groups. TLAX then determines which task from the target group should be selected to execute. The XLAX analysis indicates that a 50% threshold value results in the best performance under heavy system load. To test the performance of TLAX, initial testing is done by using a 50% threshold value regardless of the system load. TLAX-50 is used to schedule tasks under the same light, medium, and heavy system loads discussed previously. The performance of TLAX-50 is found to be comparable to that of XLAX and is summarized in Table 30. Although TLAX-50 performs slightly worse than XLAX under all three system loads, it

is a generic threshold-laxity algorithm that is easily described and implemented. Under light or medium load, the performance of TLAX-50 is comparable to that of the best traditional algorithm (i.e., EDF). However, under heavy load, TLAX-50 significantly outperforms all of the traditional scheduling algorithms by about 26%. The performance of TLAX will be further investigated in the next chapter.

Table 30: Performance summary of the TLAX algorithm

		% Met Deadlines				
System Load	Intensity Factor	RM	EDF	LLF	XLAX	TLAX-50
Light	0.5	95.46	96.46	96.36	96.63	96.32
Medium	1.0	70.98	78.49	75.30	78.81	78.47
Heavy	1.5	38.20	46.88	36.21	59.77	59.37

6.9 Chapter Summary

The results from the sensitivity analysis experiments conducted using MOSS are both interesting and instrumental in the comparison of scheduling algorithm performance and the development of improved algorithms. The analysis of the XLAX algorithm provides key insights into the best variability method to use and how to select tasks from within a common group. By summarizing the key findings regarding the performance of XLAX, a new algorithm, TLAX, is developed that is easier to understand and simpler to implement. TLAX performs comparably to the traditional scheduling algorithms (i.e., RM, EDF, and LLF) under light or medium load, but provides a significant performance boost under heavy system load. The performance of TLAX using a 50% threshold value shows promise for further analysis and study of such state-based algorithms, particularly under heavy system load. The TLAX algorithm will be studied in detail in the next chapter.

6.10 Research Contributions

The contributions presented in this chapter include:

- Evaluating and comparing the performance of traditional scheduling algorithms with an emphasis on the effects of variability
- Illustrating that the effects of variability are load-dependent
- Demonstrating that as the system utilization increases, the optimal threshold value for XLAX/TLAX also increases
- Introducing and developing the novel XLAX and TLAX algorithms, which are robust and can outperform the traditional algorithms

CHAPTER VII

THE TLAX SCHEDULING ALGORITHM

7.1 The TLAX Algorithm

The TLAX (Threshold LAXity) algorithm is derived from the sensitivity analysis results of XLAX. Like XLAX, TLAX uses a threshold-based laxity value to place tasks into one of three possible groups: N, B, or A. Tasks in group A all have positive reserve laxities above the given threshold value and are assigned the highest priority. Tasks in group B all have positive reserve laxities (below the threshold value) and are assigned the next-highest priority. Finally, tasks in group N all have negative reserve laxities and are assigned the lowest priority. TLAX differs from XLAX in that TLAX selects a task from within a group by identifying the task with reserve laxity closest to the threshold value. Therefore, from group A, TLAX selects the task with the least positive reserve laxity. From group B, TLAX selects the task with the most positive reserve laxity. Finally, from group N, TLAX selects the task with the least negative reserve laxity. Although this technique is simple, it has proven to be a powerful and effective scheduling strategy, particularly under heavy system load. The concept of a threshold-based laxity technique is a novel approach to task scheduling in real-time systems. In this chapter, sensitivity analysis is conducted on the TLAX algorithm and its performance is evaluated and compared to that of the traditional scheduling algorithms (i.e., RM, EDF, and LLF). Results show that TLAX outperforms the traditional scheduling algorithms under heavy

load situations, and that it is robust in that it is insensitive to changes in workload variability.

7.2 Performance Comparison of TLAX

The workload presented earlier consisting of four task streams is again used as a basis for the sensitivity analysis. This workload is reproduced in Table 31 for convenience. As shown in the table, S0 and S1 have the same average inter-arrival and service times for each system load condition. However, stream S0 exhibits more regularity in both its arrival and service processes when compared to those of S1. Similarly, S2 and S3 share the same average arrival and service parameters, but these values differ from those of S0 and S1. That is, streams S2 and S3 impose more relative load on the system due to their higher service requirements and lower inter-arrival times, when compared to S0 and S1.

Table 31: Workload parameters for light, medium, and heavy loads

System Load	Task Stream	Inter-arrival Time (min)	Number of Arrival Stages	Service Time (min)	Number of Service Stages
Light (IF=0.5)	S0	60.0	10	4.0	10
	S1	60.0	2	4.0	2
	S2	40.0	10	8.0	10
	S3	40.0	2	8.0	2
Medium (IF=1.0)	S0	30.0	10	4.0	10
	S1	30.0	2	4.0	2
	S2	20.0	10	8.0	10
	S3	20.0	2	8.0	2
Heavy (IF=1.5)	S0	20.0	10	4.0	10
	S1	20.0	2	4.0	2
	S2	13.33	10	8.0	10
	S3	13.33	2	8.0	2

Using MOSS, the performance of the traditional algorithms RM, EDF, and LLF can easily be compared to that of TLAX (using a 50% threshold value) under each of the three system loads. A summary of the results is shown in Figure 72. Under light load, all three algorithms exhibit similar performance, but under medium load, EDF and TLAX-50 provide a modest (i.e., 1% – 10%) performance improvement over RM and LLF. A significant performance increase occurs under heavy load, where TLAX-50 outperforms EDF by approximately 26%, and outperforms LLF and RM by 54% and 63%, respectively.

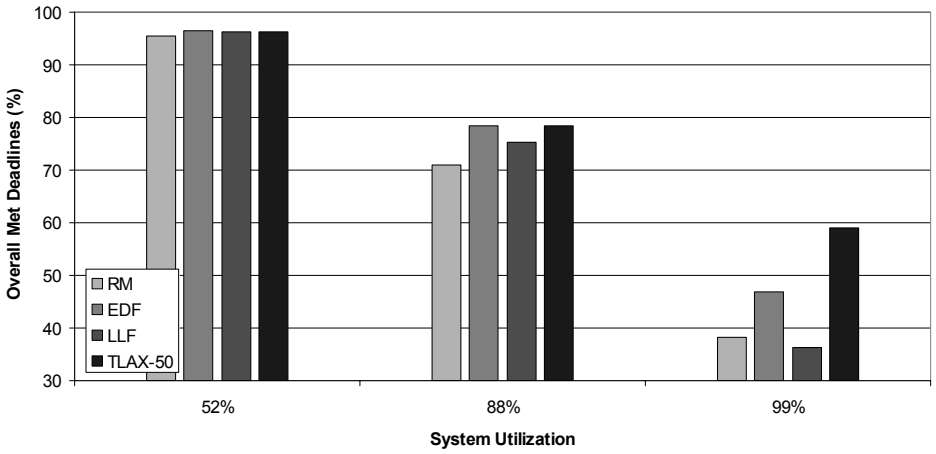

Figure 72: Performance summary of RM, EDF, LLF, and TLAX

7.3 Sensitivity Analysis of the Threshold Value

The XLAX analysis shows that a 50% threshold value provides significantly improved overall performance under heavy load, while maintaining performance under medium and light load that is comparable to that of the best traditional scheduling algorithm (i.e., EDF). To further investigate the threshold value, a sensitivity analysis is conducted to determine the optimal threshold value for light, medium, and heavy load conditions.

The results from the workloads operating under light, medium, and heavy system loads will be discussed first, followed by a more general discussion of the results from intermediate system loads. Note that TLAX-0 (i.e., when the threshold is zero) corresponds to Least Positive Laxity First, where the task with the least amount of positive laxity is selected is execute.

7.3.1 Analysis under Light Load

Under light load, few deadlines are missed by any of the scheduling algorithms, as shown in Figure 72. These results are expected because there is little contention for the processor under light load. Figure 73 provides a comparison of the percentage of deadlines met by TLAX for threshold values ranging from 0.0 to 0.70. As seen in the figure, the threshold value used by TLAX has little impact on the overall performance of the algorithm under light load. The most significant change occurs at a threshold value of approximately 0.62, but even this change is small. In addition, small variations in results are expected when conducting simulation experiments.

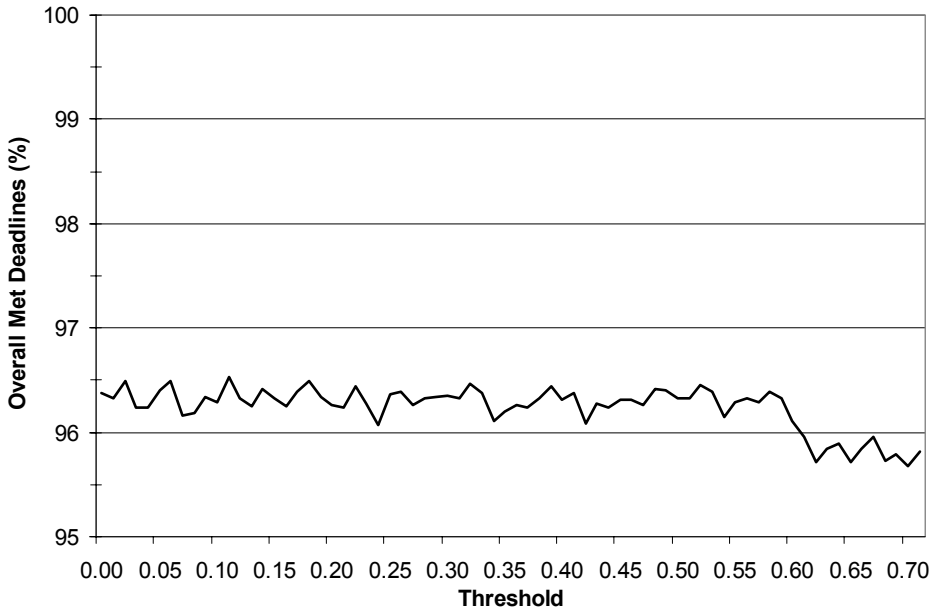

Figure 73: TLAX Threshold comparison for light load

7.3.2 Analysis under Medium Load

Under medium load, the performance of TLAX is more sensitive to changes in the threshold value when compared to that of light load. Figure 74 summarizes the performance of TLAX under medium load. From the graph, a threshold value less than about 20% results in approximately the same overall performance (i.e., about 75% met deadlines). After the boost in performance at a threshold value of about 25%, any threshold value in the range of approximately 25% – 60% results in comparable performance.

Figure 75 provides more detail by presenting the sensitivity of individual task streams within the workload, where the bold line indicates the overall system performance shown in Figure 74. Two trends are apparent in the performance curves of the individual task streams. One trend occurs at a threshold of approximately 20%, while the second occurs at a threshold of about 60%. (These trends are likely due to the discrete nature of the

number of stages and the specific task parameters.) Both of these trends coincide with the trends seen in the overall performance curve discussed previously (i.e., Figure 74). The trend that occurs at the 20% threshold shows that the performance of each task stream except S3 improves as a result of using a larger threshold value. The trend that occurs at the 60% threshold shows that the performance of streams S0 and S1 improve for larger threshold values. At the 60% threshold, the performance of streams S2 and S3 decreases. Afterwards, the performance of S3 begins to slowly improve as the threshold increases, while the performance of S2 continues to decrease for larger threshold values. In general, as the threshold value increases across the graph, the performance of S0 and S1 improve, while that of S2 decreases. The performance of S3 decreases suddenly at the 20% threshold and then monotonically increases until the 60% threshold is reached, where its performance again drops and begins increasing. From among the two streams that impose lighter relative load on the system (i.e., S0 and S1), the performance of the more variable stream (i.e., S1) improves the most. However, from among the two streams that impose heavier relative load on the system (i.e., S2 and S3), the performance of the more regular stream (i.e., S2) suffers the most. This graph demonstrates the dynamic nature of the workload and highlights the complex interactions that can occur among different streams. This reiterates the idea that the performance of individual task streams impacts the performance of other streams.

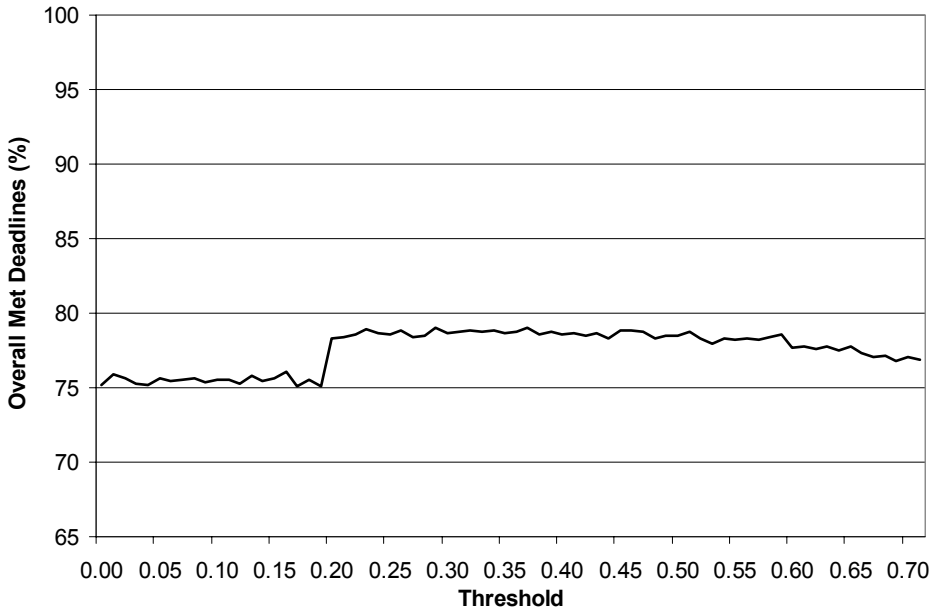

Figure 74: TLAX threshold comparison for medium load

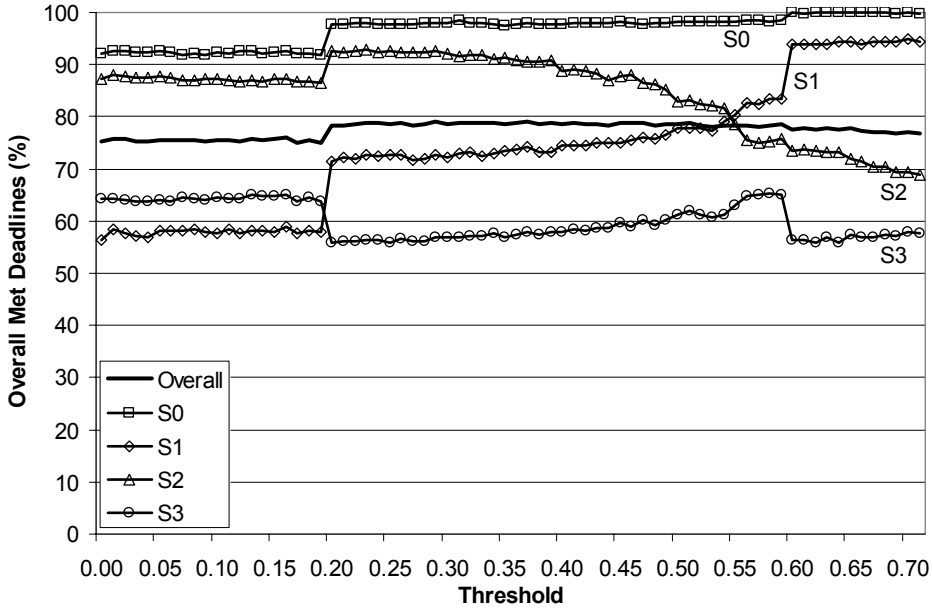

Figure 75: Detailed TLAX threshold comparison for medium load

7.3.3 Analysis under Heavy Load

Under heavy load, the overall performance is quite sensitive to changes in the threshold value. Figure 76 illustrates the change in performance as the threshold value increases. Notice that even for smaller threshold values where less laxity is held in reserve, it is advantageous to use a larger threshold value. For example, increasing the threshold value from 20% to 30% provides a relative performance increase of nearly 4% in the overall percentage of met deadlines. Overall, the performance improvement of using a larger threshold value in the range of 40% – 50% is apparent under heavy load, and provides a relative performance increase of about 13%.

Figure 77 summarizes the performance results for individual task streams. It is apparent that the task streams influence each others performance for threshold values less than about 40%. Aside from the interaction between S2 and S3 at about a 50% threshold, the performance of each task stream stabilizes for threshold values greater than 40%. The graph also shows that for threshold values greater than 50%, the performance of S3 (a relatively variable stream) continues to increase, while the performance of S2 (a relatively regular stream) decreases. This is likely a reflection of the workload variability, where increased variability generally boosts performance under heavy load, while increased regularity generally degrades performance. From the figure, it is evident that the best overall performance occurs when using a threshold value in the range 40% – 60%. Note that the fluctuation of values (i.e., jitter) seen in the figures in this chapter is typical in such simulation studies. An analytical study is presented in the next chapter that, while more limited in scope, removes the simulation jitter and validates these simulation findings.

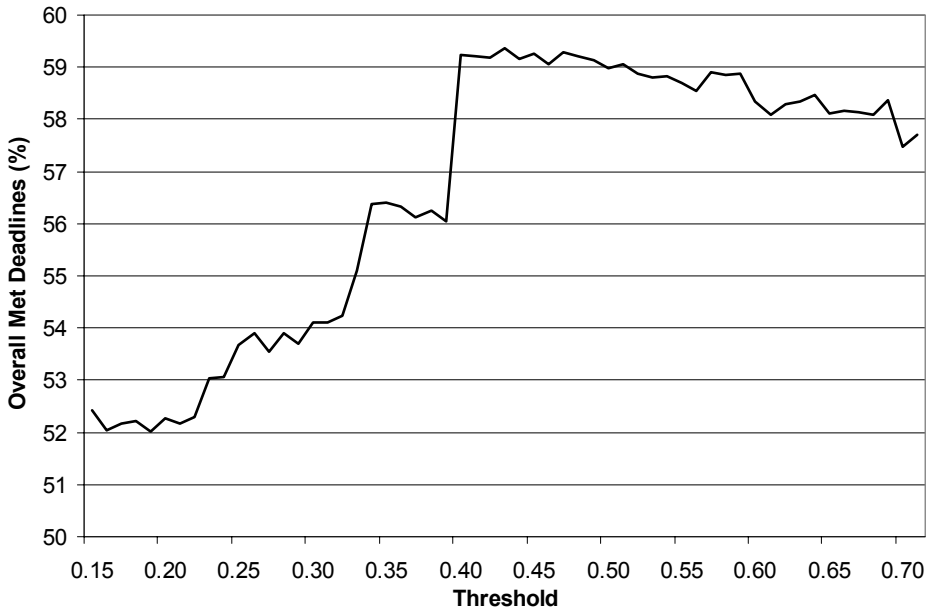

Figure 76: TLAX threshold comparison for heavy load

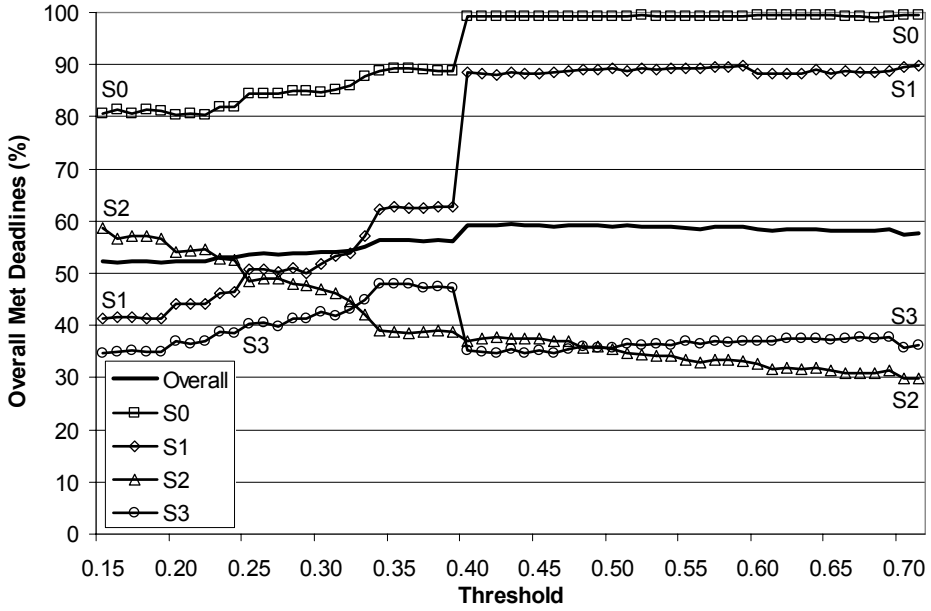

Figure 77: Detailed TLAX threshold comparison for heavy load

7.4 Evaluating Additional System Loads

In addition to light, medium, and heavy system loads, a number of additional loads are considered in order to span the gap between light and medium loads, and between medium and heavy loads. The intermediate loads between light and medium correspond to intensity factors ranging from 0.6 to 0.9. Similarly, the intermediate loads between medium and heavy correspond to intensity factors ranging from 1.1 to 1.4. For each load, TLAX is again evaluated using a range of threshold values in order to determine the preferred threshold for a given system load. For each intensity factor (i.e., system load), the best found threshold value (i.e., the one that produces the maximum percentage of met deadlines) is identified. Figure 78 shows a plot of the preferred threshold value as a function of the average system utilization. A linear (dashed) best-fit line is shown in addition to the data curve. As shown in the figure, the value of the best-found threshold increases as the system utilization increases. Increased utilization corresponds to heavier system load and therefore, more laxity should be held in reserve to maximize performance. Recall from the XLAX analysis (see Table 28 in Section 6.8.4) that a 50% threshold value results in the best overall system performance across various system loads. Additional support for this finding is seen in Figure 78, where the performance gain under heavier loads is maximized by using a 50% threshold.

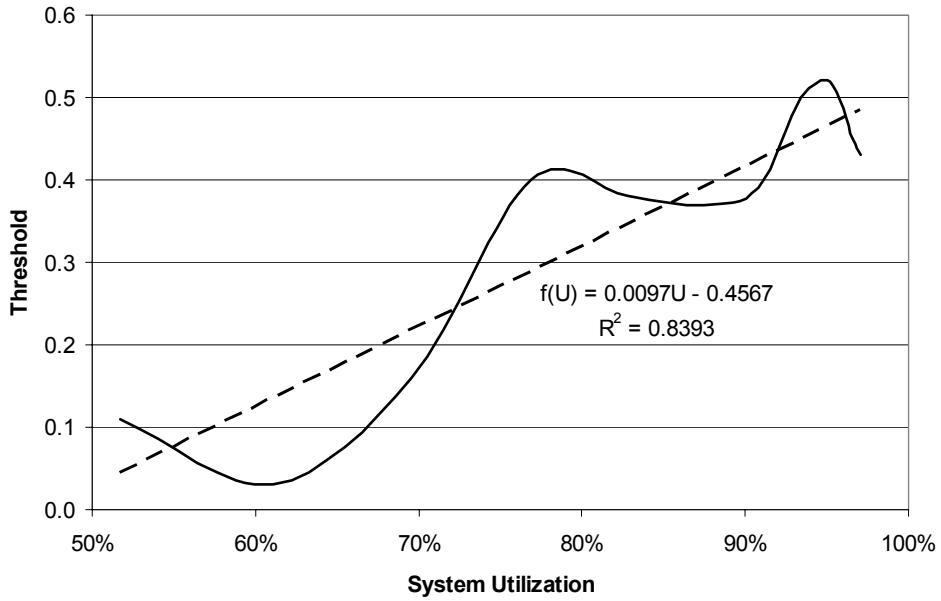

Figure 78: Preferred TLAX threshold values as a function of utilization

Figure 78 demonstrates a data curve that follows a varying pattern that generally increases from left to right, as the system utilization increases. As mentioned previously, this jitter is expected, due to the inherent randomness (e.g., random number generation) embedded into the simulation results. That is, in a series of simulations, it is expected to obtain values that are above, as well as below, the actual/correct values. Thus, an average of a set of metrics is used as an approximation for the actual/correct value. The linear best fit provides a reasonable approximation of the optimum threshold value as a function of the system utilization. The linear equation is denoted *f(U)*, where *U* represents the system utilization. Using a known utilization level as input, an estimate of the best threshold value for TLAX can be obtained using the approximation function *f(U)*. Figure 79 shows the TLAX algorithm performance comparison for different system utilizations. The performance of TLAX-50, TLAX-f(U), and the traditional scheduling algorithms are shown. For system utilizations under about 83%, there is little or no

performance gain using the TLAX algorithm when compared to the performance of EDF. However, for heavier loads, TLAX begins to outperform all of the traditional scheduling algorithms. The biggest performance boost is seen under the heaviest system loads, where the utilization approaches 100%. Under these conditions, both TLAX-50 and TLAX-f(U) outperform the next best algorithm (i.e., EDF) by about 26%. Notice that the performance of TLAX-f(U) is nearly identical to that of TLAX-50, regardless of the system utilization. This suggests that using TLAX with a 50% threshold provides a good tradeoff between necessary overhead involved with evaluating and maintaining different load data, and maximizing the overall system performance. That is, in systems where it may not be practical to implement TLAX with a dynamic threshold, a good alternative is to fix the TLAX threshold value at 50%.

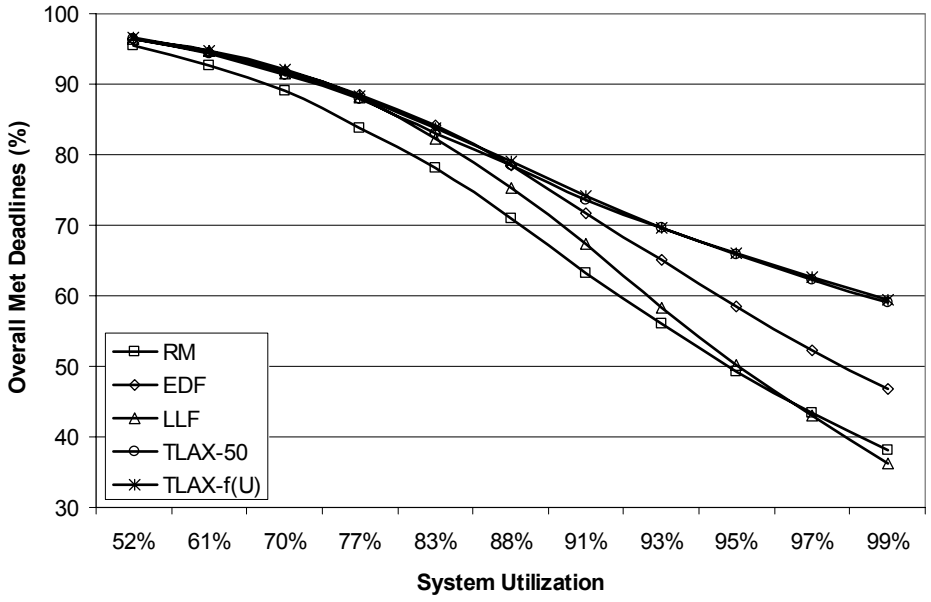

Figure 79: Performance comparison of algorithms for various system loads

As expected, as the system utilization increases, the overall percentage of met deadlines decreases due to increased load. However, the trend of any individual task stream does not necessarily follow this same trend, particularly under heavy load. Figure 80 shows a comparison of the percentage of met deadlines of the various task streams as the system utilization increases. For utilizations under 90%, the trend of each task stream follows that of the overall percentage of met deadlines (i.e., decreasing as a function of the system load). However, for utilizations above 90%, this trend does not hold. For example, the percentage of met deadlines for S1 increases for utilizations ranging from 90% to 95%. This effect is presumably the result of the relatively high variability of S1, where its variable behavior allows it to meet additional deadlines under increased load conditions. The effect of increased variability boosting system performance in heavy load conditions has been noted in related work [25]. Streams S0 and S3 also exhibit increased met deadlines within the 90% – 95% utilization range. While the performance of both S2 and S3 decreases significantly, that of S2 decreases suddenly when the utilization reaches about 90%. This effect is again attributed to the relatively high regularity of S2 under heavy load. Overall, these results indicate that individual task streams can influence each others performance and do not necessarily follow the same trend as the overall performance. For utilizations higher than about 92%, the emphasis shifts from shorter tasks (i.e., tasks with smaller service requirements) to tasks that are more regular. Therefore, workload variability is particularly important because task streams with different variability characteristics can influence the performance of other task streams, even those exhibiting very regular behavior.

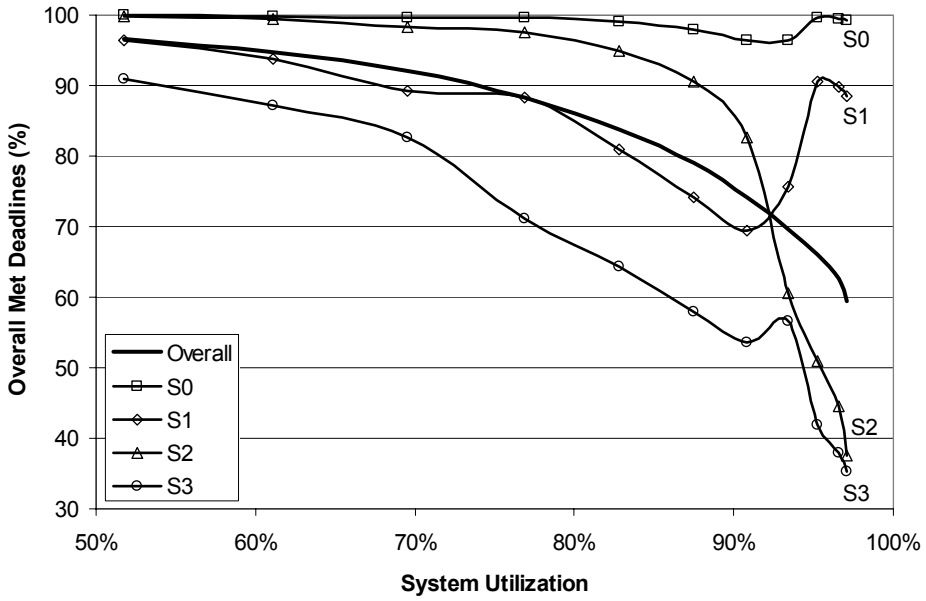

Figure 80: TLAX performance comparison for individual task streams

7.5 Evaluating Different Workloads

The previous discussion focuses on a single workload consisting of four task streams that differ in their performance characteristics. Although various system loads have been evaluated, no changes to the actual workload have been considered. In this section, five different, but related, workloads are considered. Among these workloads, the workload discussed previously will be referred to as Workload 3 (W3) because it corresponds to the middle of a workload spectrum. W3 consists of two task streams S0 and S1 that exhibit the same average arrival and service characteristics, but S0 is regular whereas S1 exhibits variability. Similarly, streams S2 and S3 exhibit the same average characteristics but S2 is regular compared to the variable S3 stream. Therefore, the workload discussed throughout this chapter (i.e., Workload 3) consists of two regular streams (i.e., S0 and S2) and two variable streams (i.e., S1 and S3). Two of the streams (i.e., S0 and S1) impose less relative load on the system, whereas the other two streams (i.e., S2 and S3) impose

more relative load on the system. A spectrum of related workloads can be constructed by modifying the composition of the task streams in each workload. Figure 81 illustrates this spectrum of workloads that range from regular to variable. On the left end of the spectrum, Workload 1 consists of four identical tasks, each of which exhibits high regularity. On the opposite end of the spectrum, Workload 5 consists of four identical streams as well, each of which exhibit high variability. Using combinations of these task streams provides a spectrum of workloads ranging in variability, regularity, and relative system load. Just as different load conditions were imposed on Workload 3, light, medium, and heavy load conditions can be imposed on each of these five workloads. Note that these load conditions are relative to each workload and a particular utilization (e.g., under light load) for one workload does not necessarily correspond to the same utilization in a different workload. Therefore, the meaning of light, medium, and heavy are relative to each other within the same workload. Table 32 summarizes the characteristics for each of these workloads, assuming relatively moderate load conditions.

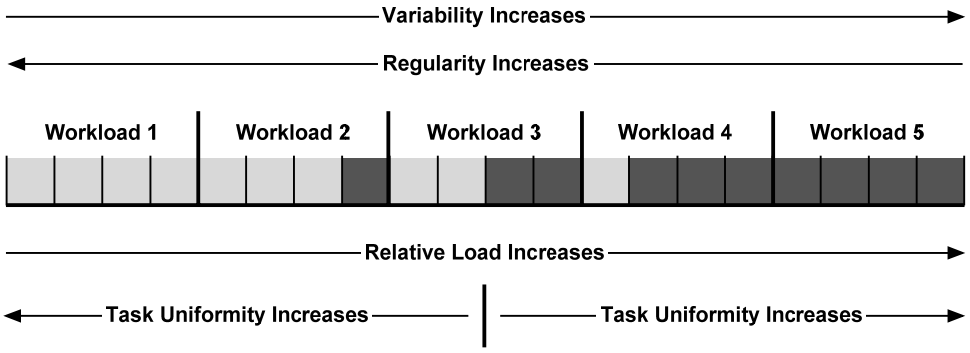

Figure 81: Workload spectrum

Table 32: Parameters for task streams along the workload spectrum

		Workload Parameters		
Workload	Task Stream(s)	Inter-arrival Time (min)	Number of Arrival/Service Stages	Service Time (min)
W1	S0, S1, S2, S3	30.00	10	4.00
W2	S0, S1, S2	30.00	10	4.00
	S3	20.00	2	8.00
W3	S0	30.00	10	4.00
	S1	30.00	2	4.00
	S2	20.00	10	8.00
	S3	20.00	2	8.00
W4	S0	30.00	10	4.00
	S1, S2, S3	20.00	2	8.00
W5	S0, S1, S2, S3	20.00	2	8.00

Workload 1 contains high regularity but imposes relatively light load on the system. This results in light, medium, and heavy loads that correspond to system utilizations of 27%, 53%, and 80%, respectively. Figure 82 provides a performance comparison for each algorithm under the three different system loads for Workload 1. For all three load conditions, the performance of EDF, LLF, and TLAX-50 is comparable with little difference. The performance of RM at higher system utilizations is somewhat lower.

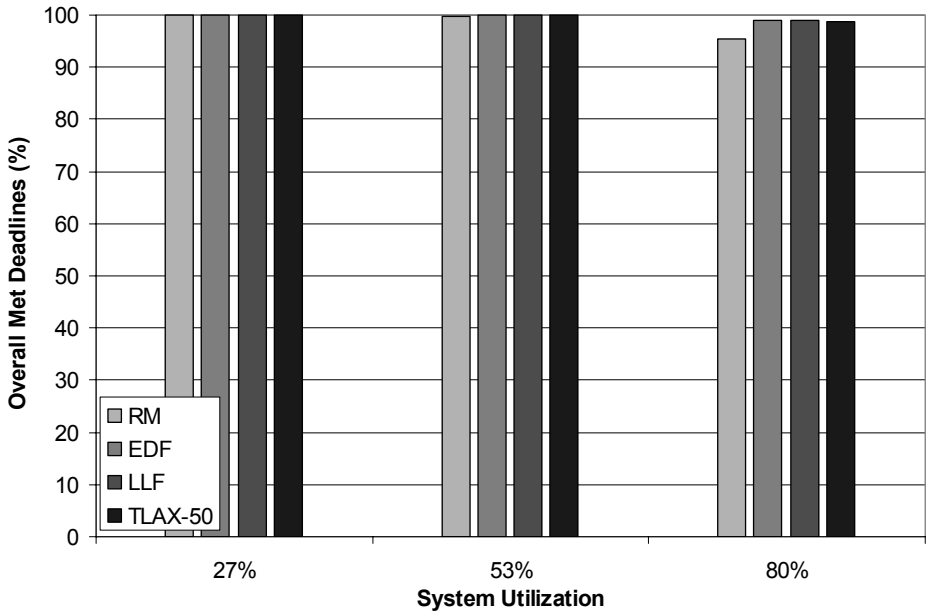

Figure 82: Performance summary for Workload 1

For Workload 2, the traditional algorithms outperform TLAX-50 under both the medium and heavy load conditions by about 1%, as shown in Figure 83. This behavior is similar to that seen throughout this chapter when discussing Workload 3. Under light load conditions, TLAX is comparable to the traditional algorithms, with less than 1% difference among all the algorithms.

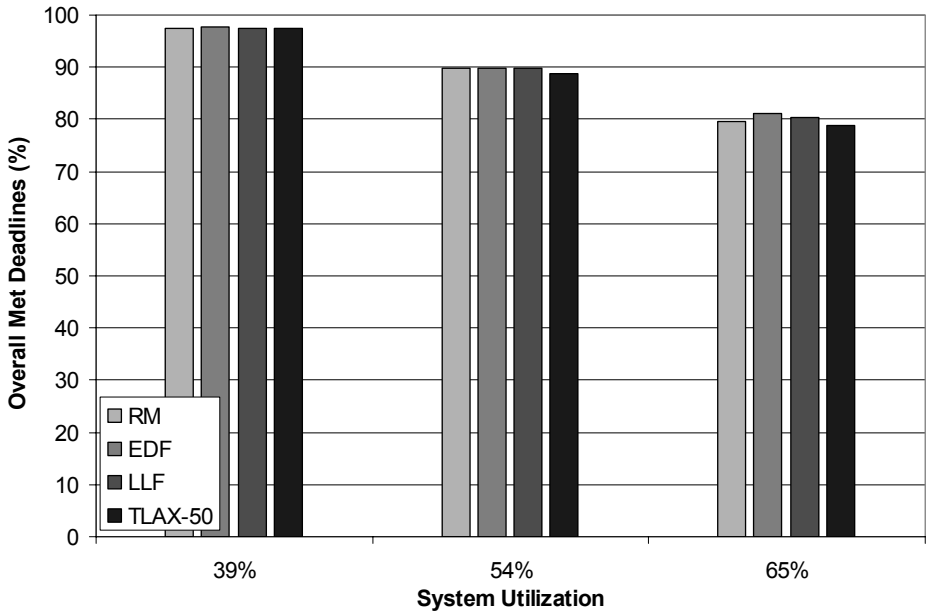

Figure 83: Performance summary for Workload 2

Workload 3 is the driving example discussed throughout this chapter but the resulting performance data is shown again in Figure 84. For the light and moderate load conditions of this workload, TLAX performs comparably to the best traditional algorithm, EDF. However, for a utilization of 99%, TLAX outperforms the traditional scheduling algorithms.

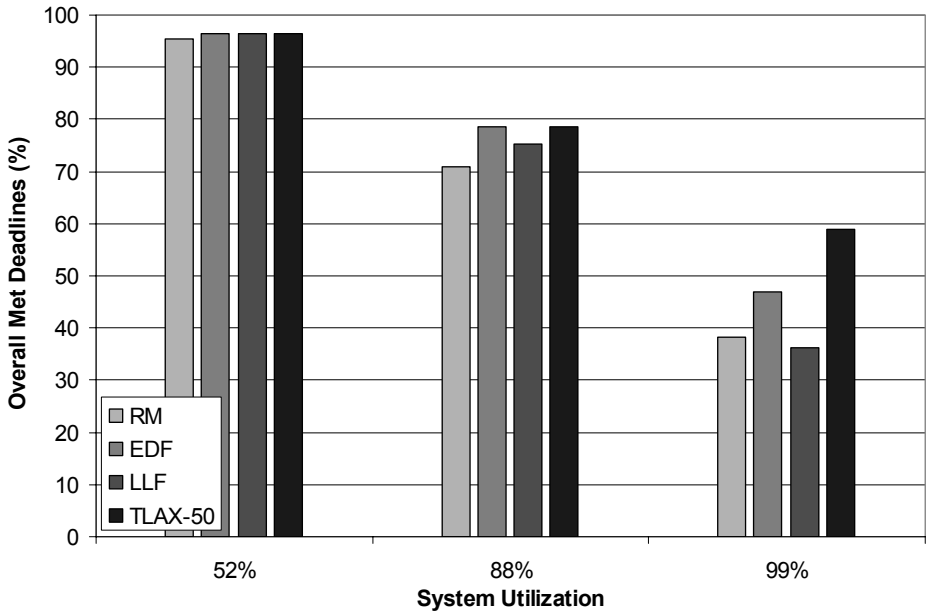

Figure 84: Performance summary for Workload 3

In Workload 4, medium and heavy load conditions correspond to higher system utilizations when compared to the medium and heavy load conditions of the first two workloads (i.e., Workload 1 and Workload 2). The performance comparison for Workload 4 is shown in Figure 85. Under light load (i.e., 61% utilization), the performance of all the algorithms is comparable. However, TLAX outperforms the traditional algorithms under the medium and heavy load conditions. Under medium load (i.e., 91% utilization), TLAX outperforms EDF by about 5%. Under heavy load (i.e., 98% utilization), TLAX outperforms RM by 36%, EDF by 13%, and LLF by 50%.

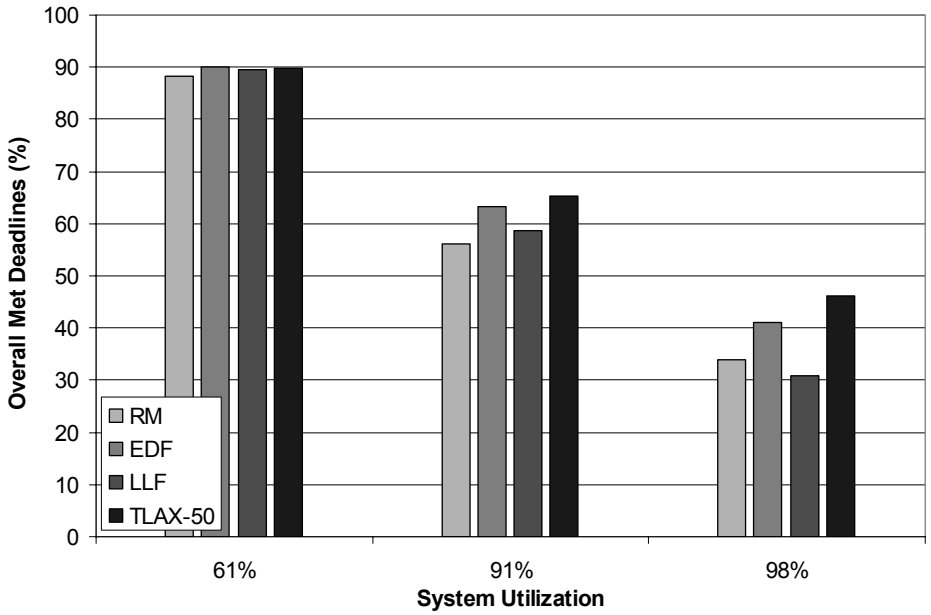

Figure 85: Performance summary for Workload 4

The results for Workload 5 are shown in Figure 86 and are similar to those observed for Workload 4. For the light load situation, EDF outperforms TLAX-50 by about 1%. For the medium and heavy system load conditions, TLAX-50 outperforms all of the traditional algorithms by about 9% in each case. From the results shown in Figures 84, 85, and 86, TLAX performs relatively better for workloads that are more diverse (i.e., workloads where the task streams are not as uniform). That is, from Workload 3 to Workload 5, the diversity decreases (and uniformity increases), as illustrated previously in Figure 81. Among these three workloads (i.e., Workload 3, Workload 4, and Workload 5), the relative performance of TLAX is best for Workload 3, slightly worse for Workload 4, and worst for Workload 5. This suggests the performance of TLAX improves as the diversity of the workload increases, suggesting it is a robust scheduling algorithm.

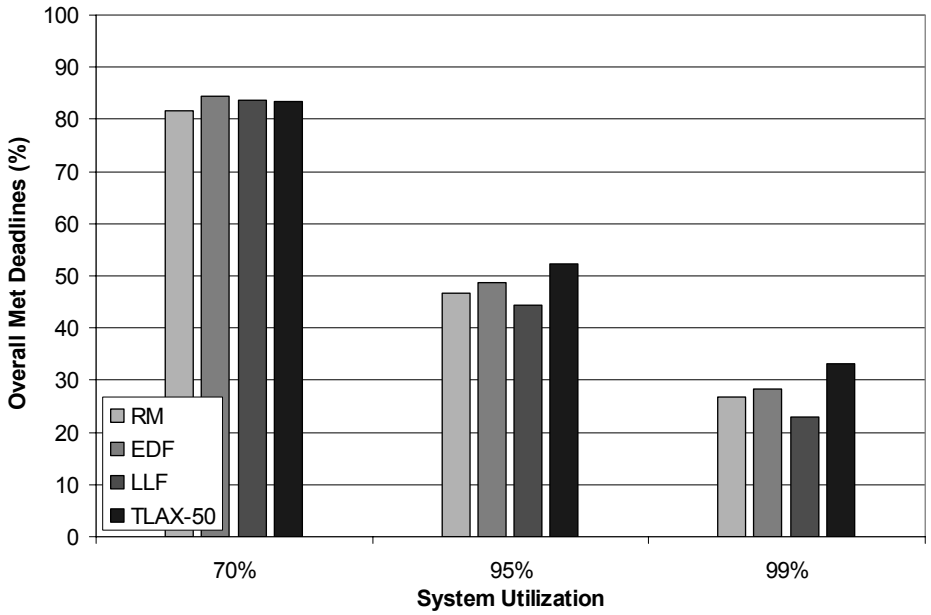

Figure 86: Performance summary for Workload 5

Figure 87 shows a performance summary for all of the workloads as a function of the system utilization as it increases from 50% to nearly 100%. For moderate system loads where the utilization is less than about 70%, all of the algorithms exhibit similar performance. For utilizations ranging from about 70% to 88%, EDF and TLAX-50 both maximize the percentage of met deadlines. At these higher utilizations, the spread among the performance of the algorithms increases, and both LLF and RM fail to match the performance of EDF or TLAX. Finally, for utilizations above 90%, TLAX outperforms all of the traditional algorithms. Again, for the three heaviest load conditions (i.e., utilizations of 98%, 99%, and 99%), TLAX outperforms the best traditional algorithm (i.e., EDF) by 13%, 26%, and 17%, respectively. The overall trend is that the performance decreases as the utilization increases, but three general exceptions occur for utilizations of 27%, 53%, and 80%. At these utilizations, the percentage of met deadlines is nearly 100% for all the scheduling algorithms. Referring to the workload descriptions,

these three situations correspond to light, medium, and heavy load conditions for Workload 1, which consists of four identical and regular tasks.

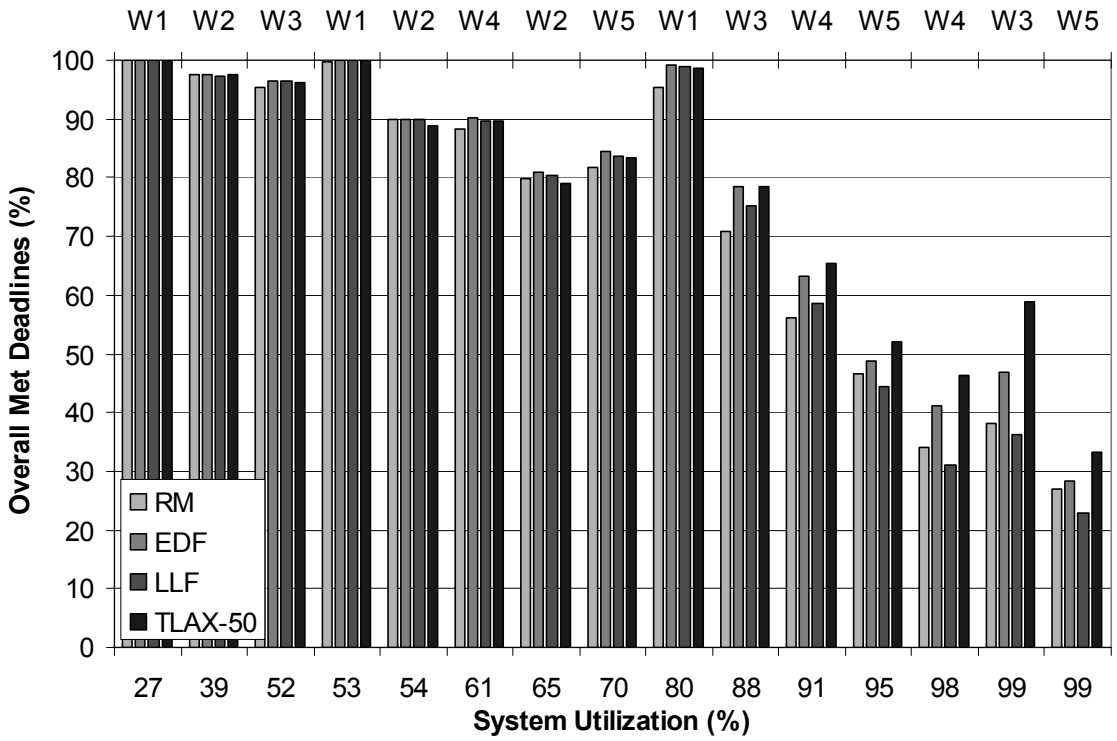

Figure 87: Performance summary for all workloads

7.6 Sensitivity to Workload Variability

The performance of the TLAX algorithm has been compared to that of the traditional scheduling algorithms by examining different workloads under different system load conditions. The impact of changes to workload variability has also briefly been considered in the analysis due to the differences in variability found within the example workloads. For example, the four identical tasks that makeup Workload 1 exhibit high regularity, while the four identical tasks that comprise Workload 5 exhibit high variability.

To better demonstrate the effects of workload variability, consider Workload 5, which consists of four identical, highly variable (i.e., low regularity) tasks. To modify the variability of this workload, the number of arrival and service stages is varied from 2 to 10, holding all other parameters constant. As the number of stages for a given parameter decreases, the variability (i.e., variance) of the parameter increases. For each number of stages, simulations are run to evaluate the performance of the scheduling algorithms discussed previously. In this way, the effect of changes in variance for each algorithm can be examined and compared to that of the other algorithms. The changes in variance are considered under light, medium, and heavy system load conditions. To provide an additional comparison for the TLAX-50 algorithm, the performance of TLAX-30 and TLAX-70 are shown as well, where each of these algorithms uses a 30% and 50% threshold, respectively. Note that in each of the load situations, the utilization varies across a different range of values for each algorithm. Therefore, the average utilization across all the variance values is used in the following discussion to provide a good estimate of the overall system utilization.

Figure 88 shows the performance results of each algorithm as the workload is incrementally changed from one with high variability to one with high regularity, assuming that the system is under light load conditions (i.e., utilization of about 70%). The number of stages as well as the CV (coefficient of variation) is shown to indicate the workload variability. As the number of stages increases, the variability of the arrival and service behavior of each task stream decreases. That is, moving from left to right across the graph corresponds to an increase in regularity. As Figure 88 shows, increased regularity results in improved performance for all of the scheduling algorithms. All of

the traditional algorithms except for RM perform similarly as the workload variability changes.

For the TLAX algorithm, TLAX-30 and TLAX-50 perform similarly, with TLAX-30 resulting in a slight performance advantage in most cases. Notice that for three to ten stages, TLAX-70 performs worse than either TLAX-30 or TLAX-50, and outperforms only the RM algorithm. Overall, Figure 88 shows that all of the algorithms are sensitive to changes in variability under light system load. To maximize the overall system performance, either the EDF or LLF traditional algorithms should be used. For a variation of TLAX, either TLAX-30 or TLAX-50 results in similar performance that is comparable to that of the best traditional algorithms (i.e., EDF and LLF).

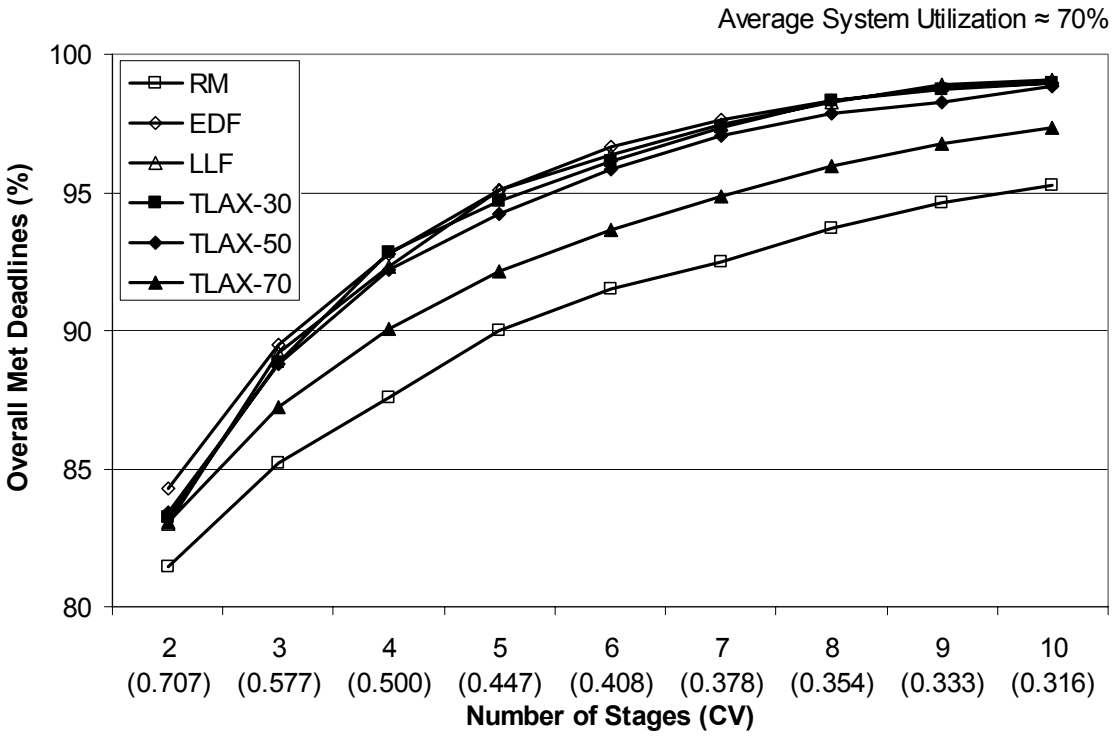

Figure 88: Effect of workload variability under light load

Figure 89 compares the algorithm performance as the variance is changed under medium load conditions (i.e., utilization of about 80%). As the number of stages increases (i.e., the variance of the workload decreases), the performance of all the traditional algorithms decreases. This is in contrast to what is seen under light load conditions, where increased regularity boosts the performance of all of the algorithms. As Figure 89 indicates, each of the traditional algorithms is much more sensitive to changes in variability when compared to the TLAX algorithms. The performance of TLAX varies, and TLAX-30 performs generally worse as the variability decreases. Both TLAX-50 and TLAX-70 are more stable, and therefore, less sensitive, to the changes in variability. The performance of TLAX-70 remains nearly constant regardless of the workload variability. Finally, TLAX-50 and TLAX-70 are both relatively insensitive to changes in variability, but TLAX-50 consistently provides slightly better performance throughout.

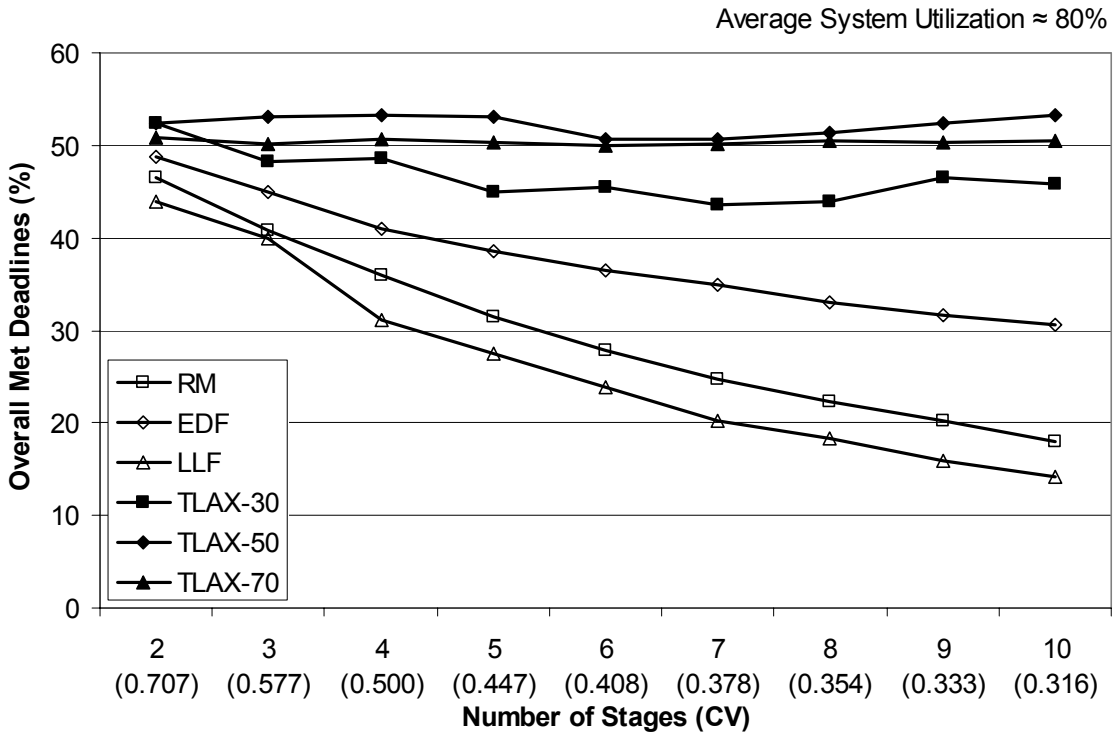

Figure 89: Effect of workload variability under medium load

Figure 90 shows the performance results for each algorithm assuming that the system is under heavy load conditions (i.e., utilization of about 98%). The performance of each of the traditional algorithms decreases significantly as the workload variability decreases. The performance of TLAX also decreases as the workload regularity increases, but this effect is less pronounced compared to the traditional algorithms. Consider the best traditional algorithm, EDF, where the percentage of met deadlines decreases from 27% for two stages, to barely 9% for ten stages. This performance decrease corresponds to nearly a 67% relative decrease in performance. Except for two stages, TLAX-50 performs best from among the three TLAX algorithms. The performance of TLAX-50 decreases from 33% for two stages to about 30% for 10 stages. This performance change

corresponds to only a 9% relative change in performance, which is a significant performance gain over the best-performing traditional algorithm (i.e., EDF).

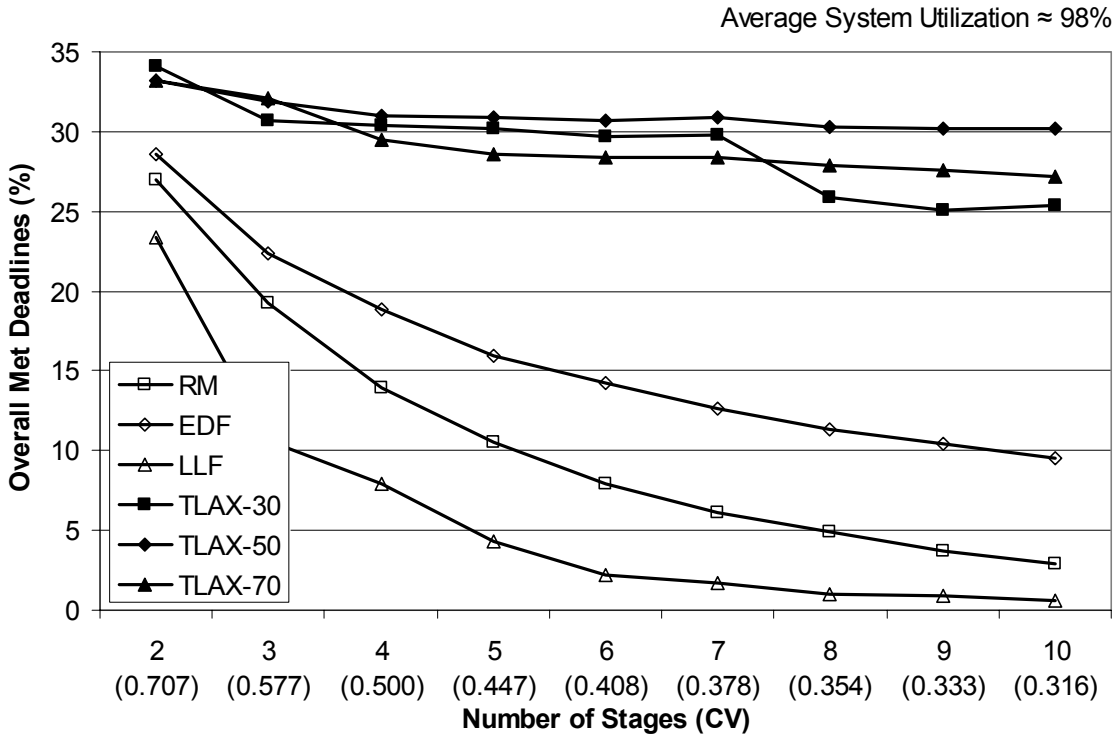

Figure 90: Effect of workload variability under heavy load

Table 33: Percent change in met deadline percentage of algorithms

		Traditional Algorithms			TLAX Algorithms		
Number of Stages	CV	RM	EDF	LLF	TLAX-30	TLAX-50	TLAX-70
2	0.707	0.00	0.00	0.00	0.00	0.00	0.00
3	0.577	-28.47	-21.91	-54.60	-9.91	-4.03	-3.30
4	0.500	-27.94	-15.81	-25.16	-1.13	-2.90	-8.05
5	0.447	-24.61	-15.45	-45.37	-0.51	-0.36	-2.98
6	0.408	-24.31	-10.39	-49.85	-1.74	-0.56	-0.82
7	0.378	-22.56	-11.38	-21.56	-4.29	0.58	-0.18
8	0.354	-20.80	-10.26	-41.62	-8.83	-1.99	-1.78
9	0.333	-22.92	-8.26	-12.45	-2.95	-0.09	-0.81
10	0.316	-22.88	-8.78	-32.45	1.20	-0.16	-1.57
Average		-24.31	-12.78	-35.38	-3.52	-1.19	-2.44

Table 33 shows a summary of the percent change in met deadline percentage for each algorithm. Among the traditional algorithms, EDF has the smallest percentage change with an average decrease of about 13%. However, the best TLAX algorithm (i.e., TLAX-50) has an average percent change of just over 1%. The results presented in the previous figure and graph illustrate that a significant performance increase results from using TLAX-50 under heavy load conditions. Further, the performance of TLAX is almost unaffected by changes in workload variability, suggesting it is a robust algorithm for use in heavy load conditions.

Overall, this sensitivity analysis shows that the performance of traditional scheduling algorithms can be significantly affected by changing only the workload variability. This effect is intensified under heavy load conditions, where many tasks are missing their deadlines. Under these conditions, where there is heavy load and unsteady workload variability, the TLAX-50 algorithm provides robust performance even in the presence of changing variability within the workload.

It should be noted that although the variability of both the arrival and service behavior are changed in the analysis presented in this section, similar effects have been demonstrated when varying only the arrival process. In [45], for example, the arrival patterns of network traffic data are studied and a second-moment characterization of traffic streams is introduced. It is shown how simple computations based on the variances of arrival patterns can be used to determine accurate QoS estimates for performance parameters. These estimates are used in new scheduling routines that result in higher system utilizations.

7.7 Chapter Summary

In this chapter, a novel scheduling algorithm named TLAX is presented that uses a threshold-based laxity value along with prioritized groups of tasks to make scheduling decisions. It is shown that a 50% threshold results in the overall best performance for TLAX across different system load conditions. Under light to moderate system load, TLAX generally performs comparably to the best traditional algorithm. However, under heavy load conditions, TLAX-50 outperforms the best traditional algorithm by as much as 26%.

Under heavy load, the TLAX algorithm is also robust in regard to changes in workload variability. That is, the performance of all the traditional scheduling algorithms decreases significantly, as the workload becomes more regular in heavy load conditions, but the performance of TLAX-50 remains nearly constant regardless of the workload variability. The TLAX algorithm, therefore, shows promise in its application to scheduling in heavy-load situations in soft real-time systems.

The results presented in this chapter are based on data from sensitivity analysis experiments conducted using MOSS. Because the output from any simulation tool is subject to jitter, an analytical approach is required in order to avoid such issues. In the next chapter, a mathematical and theoretical approach based on state-space analysis is used to evaluate the performance of TLAX and validate the results from MOSS.

7.8 Research Contributions

The contributions presented in this chapter include:

- Analyzing and comparing the TLAX algorithm based on simulation results provided from MOSS that illustrate its robustness and show that it can outperform the traditional scheduling algorithms under heavy load conditions
- Providing additional evidence that workload variance significantly affects system performance, where under light load, increased variance degrades performance, but under heavy load, increase variance boosts performance
- Illustrating that as the system utilization increases, the amount of laxity held in reserve for tasks should also increase
- Demonstrating that the performance trends of individual task streams do not necessarily follow that of the overall performance curve, where the overall system performance may increase under heavy load, but the performance of one or more task streams decreases due to their variability

CHAPTER VIII

ANALYTICAL STATE-SPACE VALIDATION OF TLAX

8.1 Introduction

In the previous chapter, the analysis of TLAX was based on simulation data obtained from MOSS. Because the output from any simulation tool is subject to jitter[17], the purpose of this chapter is to analyze the TLAX algorithm on a theoretical and mathematical basis. Using a mathematical approach, the TLAX results obtained from MOSS can be validated and further explored to investigate its behavior. Due to the precise description of the method of stages modeling approach, the technique of constructing state diagrams and using state-space analysis provides a convenient mechanism for conducting the mathematical study.

The use of state diagrams is a useful analytical technique for analyzing the details of complex systems. Using state-space analysis, details that would otherwise be difficult to analyze can easily be studied to help gain insight into system behavior. In this chapter, state-space analysis is used to study the differences among scheduling decisions made by EDF and TLAX from a theoretical, as opposed to simulation, perspective. We developed a specialized tool, the Matlab State-space Analysis Tool (MSAT), to construct and solve state-space models describing small real-time systems. Without this tool, the analysis presented in this chapter would not be possible. To keep the analysis tractable, a workload consisting of two task streams is analyzed in detail. Using this approach, it can

[17] Jitter is variability that results from simulation effects such as the underlying random number generation.

be demonstrated mathematically that TLAX outperforms the EDF algorithm under heavy loads. By studying and solving the underlying global balance equations of the state-space diagrams, insight can be gained as to how the scheduling decisions made by TLAX differ from those made by traditional algorithms, such as EDF.

In the next section, a workload consisting of two task streams is presented. The technique of state-space analysis is then revisited (see Section 2.3.8) and used to motivate the remainder of the discussion. The state diagrams are solved using MSAT and examined to gain insight into the TLAX algorithm. The results obtained analytically from solving the state-space models are used to validate the MOSS findings.

8.2 Workload Configuration

Two task streams are used in this study to keep the analysis tractable. That is, the number of states in the resulting state-space diagram increases exponentially as the complexity of the example increases. Therefore, this study serves as a proof of concept and an initial step towards broader theoretical analysis. The parameters for each of the two streams under various loads are shown in Table 34. By using only two task streams and a small number (i.e., 2 to 4) of stages for each stream, the size of the resulting state-space diagrams is reduced and the explicit performance results are tractable. Note that even in this limited system, the number of resulting simultaneous equations that must be solved is 120. Compare this to the 435,600 states that would be found in a state diagram of any of the examples presented in the previous chapter.

When using two task streams, increasing or decreasing the relative load imposed on the system is accomplished by scaling the inter-arrival times. Therefore, light, medium, and heavy loads can be imposed by using intensity factors of 0.5, 1.0, and 1.5,

respectively. For each of the two task streams, the deadline is assumed to coincide with the next arrival of the same type. That is, the deadline process of each task stream *is* its arrival process. This assumption is common in practical systems and is used here to reduce the example complexity and size of the resulting state-space diagrams.

Table 34: Workload configuration

System Load	Task Stream	Inter-arrival Time (min)	Service Time (min)	Number of Arrival/Service Stages	Arrival/Service CV
Light (IF=0.5)	S0	18.0	6.00	2	0.707
	S1	24.0	8.00	4	0.500
Medium (IF=1.0)	S0	9.0	6.00	2	0.707
	S1	12.0	8.00	4	0.500
Heavy (IF=1.5)	S0	6.0	6.00	2	0.707
	S1	8.0	8.00	4	0.500

8.3 Understanding the State-Space Model

Before constructing a state-space model, a state descriptor is defined that captures all the necessary state information for a given state. When using the method of stages, the natural state descriptor is one in which a numeric value is used to represent the current stage of each process (i.e., arrival, service, or deadline) for each task. Therefore, in the example in this chapter, the state descriptor is a four digit numeric value (i.e., the current stage of execution of the arrival and service process of each task stream).

The information captured in a state descriptor is also represented graphically in the state diagram using the method of stages notation. Figure 91 shows the state descriptor representing an example state (i.e., state 1020). Note that because the deadline of each task is its arrival, there can never be more than one task of the same type present at any given time. In the figure, the label above the circle is a compact representation of the

information shown graphically. The first half (i.e., the leftmost two digits, 10) of the state label contains information for task stream S0, while the second half (i.e., rightmost two digits, 20) contains information for task stream S1. The two digits corresponding to S0 indicate the active stage of its arrival and service process, respectively. Similarly, the two digits corresponding to S1 indicate the active stage of its arrival and service process, respectively. The active stage of each process is indicated in the state descriptor by a solid circle. In state 1020, for example, stream S0 is in its first arrival stage, while stream S1 is in its second arrival stage. The service process of each task stream is in stage 0 (i.e., neither stream has begun execution).

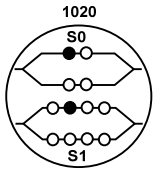

Figure 91: Example state descriptor

8.3.1 Constructing the State-Space Diagrams

At any instant, the arrival process and service process of each task may be executing in any one of their available stages. For an arrival process, one of its stages is always active, but for a service process, it is possible that none of the service stages is active. This occurs, for example, after a service process successfully completes, but before the next task arrival. Because the method of stages technique utilizes well-defined rules that describe how each process progresses through its stages, it is easy (though somewhat tedious and complex) to construct a state-space diagram for any given scheduling algorithm. Figure 92 shows a portion of a state diagram that illustrates task arrivals,

based on the task configuration presented previously. In this figure, arrival rates for S0 and S1 are indicated by the values λ_0 and λ_1, respectively. The arrival rates are scaled by the appropriate number of stages to obtain the per-stage transition rates. Note that the states shown in this partial diagram are algorithm-independent, because a scheduling decision is made only when the service processes of both tasks are active. In other words, in this diagram, there are no states shown in which the service processes of both S0 and S1 are simultaneously active. Thus, the scheduling decision is irrelevant.

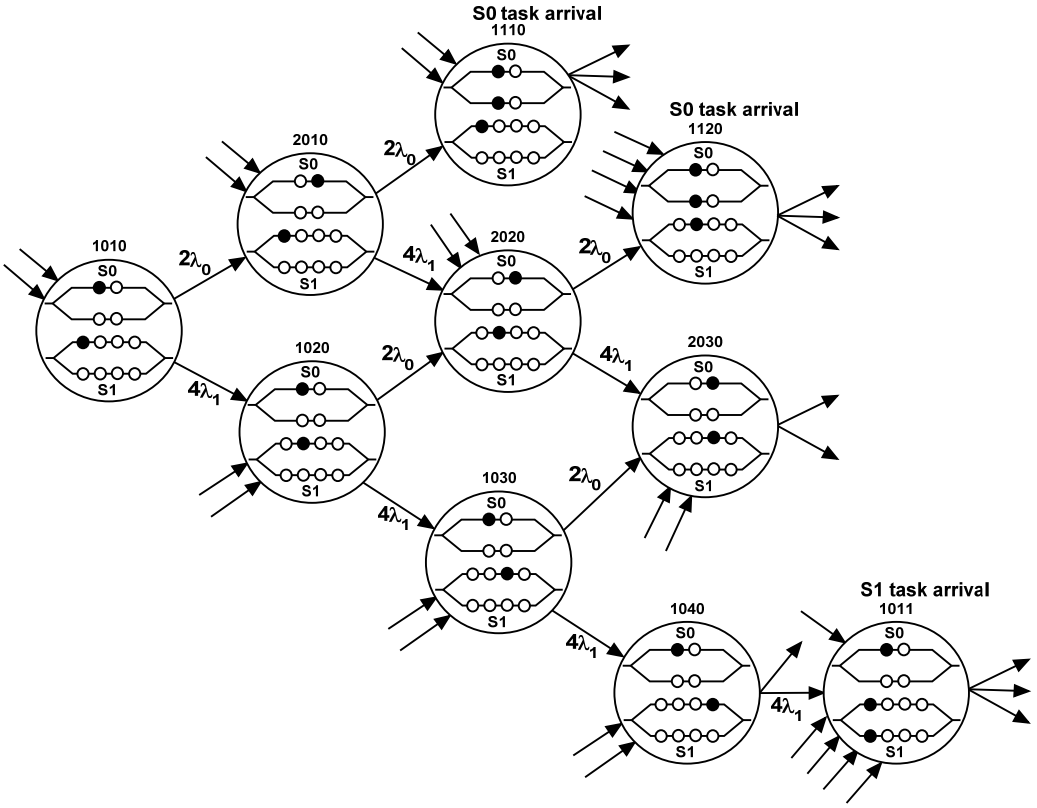

Figure 92: Partial state diagram of task arrivals

Even when analyzing only two task streams and two to four stages, the size of the state diagram is large enough that constructing and solving the model becomes tedious.

For example, consider the configuration shown in Table 34. For a task from S0, the arrival process must always be in one of its two possible stages, and its service process must be in one three possible states (i.e., inactive, in stage 1, or in stage 2). This leads to 6 (2 * 3) possible combinations. Simultaneously, a task from stream S1 must be in one of its four possible arrival stages, and its service process must be in one of five possible states (i.e., inactive or in one of its four service stages). For this task, there are 20 (4 * 5) possible combinations. Therefore, overall there are 120 (6 * 20) possible states. Note that depending on the specific scheduling algorithm used, the resulting state diagram may not encompass the entire set of possible states. That is, some states may be missing from a given state diagram depending on which scheduling algorithm is used. However, to analytically solve this system in order to obtain the expected number of met deadlines, a system of 120 (i.e., one for each state) equations must be solved simultaneously.

To make the study of the resulting state-spaces feasible, a tool (see Section 8.4) was created in Matlab that constructs the complete state-space diagram, including all state transition arcs and their associated weights. To accomplish this, the method of stages modeling technique is expressed in Matlab code. Then, configuration parameters and an initial start state are specified. Beginning with the start state, all possible reachable states are generated, along with the corresponding arcs and weights. Next, each of the newly created states is examined to determine additional reachable states. Using this recursive approach, a list of all reachable states, transition arcs, and arc weights is constructed and combined to form the complete state-space model.

8.3.2 Solving the Model

If a system is observed long enough, it will reach "steady-state," in which the flow into any given state will equal the flow out of that same state [63]. That is, in steady state, the difference obtained by subtracting the outgoing flow from the incoming flow (or vice versa) must be zero. Using this approach, each state can be examined and the expressions for the incoming and outgoing flows can be used to determine the corresponding values in a state transition matrix. Thus, the global balance equations are determined for each state and one side is subtracted from the other so that the sum is zero.

Figure 93 shows an example state (state 2020, outlined in bold) from the previous diagram, along with its associated incoming and outgoing states. In the figure, service rates for S0 and S1 are indicated by the values μ_0 and μ_1, respectively. The service rates are scaled by the appropriate number of stages to obtain the per-stage transition rates. States 2010, 1020, 2220, and 2024 are responsible for the incoming flow to state 2020, while the outgoing flow from state 2020 travels to states 1120 and 2030. Therefore, the global balance equation for state 2020 is constructed using the expressions shown in Table 35, and is given by $4\lambda_1 P_{2010} + 2\lambda_0 P_{1020} + 2\mu_0 P_{2220} + 4\mu_1 P_{2024} - (2\lambda_0 + 4\lambda_1) P_{2020} = 0$. (The term P_{abcd} denotes the probability of being in state *abcd* in steady state.)

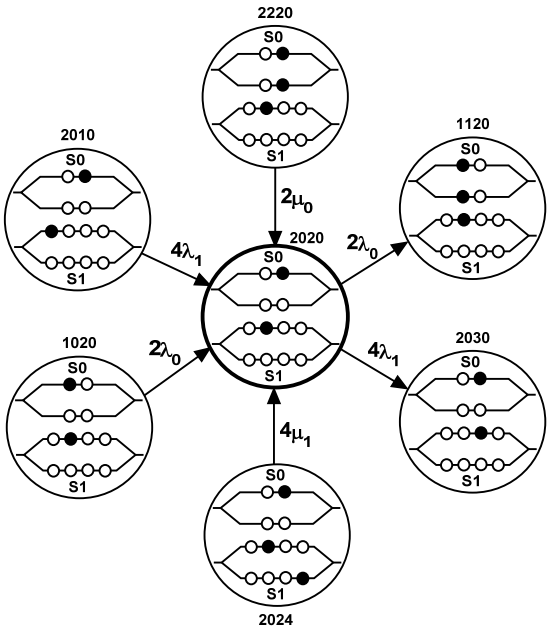

Figure 93: Example state illustrating incoming and outgoing flow

Table 35: Global state information for state 2020

Incoming Flow	Outgoing Flow
$4\lambda_1 P_{2010} + 2\lambda_0 P_{1020} + 2\mu_0 P_{2220} + 4\mu_1 P_{2024}$	$(2\lambda_0 + 4\lambda_1) P_{2020}$

In a similar manner, an equation is formed for each state, resulting in a system of 120 equations and 120 unknowns (i.e., the steady-state probabilities). In solving such a system, one additional piece of information (i.e., the fact that all the steady-state probabilities must sum to one) is used to construct the last row of a transition matrix. Overall, the resulting linear system fully describes the state-space model and in matrix form is represented by Ax=b, where A is the state transition matrix, x is the vector of unknown steady-state probabilities, and b is the solution vector consisting of all zeros, except for the last row, which contains a value of 1. This organization is illustrated in Figure 94. Given the values in Table 34 for medium load (i.e., $\lambda_0=\frac{1}{9}$, $\lambda_1=\frac{1}{12}$, $\mu_0=\frac{1}{6}$, and

$\mu_1=\frac{1}{8}$), the resulting global balance equation for state 2020 is given in equation 1. This is represented in matrix form as shown in Figure 94. Systems of this form can be solved quickly and efficiently using Matlab.

$$\frac{1}{3}P_{2010}+\frac{2}{9}P_{1020}+\frac{1}{3}P_{2220}+\frac{1}{2}P_{2024}-\frac{5}{9}P_{2020}=0 \quad (1)$$

$$\underbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots \frac{1}{3} & \cdots \frac{2}{9} & \cdots \frac{1}{3} & \cdots \frac{1}{2} & \cdots -\frac{5}{9} \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots 1 & \cdots 1 & \cdots 1 & \cdots 1 & \cdots 1 \cdots \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \vdots \\ P_{2010} \\ \vdots \\ P_{1020} \\ \vdots \\ P_{2220} \\ \vdots \\ P_{2024} \\ \vdots \\ P_{2020} \\ \vdots \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \vdots \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{b}$$

Figure 94: Matrix representation of global state equations

8.3.3 Calculating Performance Metrics

After the steady-state probabilities are obtained, many desired metrics can be computed directly from the model. For example, the overall system utilization can be obtained by summing the probabilities across all states in which at least one task is executing. Alternately, the percent of time the system is idle can be obtained by summing the

probabilities across all states in which no task is executing. The overall utilization can therefore, be computed as 1.0 – *idle_percentage*. The states corresponding to a met (or missed) deadline for a given task stream are also easily identified. Using these states, the percent of met (or missed) deadlines can be calculated by multiplying the probability of being in that state by the weight associated with the appropriate outgoing arc.

Other desired metrics can computed in a similar manner. For example, the number of tasks executing in any given state is easily obtained by observing the state descriptor (in this example, there is always zero, one, or two tasks executing). The steady-state probability for a given state corresponds to the percentage of time spent in that state. Therefore, the average number of tasks expected to be executing in a given state is found by multiplying the number of tasks executing (i.e., zero, one, or two) by the steady-state probability. Overall, the expected number of tasks executing in the system at any given time is found by computing the sum of the average number of tasks executing in each state.

8.4 The Matlab State-Space Analysis Tool

Constructing and solving even small state-space models can be tedious and complex. To resolve this issue, we developed a tool using Matlab called MSAT (Matlab State-space Analysis Tool) that can construct a state-space diagram corresponding to a real-time system composed of two task streams. The model is then solved analytically using Matlab's efficient matrix operations, allowing many performance metrics to be computed exactly. This section provides a description of the design and limitations of MSAT, as well as how it is used to obtain the analytical results presented in this chapter.

8.4.1 Tool Design

MSAT is a Matlab program that runs from the command prompt. For convenience, the interface of MSAT is provided by a configuration section in a program script file. Although a graphical user interface can easily be added, this feature is left as future work. The number of task streams is fixed at two, and the deadline of each task is its next arrival. This simplifies the tool development, shifting the focus to the engine that constructs and solves the state diagrams. Approximately 2,000 lines of code comprise the functionality of MSAT, which is broken up into functions/modules. The modular design of MSAT allows it to be easily modified and updated to support additional scheduling algorithms, the ability to solve larger models, and functionality to output a wide range of performance metrics and graphs.

8.4.2 Configuration of Parameters

For each task stream, the arrival and service behavior is specified by parameters listed in the configuration section of the MSAT script. Figure 95 shows the Matlab Editor window displaying a portion of the configuration section for MSAT. As shown, the inter-arrival time (*iat*), number of arrival stages (*nas*), service time (*st*), and number of service stages (*nss*) are specified for each task stream. The scheduling algorithm name (*alg*) is specified using a text string and a threshold value (*tlaxthresh*) can be specified for the TLAX algorithm. Additional algorithms and parameters can easily be added. For convenience, the initial start state (*states(1).descriptor*) is also a configuration parameter that allows the user to modify the beginning of the constructed state diagram. This is useful for larger models, when a specific state may be of particular interest. As with MOSS, Processor Sharing (PS) is used in the case of a tie, when two tasks have exactly

the same scheduling criterion (e.g., reserve laxity, in the case of TLAX). For convenience, the tolerance value used to test for ties between scheduling criterions is a configuration parameter (*TEST_TOL*). An output directory (*outputfolder*) specifies the directory in which output files should be created. In Figure 95, the number of specified stages is small (i.e., one for the first[18] task stream and two for the second task stream) to reduce the size of the state-space model shown later in Section 8.4.4.

```
22    % ------------------------BEGIN CONFIG SECTION-------------
23    % CONFIG FOR TASK STREAM 1
24 -  config.tasks(1).iat = 18.0;
25 -  config.tasks(1).nas = 1;
26 -  config.tasks(1).st = 6.0;
27 -  config.tasks(1).nss = 1;
28
29    % CONFIG FOR TASK STREAM 2
30 -  config.tasks(2).iat = 24.0;
31 -  config.tasks(2).nas = 2;
32 -  config.tasks(2).st = 8.0;
33 -  config.tasks(2).nss = 2;
34
35    % Output folder for text files
36 -  config.outputfolder = 'SD_output_standard';
37
38    % Tolerance value used for comparisons
39 -  config.TEST_TOL = 0.00001;
40
41    % CONFIG FOR THE SCHEDULING ALGORITHM
42 -  config.alg = 'EDF';  % choices: RM, EDF, LLF, TLAX
43 -  config.tlaxthresh = 0.50; % only used when .alg = 'TLAX'
44
45    % Specify the start state
46 -  states(1).descriptor = 1010;
47    % ------------------------END CONFIG SECTION---------------
```

Figure 95: Portion of MSAT configuration section

8.4.3 Running MSAT and Solving Models

After the configuration parameters have been specified, the tool is run from the command prompt or directly from the Matlab Editor. MSAT begins with the specified start state

[18] Note the notation change, where the two task streams are named S1 and S2, as opposed to S0 and S1. This is to maintain consistent notation in the Matlab code, as Matlab indices must begin with 1.

and constructs a list of all possible reachable states. Then, an iterative process is used to construct the reachable states for each remaining state until the complete state diagram has been constructed. For each state, a list of incoming and outgoing arcs is saved, along with all the weight values associated with the transitions. This information is stored in a compact form using Matlab's built-in *struct* data type.

The state-transition matrix (A) is constructed from the state-space information by recursively examining the list of states, arcs, and arc weights. The vector x represents the unknown steady-state probabilities, and the vector b represents the sum vector. Each entry in b represents the sum of the corresponding global state equation, where all terms in the equation have been moved to the left side, for a sum of zero. Therefore, the b vector contains all zeros except for the entry in the last row, which corresponds to the sum (i.e., 1) all of the steady-state probabilities. This results in a linear system expressed in matrix form as $Ax=b$ and is quickly and efficiently solved using Matlab. The linear system is solved to obtain the steady-state probability for each state. The solutions are checked to verify that $A*x$ equals b to within the desired level of accuracy (the chosen default value is currently 0.0000000001). The solution vector is also checked to verify that the sum of all the steady-state probabilities is 1.

8.4.4 Performance Metrics and Output

From the steady-state probabilities, the utilizations and percentage of met deadlines are automatically computed by MSAT. Support for additional metrics can be easily added. All output can be selectively displayed on the screen and/or output to files. In addition to the desired output metrics, the state-space diagram is also of interest because it is sometimes desirable to examine the diagram manually. However, it is not possible to

display the generated state diagrams in an existing format that is easily viewed. To remedy this problem, we apply a custom format to the state diagram information that allows it to be displayed in a clear, compact manner.

Figure 96 shows a portion of output from MSAT, assuming the configuration parameters specified in Figure 95, EDF scheduling, and start state of 1010 are used. In this example, there are only 12 states in order to reduce the size of the output information and state-space diagram. Information regarding the state counts is shown, along with the steady-state probabilities. Letters are used to represent the arc weights to make the state diagram more compact. The letters *A* and *B* correspond to the per-stage arrival rate and service rate, respectively, of the first task stream (i.e., S1). The letter *C* is equal to one-half of *B* and corresponds to the rate at which S1 executes when there is a tie and PS is used. The letters *D*, *E*, and *F* have similar connotation, only for the second task stream (i.e., S2). The performance metrics are shown at the bottom of the output window. The last portion of the output file (i.e., the state-space diagram) is shown in a separate figure (i.e., Figure 97) to conserve space.

```
STATE INFORMATION
  State Counts
    # total: 12
    # with at least one self arc: 6
    # with no task executing: 2
    # with S1 executing: 4
    # with S2 executing: 6
    # with S1 and S2 executing (PS): 0

  Steady-State Probabilities
    (sum of probabilities: 1.0000000000)

    State   Steady-state Probability
    1010    0.1853644596
    1110    0.0411921021
    1020    0.2926871796
    1120    0.1049877202
    1011    0.1062857944
    1111    0.0748568224
    1021    0.0227755274
    1012    0.0755188539
    1121    0.0225101268
    1022    0.0308240220
    1112    0.0167819675
    1122    0.0262154240

  Arc Weights
    (Stream 1)
    A=0.055556        (per stage arrival rate)
    B=0.166667        (per stage service rate)
    C=(B/2)=0.083333  (per stage PS rate)
    (Stream 2)
    D=0.083333
    E=0.250000
    F=(E/2)=0.125000


PERFORMANCE METRICS
  Utilizations:
    Overall: 52.19% (47.81% idle)
    Stream 1: 23.78%
    Stream 2: 28.41%

  Met Deadline Percentages:
    Overall: 74.86% (25.14% missed)
    Stream 1: 71.35% (28.65% missed)
    Stream 2: 79.53% (20.47% missed)
```

Figure 96: Portion of state information and output metrics from MSAT

Figure 97 shows the state diagram output from MSAT. In the diagram, each rectangle corresponds to a state and its related information. For a given state, its state descriptor is enclosed in parentheses and its steady-state probability is enclosed in curly

braces (i.e., {}). The labels to the left of a state descriptor correspond to incoming arc information, while the labels to the right of the descriptor correspond to its outgoing arc information. Consider state 1110, shown on the upper right of the state diagram. The label on the left (i.e., 1010[A]) indicates that state 1110 has an incoming arc with weight A from state 1010. Similarly, the labels on the right indicate that there are two arcs leaving (i.e., outgoing) from state 1110—one arc has weight D and goes to state 1120, while the other arc has weight B and travels to state 1010. In the case of self-arcs (i.e., arcs where source and destination states are the same), the weights of any such arcs are listed underneath the state descriptor. Notice that in this diagram, the letters C and F (corresponding to PS arcs) are not present because there are no states in which PS is used (i.e., there are no scheduling ties).

```
STATE DIAGRAM

+-------------------------+-------------------------+
|      {0.1853644596}     |      {0.0411921021}     |
| 1110[B]->(1010)->[A]1110 | 1010[A]->(1110)->[D]1120 |
| 1012[E]          [D]1020 |           A      [B]1010 |
+-------------------------+-------------------------+
|      {0.2926871796}     |      {0.1049877202}     |
| 1010[D]->(1020)->[A]1120 | 1110[D]->(1120)->[D]1111 |
| 1120[B]          [D]1011 | 1020[A]   A      [B]1020 |
| 1022[E]                  | 1122[E]                  |
+-------------------------+-------------------------+
|      {0.1062857944}     |      {0.0748568224}     |
| 1020[D]->(1011)->[A]1111 | 1120[D]->(1111)->[D]1121 |
| 1111[B]          [D]1021 | 1011[A]   A      [B]1011 |
| 1021[D]          [E]1012 | 1121[D]                  |
| 1022[D]                  | 1122[D]                  |
+-------------------------+-------------------------+
|      {0.0227755274}     |      {0.0755188539}     |
| 1011[D]->(1021)->[A]1121 | 1011[E]->(1012)->[A]1112 |
|                  [D]1011 | 1112[B]          [D]1022 |
|                  [E]1022 |                  [E]1010 |
+-------------------------+-------------------------+
|      {0.0225101268}     |      {0.0308240220}     |
| 1111[D]->(1121)->[D]1111 | 1021[E]->(1022)->[A]1122 |
| 1021[A]   A      [E]1122 | 1012[D]          [D]1011 |
|                          |                  [E]1020 |
+-------------------------+-------------------------+
|      {0.0167819675}     |      {0.0262154240}     |
| 1012[A]->(1112)->[D]1122 | 1121[E]->(1122)->[D]1111 |
|           A      [B]1012 | 1022[A]   A      [E]1120 |
|                          | 1112[D]                  |
+-------------------------+-------------------------+
```

Figure 97: Example MSAT output showing the state-space diagram

The information listed in the compact form of the state-space diagram can be loaded into any text editor and easily searched for state information. This format represents a convenient mechanism for representing the state information because all of the necessary information is attached to each state, unlike in a traditional state-space diagram where arcs must be examined and followed in order to determine the incoming and outgoing states. Note that although the state diagram output from MSAT contains redundant information, the internal storage representation of the diagram is very efficient and uses

only symbolic links. Overall, MSAT provides a convenient and reliable method of solving the state-space models to analytically obtain the information and metrics necessary for our study.

8.5 Analytical Results

To study the TLAX algorithm, state-space diagrams are constructed and solved for each load condition using MSAT. Because the EDF scheduling algorithm is used for relative performance comparisons of TLAX, the same procedure is repeated for the EDF algorithm. Each state-space model is then solved to obtain the steady-state probabilities and performance metrics. The metrics for each algorithm, along with any state diagram differences, are used to analytically compare and study the behavior of the algorithms.

8.5.1 Performance Comparison Summary

Table 36 provides a summary of the overall percentage of met deadlines for EDF and TLAX. The metrics obtained analytically via MSAT, as well as the metrics output from MOSS, are given in the table. Because the performance of TLAX depends on the particular threshold value used, the threshold value resulting in the maximum performance is chosen. That is, the percentage of met deadlines shown for TLAX corresponds to the maximum value observed across the range of threshold values. As Table 36 shows, the metrics obtained from MOSS closely match those obtained analytically, which validates the MOSS findings presented in the previous chapter. Similar to the results seen in the previous chapter, EDF marginally outperforms TLAX under light load by about 2%, but under heavy load conditions, TLAX outperforms EDF by over 17%.

Table 36: Performance comparison of analytical (MSAT) and simulation (MOSS) results

		Overall % Met Deadlines			
		Analytical (MSAT) Values		Simulation (MOSS) Values	
System Load	Utilization	EDF	TLAX	EDF	TLAX
Light (IF=0.5)	58.69%	83.39%	82.10%	83.43%	82.15%
Medium (IF=1.0)	85.70%	50.49%	52.91%	50.54%	52.99%
Heavy (IF=1.5)	94.76%	29.65%	34.85%	29.71%	34.91%

Table 37 lists the MOSS confidence intervals of both the EDF and TLAX algorithms. The table indicates that with 99% certainty, the MOSS estimates are accurate to within one-tenth of one percent. These results also validate the experimental values obtained from MOSS and emphasize the accuracy and stability in using MOSS for performance modeling of similar systems.

Table 37: MOSS confidence intervals for EDF and TLAX

		MOSS Confidence Intervals			
		EDF		TLAX	
System Load	Utilization	95th	99th	95th	99th
Light (IF=0.5)	58.69%	± 0.044	± 0.069	± 0.068	± 0.093
Medium (IF=1.0)	85.70%	± 0.056	± 0.090	± 0.059	± 0.091
Heavy (IF=1.5)	94.76%	± 0.049	± 0.078	± 0.055	± 0.096

Due to the relatively small state-space of this example (i.e., 120 states in this example, as opposed to the 435,600 states in the examples from the previous chapter), the performance of TLAX under heavy load is less dramatic than previously reported. That is, in this case, TLAX does not achieve the 25% relative performance increase over EDF

as with previous examples. However, if the number of stages is increased so that a wider range of variance values is possible, the performance improvement of TLAX over EDF is more noticeable. Table 38 shows a summary of the performance gains that can be achieved by increasing the number of stages of only S1, assuming heavy load conditions. Note that increasing the number of stages increases the size of the state-space (however, from our experience, MSAT can easily handle state spaces up to about 3,000 states).

Table 38: Performance summary of increased state-space size for heavy load conditions

Number of Stages for S1	Total Number of States	Utilization	Overall % Met Deadlines		Relative Performance Increase of TLAX
			EDF	TLAX	
4	120	94.76%	29.65%	34.85%	17.53%
6	252	97.75%	26.97%	33.68%	25.15%
8	432	98.39%	25.09%	32.92%	31.21%

From the table, it is seen that the performance of TLAX is not as sensitive to changes in the workload variability (i.e., number of stages) as EDF. This confirms the similar observations in Chapter VII. Also note that as the regularity increases (i.e., as the number of stages increase), the utilization/throughput also increases, but the percentage of met deadlines decreases. In real-time systems, it is typically desired to maintain low system utilization in order to improve the percentage of met deadlines. However, when heavy load situations do arise, the performance gain of using a more robust scheduling algorithm such as TLAX is evident. That is, under heavy system utilization, the performance of EDF degrades more quickly than that of TLAX.

8.5.2 Met Deadline Percentages

The TLAX algorithm is parameterized by its threshold value, and therefore, its performance is dependent upon both the threshold value and the system load. To compare the performance differences resulting from different threshold values and varying load conditions, a state-space diagram is constructed for threshold values ranging from 0% to 100% (in 1% increments) for each load condition. This is easily done analytically, using MSAT. Figure 98 shows the overall percentage of met deadlines as a function of the threshold value under light load conditions. (Note that the discrete "jump" behavior seen in the graph is not jitter, but rather a result of the discrete number of stages used to parameterize the variance.) From the figure, and noting the y-axis range, it is seen that the threshold value has only a small impact (approximately 3%) on performance, with the maximum performance obtained by using a threshold value of about 0.68 or greater. Recall from the MOSS simulation results that plots (for example, see Figure 78 in Section 7.4) of performance metrics (e.g., percentage of met deadlines) are typically characterized by jitter, where trends in the plot fluctuate (i.e., increase and decrease). Because the results shown in the figure are generated from analytical values, rather than simulation estimations, the pattern of the deadline percentages is characterized by a strictly increasing function, as expected.

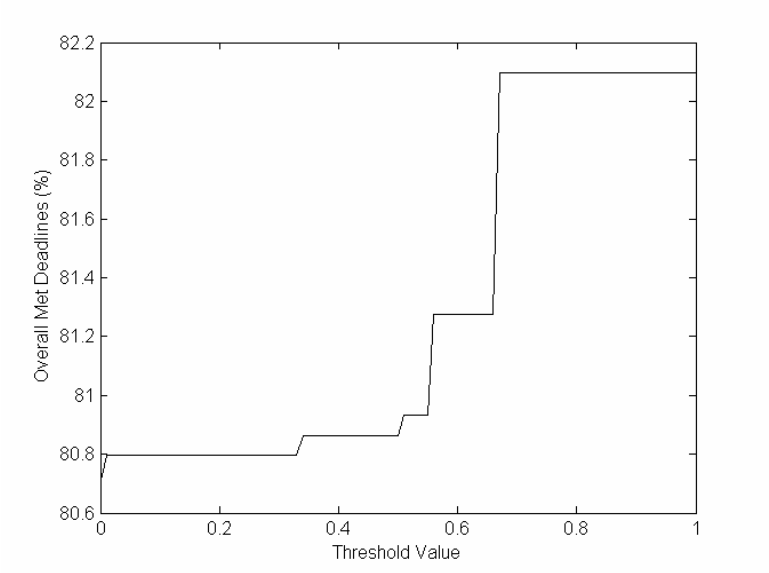

Figure 98: TLAX met deadline percentages for light load

Figure 99 shows the met deadline percentages assuming medium load conditions. Here, the performance impact of the threshold value is greater (approximately 10%) than under light load conditions. In addition, the performance reaches it maximum value earlier in terms of the threshold value. That is, the maximum percentage of met deadlines is obtained by using a threshold value of about 0.35 or greater.

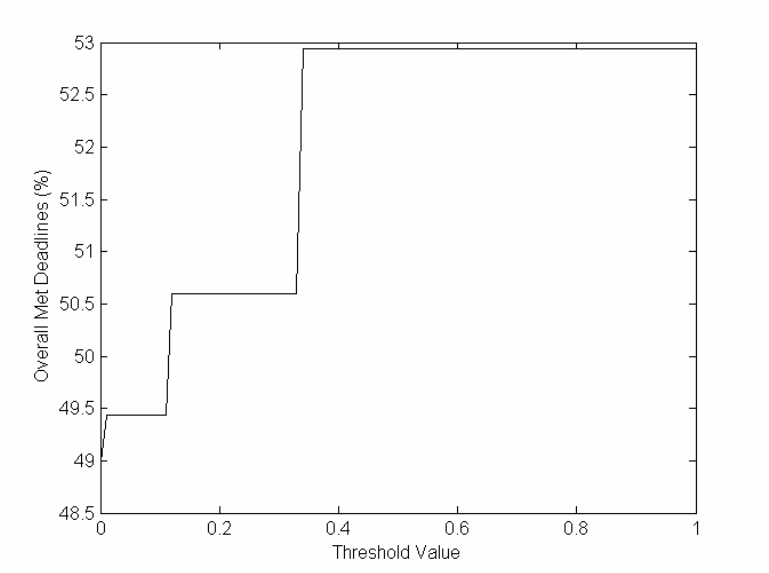

Figure 99: TLAX met deadline percentages for medium load

Figure 100 shows the percentage of met deadlines assuming heavy load conditions. Here, the performance immediately reaches it maximum value for a threshold value greater than zero. The relative impact (i.e., the change in the percentage of met deadlines) of the threshold value, however, is approximately the same (about 10%) as that seen under medium load conditions.

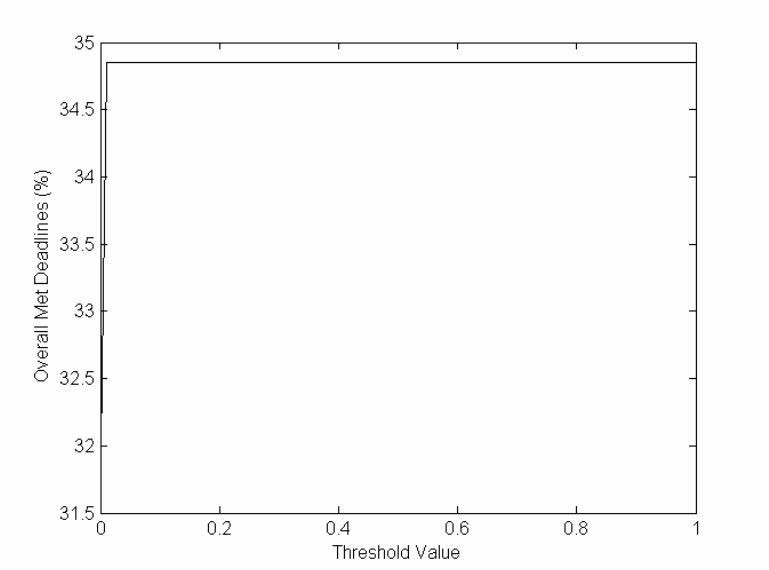

Figure 100: TLAX met deadline percentages for heavy load

Overall, it is seen that the TLAX threshold value affects its performance, as seen previously for the results obtained from MOSS. However, using analytical results, it is easier and more accurate to study the resulting trends because they are consistent and not subject to simulation jitter. In this particular example, using any threshold value other than zero under heavy load conditions will maximize the performance of TLAX. Therefore, in this example, TLAX significantly outperforms EDF under heavy load conditions and the threshold value appears to play little role.

8.5.3 Most Laxity First (MLF) Scheduling

The results presented in the previous figures seem to indicate that the higher the threshold value, the better the resulting performance will be. This implies that withholding the maximum amount of laxity, and thus a "most laxity first" scheduling policy, is a viable scheduling technique. However, it has already been demonstrated in the previous chapter that for several workloads (and particularly under heavy load conditions), this result does not hold. That is, a plot of the TLAX threshold values shows that a point is reached that

corresponds to the maximum performance gain, and after this point, increasing the threshold value any further only decreases the system performance (see Figure 76 in Section 7.3.3).

To further investigate this issue analytically, the Most Laxity First (MLF) algorithm is implemented in MSAT and the performance results are obtained for MLF under light, medium, and load conditions. Table 39 provides a comparison of the performance results for EDF, TLAX, and MLF (results for EDF and TLAX are shown again for convenience). Notice that under light load, MLF is outperformed by both EDF and TLAX. Under medium load, MLF slightly outperforms EDF but falls below TLAX. Under heavy load, MLF outperforms EDF, but MLF is outperformed by TLAX by about 9%.

Table 39: Performance comparison for MLF

		Overall % Met Deadlines		
System Load	Utilization	EDF	TLAX	MLF
Light (IF=0.5)	58.69%	83.39%	82.10%	79.61%
Medium (IF=1.0)	85.70%	50.49%	52.91%	50.97%
Heavy (IF=1.5)	94.76%	29.65%	34.85%	32.04%

These results are based on the small example of two task streams. To test MLF on a more realistic workload, the MLF algorithm is implemented in MOSS and tested on Workload 3 from Chapter VII. Using this workload, the performance of MLF is evaluated under light, medium, and heavy load conditions, the same as in the previous chapter. The performance summary provided in Chapter VII is revisited (see Figure 72) below, as show in Figure 101, with the results for MLF indicated by the slashed bar. As

seen in the figure, MLF is slightly outperformed by all the remaining algorithms under light load. Under medium load, the performance of MLF is similar to that of LLF but falls short of EDF by about 5%. Under heavy load, MLF is still outperformed by EDF by about 5% and is outperformed by TLAX by over 30%.

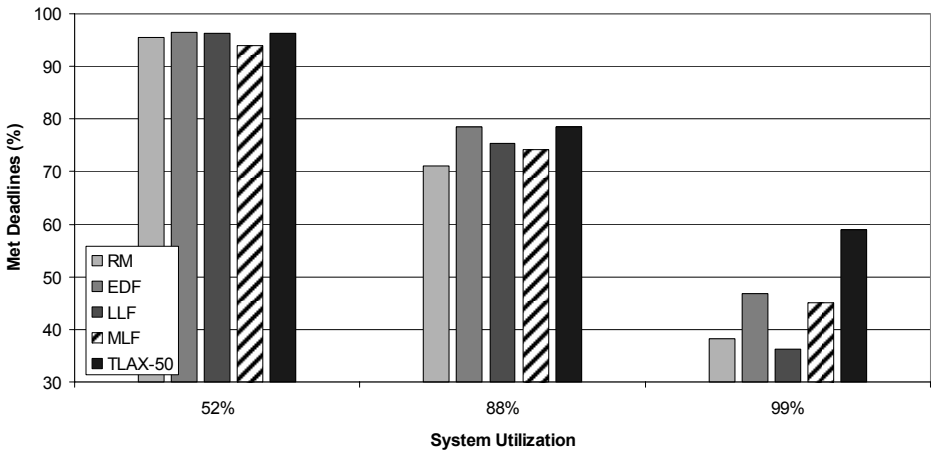

Figure 101: Revisited performance summary with MLF included

Therefore, the MLF policy fails to perform under different workloads and different load conditions. Further, MLF is sensitive to changes in workload variability, just as LLF, which reduces its performance potential. These results show that it is important to carefully examine results in broad scope and that it can be difficult to extract accurate generalizations from workload analysis involving variability.

8.5.4 Examining Missing States

To provide better insight into how the scheduling decisions made by TLAX differ from those made by EDF, information from the state diagrams is examined more closely. For example, while state diagrams for EDF each contain all of the possible 120 states, state diagrams for TLAX, in general, do not. This can be demonstrated by listing all of the

missing states in each TLAX diagram and comparing the diagrams for different TLAX threshold values. Figure 102 provides a summary of the state diagram information for TLAX under light load conditions. As mentioned previously, the full state diagram for this example contains 120 states. In the figure, the state numbers (i.e., short identifiers) are shown along the bottom, while the state descriptors (i.e., the full four-digit identifiers) are shown along the top. A state descriptor indicates the entire system state at a given moment, indicating the current stage of the arrival and service process of each task stream. In state 2130, for example, the arrival and service processes of S0 are in stages 2 and 1, respectively, while the arrival and service processes of S1 are currently in stages 3 and 0, respectively.

In Figure 102, each state is represented by a circle, where grey circles correspond to states that are present in the given diagram. Black circles correspond to states that are missing due to a task from S0 not being allowed to execute. Similarly, white circles correspond to states that are missing due a task from S1 not being allowed to execute. For example, consider state 1212 (i.e., state number 55). To enter this state, the previous state must have been either state 1112 (with an S0 task executing) or state 1211 (with an S1 task executing). However, with a sufficiently high threshold value (i.e., 0.68 or greater), priority would be given to the S1 task in state 1112 and to the S0 task in state 1211. Thus, it is impossible under TLAX to enter state 1212. A key observation is that above certain threshold values, TLAX "sacrifices" certain tasks so that it has a better chance of "saving" others.

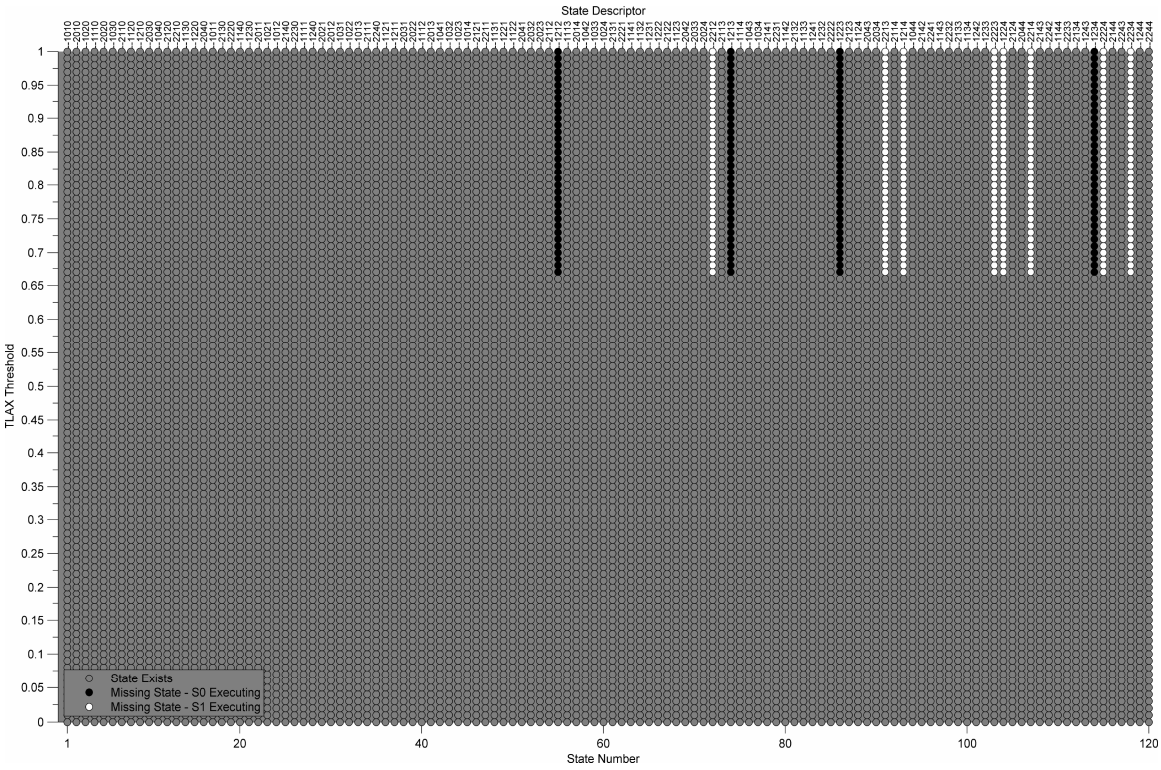

Figure 102: States missing from TLAX diagram for light load

From Figure 102, it is evident that missing states occur only at the larger threshold values. That is, for threshold values less than 0.68, none of the TLAX state diagrams is missing any states. In addition, the absence of a missing state is propagated throughout all state diagrams that correspond to larger threshold values. That is, after a particular state is missing from a diagram corresponding to a given threshold value, the same state is also missing from all diagrams that correspond to larger threshold values. This result indicates the validity of the previous observation about TLAX sacrificing certain tasks. In the figure, this pattern is indicated by the repeated sequence of missing states and appears graphically as either black or white sub-columns.

The black and white sub-columns appearing in the figure offer insight into the scheduling decisions made by TLAX. A single black state represents a missing/inefficient state, where allowing an S0 task to execute would not be allowed

under TLAX because it would lead to suboptimal performance. That is, a black state is one in which TLAX would never find itself because it has already given up on (i.e., sacrificed) the S0 task. Giving the processor to the S0 task would make it more likely that both the S0 and S1 tasks would miss their deadlines, while allocating the processor to the S1 task would make it more likely that the S1 task would meet its deadline at the cost of sacrificing the S0 task by giving up on it. From the figure, there are four black sub-columns but twice as many (i.e., eight) white sub-columns, suggesting that as the threshold value increases, TLAX gives up on an S1 task more often than an S0 task. This is also intuitive, because the S1 tasks place more relative load on the system, by definition. By contrast, the EDF algorithm never gives up on a task, and all the states missing from the TLAX diagrams are present in EDF state diagrams. Thus, EDF does not have the effective coping (i.e., threshold) mechanism that TLAX does.

Figure 103 shows the states missing from the TLAX diagrams assuming medium load conditions. In this figure, the exact same groups of missing states are present. However, under medium load, the missing states first appear in the diagrams corresponding to smaller threshold values. Here, the missing states appear for all threshold values greater than 0.33. Therefore, under medium load, TLAX makes the same decisions in terms of which task it gives up on, only the decision is made earlier (i.e., for smaller threshold values) as the load increases.

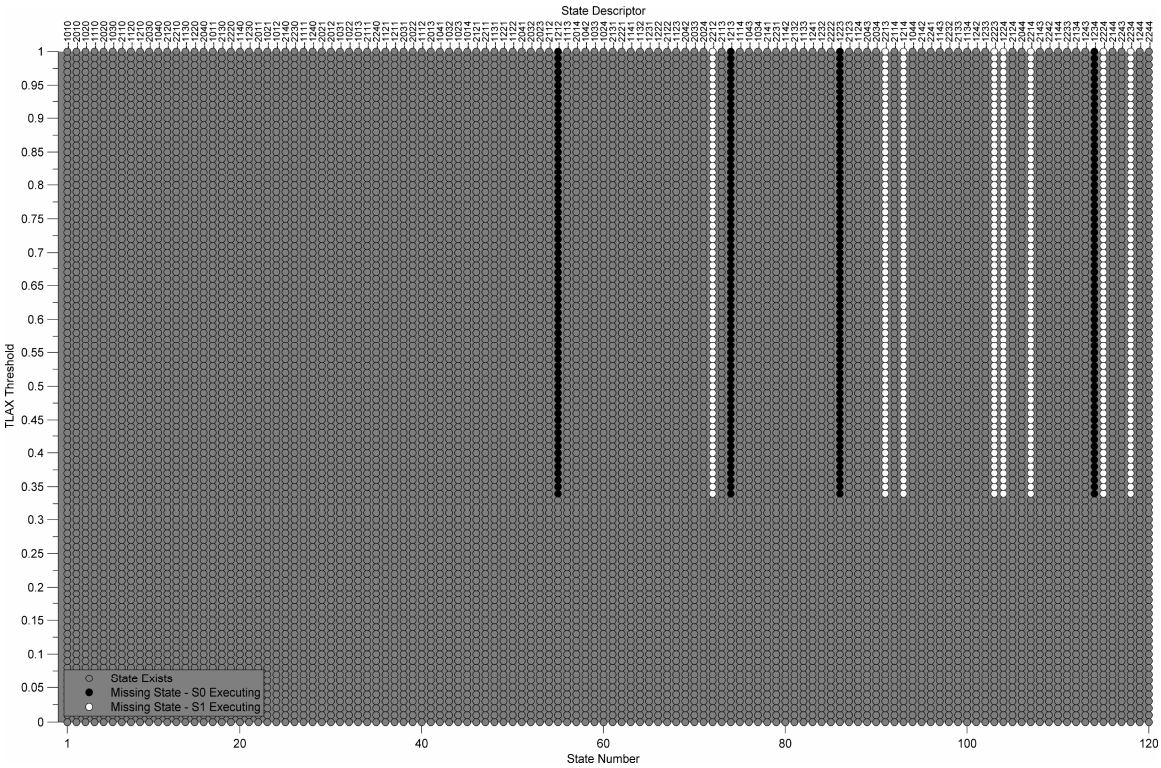

Figure 103: States missing from TLAX diagram for medium load

Figure 104 shows the missing state information for TLAX assuming heavy load conditions. Again, the exact same groups of missing states appear, but in this figure, the missing states appear for all threshold values except zero. Therefore, under heavy load, TLAX gives up on the same tasks regardless of the threshold value.

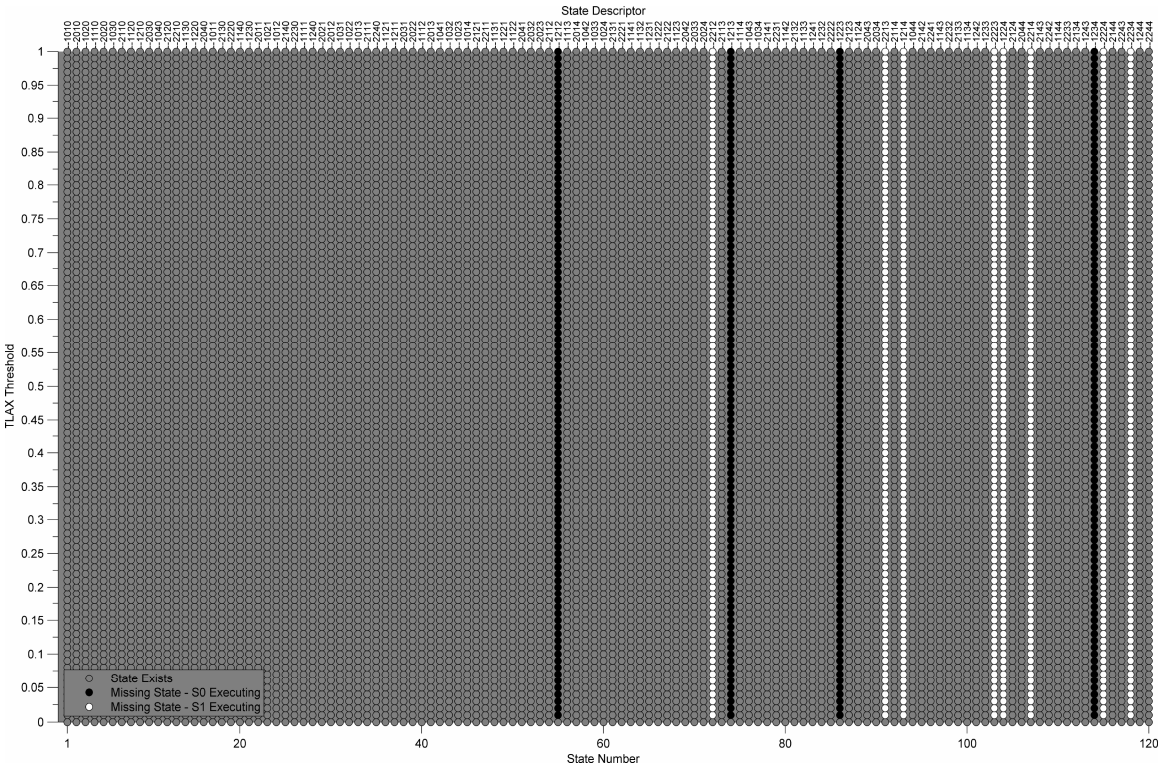

Figure 104: States missing from TLAX diagram for heavy load

After observing the missing state information for light, medium, and heavy load conditions, a correlation can be seen between the missing states and the threshold values for which maximum performance is achieved. For example, from Figure 98 it is seen that the last jump in performance (to the maximum value) occurs for a threshold value of 0.68. From Figure 102, the missing states are first eliminated from the state diagram for the same threshold value, 0.68. The same pattern is observed when comparing the results for medium and heavy loads. That is, the maximum performance jump and the point at which missing states are first eliminated are the same.

As observed previously, the specific TLAX threshold value has no effect, in this example, on the performance under heavy load conditions and yet TLAX still outperforms EDF significantly. Thus, the important thing is that there *is* a positive threshold value. That is, these results suggest that the task classification technique (i.e., a

threshold-based laxity approach) used by TLAX is also a crucial factor in its performance under heavy load. However, it should be noted that in a two-task example, the combinations of task groups is limited. Examination of additional tasks, across a broader spectrum of parameter settings, is an area of future research. The research presented here is an initial step that illustrates the importance and potential for threshold-based scheduling algorithms.

8.5.5 Comparing the State-Space Models

Another comparison method consists of contrasting the entire state-space models of EDF and TLAX. This type of theoretical analysis would not be practical for larger systems due to the large number of states. However, for two task streams, the state-space model can be summarized in a relatively compact representation. This analysis method is convenient because EDF has the same compact representation regardless of the intensity factor. Similarly, the representation corresponding to the best TLAX state diagram is the same, regardless of the intensity factor. The complete state diagrams are, of course, different because the values of arc weights change as a result of changing the intensity factor. The scheduling decision made in any given state, however, does not change as a result of changing the intensity factor. This is because the intensity factor simply scales the arrival rates, proportionally, and therefore, the decision made by a particular algorithm (e.g., EDF) in a given state (e.g., 2011) is the same under light, medium, and heavy loads. That is, the likelihood (i.e., steady state probability) of being in any particular state changes based on the intensity factor, but the decision of which task is allocated the processor in that state does not. Because some of the arc weights associated with these states change, the resulting performance metrics vary.

Figures 105 and 106 illustrate such a compact representation of the state-space diagrams for both EDF and TLAX. To make the state-space information easier to examine, the state diagrams are broken into two parts, where Figure 105 illustrates states 1 through 60 and Figure 106 illustrates states 60 through 120. For TLAX, a threshold value of 0.70 is used because this value results in the best performance regardless of the load conditions. In the figures, an idle state (i.e., one in which no task is executing) is represented by a grey circle, whereas a state missing from the corresponding diagram is represented by an X. States where either an S0 or an S1 task (but not both simultaneously) is present is represented as before, by either a black circle or a white circle, respectively. Note that in some states in the EDF diagram, both an S0 and S1 task are executing simultaneously, via processor sharing. In the given example, no two reserve laxities are ever equal and therefore, TLAX never uses PS—this issue is further addressed in the next section.

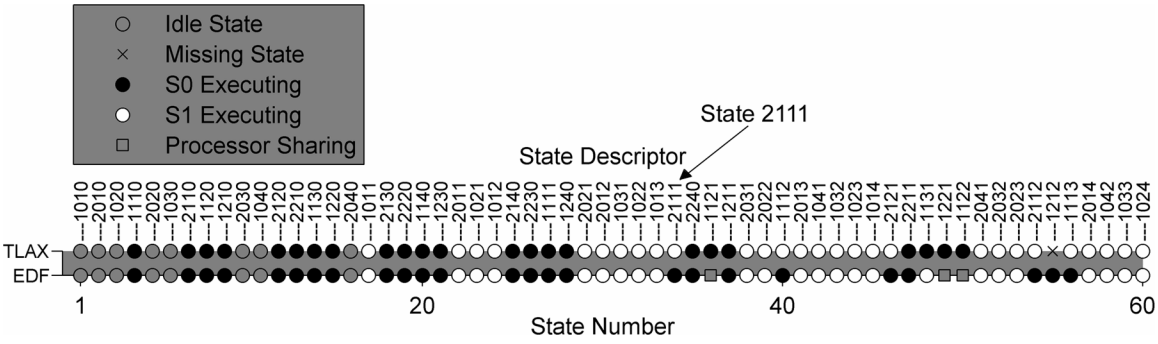

Figure 105: Comparison of state diagrams (states 1-60) for EDF and TLAX

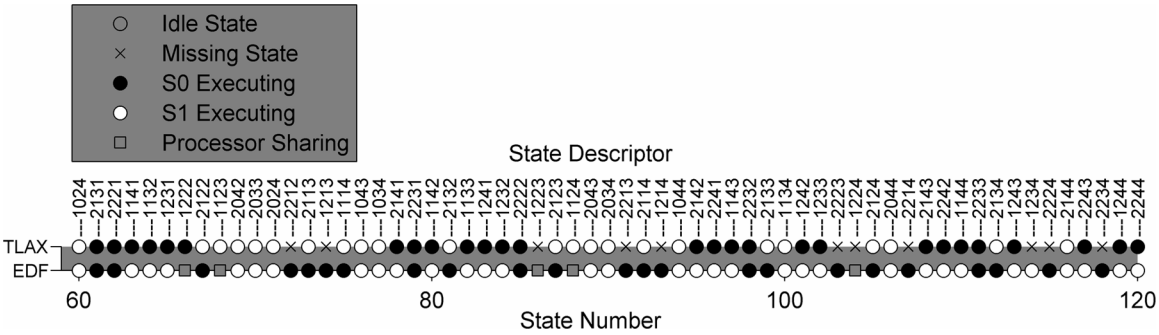

Figure 106: Comparison of state diagrams (states 60-120) for EDF and TLAX

In Figures 105 and 106, if EDF and TLAX make the same scheduling decision, their respective states are shaded the same color. Thus, it is seen that EDF and TLAX both make the same scheduling decisions in all states to the left of state 2111. In all such states (except state 1111) the service process of one or both of the task streams is inactive and therefore, no scheduling decision is necessary. (In state 1111, EDF and TLAX both choose the S0 task to execute.) For the remainder of the states, there are several similarities as well as several differences. For example, in states such as 2240 and 1211, EDF and TLAX both choose the S0 task to execute. In states such as 2013 and 1041, both algorithms choose the S1 task to execute. From the figures, it can be seen that there are more states for which EDF and TLAX both select the S1 task (i.e., matching white dots), as opposed to the states where both algorithms select the S0 task (i.e., matching black dots). There are also numerous differences among the decisions made by each algorithm.

Figure 107 presents a simplified version of the previous figures that shows only those states in which the EDF and TLAX scheduling decisions are different. Perhaps the most interesting difference corresponds to the 12 states missing from the TLAX state-space diagrams, indicated by an X in the figure. The missing TLAX states 1223 and 1224 (i.e.,

state numbers 86 and 104, respectively) correspond to EDF states in which PS is used. Similarly, the TLAX state 1234 (state number 114) corresponds to a state in which EDF selects the S1 task to execute. For all nine of the remaining missing states, EDF chooses the S0 task to execute. However, in the same set of missing states, TLAX would choose the S0 task in only two of those states—1212 and 1213. (See Figure 102, for example.) In the seven other states, TLAX would instead choose the S1 task to execute.

For the majority of the missing states, TLAX has already given up on S1, whereas in the same corresponding states, EDF has not given up. Instead, EDF allocates the processor to the task stream (i.e., S1) TLAX has already given up on. Therefore, most of the missing states correspond to situations in which the S1 task has more reserve laxity (according to TLAX) than S0, but an earlier deadline (according to EDF) than S0. This perhaps suggests that EDF negatively affects its own performance by allowing the S1 task to execute in these states. (Recall that the S1 stream imposes more relative load on the system, compared to S0—see Table 34.) Results such as these help compare and contrast the scheduling decisions made by EDF and TLAX, and illustrate the importance of analytical, as well as simulation, approaches.

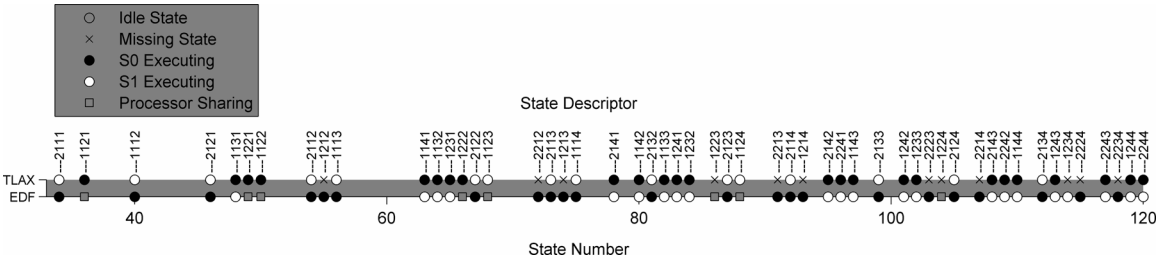

Figure 107: Differences in state diagrams for EDF and TLAX

8.5.6 The Significance of Processor Sharing

Another obvious difference among the state diagrams is the presence of states in which EDF uses processor sharing. In the MOSS implementation of scheduling algorithms, ties are broken by using processor sharing (PS) and to be consistent, MSAT is implemented in the same manner. In order to determine the significance of using PS for ties, MSAT is modified to test the effect of two other tie-breaking strategies for EDF. Note that tie breaking is not an issue for TLAX in this example because there is never a tie in the reserve laxity values of tasks. For EDF, the state-space model is reconstructed assuming that in the case of a tie, priority is always given to the S0 task. This technique is then repeated assuming that ties are broken by instead always giving priority to the S1 task. Table 40 shows a summary of the performance effect for EDF when using different tie-breaking rules.

Table 40: Performance effect of changing the EDF tie-breaking rule

	Overall % Met Deadlines		
	Stream Given Priority in Case of Equal Deadlines		
System Load	S0	S1	Both (PS)
Light (IF=0.5)	83.52%	83.33%	83.39%
Medium (IF=1.0)	50.96%	50.42%	50.49%
Heavy (IF=1.5)	30.43%	29.39%	29.65%

From Table 40, it is seen that using a different tie-breaking rule has minimal effect on the performance of EDF, when compared to the met deadline percentages obtained using PS. Considering that TLAX achieves a relative performance increase over EDF that ranges from 17% to 31% (depending on the number of stages), the fact that EDF was

tested using PS for tie breaking is negligible. In addition, tasks in real-world systems rarely have the exact same scheduling criterion (e.g., deadline) and therefore, this seems like a minor implementation artifact.

8.6 Chapter Summary

In this chapter, a mathematical and theoretical approach is used to analytically compare and contrast the performance of EDF and TLAX. Because the output from MOSS and other simulators is subject to jitter, the MSAT tool is used to analytically solve state-space models describing real-time systems based on the method of stages modeling approach. A small workload is used in this chapter to keep the analysis tractable. The study serves as a proof of concept and an initial step towards broader theoretical analysis.

Using MSAT, state-space models based on the sample workload are constructed and solved analytically. MSAT outputs compact forms of state-space diagrams and calculates desired performance metrics exactly. The state-space models are used to compare and contrast both EDF and TLAX scheduling decisions. It is observed that while TLAX gives up on, and sacrifices, tasks after a certain threshold value is reached, EDF never gives up on tasks. The sacrifices made by TLAX correspond to missing states in the state diagrams and help gain insight into the differences between EDF and TLAX. The task classification technique (i.e., a threshold-based laxity approach) used by TLAX is a unique and important criterion in making scheduling decisions and warrants future investigation.

The analytical results obtained from MSAT validate the MOSS findings presented earlier in Chapter VII and confirm that TLAX outperforms EDF under heavy load conditions. Based on graphs presented early in the chapter, a Most Laxity First (MLF)

scheduling policy appears to be a simple and viable scheduling strategy, but with further analysis, it is demonstrated that MLF fails to perform well (consistently) under varying workloads. This observation emphasizes the importance of further exploration and demonstrates that interpreting results involving variability studies can be difficult.

8.7 Research Contributions

The contributions presented in this chapter include:

- Development, description, and demonstration of the Matlab State-space Analysis Tool (MSAT)
- Description and demonstration of the compact form of state-space representation used in MSAT
- State-space analysis of the TLAX algorithm
- Analytical validation of the TLAX findings obtained from MOSS

CHAPTER IX

CONCLUSION

9.1 Motivation Revisited

As performance analysis and capacity planning of distributed and real-time systems become more pervasive, the effects of variability within the workloads to which these systems are subjected play an increasingly important role in system performance. In the past, the effects of variance have typically been minimized or ignored in performance modeling, especially in the area of task scheduling algorithms. By studying the effects of variance on performance parameters such as inter-arrival times, service times, and deadline times, the results can be incorporated into new hybrid scheduling strategies that are either immune to workload variability, or adapt their performance in light of it. In particular, new hybrid scheduling techniques that utilize threshold-based laxity approaches and key insights gained from state-based performance analysis show promising results for future work. This work emphasizes the importance of explicit incorporation of variance in performance modeling, particularly in the field of real-time task scheduling algorithms.

9.2 Summary of Results

In performance modeling, the mean of workload parameters often receives much of the focus of research with little or no attention being given to the higher moments of parameters. As seen in the case study detailed in Chapter III, matching only the mean is

often insufficient in developing an accurate performance model. However, even after a performance model is validated and higher-moment matches are achieved for performance parameters, the effects of variability can still have significant effects on system performance. Thus, it is important and beneficial to study the effects of higher moments (e.g., variance) in performance modeling and evaluation.

The MOSS simulator presented in Chapter IV is useful in conducting various sensitivity analysis experiments to determine the effects of variability on workload parameters. In doing so, many interesting results have been presented. However, these results represent only the initial steps in the exploration of the higher moment effects in performance modeling. Through simulation and analytical techniques, the role of parameter variance in evaluating the performance of systems can be further investigated.

Variability is inherently found in most real-time systems and it can be introduced by the workload itself, hardware failures, and uncontrollable environmental factors. It is shown that by incorporating the effects of variability directly into scheduling routines (Chapters VI and VII), the performance of scheduling algorithms can be improved. For example, the TLAX algorithm can outperform EDF by as much as 50% in heavy load conditions. Further, TLAX is much less susceptible to changes in the workload variability when compared to traditional scheduling algorithms. These results can guide the development of scheduling algorithms that maintain consistent performance regardless of system load or changes in workload variability.

The method of stages framework, MOSS, and MSAT are all powerful modeling tools that can be further exploited to provide insights into the effects of workload variability on system performance. In the last year, it has been demonstrated that variable behavior

found in task workloads is often self-similar and can exhibit irregularity on many different scales [30]. This type of study helps to revive interest in topics such as workload characterization and analysis, evaluation of scheduling algorithms, and performance improvement techniques. We hope this work will help guide future research efforts and motivate the explicit incorporation of variance in performance modeling.

9.3 Summary of Research Contributions

The main contributions of this work are summarized as follows:

1. Case Study of an Enterprise Grid Environment

   We developed a performance model of a large grid environment using Colored Petri Nets. The difficulty encountered in validating the performance model led to the novel development of a three-step job generator that accurately achieves a two-moment match for performance parameters. Using capacity-planning scenarios, we demonstrate that the validated model is a good tool for predicting the effects of changes made to the real-world system. It is observed that changing the variance of a single workload parameter significantly affects the estimated system performance. This motivated the further study of the effects of variability on system performance and led us to the method of stages modeling technique.

2. Uniform Sensitivity Analysis Experiments

   We adopt the method of stages modeling technique and apply it to performance analysis and evaluation in soft real-time systems. This technique provides a flexible and powerful modeling framework that allows the first two moments of performance parameters to be matched. In addition, the workload variance can be systematically controlled and investigated in a uniform manner, allowing the effects of variance to

be methodically investigated. In this work, several interesting observations are made and heuristics (see Section 9.4) are developed to summarize many of the important findings from our work. The basic framework of the method of stages, where real-world processes are modeled by a series of discrete stages, can be applied to other areas of research outside of distributed or real-time systems. For example, warehouse management systems, supermarket distribution mechanisms, and even parking lot customer behavior all contain variability that can be modeled by discrete processes using the method of stages technique.

3. Method of Stages Simulator

We develop a simulation tool called MOSS (Method Of Stages Simulator) that uses the method of stages to help analyze the effects of variance in real-time scheduling. MOSS includes many advanced features, such as a setup and installation program, extensive help documentation, an intuitive graphical user interface, and a built-in configuration file editor. Future enhancements to MOSS have been explored via a discussion of the enhanced prototype currently in development. MOSS achieves a two-moment match for the inter-arrival times, service times, and deadline times of each task stream comprising a workload. By changing a single parameter, the number of stages, the user can model various distributions and simulate conditions ranging from soft to hard real-time environments. Using MOSS to study the effects of variance on existing scheduling algorithms, as well as to discover new state-based algorithms, has produced new and interesting results. In particular, the development and testing of the TLAX algorithm has proven it a robust algorithm that outperforms the traditional scheduling algorithms.

4. The TLAX Algorithm

With the help of MOSS, we develop the XLAX scheduling technique and compare its performance to that of traditional scheduling algorithms. With further inspection, we propose the TLAX algorithm, which is simpler to describe and easier to implement. Through sensitivity analysis studies conducted using MOSS, we show experimentally that TLAX can outperform the traditional scheduling algorithms, particularly under heavy load conditions. These results help demonstrate the importance of explicitly considering the variance in the development of performance models.

5. Matlab State-Space Analysis Tool

In order to help analytically validate results obtained from MOSS, we develop the Matlab State-space Analysis Tool (MSAT) that constructs and solves small state-space models describing real-time environments. A simple, but novel, representation of state-space diagrams is used to output compact state-space information. Specifically, the tool can be used to construct a complete state diagram for any model based on the method of stages technique, where two task streams are allowed. A number of different scheduling algorithms can be evaluated using this tool, and the models are solved analytically to obtain exact (theoretical) performance metrics. The design of MSAT provides great flexibility and makes it easy to add and test new scheduling algorithms, compute additional performance metrics, and output new summary information such as charts, graphs, and spreadsheets.

6. Analytical Validation

Using MSAT, we analytically validate several of the results obtained from MOSS and demonstrate the feasibility and accuracy of using such modeling tools. The analytical

validation strengthens our confidence in the MOSS tool and we hope this and similar tools will be used to investigate the effects of variance in the future.

9.4 Observations and Heuristics

An interesting and applicable result from our work is that the changes in variability alone can significantly affect system performance, and these changes are load-dependent. That is, depending on the system utilization, variability can both positively and negatively impact the overall system performance. By studying the results from sensitivity analysis experiments, it is seen that the variance of task parameters has about a 15% performance impact on the percentage of met deadlines for systems under light load. In general, increased variability degrades performance in light load situations.

As the system load increases, the average laxity of tasks naturally decreases. However, the optimal amount of reserve laxity is not zero, as might be expected. Instead, our work suggests that it is important to use a nonzero threshold value for laxity-based scheduling routines. The performance of the TLAX algorithm suggests that the best threshold value is about 50%. Further, the amount of laxity to hold in reserve increases as the system utilization increases. Maintaining about 50% laxity under heavy loads allows TLAX to outperform traditional scheduling algorithms, and it is also robust in that it is insensitive to changes in workload variability.

When the system utilization is high, increased workload variability actually boosts system performance. As this trend is opposite that seen under light load, the system utilization is a key criterion in monitoring and improving system performance of scheduling algorithms. Ideally, the intensity of workloads a system is subjected to would be controlled in order to maximize performance, but this option is typically not feasible

in practice. However, our work shows that workload intensity factors correspond to ranges of overall system utilization levels. Thus, for any given scheduling algorithm, there exists a preferred target utilization range that not only minimizes the effects of variance, but also maximizes the overall system performance. These utilization ranges can be determined experimentally a priori by analysis tools such as MOSS.

9.5 Future Work

In this current work, we focus on workloads composed of tasks whose deadlines coincide with their next arrivals. That is, we have explored only workloads for which the deadline of a task is its next arrival. A deadline can also be modeled by a distinct deadline process that is characterized by stage-type distribution parameters that are either the same or different from those of the arrival process. Thus, the role a unique deadline process plays in task behavior can be explored, as well as its effect on the overall system performance. MOSS already supports this feature and further investigation of the deadline process may lead to additional insights in task scheduling.

The effects of variability using distributions other than Erlang-k distributions (e.g., Coxian distributions) can be explored as well. The next version of MOSS is already in development and this feature, among others, will be supported. This will increase the usefulness of MOSS and allow the study of a wider range of realistic workloads while maintaining the systematic approach (i.e., method of stages) to modeling the effects of variance.

Also of importance is investigating the significance of the third, fourth, and higher moments of performance parameters to determine their impact on system performance and compare it to the results related to variance. In fields such as civil and environmental

engineering, the position and shape of strength distribution curves have been shown to play an important role in predicting structural reliability [19]. Just as the variance of parameters is important, the less investigated higher moments that further determine the shape (e.g., skewness, kurtosis) of distributions may also play an important role in evaluating and predicting system performance. Tools such as MOSS can help make the important initial steps needed in order to expand such research efforts in real-time studies and other related fields.

We have provided validation of the MOSS findings based on the analytical MSAT tool. In future work, an exhaustive validation of the MOSS simulator should be done, in the hopes of making the use of MOSS more widespread, as well as motivating the development of other modeling tools based on the method of stages framework.

Other possible topics for future work include multi-processor systems, development of additional state-based scheduling algorithms, conducting sensitivity analysis on additional scheduling algorithms (e.g., group EDF [56] and fuzzy LLF [29]), and experimentally evaluating the performance of TLAX and similar state-based algorithms in real-world environments.

LIST OF PUBLICATIONS

Textbook Chapters

1. D. Hoffman, A. Apon, L. Dowdy, B. Lu, N. Hamm, L. Ngo, and H. Bui: Chapter 9 of the textbook:

   Y. Chan, J. Talburt, and T. Talley. *Data Engineering: Mining, Information and Intelligence*, Springer, 2009.

Conference Papers

1. N. Hamm and L. Dowdy: The Method Of Stages Simulator (MOSS) for Modeling System Variance in Soft Real-Time Systems. *IEEE International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (2010)
2. N. Hamm, and L. Dowdy: Performance Modeling of Grid Systems Using High-Level Petri Nets. *The 7th Annual ALAR Conference on Applied Research in Information Technology,* (2008)
3. N. Roy, N. Hamm, M. Madhukar, L. Dowdy, and D. Schmidt: The Impact of Variability on Soft Real-Time System Scheduling. *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009.
4. N. Roy, A. Dabholkar, N. Hamm, L. Dowdy, and D. Schmidt: Modeling Software Contention Using Colored Petri Nets. *The 16th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS),* (2008)

5. L. Ngo, B. Lu, H. Bui, A. Apon, N. Hamm, L. Dowdy, et al.: Application of Empirical Mode Decomposition to the Arrival Time Characterization of a Parallel Batch System Using System Logs. *The 2009 Conference on Modeling, Simulation, and Visualization Methods,* (2009)
6. B. Lu, L. Ngo, H. Bui, A. Apon, N. Hamm, L. Dowdy, et al.: Capacity Planning of a Commodity Cluster in an Academic Environment: A Case Study. *The 9th LCI International Conference on High-Performance Clustered Computing,* (2008)
7. B. Lu, L. Ngo, H. Bui, A. Apon, N. Hamm, L. Dowdy, et al.: Capacity Planning of Supercomputing Resources. *The 7th Annual ALAR Conference on Applied Research in Information Technology,* (2008)
8. B. Lu, L. Ngo, H. Bui, A. Apon, N. Hamm, L. Dowdy, et al.: Workload Characterization Using Empirical Mode Decomposition. *The 7th Annual ALAR Conference on Applied Research in Information Technology*, (2008)

REFERENCES

[1] *Standard Performance Evaluation Corporation SPECjAppServer2001 Documentation*. 2010; Available from: http://www.spec.org/psg/jAppServer/.

[2] L. Abeni and G. Buttazzo, *Integrating Multimedia Applications in Hard Real-Time Systems.* Proceedings of the 19th IEEE Real-Time Systems Symposium, 1998.

[3] L. Abeni and G. Buttazzo, *Resource Reservations in Dynamic Real-Time Systems.* Real-Time Systems, 2004. **27**(2): p. 123 - 165.

[4] R.B. Abernethy, *The New Weibull Handbook*. 1996, New York: Gulf Publishing Company.

[5] T. Agerwala and M. Flynn, *Comments on Capabilities, Limitations, and 'Correctness' of Petri Nets.* Proceedings of the 1st Annual Symposium on Computer Architecture, 1973: p. 81 - 86.

[6] A.K. Atlas and A. Bestavros, *Statistical Rate Monotonic Scheduling.* Proceedings of the 19th IEEE Real-Time Systems Symposium, 1998: p. 123 - 132.

[7] S. Baarir, M. Beccuti, D. Cerotti, et al., *The GreatSPN Tool: Recent Enhancements.* ACM SIGMETRICS Performance Evaluation Review, 2009. **36**(4): p. 4-9.

[8] F. Baccelli, G. Balbo, R. Boucherie, et al., *Annotated Bibliography on Stochastic Petri Nets.* Performance Evaluation of Parallel and Distributed Systems - Solution Methods, 1994.

[9] T.P. Baker, *Stack-Based Scheduling of Real-Time Processes.* Real-Time Systems, 1991. **3**(1): p. 67 - 100.

[10] F. Bause and P.S. Kritzinger, *Stochastic Petri Nets: An Introduction to the Theory*. 2 ed. 2002: Springer Verlag. 204 - 227.

[11] M. Bertoli, G. Casale and G. Serazzi. *User-Friendly Approach to Capacity Planning Studies With Java Modelling Tools*. in *International Conference on Simulation Tools and Techniques*. 2009.

[12] S. Cheng, J.A. Stankovic and K. Ramamritham, *Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey.* Hard Real-Time Systems: A Tutorial, 1988: p. 150 - 173.

[13] S. Christensen, L. Kristensen, K. Mortensen, et al., *Capacity Planning of Web Servers Using Timed Hierarchical Couloured Petri Nets.* Proceedings of HP-OVUA, 1999.

[14] F. Cottet and J. Delacroix, *Scheduling in Real-Time Systems*. 2002: Wiley.

[15] D.R. Cox, *Some Statistical Methods Connected With Series of Events.* J. R. Statist Soc., 1955: p. 129 - 164.

[16] M.L. Dertouzos, *Control Robotics: the Procedural Control of Physical Processes.* Information Processing, 1974.

[17] M.L. Dertouzos and A.K.L. Mok, *Multiprocessor On-Line Scheduling of Hard Real-Time Tasks.* IEEE Transactions on Software Engineering, 1989. **15**(12): p. 1497 - 1506.

[18] P. Fowler and L. Levine, *Technology Transition Push: A Case Study of Rate Monotonic Analysis*, in *Technical Report CMU/SEI-93-TR-29*. 1993.

[19] S.W. Freiman and C.M. Hudson, *Methods for Assessing the Structural Reliability of Brittle Materials*. 1984: Library of Congress.

[20] S. Gaonkar, K. Keefe, R. Lamprecht, et al., *Performance and Dependability Modeling With Mobius.* ACM SIGMETRICS Performance Evaluation Review, 2009. **36**(4): p. 16-21.

[21] M. Hack, *The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri Nets and Vector Addition Systems.* FOCS, 1974: p. 156 - 164.

[22] M.T. Hack, *Decidability Questions for Petri Nets.* Ph.D. Thesis, M.I.T., 1976.

[23] N. Hamm. *MOSS*. 2009; Available from: http://www.vuse.vanderbilt.edu/~hamm/MOSS.

[24] N. Hamm. *Petri Net Model*. 2009; Available from: http://www.vuse.vanderbilt.edu/~hamm/PNM.

[25] N. Hamm and L. Dowdy. *The Method of Stages Simulator for Modeling Variance in Soft Real-Time Systems*. in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. 2010. Reprinted with the Permission of the Society for Modeling and Simulation International (SCS).

[26] N. Hamm and L. Dowdy. *Performance Modeling of Grid Systems Using High-Level Petri Nets*. in *The 7th Annual ALAR Conference on Applied Research in*

Information Technology. 2008. Reprinted with the Permission of Acxiom Corporation.

[27] M.G. Harbour, M.H. Klein and J.P. Lehoczky, *Fixed Priority Scheduling of Periodic Tasks With Varying Execution Priority.* Proceedings of the 12th IEEE Real-Time Systems Symposium, 1991.

[28] P.K. Harter, *Response Times in Level-Structured Systems.* ACM Transactions on Computer Systems, 1987. **5**(3): p. 232 - 248.

[29] X. He, *An Improved LLF Scheduling Algorithm Based on Fuzzy Inference in the Uncertain Environments.* IEEE International Conference on Computer Science and Information Technology, 2010: p. 211 - 215.

[30] E. Hernandez-Orallo and J. Vila-Carbo, *Analysis of Self-Similar Workload on Real-Time Systems.* Real-Time and Embedded Technology and Applications Symposium, 2010: p. 343-352.

[31] A. Horvath and M. Telek, *Matching More Than Three Moments with Acyclic Phase Type Distributions.* Stochastic Models, 2007. **23**(2): p. 167-194.

[32] M.D. Jeng, *Modular Synthesis of Petri Nets for Modeling Flexibile Manufacturing Systems.* International Journal of Flexibile Manufacturing Systems, 1995. **7**(3).

[33] K. Jensen, *Analysis Methods*. Coloured Petri Nets. Basic Concepts, Analysis Methods, and Practical Use. Vol. 2. 1994: Springer-Verlag.

[34] K. Jensen, *Basic Concepts*. Coloured Petri Nets. Basic Concepts, Analysis Methods, and Practical Use. Vol. 1. 1992: Springer-Verlag.

[35] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*. Monographs in Theoretical Computer Science. 2003: Springer.

[36] K. Jensen, *An Introduction to the Theoretical Aspects of Coloured Petri Nets*. Lecture Notes in Computer Science. Vol. 803. 1994: Springer-Verlag.

[37] K. Jensen, *Practical Use*. Coloured Petri Nets. Basic Concepts, Analysis Methods, and Practical Use. Vol. 3. 1997: Springer-Verlag.

[38] M.A. Johnson and M.R. Taaffe, *Matching Means to Phase Distributions: Mixtures of Erlang Distributions of Common Order.* Stochastic Models, 1990. **5**(4): p. 141-149.

[39] M.A. Johnson and M.R. Taaffe, *Matching Moments to Phase Distributions: Nonlinear Programming Approaches* Stochastic Models, 1990. **6**(2): p. 259-281.

[40] N.L. Johnson and N. Balakrishnan, *Advances in the Theory and Practices of Statistics*. 1997, Ontario: Wiley & Sons.

[41] K. K Jeffay, *Scheduling Sporadic Tasks With Shared Resources in Hard Real-time systems.* Proceedings of the 13th IEEE Real-Time Systems Symposium, 1992.

[42] R.M. Karp and R.E. Miller, *Parallel Program Schemata.* Journal of Computer and System Sciences, 1969. **3**: p. 147 - 195.

[43] A.A.P. Kazem, H. Seifzadeh, M. Kargahi, et al., *Maximizing the Accrued Utility of an Isochronal Soft Real-Time System Using Genetic Algorithms.* Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science 2009: p. 65 - 69.

[44] L. Kleinrock, *Queuing Systems, Volume I: Theory*. 1975: Wiley-Interscience.

[45] E. Knightly, *Second Moment Resource Allocation in Multi-Service Networks.* Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems 1997: p. 181-191.

[46] G. Koren and D. Shasha, *Skip-Over: Algorithms and Complexity for Overloaded Systems That Allow Skips.* Proceedings of the IEEE Real Time System Symposium, 1995: p. 110 - 117.

[47] S.R. Kosaraju, *Decidability of Reachability in Vector Addition Systems.* Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982: p. 267 - 281.

[48] S. Kounev, *Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queuing Petri Nets.* IEEE Transactions on Software Engineering, 2006.

[49] S. Kounev, *QPME: A Performance Modeling Tool Based on Queueing Petri Nets.* ACM SIGMETRICS Performance Evaluation Review, 2009. **36**(4): p. 46-51.

[50] S. Kounev and A. Buchmann, *Performance Modelling of Distributed E-Business Applications Using Queuing Petri Nets.* Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software, 2003.

[51] S. Kounev, C. Dutz and A. Buchmann, *QPME - Queuing Petri Net Modeling Environment.* 3rd International IEEE Conference on the Quantitative Evaluation of Systems, 2006.

[52] J. Lehoczky, L. Sha and Y. Ding, *The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior.* IEEE Real-Time Systems Symposium, 1989: p. 166 - 171.

[53] J.P. Lehoczky, *Fixed Priority Scheduling of Periodic Task Sets With Arbitrary Deadlines.* Proceedings of the 11th IEEE Real-Time Systems Symposium, 1990: p. 201-209.

[54] J.P. Lehoczky, *Real-Time Queueing Theory.* Proceedings of IEEE Real-Time Systems Symposium, 1996: p. 186 - 195.

[55] J.P. Lehoczky, L. Sha and J.K. Stronsnider, *Enhanced Aperiodic Responsiveness in a Hard Real-Time Environment.* Proceedings of the 8th IEEE Real-Time Systems Symposium, 1987: p. 261 - 270.

[56] W. Li, K. Kavi and R. Akl, *A Non-Preemptive Scheduling Algorithm for Soft Real-Time Systems.* Computers and Electrical Engineering, 2007. **33**(1): p. 12 - 29.

[57] R.J. Lipton, *The Reachability Problem Requires Exponential Space.* Information Processing, 1976.

[58] C.L. Liu and J.W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* Journal of the ACM, 1973. **20**: p. 46 - 61.

[59] M. Marsan, A. Bobbio and S. Donatelli, *Petri Nets in Performance Analysis: An Introduction*. Lecture Notes In Computer Science. 1998: Springer Berlin.

[60] M. Marsan and G. Conte, *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems.* ACM Transactions on Computer Systems, 1984. **2**(2): p. 93-122.

[61] E.W. Mayr, *An Algorithm for the General Petri Net Reachability Problem.* SIAM Journal of Computing, 1984. **13**(3): p. 441 - 460.

[62] D. Menasce, V. Almeida and L. Dowdy, *Capacity Planning and Performance Modeling*. 1994: Prentice Hall.

[63] D. Menasce, V. Almeida and L. Dowdy, *Performance by Design - Computer Capacity Planning by Example*. 2004, Upper Saddle River, NJ: Prentice Hall.

[64] A.K. Mok, *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA., 1983.

[65] T. Murata, *Petri Nets: Properties, Analysis, and Applications*. IEEE, 1989. **77**(4): p. 541-580.

[66] N. Navet and S. Merz, *Modeling and Verification of Real-Time Systems*. 2008: Wiley.

[67] M.F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. 1995: Dover Publications.

[68] S. Peng, X. Liu, Q. Dai, et al., *An Improved RM Algorithm for Preventing Streaming Media Tasks from Starvation.* International Conference on Multimedia and Expo, 2003: p. 589 - 592.

[69] J. Peterson, *Petri Net Theory and the Modeling of Systems*. 1981, Englewood Cliffs: Prentice Hall, Inc.

[70] C.A. Petri, *Communication With Automata*, in *Technical Report RADC-TR-65-377*. 1966: New York.

[71] C.A. Petri and W. Reisig, *Petri net*, in *Scholarpedia*. 2008.

[72] C. Rackoff, *The Covering and Boundedness Problem for Vector Addition Systems.* Theoretical Computer Science, 1978. **6**: p. 223 - 231.

[73] R. Rajkumar, L. Sha and J.P. Lehoczky, *Real-Time Synchronization Protocols for Shared Memory Multiprocessors.* Proceedings of the 10th International Conference on Distributed Computing, 1990: p. 116 - 125.

[74] R.L. Rajkumar, L. Sha and J.P. Lehoczky, *Real-Time Synchronization Protocols for Multiprocessors.* Proceedings of the 9th IEEE Real-Time Systems Symposium, 1988: p. 259 - 269.

[75] A. Ratzer, L. Wells, H. Lassen, et al., *CPN Tools for Editing, Simulating, and Analyzing Coloured Petri Nets*. Lecture Notes In Computer Science. 2003.

[76] L.E. Rosier and H.C. Yen, *A Multiparameter Analysis of the Boundedness Problem for Vector Addition Systems.* Journal of Computer and System Sciences, 1986. **32**: p. 105 - 135.

[77] B. Schroer, G. Gaxiola, G. Mackulak, et al. *A Comparison of the Exponential and the Hyperexponential Distributions for Modeling Move Requests in a Semiconductor Fab*. in *Winter Simulation Conference*. 1999.

[78] L. Sha, J.P. Lehoczky and R. Rajkumar, *Task Scheduling in Distributed Real-Time Systems.* Proceedings of the IEEE Industrial Electronics Conference, 1987.

[79] M. Sinnema and S. Deelstra, *Classifying Variability Modeling Techniques.* Information and Software Technology, 2007. **49**: p. 717-739.

[80] D. Smarkusky, R. Ammar, I. Antonios, et al., *Hierarchical Performance Modeling for Distributed System Architectures.* IEEE International Symposium on Computers and Communications, 2000.

[81] B. Sprunt, L. Sha and J.P. Lehoczky, *Aperiodic Task Scheduling for Hard Real-Time Systems.* The Journal of Real-Time Systems, 1989: p. 27 - 60.

[82] M. Spuri and G. Buttazzo, *Efficient Aperiodic Service Under the Earliest Deadline Scheduling.* Proceedings of the IEEE Real-Time Systems Symposium, 1994.

[83] M. Spuri and G. Buttazzo, *Scheduling Aperiodic Tasks in Dynamic Priority Systems.* Real-Time Systems, 1996. **10**(2): p. 179 - 210.

[84] J.A. Stankovic, K. Ramamritham, M. Spuri, et al., *Deadline Scheduling for Real-Time Systems.* Boston-Dordrecht-London: Kluwer, 1998.

[85] W. Stewart, *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. 2009, Princeton, NJ: Princeton University Press.

[86] M. Westergaard. *CPNTools*. 2009; Available from: http://www.daimi.au.dk/CPNTools/.

[87] M. Zhou and F. Dicesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. The Springer International Series in Engineering and Computer Science. 1992: Springer.