

**RESPONSE TIME AND PARALLELISM IN PARALLEL
PROCESSING SYSTEMS WITH CERTAIN SYNCHRONIZATION
CONSTRAINTS**

Abdelfettah Belghith

**December 1986
CSD-860015**



UCLA-CSD-860015

December 1986

**RESPONSE TIME AND PARALLELISM IN PARALLEL PROCESSING
SYSTEMS WITH CERTAIN SYNCHRONIZATION CONSTRAINTS**

by

Abdelfettah Belghith


This research, conducted under the chairmanship of Professor
Leonard Kleinrock, was sponsored by the Defense Advanced
Research Projects Agency, Department of Defense.

**Computer Science Department
School of Engineering and Applied Science
University of California
Los Angeles**

The dissertation of Abdelfertah Belghith is approved.




Mario Gerla



Richard R. Muntz



Bruce Rothschild



R. Clay Sprows



Leonard Kleinrock, Committee Chairman

University of California, Los Angeles
1987

To my wife Sihem

Table of Contents

	page
SIGNATURE PAGE	ii
DEDICATION PAGE	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vii
ACKNOWLEDGEMENTS	x
VITA	xi
ABSTRACT	xiii
1 INTRODUCTION	1
1.1 Overview of Distributed and Parallel Processing Systems	1
1.2 Statement of the Problem and Preliminary Definitions	4
1.3 Previous Work in the Area	7
1.4 Contribution of this Dissertation	13
1.5 Multiprocessing Applications	16
1.5.1 Predictive Modeling and Simulations	16
1.5.2 Medical and Military Research	18
1.5.3 Energy Resource Exploration	19
2 NUMBER OF OCCUPIED PROCESSORS IN PARALLEL PROCESSING SYSTEMS	20
2.1 Model Description	21
2.2 Fixed Process Graphs	24
2.2.1 Distribution of the Number of Occupied Processors	24
2.2.2 Average Number of Occupied Processors	27
2.2.3 Variance of the Number of Occupied Processors	27
2.3 Semi Random Process Graphs with two or more Levels	32
2.3.1 Distribution of the Number of Occupied Processors	32
2.3.2 Average Number of Occupied Processors	35
2.3.3 Variance of the Number of Occupied Processors	39
2.4 Random Process Graphs	43
2.4.1 Distribution of the Number of Occupied Processors	43
2.4.2 Average Number of Occupied Processors	46
2.4.3 Variance of the Number of Occupied Processors	49
2.5 Conclusion	50
3 AVERAGE NUMBER OF OCCUPIED PROCESSORS THE GENERAL CASE	51
3.1 Infinite Number of Processors	52
3.1.1 The Task Interarrival Time Process	52
3.1.1.1 Average Interarrival Time Between Super Tasks	57
3.1.1.2 Variance of the Interarrival Time Between Super Tasks	58

3.1.2 Expected Number of Busy Processors	60
3.2 Finite Number of Processors	61
3.3 Conclusion	62
4 AVERAGE RESPONSE TIME IN PARALLEL PROCESSING SYSTEMS	66
4.1 Model Description	67
4.2 The Infinite Number of Processors Case	76
4.3 The Uniprocessor Case	80
4.3.1 Completeness of the 1-DPS-WF family of Scheduling Strategies	81
4.3.2 The Job Average Response Time	83
4.3.3 The Conservation Law	88
4.4 The P-DPS-WF Multiprocessing System with the Same Concurrency Degree for all Stages	90
4.5 Average Response Time in the P-DPS-WF System	96
4.5.1 Job Average Response Time Through an Empty P-DPS-WF System	96
4.5.2 Approximation of the Job Average Response Time	98
4.5.3 Achievable Parallelism in a P-DPS-WF System	100
4.6 Conclusion	112
5 PARALLEL PROCESSING WITH CERTAIN SYNCHRONIZATION CONSTRAINTS	113
5.1 Introduction and Previous Work	113-
5.2 Model Description	114
5.3 Bounds on the Response Time of a Bulk Job	120
5.4 Approximate Analysis Using M/G/1 Theory	124
5.5 Analysis of the Modified $M/G_c/1$ System	132
5.6 Analysis of the $M/G_c/1$ System Using a Load Adjustable Service Time Distribution	136
5.7 Analysis of the $M/G_c/1$ System Using Independent Start-up and Finishing-up Delays	139
5.7.1 Analysis of the Modified System with Start-up Periods	140
5.7.2 Analysis of the Modified System with Finishing-up Periods	147
5.7.3 Comparison and Application to the Study of the FJ-2-M/M/1 System	152
5.8 Approximate Analysis of the FJ-P-M/M/1 System	157
5.9 Conclusion	162
6 PARALLEL PROCESSING WITH SYNCHRONIZATION CONSTRAINTS AND JOB FEEDBACK	164
6.1 Model Description	166
6.2 Chain Shaped Stage Graphs	174
6.2.1 Conditional Expected Total Time Spent in the System for Endogenous Stages	175
6.2.2 Conditional Expected Total Time in System for Exogenous Stages	177
6.2.3 Conditional System States at Stage Feedback Times	178
6.2.3.1 Conditional System States at Completion Times of Endogenous Stages	179
6.2.3.2 Conditional System States at Completion Times of Exogenous Stages	180
6.2.4 System States at Service Completion Times	182

6.2.5	Steady State Equations for the Expected Total Time in System	182
6.2.6	Job Average Sojourn Time	185
6.3	Tree Shaped Stage Graph	188
6.3.1	Conditional Expected Total Time in System	189
6.3.2	Conditional System States at Stage Feedback Times	190
6.3.3	System States at Service Completion Times	192
6.3.4	Steady State Equations for the Expected Total Time in System	194
6.3.5	Generalized Conservation Law	196
6.3.6	Job Average Sojourn Time	198
6.4	Conclusion	202
7	CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK	204
Appendix A	207
Appendix B	209
Appendix C	215
References	217

List of Figures

page

2.1: Example of a Process Graph	22
2.2: Analysis of Fixed Process Graphs	25
2.3: Discrete Well Shaped Process Graph with Odd Number of Levels	30
2.4: Discrete Well Shaped Process Graph with Even Number of Levels	30
2.5: Analysis of Semi Random Process Graphs	32
2.6: Variance of the Number of Occupied Processors versus the Number of Levels; for Semi Random Process Graphs	42
2.7: Analysis of Random Process Graphs	44
3.1: Job Arrival and Task Arrival Time Diagram: (a) Process Graph Description, (b) Arrival Processes Time Diagram	53
3.2: Super-task Interarrival Time Diagram, case of $0 \leq t < \bar{X}$	55
3.3: Super-task Interarrival Time Diagram, case of $t > \bar{X}$	57
3.4: System Utilization and Average Number of Occupied Processors versus P , $\lambda \bar{N} \bar{X} < 1$	63
3.5: System Utilization and Average Number of Occupied Processors versus P , $\lambda \bar{N} \bar{X} \geq 1$	64
3.6: System Time and Average Number of Occupied Processors versus λ	65
4.1: Process Graph with Identity Set $\Omega = \{A, B, C, D, E, F, G\}$	69
4.2: The Execution Graph of the Process Graph of Figure 4.1	70
4.3: Process Graphs and their Corresponding Execution Graphs for $N=4$, (a) Process Graphs, (b) Execution Graphs	85
4.4: Process Graphs and their Corresponding Execution Graphs for $N=5$, (a) Process Graphs, (b) Execution Graphs	85
4.5: Average Response Time in a 1-DPS-WF System	87
4.6: Average Response Time in a P-DPS-WF System with Constant Concurrency Degree	97
4.7: Average Response Time in a P-DPS-WF System	101

4.8: Achievable Parallelism in a P-DPS-WF System	104
4.9: Achievable Parallelism versus the Number of Processors	105
4.10: Efficiency per Processor in a P-DPS-WF System	107
4.11: Power Function in a P-DPS-WF Multiprocessor System	109
4.12: Maximum Power in a P-DPS-WF System	110
4.13: Achievable Parallelism at Optimal Operating Points	111
5.1: A Representation of the P-Dimensional-Fork-Join System: (a) The P-Dimensional-Fork-Join System, (b) The Actual Process Graph in a P-Dimensional-Fork-Join System	116
5.2: The FJ-2-M/M/1 Parallel Processing System: (a) A Representation of the FJ-2-M/M/1 System, (b) A Service Period of the FJ-2-M/M/1 System	125
5.3: Approximation Using an $M/G_c/1$ Representation of the System	130
5.4: Discrepancy Between the Average Response Times Given by Formulae (5.9) and (5.28)	131
5.5: The Ratio of the Service Time Variances given by Equations (5.34) and (5.35)	134
5.6: The Average Response Time Using the Modified $M/G_c/1$ Analysis	135
5.7: Approximation Using a Load Adjustable Service Time Distribution	138
5.8: Behavior of the Unfinished Work in the System With and Without Start-up Delays: (a) Event Time Diagram for the Pure $M/G/1$ System, (b) Event Time Diagram for the Modified $M/G/1$ System	142
5.9: Behavior of the Unfinished Work in the System With and Without Finishing-up Delays: (a) Event Time Diagram for the Pure $M/G/1$ System, (b) Event Time Diagram for the Modified $M/G/1$ System	148
5.10: An Event Time Diagram for the System With Start-up Periods: (a) Idle System, (b) Busy System	153
5.11: An Event Time Diagram for the System With Finishing-up Periods: (a) Idle System, (b) Busy System	155
5.12: Representation of the FJ-2-M/M/1 System as Two Queues in Tandem	156
5.13: Approximation Using a Tandem Representation of the FJ-2-M/M/1 System	158
5.14: Validation of the Approximation of the FJ-P-M/M/1 System	161
6.1: Examples of Chain and Tree Shaped Stage Graphs	169

6.2: Computation of the Expected Sojourn Time in an Empty System for the Tree Shaped Stage Graph of Figure 6.1:(b)	171
6.3: Computation of the Expected Sojourn Time in an Empty System for the Tree Shaped Stage Graph of Figure 6.1:(c)	172
6.4: Expected Sojourn Time of the Chain Shaped Stage Graph of Figure 6.1:(a)	187
6.5: Expected Sojourn Time of the Tree Shaped Stage Graph of Figure 6.1:(b)	200
6.6: Expected Sojourn Time of the Tree Shaped Stage Graph of Figure 6.1:(c)	201
6.7: Comparison of the Expected Sojourn Times	203

ACKNOWLEDGEMENTS

I would like to express my appreciation to my doctoral committee consisting of Professors Leonard Kleinrock, Mario Gerla, Richard R. Muntz, Bruce Rothschild, and R. Clay Sprowls. I am especially grateful to my advisor and committee chairman Professor Leonard Kleinrock for his encouragement, his continued and enlightening guidance, and innumerable and stimulating discussions throughout the course of this dissertation work.

A debt of gratitude is owed to Dr. Mohamed Ben Ahmed, Professor at the Faculté des Sciences de Tunis, Tunisia, to Dr. Ali Hili, former Dean of the Faculté des Sciences de Tunis, and to Dr. Farouk Kamoun, Chairman of the board of the Center National d'Informatique (CNI), Tunis, for their help in making my program of studies at UCLA possible.

A debt of appreciation is owed to Dr. Mahmoud Triki, former director of the Scientific Mission of Tunisia, Washington, D.C., Mr. Radhouane Nouicer, present director of the Mission, and the rest of the personnel of the Mission, for their help and assistance in making my stay in the United States as pleasant as possible.

The support by the Advanced Research Projects Agency of the Department of Defense (Contract number MDA-903-77-C-0272) for this work is greatly appreciated, and I owe a debt of gratitude to the many employees of that contract. I would especially like to thank Lily Chien, Jodi Feiner, Saba Hunt, Lillian Larijani, Verra Morgan, and Ruth Pordy for their continued friendship and administrative assistance.

To the former ATS group members, Dr. Yehuda Afek, Dr. Richard Gail, Dr. Kenneth Kung, Dr. Hanoeh Levy, Dr. Randolph Nelson, and Dr. Hideaki Takagi, and to the present ATS members, Bob Felderman, Christopher Ferguson, Joseph Green, Jau-Hsiung Huang, and Willard Korfhage, I offer my sincere thanks for making the environment stimulating and exciting.

I dedicate this dissertation to Sihem, my wife, for her constant understanding, moral support and encouragement. Life would not be the same without her true love and joyful playfulness. My last and greatest gratitude goes to my parents, Salah and Naima Belghith, for their constant encouragement through the years, and their everlasting love.

VITA

January 10, 1954	Born in Sfax, Tunisia
June 1975	University of Tunisia, Faculté des Sciences de Tunis Diplome Universitaire des Etudes Supérieures (DUES)
June 1979	University of Tunisia, Faculté des Sciences de Tunis, and Institut National de Recherche en Informatique et Automatique, France Diplome D'Ingenieur Principal (Engineering Main Diploma) in Computer Science
June 1979	Awarded Presidential Prize (ranked first in Computer Science throughout Tunisia)
October 1979	University of Tunisia, Faculté des Sciences de Tunis Diplome Universitaire des Etudes Approfondies (DEA)
October 1979 - December 1980	Teaching Associate Computer Science Department University of Tunisia Faculté des Sciences de Tunis, Tunisia
1981 - 1982	University of California, Los Angeles M.S. in Computer Science
June 1981 - June 1984	Guest member of the Experimental Packet Radio Network "PRNET" Working Group, The Defense Advanced Research Projects Agency (DARPA)
June 1983 - December 1985	Chief System Analyst Technology Transfer Institute (TTI), Santa Monica, California
June 1981 - September 1986	Post Graduate Research Engineer Advanced Teleprocessing Systems Research Group Computer Science Department University of California, Los Angeles
September 1986 - present	Staff Research Associate Advanced Teleprocessing Systems Research Group Computer Science Department University of California, Los Angeles

PUBLICATIONS

- A. Belghith and L. Kleinrock, "Average Response Time in Parallel Processing Systems with Certain Synchronization Constraints," UCLA Report # UCLA-CSD-860033, November 1986.
- A. Belghith and L. Kleinrock, "Analysis of the Number of Occupied Processors in a Multiprocessing System," UCLA Report # UCLA-CSD-850027, August 1985.

- A. Belghith and L. Kleinrock, "Distributed Routing Scheme with Mobility Handling in Stationless Multi-hop Packet Radio Networks," ACM SIGCOMM 1983, Symposium on Communications, Architectures and Protocols, University of Texas at Austin, March 1983.
- A. Belghith and L. Kleinrock, "Distributed Routing Scheme and Transport of Real-time Data in Highly Mobile Stationless Multi-hop Packet Radio Networks," Master's Thesis, Computer Science Department, University of California, Los Angeles, December 1982.
- F. Kamoun, A. Belghith and J.L. Grangé, "Congestion Control with a Buffer Management Strategy Based on Traffic Priorities," Proceedings of the Fifth International Conference on Computer Communications (ICCC), Atlanta, Georgia, October 27-28, 1980.
- A. Belghith, "Etude de Congestion dans les Réseaux à Commutation de Packets," Memoire D'Ingenieurat Institut National de Recherche en Informatique et Automatique (INRIA), Rocquencourt, France, June 1979.

ABSTRACT OF THE DISSERTATION

Response Time and Parallelism in Parallel Processing Systems with Certain Synchronization Constraints

by

Abdelfettah Belghith
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1987
Professor Leonard Kleinrock, Chair

What makes the exact analysis of parallel processing systems so problematic is the internal parallelism within jobs, namely, a job may need and consequently may hold more than one processor at a time. We view a multiprocessor system as a set of P cooperating processors, and a computer job as a set of tasks partially ordered by some precedence relationships. For a finite number of processors, we assume that the total system capacity is shared among the jobs proportionate to the number of their ready tasks. This is non-egalitarian Processor Sharing, as compared with the usual egalitarian Processor Sharing technique.

Two fundamental performance measures of concern in a multiprocessor system are the expected job sojourn time and the achievable system speedup. We construct models and methodologies to analyze these two measures by exploiting the underlying stochastic processes. In those cases where the exact analysis fails, we provide bounds and/or approximate solutions backed up by simulations of the exact models.

Specifically, we investigate the probability distribution of the number of occupied processors, the generating function of this distribution and its first two moments. In particular, the expected number of busy processors is found to be dependent only on the average number of tasks per job, the job average arrival rate, and the task average processing requirement. Significant reductions in the expected job sojourn time can be realized by executing a job on a multiprocessor system. This effect is known as the speedup factor, which typically increases with the number of processors used. Along with an increase in the speedup factor, comes a decrease in the efficiency of the processors. While a large speedup factor may appear as a delight to the users, the efficiency of the processors is also extremely important. We first formulate an accurate and yet simple parametric approximation of the sojourn time, and then investigate the tradeoff between speedup and processor efficiency.

A special case of parallel processing systems is that in which a job, upon arrival, splits into exactly P tasks, each of which is attended by a separate processor. The performance measure of interest in such systems is the expected job sojourn time, defined as the expected time spent in the system between a job arrival time and the execution completion of all its clones. We extend the already known results, and investigate the case where a job may feed back into the system according to the prespecified partial ordering among its tasks.

CHAPTER 1 INTRODUCTION

In the past few years, we have seen a strong movement towards distributed computing systems. This movement is a natural part of evolution, fueled by technology and driven by real needs that cannot be satisfied by centralized computing systems. Advances in hardware technology have made it economically feasible to build distributed systems and multi-processor systems comprising thousands of processors [Hill85].

1.1 Overview of Distributed and Parallel Processing Systems

In many real time applications such as meteorology, cryptography, image processing and sonar and radar surveillance, the quality of the answer a processing system returns is proportional to the amount of computation performed [Hayn82a, Hayn82b, Pot83, Rose83]. Such real time applications need an instruction execution rate of more than one billion floating point instructions per second [Feng77]. Concurrent processing of data items is considered a proper approach for significantly increasing processing speed. This direction has already been taken by the Japanese Fifth Generation Computer Project. Array processing and multiprocessing have been utilized in attempt to provide such processing concurrency.

State-of-the-art parallel computer systems can be conceptually characterized into four structural classes: *Pipeline Computers*, *Array Processors*, *Multiprocessor Systems*, and *Data Flow Computers*.

Pipeline Computers

A pipeline computer [Grah70, Hall72, Rama77, Chen80] performs overlapped computations to exploit *temporal parallelism*. Generally, the four major steps or execution stages involved in the process of executing an instruction in a digital computer are: *instruction fetch* from the main memory, *instruction decoding* identifying the operation to be performed, *operand fetch* if needed in the execution, and the *execution* of the decoded arithmetic logic operation. A pipeline computer executes successive instructions in an overlapped fashion, as

opposed to a nonpipeline computer, where the four execution stages defined above must be completed before the next instruction can be initiated. The operation of the execution stages is synchronized under a common clock control. While a nonpipeline computer necessitates four pipeline cycles to complete one instruction, a pipeline computer may complete one instruction per pipeline cycle, provided the pipeline is full. Theoretically, a k-stage linear pipeline computer could be at most k times faster than a nonpipeline processor. However, this ideal speedup is merely an upper bound due to memory conflicts, data dependency, control instructions, and interrupts.

In addition to the *instruction pipelining* described above, some pipeline processors also partition the execution stage into a multiple stage arithmetic logic pipeline. This is usually done for some CPU-bound instruction such as sophisticated floating-point instructions. Due to the overlapping of instruction and arithmetic executions, pipeline processors are obviously better tuned to perform the same operations repeatedly through the pipeline. Pipeline computers are then very attractive and efficient for vector processing, where component operations are to be repeated many times. Examples of commercially available pipeline processors include the early vector processors, such as the Control Data Star-100 [Hint72, Corp73] and the Texas Instrument Advanced Scientific Computer (ASC) [Wats72, Wats74], the attached pipeline processors, such as the IBM 3838 [Corp76], and recent vector processors, such as the CRAY-1 processor [Bask77, Dorr78, Russ78], the CYBER-205 [Corp80], and the Fujitsu processor VP-200 [Miur83].

Array Processors

An array processor uses multiple synchronized Arithmetic Logic Units (ALU) to achieve *spatial parallelism*, as opposed to the *temporal parallelism* achieved by pipeline processors. A *Processing Element* (PE) is an ALU along with some registers and a local memory. The PEs are interconnected by a data-routing network. The PEs are synchronized in a lock-step fashion to perform the same function at the same time. The PEs, as well as the interconnection pattern to be established for a specific computation, are under the sole control of the *Control Unit* (CU). The CU also executes the scalar and control instructions, and broadcasts to the PEs vector instructions to be executed in parallel. The PEs are passive machines without instruction decoding capabilities.

Different array processors may use different interconnection networks between the PEs. The Burroughs Scientific Processor (BSP), for example, uses a crossbar network [Kuck82]. The Illiac-IV computer, on the other hand, uses a mesh-structured network [Barn68, Kuck68].

Multiprocessor Systems

A multiprocessor system is a system composed of two or more processors of approximately comparable capabilities. Each processor has its own local memory and perhaps private devices. Common sets of memory modules, I/O channels, and peripheral devices are also shared among the processors. The entire system is controlled by a single integrated operating system, providing interactions between the processors and their programs at various levels.

The interconnection structure used between the shared common memories and the processors (also between the common memories and the I/O channels) defines the multiprocessor architectural organization. Different interconnections have been practiced, such as the time-shared common bus, the cross-bar switch network, and the multiport memories. We distinguish two architectural models: *Tightly Coupled* and *Loosely Coupled* multiprocessor systems. Processors in a tightly coupled multiprocessor system communicate through a shared main memory; consequently, the rate at which data can be exchanged from one processor to the other is on the order of the bandwidth of the memory. In loosely coupled multiprocessor systems, however, processors communicate by exchanging messages through a message-transfer system; hence, the degree of coupling is very loose. The determinant of the degree of coupling is the communication topology of the associated message-transfer system. Loosely coupled systems are usually efficient when the interaction between parallel processes is minimal; generally they are referred to as *Distributed Systems*. Tightly coupled systems, on the other hand, can tolerate a higher degree of interaction between parallel processes without significant deterioration in performance. They are generally referred to as *Parallel Systems*.

Examples of multiprocessor systems include the CRAY-X-MP and the CRAY-2 systems [Chen83], the HEP system [Kowa85], the c.mmp system developed at Carnegie Mellon University [Saty80], the S-1 multiprocessor system developed at the Lawrence Livermore National Laboratory [Widd80], the IBM system 370/Model-168-MP [Case78], the Univac 1100/80-MP and 1100/90-MP, and many other multiprocessor systems [Saty80].

Data Flow Computers

To exploit maximal parallelism in a program, data flow computers were suggested in recent years. The basic concept is to enable the execution of an instruction whenever its required operands become available. Therefore, no program counters are needed in data-driven computations, as opposed to the conventional Von Neuman computers, which are control flow computers. Since instructions are executed upon the data availability, maximal concurrency is only constrained by the hardware resource availability. To the best of our knowledge, no data flow computer is commercially available yet. However, some laboratory prototypes are being investigated [McGr80, Taka83].

Several computer architectural classification schemes exist in the literature. Among them are: Flynn's classification scheme [Fly66], which is based on the multiplicity of instruction streams and data streams in a computer system, Feng's classification scheme [Feng74], which is based on serial versus parallel processing, and Handler's classification [Hand77], which is determined by the degree of parallelism and pipelining at the various subsystem levels. Following Flynn's classification scheme, array processors are commonly known as *Single Instruction stream-Multiple Data stream* (SIMD) computers. Multiprocessing systems, on the other hand, handle multiple instructions and multiple data streams, and hence are called *Multiple Instruction stream-Multiple Data stream* (MIMD) processors.

Many programming languages provide the user with the ability to write concurrent programs [Stot82]. Languages such as Ada [Ada81], CSP [Hoar78], and Concurrent Pascal [Hans75] provide for procedure level and statement level granularity. Granularity refers to the size of the operations that are executed in parallel. In contrast, some data flow languages [Acke82, McGr82] provide for operator level concurrency. The program is specified without reference to concurrency. The compiler breaks a program down into its component operations often to as small as add or multiply. When the program executes, all operators can be processed in parallel; an operator can execute as soon as it has received values for all its input operands. Upon termination, it forwards its result to those operators requiring such a result as their input.

1.2 Statement of the Problem and Preliminary Definitions

The problem we will study in this dissertation concerns the performance evaluation of loosely and tightly coupled multiprocessing systems, in which jobs are composed of a collection of tasks to be performed in a certain prescribed order described by a structural graph.

Task: A task is a well defined set of operations which can be performed on some input data, and which may generate some output data. Examples of a task include a single instruction, a library function, and a subroutine. A task may be *active* or *dormant*. A dormant task cannot begin execution until it acquires the *ready-for-service status*. An active task, on the other hand, is a task that is being executed. The identity of the tasks, the order of acquisition of the ready-for-service status, and the acquisition of the ready-for-service status are specified according to a given algorithm, as defined below. The scheduler of the multiprocessor operating system, on the other hand, handles the passage of the ready-for-service-tasks to the active state. The task service time (also referred to as the task service requirement) is the time needed to complete the task execution at a rate of one second per second. Throughout this dissertation, we shall consider various scheduling schemes and various task service time distributions.

Algorithm: An algorithm is a set of predicate dependent rules designed to carry out a given finite set of tasks. An algorithm, once performed, will accomplish a well defined objective in a period of finite time. The set of tasks may be as large as needed, but must be finite. A computer program is an example of an algorithm.

We distinguish two categories of relations among the set of tasks, namely, the *Precedence Relationships* and the *Parallel Relationships*. The precedence relationships between tasks specify the order of acquisition of the ready-for-service status. The parallel relationships specify the dynamic interactions that govern the actual execution of the active tasks. Whereas precedence relationships concern successive tasks (i.e., non-concurrent tasks), parallel relationships concern parallel tasks (i.e., concurrent tasks). Examples of parallel relationships include synchronizations, mutual exclusions, shared memory and/or communication conflicts, and interruptions (e.g., the ability of a given task to interrupt, stop, or terminate another parallel active task). In this dissertation, we concentrate only on the precedence relationships among the finite set of tasks. Following [Kana85], we formulate the five following types of precedence relationships. In the following definitions, we use the symbols T_a and T_b to indicate two distinct tasks, and the sets $\{T_i\}$ and $\{T_j\}$ to indicate two distinct sets of tasks with perhaps some common members.

Sequential Relationship: A Sequential relationship $S:T_a \rightarrow T_b$ specifies that task T_a must be completed before task T_b can acquire the ready-for-service status. Upon the completion of task T_a , task T_b assumes the ready-for-service status, and consequently may become active.

If-then-else Relationship: An If-then-else relationship $IF:T_a \rightarrow \{T_j\}$ specifies that task T_a must be completed before any tasks in $\{T_j\}$ can begin. Upon the completion of task T_a , one and only one task in the set $\{T_j\}$ acquires the ready-for-service status.

The selection of such a task is assumed throughout the dissertation to be accomplished according to a stochastic selection procedure.

Merge Relationship: A Merge relationship $M:\{T_i\} \rightarrow T_b$ specifies that only one task in the set $\{T_i\}$ (i.e., the task having the ready-for-service status) is actually executed, and that upon the execution of such a task, task T_b acquires the ready-for-service status and may thus become active.

Fork Relationship: A Fork relationship $F: T_a \rightarrow \{T_j\}$ specifies that task T_a must be executed before the tasks in $\{T_j\}$ may begin. Upon the completion of T_a , all the tasks in the set $\{T_j\}$ acquire the ready-for-service status, and consequently may become active.

Join Relationship: A Join relationship $J: \{T_i\} \rightarrow T_b$ specifies that all the tasks in the set $\{T_i\}$ must be completed before task T_b may begin. Upon the completion of all the tasks in $\{T_i\}$, task T_b acquires the ready-for-service status, and consequently may become active.

Algorithms may be classified into two categories: acyclic algorithms and cyclic algorithms. In acyclic algorithms, a task is never executed more than once. For cyclic algorithms, we assume that the number of times a loop of tasks is performed is an independent stochastic process (in particular, independent of any input data).

Process Graph: A process graph represents a given execution instance of an algorithm. From the preceding definition of an algorithm, a process graph is thus an acyclic directed graph with nodes representing the tasks, and edges representing the precedence relationships among these tasks. The only types of precedence relationships used in a process graph are the three basic types: sequential, fork, and join relationships. Tasks may be represented several times in a process graph to undo any loop in the algorithm. Note that the merge and if-then-else precedence relationships do not appear in the process graph, and that a given algorithm on a given finite set of tasks may generate several different process graphs.

Job: A job is an exogenous request to execute a certain algorithm. Consequently, there exists a one-to-one correspondence between a process graph and a job. Throughout the dissertation, we consider that jobs are represented by process graphs, and arrive exogenously to the system according to a prespecified arrival process. Different types of process graphs will be investigated throughout the dissertation. The notion of an algorithm is only introduced here to define the process graph and shall be ignored in the sequel.

Parallelism in a multiprocessor system is accomplished through both the pipelining of different jobs to the processors (i.e., *Pipelined Processing*) and the cooperation of the processors in executing parallel tasks of a given job (i.e., *Concurrent Processing*). Pipelined processing alone can be accomplished by a set of noninteracting and asynchronous computers. We shall refer to this model as the multicomputer model system. Such a multicomputer model shall serve

as a basis for comparison to ascertain the speedup and the gain in the job response time achieved by multiprocessing systems.

Since we are considering only precedence relationships among the tasks, we may incorporate the communication involved between two consecutive tasks in their corresponding service times. Throughout the dissertation, we shall then assume that the communication between tasks is accounted for in the task service times and, more importantly, that the queuing delays emerging from such communications are negligible.

1.3 Previous Work in the Area

In this section, we briefly review some of the related research in the area of distributed, multiprocessing and parallel processing performance evaluations.

K.C. Kung [Kung84] defined a common concurrency measure which gives a comparison of how much parallelism can be achieved in a multiprocessing system. A Job is represented by a directed acyclic graph of computation where the nodes represent the tasks and the directed edges represent the dependencies among the tasks. He defined the concurrency measure as the ratio between the average system time in a P processor system, and the average system time obtained by just using one of the processors. The speedup factor, which measures the amount of parallelism within a particular job, is thus defined as the inverse of the concurrency measure.

For an infinite number of processors, he gave an exact formula for the concurrency measure and derived upper and lower bounds for the average system time. Upper bounds are calculated by synchronizing the execution at each level by forcing all the tasks in the next level to wait for the slowest task in the current level to complete before they all start execution. He called the time between the synchronization of two neighboring levels the *Forced Synchronization Time (FST)*. It follows that by summing up the FSTs at each level of a process graph, an upper bound for the average time in a system with an infinite number of processors is obtained. For a lower bound, he found the average time required to execute the tasks in the longest path from the initial node to the terminating node in the process graph.

For a finite number of processors, he assumes a specific shape of the process graph, namely, a diamond shape with a continuum of tasks within the diamond. The service times of tasks are assumed to be constant. The system is assumed to initially contain a fixed number of jobs to be executed only once. Two task assignments to processors are studied: the Depth First Assignment, where all available processors are first assigned to the tasks in a job that is closest to being completed, and the Breadth First Assignment, where all available processors are first

assigned to those jobs that have received the least amount of processing. The ratio of the average system times of such assignments is investigated.

Using the Chernoff Bound on the tail probability [Klei75], Kung found that, as the number of tasks in a randomly chosen process graph increases, the number of levels in such a process graph approaches half that number of tasks with probability one. Consequently, he shows for the case of a randomly chosen process graph with a fixed but very large number of tasks, say N , and an infinite number of processors, that the concurrency measure, say S , is bounded by: $\frac{1}{2} \frac{N}{\mu} \leq S \leq \frac{3}{4} \frac{N}{\mu}$, where $\frac{1}{\mu}$ is the average task execution time.

L. Kleinrock [Klei84] considered a distributed processing environment in which a total processing capacity, say C operations/second, is split into a number of smaller processing units of the same total capacity, and which collectively process a stream of jobs arriving to the system according to a Poisson distribution with a fixed rate. He studied the performance ratio of the mean response time, say T , seen by jobs in the distributed environment, and the mean response time, say T_0 , seen by a job when it is processed by a single processor of the same total capacity. The most general distributed configuration studied is that of a series-parallel topology. He considered m parallel chains, the k th of which contains n_k processors in series, each of capacity C_k operations/second. Jobs are assumed to arrive at the k th chain from a Poisson source and each requires an independent, exponentially distributed number of operations from each processor in that chain. For equal loading on each series chain, he showed that $\frac{T(\rho)}{T_0(\rho)} = \sum_{k=1}^m n_k$, where ρ is the system utilization. He also addressed the problem of optimizing such a performance ratio. He found the optimal distribution of traffic among the chains; one property of this solution is that some of the series chains carry zero traffic. Such a property is also studied by A. Agrawala, E. Coffman, M. Garey, and S. Tripathi [Agra84]. Also, by considering the optimization of the performance measure subject to traffic patterns and processor capacities, he found that $\frac{T(\rho)}{T_0(\rho)} = \min_k \{n_k\}$. He concluded that distributed processing never improves such a performance ratio unless some dis-economy of scale in the cost of processing is introduced.

F.A. Tobagi and H. Kanakia [Kana85, Toba85] considered a job as a collection of m tasks defined by a structural graph. Requests to execute the algorithm arrive from some external source according to some aggregate Poisson distribution. Their multiprocessing system consists of m processors, the same number as the number of tasks within a job, the i th of which can only perform the i th task of the algorithm. Each processor runs asynchronously and is modeled as a single server with infinite capacity room. As in [Klei84], the total system capacity is constrained to a fixed value, the same for both the distributed and the centralized systems. Their model of a centralized system consists of a single processor which can perform all tasks and is modeled as a single server with infinite waiting room. In the case of single stage service distribution in the centralized system, they postulated that such systems always outperform

distributed systems. This result is merely a generalization and an extension of the result found by L. Kleinrock in [Klei84]. For the case of multi-stage type service distribution in the centralized system, they showed that there are cases of algorithms and models of processing systems where distributed processing outperforms centralized processing.

Queueing networks are important as performance models of computer and communication systems since the performance of these systems is usually affected by contention for resources. Since the original work of Jackson [Jack63], it has been shown that the product form solution exists for networks with heterogeneous jobs, several important scheduling techniques and state dependent behaviors [Bask75, Chan77, Tows80]. However, there are a number of system characteristics which preclude a product form solution. Among the most important of these are priorities and simultaneous resource possession. This unfortunate fact obliges us to settle for approximate solutions or to use simulation. Indeed, this direction has been followed by many researchers to model and investigate distributed and multiprocessing system performance.

P. Heidelberger and K.S. Trivedi [Heid82] considered a queueing network model of a computer system in which a job subdivides into two tasks at some point during its execution. This is known as the Fork operation. Except for queueing effects, the tasks are supposed to execute independently of one another and do not require any synchronization. Because of the inherent parallelism, the model does not have an analytically tractable solution. The workload of the system consists of a set of statistically identical jobs, where each job consists of primary tasks and zero or one secondary task. A secondary task is spawned whenever a primary task enters the specified Fork node. The system is assumed to contain a fixed number of primary tasks at all times. The network model which they used for the approximate system has two chains: one closed chain to model the behavior of primary tasks, and one open chain to model the behavior of secondary tasks with arrival rate equal to the primary task throughput of the closed chain at the specified Fork node. The exogenous arrivals to the open chain are assumed to be Poisson with fixed rate. This approximation is found to be quite accurate, unless the system under consideration is highly unbalanced.

D. Towsley, K.M. Chandy and J.C. Browne [Tows78] have studied computer models in which CPU and I/O (or I/O and I/O) activities can be overlapped. However, their model, in contrast with the model studied in [Heid82], requires tight synchronization between the concurrent tasks in the sense that both the CPU and the I/O (or I/O and I/O) tasks must be completed before processing can continue. This is known as the Join operation. They concluded that the performance gain due to this type of overlap is greatest for balanced systems and relatively low levels of multiprogramming.

In [Heid83], P. Heidelberger and K.S. Trivedi extended their work of [Heid82] to contain, in addition to the Fork node, a Join node (the same node as the Fork node) to enforce synchronization between primary tasks and their spawned secondary tasks. As in [Heid82], the system consists of a finite number of jobs (i.e., primary tasks). Synchronization is achieved by requiring all siblings to complete execution before the job can continue processing. Two approximation techniques are presented. The first technique, called *The Decomposition Approximation*, is based on the observation that if the primary and secondary tasks are relatively long, in the sense that many processors are visited before the tasks complete, then changes in the number of primary and secondary tasks in the system will occur infrequently as compared to changes in the queue lengths of tasks at the processors. In such cases, the authors assumed that the queue length distributions converge to steady state distributions prior to the next change in the number of primary and secondary tasks in the system. Their second approximation technique, called *The Method of Complementary Delays*, consists of iteratively solving a sequence of mathematically tractable queueing networks. For such a technique, they introduced three fictitious servers: one server to represent the waiting time for a secondary task to be spawned, one server to represent the waiting time of a secondary task for the completion of its siblings, and one server to model the waiting time of a primary task for all its spawned secondary tasks to complete. The fictitious servers are modeled by delay type servers (i.e., Infinite Servers). The resulting queueing network belongs to the Product-Form class and is hence solved rather easily. However, the fictitious servers' mean delays are to be determined from the solution of the network itself. For that, they applied an iterative procedure.

Other modeling methods for distributed and multiprocessing systems in which jobs require two or more resources simultaneously before proceeding have been investigated. Such models include I/O channels and disks, memory partitions and busses, and memory partitions and processors. These types of models are not usually mathematically tractable due to the inherent simultaneous resource possession phenomenon. Approximate methods with high accuracy and low computational cost are therefore needed, and indeed have been developed and used by several researchers. Virtually all such approximation techniques for such models use an adjusted aggregate queue-dependent server to represent and model the resources simultaneously required.

K.M. Chandy, U. Herzog and L. Woo [Chan75a] introduced the so-called *Norton's theorem for queueing networks* as an approximate iterative technique for the analysis of complex queueing networks. They showed that Norton's theorem provides an exact analysis when applied to queueing networks which satisfy local balance [Chan77]. However, for queueing networks which do not satisfy local balance, Norton's theorem provides only an approximate solution. In [Chan75b], the same authors extended Norton's theorem to the study of queueing networks with heterogeneous jobs (several job classes).

Norton's theorem for queueing networks can be used to approximately analyze systems with simultaneous or overlapping resource possession. As noted in [Jaco83], this approach works well when the primary resource consists of a number of identical units and a job may use any available unit. C.H. Sauer [Sauer81] used such a technique to analyze computer systems in which a job holds a memory partition and the processor simultaneously.

P.A. Jacobson and E.D. Lazowska [Jaco83] developed a new approach to modeling the simultaneous or overlapping resource possession. Their method is called "The Method of Surrogate Delays". This approach is applicable to queueing networks in which some resource, called primary resource, must be obtained and held while some other series of resources, called the secondary subsystem, is used. The key concept of their method is the iteration between two closed queueing network models. In each of these models, the portion of the overall system in which simultaneous resource possession occurs will be replaced by two components. In the first model, one component is an explicit representation of the primary resource and the other component is a delay server acting as a surrogate queueing delay due to congestion in the secondary subsystem. In the second model, one component is a flow equivalent server obtained from analyzing the secondary subsystem in isolation and the other component is a delay server acting as a surrogate for queueing delays at the primary resource when the secondary subsystem is not congested. In [Jaco83], the authors displayed the use of such a technique to model a loosely-coupled multiprocessor system consisting of P identical processors, each with its own local memory, M identical shared memories equally likely referenced by each of the P processors, and B shared busses, each of which can connect any processor to any shared memory. This system is often characterized as the P - M - B system. In the same paper, they also considered the case of a computer system with an I/O system consisting of D disks sharing a single common channel. For this system, they assumed that the channel is occupied whenever any of the disks is either searching for the start of the data or transferring such data.

D. Towsley in [Tows83] used the method of surrogate delays to analyze a P - M - B system in which he assumed constant memory access times, arbitrary memory access patterns and bus contention. He concluded that the throughput predictions from this model are very accurate and within 1% of those given by simulation.

F. Baccelli and A.M. Makowski [Bacc85a] and the same authors with A. Schwartz [Bacc85b] considered a K -dimensional Fork-Join queue: a K parallel servers with a synchronization constraint on the arrivals and the departures of jobs from the system. A job, upon arrival to the system, is split into K subtasks, each of which attends one of the K parallel servers. Synchronization at departures is achieved by parking already serviced subtasks in an auxiliary infinite queue, where they wait to be reunited to not yet serviced siblings of the same job. The K parallel servers are assumed to be heterogeneous, and the arrival and service processes to be of general type distributions. They obtained bounds on the response time of a job, which is defined to be the delay between the Fork and Join dates. Their upper bound assumes K

mutually independent GI/G/1 parallel queueing systems, whereas their lower bound assumes K mutually independent D/G/1 parallel queueing systems. In [Bacc85b], some numerical examples are given for the 2-dimensional case; however, the tightness of the bound is not investigated.

L. Flatto and S. Hahn [Flat84] considered the exact analysis of the 2-dimensional Fork-Join queue system. Their system consists of two parallel heterogeneous exponential servers, each one with its own infinite waiting room, and a Poisson job arrival process. Each job, upon arrival, splits into exactly two subtasks, each of which joins one of the two parallel servers. They obtained a formula for the double generating function of the system occupancy. They also derived asymptotic formulae for the system occupancy as either one of the queues becomes congested. Such asymptotic results are employed to study the interdependency of the occupancies of the two queues and to derive limit laws for the expectation and the distribution of one of the queue lengths conditioned on the other [Flat85].

R. Nelson and A.N. Tantawi [Nels85] developed a new technique called *The Scaling Approximation*. This technique is based on the observation that, if P_K is a performance metric to be evaluated, where $\lim_{K \rightarrow \infty} P_K = \infty$, and if there exist upper and lower bounds that grow at rate $O(f(K))$, then it must be the case that P_K also grows at a rate $O(f(K))$. An approximation of P_K is then of the form $P_K = G(K-i+1)P_i$, where $G(K)$ is a function that grows at a rate $O(f(K))$ and P_i is a known (approximate or exact) evaluation of the statistic P_K for the value $i, i=1, \dots, K-1$. They applied this approximation technique to analyze a multi-dimensional Fork-Join queue system with homogeneous exponential parallel servers and Poisson job arrivals. Using results from [Flat84, Flat85], they derived exact, asymptotic and approximate expressions for the mean response time of a job in such a system for the case $K=2$. They then applied the Scaling Approximation to study the K-dimensional case. However, they used simulation to determine one parameter of the scaling function $G(K)$, and to study the accuracy of such an approximation. They concluded that the relative error in the approximation is less than 0.05 for $K \leq 32$.

J. Le Boudec [Boud85] considered a multiclass multiserver queueing system consisting of B identical exponential servers with constant rate and in which the classes of customers are sorted into M concurrency groups. He assumed an FCFS service discipline but with the restriction that two customers of the same group cannot be served simultaneously. The author derived closed form expressions for the steady state probabilities and showed that product form is maintained when such a system is inserted in a BCMP network.

L. Green [Gree80] considered a multiserver system where each customer requests service from a random number of identical servers. Customers cannot start service until all required servers are available. The servers are assumed to be identical and independent. Customers arrive to the system according to a Poisson process. The service completion time at any server is assumed to be exponentially distributed. The author converted such a system to a

single server M/G/1 system. She derived the distribution of waiting times, the average waiting time and the distribution of busy servers. A.F. Seila [Seil84], by using results from [Gree80], derived the second moment of the waiting time in the queue.

S. Pincus [Pinc84] studied a multiserver queueing system which responsively adjusts the number of servers based on the number of customers awaiting service (i.e., in the queue). The arrivals to the system are assumed to be Poisson, and the service times to be exponentially distributed. Moreover, he considered the case of nondistinguishable servers and distinguishable servers where the same extra servers that are added at a threshold are the ones to be removed from the system when the queue shortens sufficiently. He analyzed the system by formulating and solving a two-dimensional Markov chain.

Petri Nets (PN) [Pete81] were designed to model systems with interacting and concurrent components. A PN comprises a set of *Places*, a set of *Transitions*, and a set of *Directed Arcs*. Places may contain *Tokens*. The state of a PN is its *Marking*, defined by the number of tokens contained in each one of the places. Stochastic Petri Nets (SPN) are introduced by M.K. Molloy [Moll82, Moll81]. SPNs are obtained by associating with each transition in a PN an exponentially distributed firing time. M.K. Molloy has shown that SPNs are isomorphic to continuous time Markov chains and that the markings in SPNs correspond to the states in Markov chains. SPNs, though a very useful tool for the analysis of computer systems, are limited to the modeling of very simple and yet small systems. This limitation is mostly due to the fact that the graphical representation of a system rapidly becomes very difficult when the system size and complexity increase. Moreover, the number of states of the associated Markov chain grows very quickly with the dimension of the graphs.

Resource allocation (e.g., task assignment, load sharing, and load balancing) in distributed systems and its relationship to systems performance is also a major issue associated with the design of distributed systems. A large body of studies in this field appeared in the literature and several approaches have been suggested. These approaches can be classified into three major categories, namely, mathematical programming, such as in; [Chu80] graphic-theoretic, such as in; [Ston78] and heuristic methods, such as in [Efe82]. Y.T. Wang and R.J.T. Morris in [Wang85] presented a unified approach to this problem, as well as a literature survey and an extensive list of references on the subject.

1.4 Contribution of this Dissertation

In this section, we outline the main contributions of this dissertation. In Chapter 2, we analyze the multiprocessing system with an infinite number of processors. The job arrival process is considered to be Poisson, the task service time is assumed to be constant: the same for all the tasks. Jobs are described by an acyclic directed computational graph, namely, a process

graph with a fixed number of nodes (i.e., tasks). Three different graph models are investigated. These are the fixed process graph model, where the topology of the graph is fixed and prespecified; the semi random process graph model, where the number of levels is fixed but the repartition of the nodes among the levels may vary from one job to another; and finally, the random process graph model where, both the number of levels and the repartition of tasks among these levels are considered to be random. For each of the three graph models, we derive a closed form expression for both the probability density function and the Z-transform of the number of occupied processors. Using the Z-transform, we derive for all three cases the average and the variance of the number of busy processors in the system. We have found a rather interesting result, stating that the expected number of busy processors is only a function of the job Poisson arrival average rate, the task constant service time, and the fixed number of nodes in the process graph. For all the three process graphs, we therefore obtain the same expected number of busy processors.

The question naturally arises as to what extent this result can be generalized. In Chapter 3, we address this issue. The goal here is to free the multiprocessing model of the second chapter from its imposed restrictions. We then consider a very general model with arbitrary precedence relationships among the nodes in the process graph, arbitrary distribution of the number of tasks per job, arbitrary conditional distribution of the number of levels in the process graph, arbitrary repartition of the tasks among the levels, arbitrary task service requirements, perhaps different requirements for the different tasks, and a general job arrival process. We first pursue the case of an infinite number of processors, and then investigate the finite number of processors case. We prove that in the generalized model just described, and for both cases of an infinite and finite number of processors, the expected number of busy processors remains only a function of the job arrival average rate, the task average service requirement, and the average number of tasks per job.

In Chapter 4, we aim for the job average response time in a P processors multiprocessor system. We distinguish three cases depending on the value of P . For the infinite number of processors case, we investigate and find the process graphs, among all possible process graphs, which provide respectively the upper and lower bounds on the job average response time. For the finite number of processors case, we introduce a new scheduling strategy, termed the P -dimensional-Discriminatory-Processor-Sharing-With-Feedback (P -DPS-WF). For the uniprocessor case, we first prove that the 1-DPS-WF scheduling policy forms a complete family of scheduling strategies, in the sense that any response time requirement that can be satisfied at all, can be achieved by a strategy from the family. Then, we prove a conservation law which puts a linear equality constraint on the set of average system times of the different stages during the job execution through the 1-DPS-WF system. For any finite number of processors, we first analyze the P -DPS-WF system with the same constant concurrency degree for all the execution stages. We then find the process graphs, among all possible process graphs, which provide respectively the upper and lower bounds on the job average response time. This enable us to

formulate a rather accurate and parametric approximation of the job average response time. Simulations are again used to prove the accuracy of our approximation. We conclude chapter 4 by introducing and studying the achievable parallelism in the P-DPS-WF systems. The efficiency of the processors and the optimal operating points of the P-DPS-WF parallel processing system are also investigated.

Models of parallel processing systems in which a job, upon arrival, spawns into two or more tasks, each one to be executed independently on a different processor, arise in many practical situations and application areas, such as flexible manufacturing and concurrent and distributed processing models. In the context of production systems, a customer order may be viewed as a bulk of suborders, each one to be attended by a separate device or facility. In parallel architecture computer systems, a bulk job can be viewed as a program composed of several concurrent subroutines, each one to be executed on a dedicated processor. In distributed replicated data base, update requests arriving to a given site must be performed by all the sites to maintain the data base integrity.

In Chapter 5, we seek to determine the bulk job average response time through such a multiprocessing system with synchronization constraints. The job average response time is defined as the expected delay incurred between the job arrival and its completion times. This type of multiprocessing system with synchronized arrivals is regarded as an $M/G/1$ queueing system with correlated consecutive service times. The purpose of Chapter 5 is, in essence, two fold: to devise an approach to derive an approximate value of the job average response time through such a multiprocessing system with synchronized arrivals, and to present a unified way of approximating the average response time in an $M/G/1$ queueing system with correlated consecutive service times and with a coefficient of variation greater than one. We investigate four different approaches based on the inherent characteristics of such multiprocessing systems. In the last section of the chapter, we extend our results to any finite number of processors.

In Chapter 6, we extend the model of multiprocessing systems with synchronized arrivals studied in Chapter 5. We now allow the job to feed back as many times as needed. Two cases are distinguished depending on the description of the feedback policy: the chain feedback case, where jobs are allowed to feedback a given number of times, and the tree feedback policy, where jobs are allowed to feedback according to any given acyclic directed graph description. Approximate solutions of the job average response time are presented and discussed for both cases of job feedbacks. Extensive simulations are used to validate and support our approximation. The job approximate average response time is found to be well within 5% of the real value.

The chapters of this dissertation have been written in a self-sufficient manner, to allow the reader to selectively read portions he finds most interesting without the necessity of reading all the previous material. This, of course, implies that there is a certain amount of redundancy and repeated definitions in the text, but it is hoped that these repetitions will serve to clarify the material rather than to bore the reader.

1.5 Multiprocessing Applications

We conclude this first chapter by providing and discussing some representative examples of the widespread applications of high performance computers, and the ever growing greediness for more computing speed. Without the use of superpower computers, several advances in human civilization could barely be accomplished. Fast and efficient multiprocessing systems are in very high demand in many scientific, engineering, energy resource, medical, military, artificial intelligence, and basic research areas. Large-scale computations are needed and performed in these application areas.

Obtaining a solution to a large scientific problem generally involves three interactive disciplines: theories, experiments, and computations. Theoretical scientists develop mathematical models which computer engineers solve numerically. The numerical results may then suggest new approaches and theories. Experiments provide data and can model processes that are hard to approach in the laboratory. Indeed, computer simulations are far cheaper and faster than physical experiments. Computers can solve a much wider range of problems than specific laboratory experiments are capable of. Computational approaches are only limited by computer speed and main memory capacity, while physical experiments have several inherent practical constraints.

Theoretical and experimental scientists are potential users of large program codes and large data manipulations in several areas. Some representative applications are discussed below.

1.5.1 Predictive Modeling and Simulations

Predictive modeling is performed through extensive computer simulation experiments, which often involve large scale computations to achieve the desired accuracy and turn-around time.

Numerical Weather Forecasting

Weather modeling is necessary for short-range forecasts, as well as long-range hazard predictions, such as floods and environmental pollution. Weather and climate researchers are in pressing need of very fast and efficient computers [Suga80, Whit85, Dick82, Faro83]. Weather and climate analysts need to solve *general circulation model* equations, which necessitate huge amounts of data manipulation and very large scale computations. In such models, the atmospheric state is represented by the surface pressure, the wind field, the temperature, and the water vapor mixing ratio. These state variables are governed by the Navier-Stokes fluid dynamics equations in a spherical coordinate system.

Computations are carried out on a three-dimensional grid which partitions the atmosphere vertically into I levels and horizontally into J intervals of longitude and K intervals of latitude. The number, N, of time steps used in the simulation forms a fourth dimension. Currently 24-hour computer forecasts are made on an approximately 270-mile grid, which is roughly the distance between New York and Washington, D.C. Such a 24-hour forecast would perform about 100 billion data operations, and consequently would require 100 minutes if executed on a 100 megaflop computer such as the CRAY-1 or the CYBER-205.

This 270-mile grid gives the forecast between New York and Washington, D.C., but not for Philadelphia, about half way in between. Increasing the forecast by halving the grid in all the four dimensions would necessarily increase the computation volume at least 16 times. A 100 megaflop computer such as the CRAY-1 would then take at least 24 hours to complete the 24-hour forecast. Reliable and accurate long-range forecasts and predictions require a finer grid and a smaller time step, and hence require a much more powerful computer than the CRAY-1, the HEP, the CYBER-205, or any existing super computer for that matter.

Oceanography and astrophysics are other potential customers of super power computers. Oceans store and transfer heat and exchange it with the atmosphere. A better understanding of our oceans would undoubtedly help in areas such as climate prediction analysis, fishery management, ocean resource exploration, and coastal dynamics and tides. Oceanographic studies use a larger scale time variability and a smaller grid size than those used for atmospheric studies. A complete simulation of the pacific ocean with adequate resolution (1 degree grid) for 50 years would take about 40 days on a CYBER-205 super computer.

Socioeconomics and Government Use

Econometrics, social engineering, government census, and crime control are areas in great demand of very large computers [Suga83, Rodr80]. In the United States, the FBI uses large computers for crime control; the IRS uses large mainframes for the tax collection and auditing. Super computers are also used extensively for national census and general public

opinion polls. It has been estimated that about 60% of the large scale computers manufactured in the United States have been used by the U.S. government.

The United Nations supported a world economic simulation which suggests how a system of international economic relations that features a partial disarmament could narrow the gap between the rich and the poor. This simulation is based on an input-output model of world economy proposed by the Nobel Laureate W.W. Leontief (1980).

1.5.2 Medical and Military Research

Super power computers are needed in medical areas such as computer-assisted tomography (CAT), artificial heart design, liver diagnosis, brain damage estimation, and genetic engineering studies. Super computers are also sorely needed in military and defense areas, such as weapon design and other electronic warfare.

Computer-Assisted Tomography

The human body can be modeled by computer-assisted tomography scanning [Alex83, Suga80, IEEE83a]. At the Courant Institute of Mathematical Sciences, scientists are seeking an array processor for time-sequence, three-dimensional modeling of blood flow in the heart, with the goal of understanding how best to make an artificial heart valve. Similar approaches can be applied to eventually reveal and understand the secrets of our organs in real-time.

The image reconstruction of human anatomy in present computer-assisted tomography scanners is two-dimensional, but there is a strong need for three-dimensional scanners. The Mayo Clinic in Rochester, Minnesota, is developing a research CAT scanner for three-dimensional, stop-action, cross-action viewing of the heart. This Mayo Clinic scanner is expected to have 2,000 to 3,000 megaflops speed.

Weapon Research and Defense

To date, the military and defense research agencies have been using the majority of the existing super computers [Boot83, Fors83, IEEE83b, Kowa85]. Defense related military applications of super power computers involve but are not limited to multiwarhead nuclear weapon design, cartographic data processing for automatic map generation, sea surveillance for antisubmarine warfare, radar signal processing, and simulation of atomic weapon effects by solving hydrodynamic and radiation problems. Many types of super computers have been used including the CRAY-1, the CYBER-205, the PEPE, the Staran, and the S-i multiprocessors systems.

1.5.3 Energy Resource Exploration

Using computers in the energy area results in lower production costs and higher safety measures. Super power computers play an important role in the discovery of oil and gas, the development of workable plasma fusion energy, and molecular reactor safety.

Seismic Exploration

Seismic exploration sets off a sonic wave via explosive or by jamming a heavy hydraulic ram into the ground, and vibrating it in computer-assisted patterns. Echoes are picked up by using a few thousands phones which are scattered around the designated spot. The echo data are used to depict two-dimensional cross-sections displaying the geometrical underground strata. The strata types which may bear oil or gas can be identified using reconstruction techniques. Many oil companies are investing about 10% of their budget in the use of attached array processors or vector super computers for seismic data processing. A typical field record for the earth response to a sonic input has around 3,000 different time values, each at about 48 different locations, which produces 2 to 5 million floating-point numbers per kilometer along a survey line [Suga80, IEEE84].

Plasma Fusion Power

At the Laurence Livermore National Laboratory and at the Princeton Plasma Physics Laboratory, nuclear fusion researchers are using vector super computers extensively [Bren83, Rodr82]. The potential for magnetic fusion to provide an alternative source of energy is now closer to becoming a reality. The United States National Magnetic Fusion Energy Computer Center is currently using two CRAY-1's and one CDC-7600 to assist in its controlled plasma experiments. Supercomputers are an indispensable tool in magnetic fusion energy exploration.

Several other engineering design, automation and basic research areas have been in high demand for super computers. Engineering design problems are in pressing need of large-scale computing systems in fields such as the finite-element analysis needed for structural designs and wind tunnel experiments for aerodynamic studies. Artificial intelligence and automation require hundreds of megaflop computers in fields such as image processing, pattern recognition, computer vision, knowledge engineering, speech understanding, expert computer systems and intelligent robotics. Many other application areas related to basic scientific research fields demand the use of super power computers such as problems on quantum mechanics, statistical mechanics, polymer chemistry, and the study of fluid and molecular dynamics.

CHAPTER 2 NUMBER OF OCCUPIED PROCESSORS IN PARALLEL PROCESSING SYSTEMS

In this chapter, we derive and investigate the distribution, the Z-transform, the average and the variance of the number of occupied processors in a parallel processing system. Many parameters are in play to characterize the terrain of the parallel processing system under investigation. These are:

1. the job arrival process,
2. the process graph description,
3. the task processing requirement, and
4. the number of processors involved.

This chapter concerns the Poisson job arrival process, the constant task service time (which is the same for all tasks), and the infinite number of processors case. The process graph however, can be fixed, semi random, or random. The chapter is organized into 4 sections. In Section 2.1, we define the parallel processing model to be used throughout the chapter; this concerns the description of the process graphs representing the jobs to be processed, and the processors forming our parallel processing system. In Section 2.2, we shall investigate the case of fixed process graphs. All jobs have the same process graph with a fixed number of tasks and a fixed number of levels. The probability density function, the Z-transform, the average and the variance of the number of occupied processors will be derived. Section 2.3 deals with semi-random process graphs with two or more levels. The case of just one level, being a fixed process graph case, has already been treated in Section 2.2. Each job has a process graph with a fixed number of tasks and a fixed number of levels, the distribution of tasks among the levels, however, varies from one job to another. In this section, we shall also derive the probability density function, the Z-transform, the average and the variance of the number of occupied processors. Section 2.4 will deal with the case of random process graphs. Each job has a random process graph with a fixed number of tasks but a random number of levels (not exceeding the number of tasks). The Z-transform, the average, and the variance of the number of occupied processors will be derived. In the following chapter, we shall generalize our results to random process graphs with a random number of tasks, general task service times, a general job arrival process, and a finite number of processors.

2.1 Model Description

A computer job is a set of tasks partially ordered by some precedence relationships and represented by a Process Graph (PG). A node in the process graph represents a given task, and an edge (i,j) between node i and node j represents the precedence relationship between task i and task j. Edge (i,j) is used to prevent the start of task j execution unless task i execution has been completed. The tasks (i.e., nodes) in the process graph are therefore distributed into levels. Tasks at level one are said *starting tasks*, and tasks at the last level in the process graph are said *terminating tasks*. Any two tasks can be executed concurrently (i.e., in parallel) if and only if every predecessor of one task does not include the other task, and vice versa. Figure 2.1 gives an example of such a process graph, where the edges are implicitly directed downwards. The job arrival process is assumed throughout the chapter to be Poisson with aggregate rate λ .

The multiprocessor system under consideration consists of an infinite number of homogeneous and identical processors. Each processor is capable of executing any task. Let N denote the total number of tasks in a process graph. Throughout this chapter, we consider that N has a fixed given value. Let \bar{r} represent the random variable counting the number of levels in a job, such that $1 \leq \bar{r} \leq N$, \bar{X} be the task constant service time, and \bar{Y} denote the random variable representing the number of occupied processors in the system.

Since each level in the process graph must have at least one task in it, it follows that the total number of process graphs having N tasks and r levels is equal to the number of ways to distribute $(N-r)$ tasks among the r levels. This number of ways is* :

$$\binom{(N-r)+r-1}{r-1} = \binom{N-1}{r-1} \quad (2.1)$$

* In fact, the ordinary generating function of such a number of ways is :

$$(x + x^2 + \dots + x^k + \dots)^r = x^r (1-x)^{-r}$$

since each level can have from 1 to $N-r+1$ tasks in it, the number of ways to distribute N tasks among the r levels such that no level is left empty is the coefficient of x^N in the generating function. By the binomial theorem [Liu68], we have:

$$(1-x)^{-r} = \sum_{i=0}^{\infty} \binom{r+i-1}{i} x^i \quad \text{and therefore}$$

$$x^r (1-x)^{-r} = \sum_{i=0}^{\infty} \binom{r+i-1}{i} x^{r+i} = \sum_{N=r}^{\infty} \binom{N-1}{r-1} x^N$$

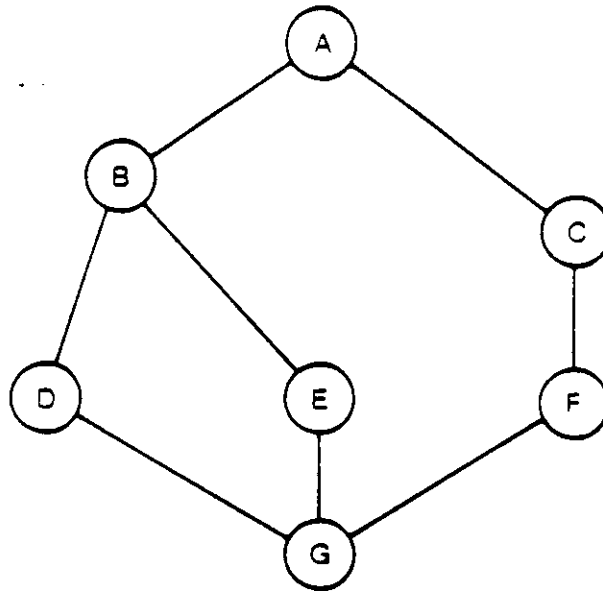


Figure 2.1: Example of a Process Graph

Proposition 2.1

For a fixed number of tasks per job, say N , and a fixed number of levels, say r , and for $1 \leq n \leq N-r+1$ and $1 \leq k \leq r$, the probability of having n tasks at level k is given by :

1. If $r=1$ then:
$$P\{n \text{ tasks at level } 1\} = \begin{cases} 0 & \text{if } n \neq N \\ 1 & \text{if } n = N \end{cases}$$

2. If $r \geq 2$ then:
$$P\{n \text{ tasks at level } k\} = \frac{\binom{N-n-1}{r-2}}{\binom{N-1}{r-1}}$$

Proof

The proof of case 1 is trivial since for $r=1$, all tasks must be at such a level. For N and r fixed and from equation (2.1), we know that the total number of process graphs that we can have is given by: $\binom{N-1}{r-1}$. Consider now level k , we want to have n tasks at this level where $1 \leq n \leq N-r+1$. Hence it remains $(N-n)$ tasks for the other $(r-1)$ remaining levels. The number of ways to distribute these $(N-n)$ tasks among the $(r-1)$ levels such that no level is left empty (i.e., the number of process graphs with $(r-1)$ levels and $(N-n)$ tasks), is given by :

$$\binom{\binom{(N-n)-(r-1)}{(r-1)-1} + (r-1) - 1}{(r-1) - 1} = \binom{N-n-1}{r-2}$$

This number also represents the number of possibilities for level k , $1 \leq k \leq r$ to have n tasks out of a total of $\binom{N-1}{r-1}$ possibilities. The probability P (n tasks at level k) is therefore the ratio between them.

Notice that level k can be any level, that is $1 \leq k \leq r$. The minimum number of tasks any level can have is one, and therefore the maximum number of tasks any level can have is $(N-r+1)$.

Proposition 2.2

For a fixed number of tasks per job, say N , the probability that a randomly chosen process graph has r levels, $1 \leq r \leq N$, is given by :

$$P \{\text{process graph has } r \text{ levels}\} = \frac{\binom{N-1}{r-1}}{2^{N-1}}$$

This conditional probability of having a process graph with r levels given that the number of tasks is fixed to N , is the binomial distribution $b(r-1, N-1, \frac{1}{2})$.

Proof

Since the total number of process graphs with N tasks is readily given by $\sum_{r=1}^N \binom{N-1}{r-1} = \sum_{r=0}^{N-1} \binom{N-1}{r} = 2^{N-1}$, and since the total number of process graphs with r levels and N tasks is $\binom{N-1}{r-1}$, it follows that:

$$P \{\text{process graph has } r \text{ levels} \mid N \text{ tasks in it}\} = \frac{\binom{N-1}{r-1}}{2^{N-1}} = \binom{N-1}{r-1} \left[\frac{1}{2}\right]^{r-1} \left[\frac{1}{2}\right]^{N-r}$$

$$= b(r-1, N-1, \frac{1}{2})$$

Let \bar{r} and σ_r^2 denote respectively the mean number of levels and the variance of the number of levels in a randomly chosen process graph comprising N tasks. From Proposition 2.2, we

readily have:

$$\bar{r} = \frac{N-1}{2} \quad \text{and} \quad \sigma_r^2 = \frac{N-1}{4}$$

2.2 Fixed Process Graphs

Throughout this section, the process graph is assumed to have a fixed description, the same for all jobs. All jobs have the same process graph with a fixed number of tasks, N , and a fixed number of levels, r . Moreover, if we let $J(n_1, n_2, \dots, n_r)$ be the description of the process graph where n_i is the number of tasks at level i in the process graph, then we require that all jobs have the same process graph description. First, we provide an expression for the probability density function of the number of occupied processors. Then, we derive a closed form expression of the Z-transform of the distribution of the number of occupied processors. Closed form expressions for the average and the variance of the number of occupied processors will also be derived.

2.2.1 Distribution of the Number of Occupied Processors

Let I represent the closed time interval $[t - \bar{X}r, t]$ as depicted in Figure 2.2. Interval I is divided into r slots of width \bar{X} each (recall that \bar{X} represents the constant service time per task). Jobs arriving before time $(t - \bar{X}r)$ complete execution before time t , and hence do not occupy any processor at time t . On the other hand, a job arriving in slot i of the interval I , participates at time t with the tasks of its i th level. This job will then occupy n_i processors at time t . Hence,

$$P\{\bar{Y} = y\} = \sum_{s.t. \sum_{i=1}^r k_i n_i = y} P\{k_1 \text{ jobs arrived in slot 1}, \dots, k_r \text{ jobs arrived in slot } r\}$$

Since the job arrival process is Poisson with aggregate rate λ , and the slot width is \bar{X} , we obtain:

$$P\{\bar{Y} = y\} = e^{-\lambda\bar{X}} \sum_{s.t. \sum_{i=1}^r k_i n_i = y} \frac{(\lambda\bar{X})^{\sum_{i=1}^r k_i}}{k_1! \cdots k_i! \cdots k_r!} \quad (2.2)$$

Although equation (2.2) is not an explicit expression, it allows to numerically compute the probability density function of the number of occupied processors in the system. In fact, since $\sum_{i=1}^r k_i \leq y$, the number of possibilities $(k_1, \dots, k_i, \dots, k_r)$ such that $\sum_{i=1}^r k_i n_i = y$ is

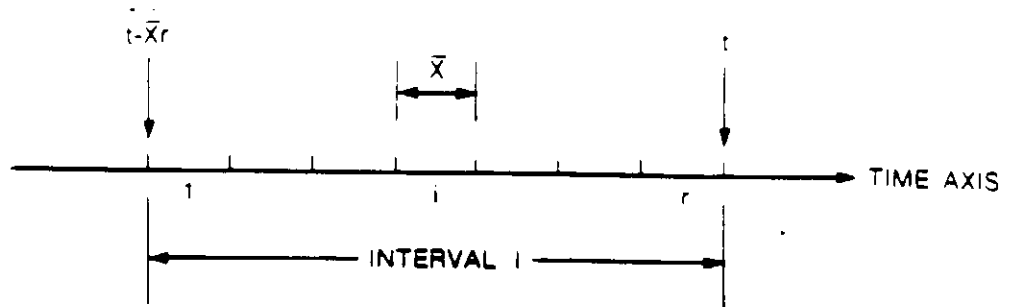


Figure 2.2: Analysis of Fixed Process Graphs

at most equal to the number of ways to distribute y objects among r cells. Hence for a small y , not too many computations are involved in computing $P\{\bar{Y} = y\}$. If the job arrival rate, λ , is small, only $P\{\bar{Y} = y\}$ for small values of y are of interest (of any significance). For a large value of λ however, we can see the system as composed of several independent subsystems, say j subsystems, each one comprising an infinite number of processors, and having a Poisson job arrival process with average rate $\frac{\lambda}{j}$. Now the probability density function of the number of occupied processors can be computed using equation (2.2) and the following:

$$P\{\bar{Y} = y\} = \sum_{\substack{j \\ \sum_{k=1}^j y_k = y}} P\{\bar{Y}_k = y_k\}$$

where \bar{Y}_k , $k=1, \dots, j$ represents the number of occupied processors in the k th subsystem.

Now, we proceed to compute the Z-transform of the number of occupied processors in the system. Let \bar{Y}_i , $i=1, \dots, r$ be the random variable counting the number of processors occupied at time t by the jobs that arrived in slot i of the interval I , we have :

$$\bar{Y} = \sum_{i=1}^r \bar{Y}_i \quad (2.3)$$

since

$$P\{\bar{Y}_i = x\} = \sum_{k=0}^{\infty} P\{\bar{Y}_i = x \mid k_i \text{ jobs arrived in slot } i\} \cdot P\{k_i \text{ jobs arrived in slot } i\}$$

and,

$$P[\bar{Y}_i = x \mid k_i \text{ jobs arrived in slot } i] = \begin{cases} 1 & \text{iff } x = k_i \\ 0 & \text{otherwise} \end{cases}$$

we obtain:

$$P[\bar{Y}_i = k_i \mid n_i] = P[k_i \text{ jobs arrived in slot } i]$$

and since the job arrival process is Poisson with rate λ , and the slot width is \bar{X} (the same as the task service time), we obtain:

$$P[\bar{Y}_i = k_i \mid n_i] = \frac{(\lambda \bar{X})^{k_i}}{k_i!} e^{-\lambda \bar{X}} \quad (2.4)$$

Let us define by $Y_i(Z)$ the Z-transform of \bar{Y}_i ; that is :

$$Y_i(Z) \triangleq \sum_{j=0}^{\infty} P[\bar{Y}_i = j] Z^j \quad i=1, \dots, r$$

From equation (2.4) we get:

$$Y_i(Z) = \sum_{k=0}^{\infty} P[\bar{Y}_i = k \mid n_i] Z^{k n_i} = \sum_{j=0}^{\infty} \frac{(\lambda \bar{X})^j}{j!} e^{-\lambda \bar{X}} Z^{j n_i}$$

which amounts to:

$$Y_i(Z) = \exp \left\{ -\lambda \bar{X} (1 - z^{n_i}) \right\} \quad (2.5)$$

Since the job arrival process is Poisson with parameter λ , then the arrivals and the number of such arrivals in any slot i , $i=1, \dots, r$ are independent random variables. It follows that :

$$Y(Z) = \prod_{i=1}^r Y_i(Z)$$

using (2.5) we obtain :

$$Y(Z) = e^{-\lambda \bar{X} r} e^{\lambda \bar{X} \sum_{i=1}^r z^{n_i}} \quad (2.6)$$

Notice from (2.6) that $Y(0) = e^{-\lambda \bar{X} r} = P_0$, where $P_0 = P$ [no job arrivals in the interval I].

2.2.2 Average Number of Occupied Processors

We now proceed to derive the average number of occupied processors, we have $\bar{Y} = \frac{d}{dZ} Y(Z) \Big|_{z=1}$ and using equation (2.6), we get:

$$\begin{aligned} \bar{Y} &= e^{-\lambda \bar{X} r} \left\{ \frac{d}{dZ} \left[\lambda \bar{X} \sum_{i=1}^r Z^{n_i} \right] \cdot e^{\lambda \bar{X} \sum_{i=1}^r Z^{n_i}} \right\} \Big|_{z=1} = e^{-\lambda \bar{X} r} \left\{ \lambda \bar{X} \sum_{i=1}^r n_i z^{n_i-1} \cdot e^{\lambda \bar{X} \sum_{i=1}^r Z^{n_i}} \right\} \Big|_{z=1} \\ &= e^{-\lambda \bar{X} r} \left\{ \left[\lambda \bar{X} \sum_{i=1}^r n_i \right] \cdot e^{\lambda \bar{X} r} \right\} = \lambda \bar{X} \sum_{i=1}^r n_i \end{aligned}$$

Therefore, we have:

$$\bar{Y} = \lambda \bar{X} N \tag{2.7}$$

2.2.3 Variance of the Number of Occupied Processors

Now, we proceed to derive the variance of the number of occupied processors, we have $\frac{d^2}{dZ^2} Y(Z) \Big|_{z=1} = \bar{Y}^2 - \bar{Y}$. Using equation (2.6), we obtain:

$$\begin{aligned} \frac{d^2}{dZ^2} Y(Z) &= \frac{d}{dZ} \left[\frac{d}{dZ} Y(Z) \right] \\ &= e^{-\lambda \bar{X} r} \left\{ \lambda \bar{X} \sum_{i=1}^r n_i (n_i - 1) Z^{n_i - 2} \right\} e^{\lambda \bar{X} \sum_{i=1}^r Z^{n_i}} + e^{-\lambda \bar{X} r} \left\{ \lambda \bar{X} \sum_{i=1}^r n_i Z^{n_i - 1} \right\}^2 \cdot e^{\lambda \bar{X} \sum_{i=1}^r Z^{n_i}} \end{aligned}$$

and by taking $Z=1$, we obtain:

$$\begin{aligned} \bar{Y}^2 - \bar{Y} &= e^{-\lambda \bar{X} r} \left[\lambda \bar{X} \sum_{i=1}^r n_i (n_i - 1) \right] e^{\lambda \bar{X} r} + e^{-\lambda \bar{X} r} \left[\lambda \bar{X} \sum_{i=1}^r n_i \right]^2 e^{\lambda \bar{X} r} \\ &= \lambda \bar{X} \sum_{i=1}^r n_i (n_i - 1) + (\lambda \bar{X} N)^2 = \lambda \bar{X} \sum_{i=1}^r n_i^2 - \lambda \bar{X} N + (\lambda \bar{X} N)^2 \end{aligned}$$

Since $\sigma_Y^2 = \bar{Y}^2 - \bar{Y} = \frac{d^2}{dZ^2} Y(Z) \Big|_{z=1} + \bar{Y} - \bar{Y}^2$, and using equation (2.7), we get:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} \sum_{i=1}^r n_i^2 \quad (2.8)$$

An upper bound for $\sigma_{\bar{Y}}^2$ can be found by using the loose inequality $\sum_i x_i^2 \leq \left[\sum_i x_i \right]^2$. We get from equation (2.8):

$$\sigma_{\bar{Y}}^2 \leq \lambda \bar{X} N^2 \quad \text{or equivalently} \quad \sigma_{\bar{Y}}^2 \leq N \bar{Y}$$

Indeed *, this forms a tight upper bound, for it is accomplished by the process graph comprising only one level (i.e., $r=1$). On the other hand, the process graph with N levels (i.e., each level has one task) provides the lower bound for the variance of the number of occupied processors; that is $\sigma_{\bar{Y}}^2 = \lambda \bar{X} N$. Hence, we have:

$$\lambda \bar{X} N \leq \sigma_{\bar{Y}}^2 \leq \lambda \bar{X} N^2 \quad \text{or equivalently} \quad \bar{Y} \leq \sigma_{\bar{Y}}^2 \leq N \bar{Y} \quad (2.9)$$

In the above analysis, no restriction was assumed as to the choice of the shape of the PG(N,r). Equations (2.7), (2.8), and (2.9) are valid for any given shape of the process graph PG(N,r). The only restriction is that all jobs have the same process graph description $J(n_1, n_2, \dots, n_r)$.

* Formerly, to obtain an upper bound on $\sigma_{\bar{Y}}^2$, we have to solve the following maximization problem:

$$\begin{aligned} & \text{maximize} \left\{ \sum_{i=1}^r n_i^2 \right\} \\ & \text{subject to: } \sum_{i=1}^r n_i = N \quad \text{for } 1 \leq r \leq N \end{aligned}$$

For a given value of r , the solution to this maximization problem is simply $n_k = N - r + 1$, $n_j = 1$, $j=1, \dots, N$ and $j \neq k$. It is also easy to see that $r=1$ gives the upper bound. Likewise, to obtain a lower bound on $\sigma_{\bar{Y}}^2$, we have to solve the following minimization problem:

$$\begin{aligned} & \text{minimize} \left\{ \sum_{i=1}^r n_i^2 \right\} \\ & \text{subject to: } \sum_{i=1}^r n_i = N \quad \text{for } 1 \leq r \leq N \end{aligned}$$

For a given value of r , the solution to this minimization problem is simply $n_i = \frac{N}{r}$ for all $i=1, \dots, r$. It is easy to see that $r=N$ gives the lower bound.

Examples

Let us take *the discrete well shaped diamond* process graph denoted by $PG(r)$ (as a function of r only); examples of which are depicted in Figure 2.3 and Figure 2.4. Two cases may be distinguished depending on the value of r being odd or even.

Case of an odd number of levels

Figure 2.3 gives some examples of such a process graph. In this case, the number of tasks at level i , $i=1, \dots, r$ is given by:

$$n_i = \begin{cases} i & 1 \leq i \leq \frac{r+1}{2} \\ r-i+1 & \frac{r+1}{2} < i \leq r \end{cases}$$

and hence, the total number N of tasks in the well shaped process graph with an odd number of levels is :

$$N = \sum_{i=1}^r n_i = \left[1+2+\dots+\frac{r+1}{2} \right] + \left[\frac{r-1}{2}+\dots+2+1 \right] = \frac{1}{2} \frac{r+1}{2} \frac{r+3}{2} + \frac{1}{2} \frac{r-1}{2} \frac{r+1}{2}$$

which finally yields:

$$N = \left[\frac{r+1}{2} \right]^2 \quad (2.10)$$

replacing N in equation (2.7) by the above expression, yields:

$$\bar{Y} = \frac{\lambda \bar{X} \left[\frac{r+1}{2} \right]^2}{4} \quad (2.11)$$

On the other hand, using the known identity $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$, we get :

$$\sum_{i=1}^r n_i^2 = 2 \sum_{i=1}^{\frac{r-1}{2}} i^2 + \left[\frac{r+1}{2} \right]^2 \left[\frac{r+1}{2} \right]^2 \left[\frac{r}{3} + 1 \right] - \frac{r}{3} \left[\frac{r+1}{2} \right]^2$$

Now, by using equation (2.10) we obtain :

$$\sum_{i=1}^r n_i^2 = N \frac{r+3}{3} - \frac{r}{3} \sqrt{N}$$

which along with equation (2.8) gives:

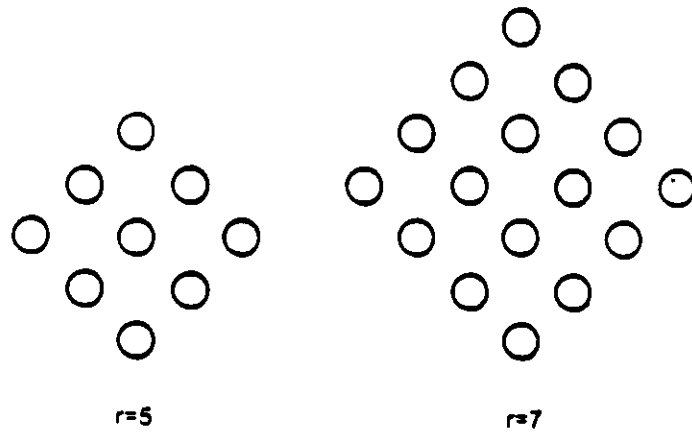


Figure 2.3: Discrete Well Shaped Process Graph with Odd Number of Levels

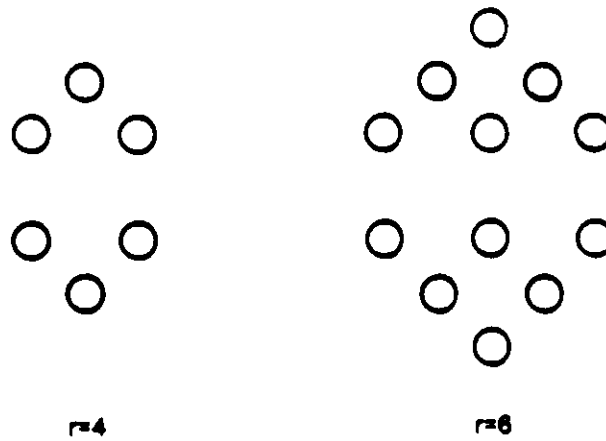


Figure 2.4: Discrete Well Shaped Process Graph with Even Number of Levels

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} N \frac{r+3}{3} - \frac{\lambda \bar{X} r}{3} \sqrt{N} \quad (2.12)$$

Case of an even number of levels

Figure 2.4 gives some examples of such a process graph. In this case, the number of tasks at level i , $i=1, \dots, r$ is given by:

$$n_i = \begin{cases} i & 1 \leq i \leq \frac{r}{2} \\ r-i-1 & \frac{r}{2}+1 \leq i \leq r \end{cases}$$

and hence, the total number N of tasks in the well shaped process graph with an even number of levels is :

$$N = \sum_{i=1}^r n_i = \left[1+2+\dots+\frac{r}{2} \right] + \left[\frac{r}{2}+(\frac{r}{2}-1)+\dots+2+1 \right]$$

which amounts to :

$$N = \frac{r(r+2)}{4} \quad (2.13)$$

replacing N in equation (2.7) by the above expression, yields:

$$\bar{Y} = \frac{\lambda \bar{X} r(r+2)}{4} \quad (2.14)$$

On the other hand, using the known identity $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$, we get:

$$\sum_{i=1}^r n_i^2 = 2 \sum_{i=1}^{\frac{r}{2}} i^2 = \frac{r(r+2)}{4} \frac{r+1}{3}$$

now, using equation (2.13) along with equation (2.8), we obtain:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} N \frac{r+1}{3} \quad (2.15)$$

or equivalently,

$$\sigma_{\bar{Y}}^2 = \bar{Y} \frac{r+1}{3}$$

Equations (2.11) and (2.14) provide explicit expressions of the average number of occupied processors as a function of the number of levels, for well shaped process graphs having respectively an odd and an even number of levels. Equations (2.12) and (2.15) on the other hand,

ascertain their variances. Other examples of interest are studied in [Belg85].

2.3 Semi Random Process Graphs with two or more Levels

We now proceed to analyze the case of semi random process graphs. Each job has a process graph with a fixed number of tasks, N , and a fixed number of levels, $r \geq 2$. However, jobs do not necessarily have the same process graph description $J(n_1, n_2, \dots, n_r)$ where n_i , $i=1, \dots, r$ denotes the number of tasks at level i in the process graph. We shall first derive closed form expressions of the probability density function and the Z-transform of the number of occupied processors in the system. Closed form expressions for the average and the variance of the number of occupied processors will also be derived.

2.3.1 Distribution of the Number of Occupied Processors

Let I define the interval of time $(t - \bar{X}r, t]$, \bar{Y}_k denote the random variable counting the total number of occupied processors at time t given that k jobs arrived in the interval I .

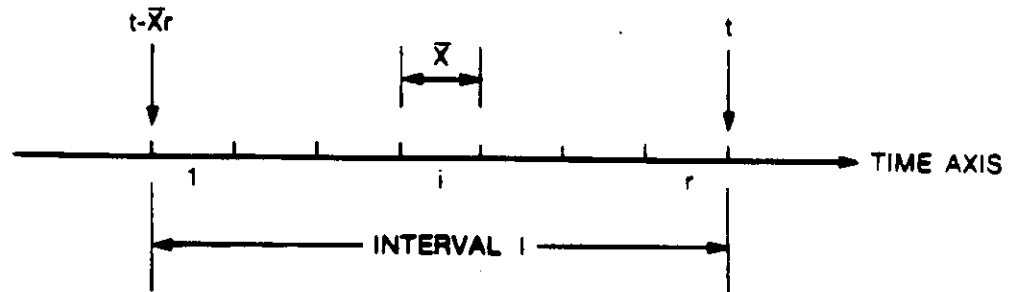


Figure 2.5 Analysis of Semi Random Process Graphs

It is easy to see from Figure 2.5 that all jobs which arrived before time $(t - \bar{X}r)$ had finished before time t . Such jobs will not occupy any processor at time t . Those jobs which occupy some processors at time t , are the jobs that arrive in the interval I . Therefore :

$$P\{\bar{Y} = y\} = \sum_{k=0}^{\infty} P\{\bar{Y}_k = y\}P\{k \text{ jobs arrived in } I\}$$

Since, if no job arrived in the interval I , then no processor will be occupied at time t , it follows that:

$$P[\bar{Y} = 0] = e^{-\lambda \bar{x}_r} \quad (2.16)$$

On the other hand, if k jobs arrived in the interval I , then at least k processors will be occupied at time t . The number of job arrivals in the interval I and the number of occupied processors at time t are therefore related by the following double sided inequality:

$$k \leq y \leq k(N-r+1)$$

and hence for $y \geq 1$, we have:

$$P[\bar{Y} = y] = \sum_{k=1}^y P[\bar{Y}_k = y] P[k \text{ jobs arrived in } I] \quad y \geq 1 \quad (2.17)$$

To explicitly express the probability density function of the number of occupied processors in the system, we need to evaluate $P[\bar{Y}_k = y]$ for the values $1 \leq k \leq y$. We have:

$$P[\bar{Y}_k = y \text{ s.t. } 1 \leq k \leq y] = \sum_{\substack{\text{s.t. } \sum_{j=1}^k n_j = y \\ 1 \leq n_j \leq N-r+1, \forall j=1, \dots, k}} P[\text{job } j \text{ participates with } n_j]$$

Proposition 2.1 readily gives the probability of having n tasks at a given level k given that we have N tasks and r levels in such process graph $n=1, \dots, N-r+1$ and $k=1, \dots, r$. We then obtain:

$$P[\bar{Y}_k = y \text{ s.t. } 1 \leq k \leq y] = \left[\frac{1}{\binom{N-1}{r-1}} \right]^k \sum_{\substack{\text{s.t. } \sum_{j=1}^k n_j = y \\ 1 \leq n_j \leq N-r+1, \forall j=1, \dots, k}} \prod_{j=1}^k \binom{N-n_j-1}{r-2} \quad (2.18)$$

Let us define the following quantities: $L=y-k$, $M=N-2$, and $R=r-2$. Since $r \geq 2$, $N \geq 2$, and $k \leq y \leq k(N-r+1)$, it follows that $M \geq 0$, $R \geq 0$, and $0 \leq L \leq (k-1)(N-r+1)$. Using these quantities, equation (2.18) may be rewritten as:

$$P[\bar{Y}_k = y \text{ s.t. } 1 \leq k \leq y] = \left[\frac{1}{\binom{N-1}{r-1}} \right]^k \sum_{\substack{\text{s.t. } \sum_{j=1}^k n_j = L \\ n_j \geq 0, \forall j=1, \dots, k}} \prod_{j=1}^k \binom{M-n_j}{R}$$

where by definition $\binom{n}{i} \triangleq 0$ whenever $n < i$. Let the bivariate function $a_{k,l}$ be defined as:

$$a_{k,l} = \sum_{\substack{\text{s.t. } \sum_{j=1}^k n_j = l}} \prod_{j=1}^k \binom{M-n_j}{R}$$

which amounts to :

$$a_{k,l} = \sum_{i=0}^l \left\{ \binom{M-i}{R} \sum_{\substack{s.t. \sum_{j=1}^{k-1} n_j = l-i \\ \forall n_j \geq 0, j=1, \dots, k-1}} \prod_{j=1}^{k-1} \binom{M-n_j}{R} \right\}$$

Notice that the inner summation is exactly $a_{k-1, l-i}$, and hence we obtain the following recurrence relation on the bivariate function $a_{k,l}$:

$$a_{k,l} = \sum_{i=0}^l \binom{M-i}{R} a_{k-1, l-i} \quad 2 \leq k \leq y \quad (2.19)$$

with the following boundary condition for $k=1$:

$$a_{1,l} = \binom{M-l}{R} \quad \forall l \geq 0 \quad (2.20)$$

since this is the case where only one job arrived in the interval I. Finally, using equations (2.17), (2.19), and (2.20), and the fact that the job arrival process is Poisson with aggregate rate λ , we obtain:

$$P[\bar{Y} = y] = \sum_{k=1}^y \left[\frac{\lambda \bar{X}_r}{\binom{M+1}{R+1}} \right]^k \frac{1}{k!} e^{-\lambda \bar{X}_r} \sum_{i=0}^L \binom{M-i}{R} a_{k-1, L-i} \quad y \geq 1 \quad (2.21)$$

Equation (2.16) and equation (2.21) provide then the probability density function of the number of occupied processors in the system. In the sequel, we derive the Z-transform of the number of occupied processors, denoted hereafter by $Y(Z)$. Since by definition $Y(Z) \triangleq \sum_{y=0}^{\infty} P[\bar{Y} = y] Z^y$, we can use equations (2.21) and (2.16) to derive such a Z-transform. In the sequel, however, we shall take a rather simpler and more elegant way. Let $\bar{X}_i, i=1, \dots, k$ denote the random variable counting the number of occupied processors at time t , and by the i th job arriving in the interval I. Since only the jobs that have arrived in the interval I will occupy some processors at time t , we have:

$$\bar{Y}_k = \sum_{i=1}^k \bar{X}_i$$

and from Proposition 2.1, we already have:

$$P[\bar{X}_i = n_i] = \frac{\binom{N-n_i-1}{r-2}}{\binom{N-1}{r-1}} \quad \forall n_i = 1, \dots, N-r+1$$

Let $X_i(Z)$ denote the Z-transform of the random variable \bar{X}_i , and $Y_k(Z)$ denote the Z-transform of the random variable \bar{Y}_k . We therefore have:

$$X_i(Z) \triangleq \sum_{j=1}^{N-r+1} P[\bar{X}_i=j] Z^j$$

which amounts to:

$$X_i(Z) = \frac{1}{\binom{N-1}{r-1}} \sum_{j=0}^{N-r} \binom{N-j-2}{r-2} Z^{j+1} \quad (2.22)$$

Since the random variables \bar{X}_i 's, $i=1, \dots, k$ are independent and identically distributed, we may drop the index i in $X_i(Z)$, and we get :

$$Y_k(Z) = [X(Z)]^k$$

and since the job arrival process is Poisson with an average rate λ , we get :

$$Y(Z) = \sum_{k=0}^{\infty} [X(Z)]^k \frac{(\lambda \bar{X} r)^k}{k!} e^{-\lambda \bar{X} r}$$

which amounts to:

$$Y(Z) = e^{-\lambda \bar{X} r} [1 - X(Z)] \quad (2.23)$$

Finally, by using the expression of $X(Z)$ as given by equation (2.22) into equation (2.23), we obtain the following expression for the Z-transform $Y(Z)$ of the number of occupied processors:

$$Y(Z) = \exp \left\{ -\lambda \bar{X} r \left[1 - \frac{Z \sum_{j=0}^{N-r} \binom{N-j-2}{r-2} Z^j}{\binom{N-1}{r-1}} \right] \right\} \quad (2.24)$$

2.3.2 Average Number of Occupied Processors

We now proceed to derive the average number of occupied processors in the case of semi random process graphs. We have $\bar{Y} = \frac{d}{dz} Y(Z) \Big|_{z=1}$, where $Y(Z)$ is as given by equation (2.24). Let $b(Z)$ and a be defined as:

$$b(Z) = \sum_{i=0}^{M-R} \binom{M-i}{R} Z^i \quad (2.25)$$

and,

$$a = \frac{\lambda \bar{X} r}{\binom{M+1}{R+1}} \quad (2.26)$$

Therefore, the Z-transform of the number of occupied processors given by equation (2.24) can be rewritten as follows:

$$Y(Z) = \exp \left\{ -\lambda \bar{X} r + a Z b(Z) \right\} \quad (2.27)$$

from the above equation, the average number of occupied processors is then given by:

$$\bar{Y} = ab(1) + a \frac{d}{dZ} b(Z) \Big|_{z=1} \quad (2.28)$$

Let us first compute $b(1)$ and $b'(1) = \frac{d}{dZ} b(Z) \Big|_{z=1}$. From equation (2.25), we get:

$$b'(1) = \sum_{i=0}^{M-R} i \binom{M-i}{R} \quad (2.29)$$

Now let us compute $b(1)$, from equation (2.25), we get :

$$b(1) = \sum_{i=0}^{M-R} \binom{M-i}{R} = \binom{M}{R} + \binom{M-1}{R} + \binom{M-2}{R} + \dots + \binom{R}{R}$$

which amounts to:

$$b(1) = \sum_{i=0}^{M-R} \binom{R+i}{R}$$

Consider now the function $\beta(x)$ defined by :

$$\beta(x) = (1+x)^R + (1+x)^{R+1} + \dots + (1+x)^M$$

since the coefficient of x^R in $(1+x)^{R+i}$ is $\binom{R+i}{R}$, it follows that the coefficient of x^R in $\beta(x)$ is exactly $b(1)$. First, let us rewrite the function $\beta(x)$ as :

$$\beta(x) = \frac{(1+x)^{M+1} - (1+x)^R}{x}$$

Thus, the coefficient of x^R in $\beta(x)$ is $\binom{M+1}{R+1}$, and therefore we obtain :

$$b(1) = \begin{Bmatrix} M+1 \\ R+1 \end{Bmatrix} \quad (2.30)$$

by replacing M and R by their respective values, it can easily be seen that $X_i(Z)|_{z=1} = 1$. Now, by using equations (2.28), (2.29), and (2.30) we get:

$$\frac{d}{dZ} Y(Z)|_{z=1} = \lambda \bar{X}_r \left[1 + \frac{b'(1)}{\begin{Bmatrix} M+1 \\ R+1 \end{Bmatrix}} \right] \quad (2.31)$$

Now, let us compute $b'(1)$, we have:

$$\begin{aligned} b'(1) &= \sum_{i=0}^{M-R} i \begin{Bmatrix} M-i \\ R \end{Bmatrix} = 0 \begin{Bmatrix} M \\ R \end{Bmatrix} + 1 \begin{Bmatrix} M-1 \\ R \end{Bmatrix} + \dots + i \begin{Bmatrix} M-i \\ R \end{Bmatrix} + \dots + (M-R) \begin{Bmatrix} R \\ R \end{Bmatrix} \\ &= \sum_{i=0}^{M-R} (M-R-i) \begin{Bmatrix} R+i \\ R \end{Bmatrix} = (M-R) \sum_{i=0}^{M-R} \begin{Bmatrix} R+i \\ R \end{Bmatrix} - \sum_{i=0}^{M-R} i \begin{Bmatrix} R+i \\ R \end{Bmatrix} \\ &= (M-R)b(1) - \sum_{i=0}^{M-R} i \begin{Bmatrix} R+i \\ R \end{Bmatrix} \end{aligned}$$

which by using equation (2.30), amounts to:

$$b'(1) = (M-R) \begin{Bmatrix} M+1 \\ R+1 \end{Bmatrix} - \sum_{i=0}^{M-R} i \begin{Bmatrix} R+i \\ R \end{Bmatrix} \quad (2.32)$$

Now we proceed to derive a closed form expression for the summation in the right hand side of the above equation. Let us define the following :

$$a_n = \sum_{i=0}^n i \begin{Bmatrix} R+i \\ R \end{Bmatrix} \quad n \geq 0 \quad (2.33)$$

which then results in the following recurrence relation:

$$a_n = a_{n-1} + n + \begin{Bmatrix} R+n \\ R \end{Bmatrix} \quad n \geq 1 \quad (2.34)$$

with the boundary condition:

$$a_0 = 0 \quad (2.35)$$

Define the generating function of a_n by $A(x)$; that is $A(x) \triangleq \sum_{n=0}^{\infty} a_n x^n$. Thus, equation (2.34) yields:

$$\sum_{n=1}^{\infty} a_n x^n = \sum_{n=1}^{\infty} a_{n-1} x^n + \sum_{n=1}^{\infty} n \binom{R+n}{R} x^n$$

That is,

$$A(x) - a_0 = xA(x) + x \sum_{n=0}^{\infty} n \binom{R+n}{R} x^{n-1} = xA(x) + x \frac{d}{dx} \left\{ \sum_{n=1}^{\infty} \binom{R+n}{R} x^n \right\}$$

and using equation (A.5) of Appendix (A), yields:

$$A(x) - a_0 = xA(x) + x \frac{d}{dx} \left\{ \frac{1}{(1-x)^{R+1}} - 1 \right\} = xA(x) + x + \frac{R+1}{(1-x)^{R+2}}$$

which finally amounts to:

$$A(x) = \frac{(R+1)x}{(1-x)^{R+3}}$$

We can rewrite A(x) as :

$$A(x) = \frac{R+1}{(1-x)^{R+3}} - \frac{R+1}{(1-x)^{R+2}} \quad (2.36)$$

Equation (2.36) can be inverted [Klei75] to give :

$$a_n = (R+1) \binom{n+R+2}{R+2} - (R+1) \binom{n+R+1}{R+1}$$

and, by using the well known formula $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$, we get :

$$\binom{n+R+2}{R+2} = \binom{n+R+1}{R+2} + \binom{n+R+1}{R+1}$$

hence, we obtain :

$$a_n = (R+1) \binom{n+R+1}{R+2} \quad (2.37)$$

Now, using equation (2.32) and equation (2.37), we get:

$$\frac{d}{dz} b(Z) \Big|_{z=1} = (M-R) \binom{M+1}{R+1} - (R+1) \binom{M+1}{R+2} \quad (2.38)$$

and using equations (2.31) and (2.38), we obtain :

$$\frac{d}{dz} Y(Z) \Big|_{z=1} = \lambda \bar{X}_r \frac{M+2}{R+2}$$

Finally, since $M=N-2$ and $R=r-2$ we obtain :

$$\bar{Y} = \lambda \bar{X} N \quad (2.39)$$

2.3.3 Variance of the Number of Occupied Processors

Now, we proceed to derive the variance of the number of occupied processors, denoted by σ_Y^2 . We have:

$$\sigma_Y^2 = \frac{d^2 Y(Z)}{dZ^2} \Big|_{z=1} + \frac{dY(Z)}{dZ} \Big|_{z=1} - \left[\frac{dY(Z)}{dZ} \Big|_{z=1} \right]^2$$

Using equation (2.27), we obtain:

$$\frac{d^2 Y(Z)}{dZ^2} \Big|_{z=1} = 2ab'(1) + ab''(1) + [ab(1) + ab'(1)]^2 \quad (2.40)$$

To evaluate the above expression, we need to compute $b''(1) = \frac{d^2 b(Z)}{dZ^2} \Big|_{z=1}$. From equation (2.25), we obtain :

$$\begin{aligned} b''(1) &= \sum_{i=0}^{M-R} i(i-1) \binom{M-i}{R} \\ &= 0 \cdot \binom{M}{R} + 0 \cdot \binom{M-1}{R} + 2 \cdot \binom{M-2}{R} + 6 \cdot \binom{M-3}{R} + \dots + (M-R)(M-R-1) \cdot \binom{R}{R} \\ &= \sum_{i=0}^{M-R} (M-R-i)(M-R-i-1) \binom{R+i}{R} \end{aligned}$$

which can be rewritten as:

$$b''(1) = (M-R)(M-R-1) \sum_{i=0}^{M-R} \binom{R+i}{R} - 2(M-R) \sum_{i=0}^{M-R} i \binom{R+i}{R} + \sum_{i=0}^{M-R} i(i+1) \binom{R+i}{R} \quad (2.41)$$

We then need to compute $\sum_{i=0}^{M-R} i(i+1) \binom{R+i}{R}$. Let:

$$a_n = \sum_{i=0}^n i(i+1) \binom{R+i}{R} \quad n \geq 0 \quad (2.42)$$

which results into the following recurrence relation :

$$a_n = a_{n-1} + n(n+1) \binom{R+n}{n} \quad n \geq 1 \quad (2.43)$$

with the boundary condition

$$a_0 = 0 \quad (2.44)$$

Let $A(x)$ define the generating function of a_n ; that is $A(x) \triangleq \sum_{n=0}^{\infty} a_n x^n$. Therefore, equation (2.43) yields:

$$\sum_{n=1}^{\infty} a_n x^n = \sum_{n=1}^{\infty} a_{n-1} x^n + \sum_{n=1}^{\infty} n(n+1) \binom{R+n}{R} x^n$$

which gives :

$$A(x) = \frac{x}{1-x} \sum_{n=1}^{\infty} n(n+1) \binom{R+n}{R} x^{n-1} \quad (2.45)$$

On the other hand, we have:

$$\sum_{n=1}^{\infty} n(n+1) \binom{R+n}{R} x^{n-1} = \frac{d^2}{dx^2} \left\{ \sum_{n=1}^{\infty} \binom{R+n}{R} x^{n+1} \right\} = \frac{d^2}{dx^2} \left\{ x \sum_{n=1}^{\infty} \binom{R+n}{R} x^n \right\}$$

Now, using equation (A.5) of Appendix (A), we obtain:

$$\sum_{n=1}^{\infty} n(n+1) \binom{R+n}{R} x^{n-1} = \frac{d^2}{dx^2} \left\{ x \left[\frac{1}{(1-x)^{R+1}} - 1 \right] \right\} = \frac{R(1+R)x + 2(1+R)}{(1-x)^{R+3}}$$

hence equation (2.45) becomes:

$$A(x) = \frac{R(1+R)x^2 + 2(1+R)x}{(1-x)^{R+4}}$$

which can be rewritten as

$$A(x) = \frac{R(1+R)}{(1-x)^{R+2}} - \frac{2(1+R)^2}{(1-x)^{R+3}} + \frac{(1+R)(2+R)}{(1-x)^{R+4}} \quad (2.46)$$

The above equation can be easily inverted [Klei75] to obtain:

$$a_n = 2(1+R) \binom{R+n+1}{R+2} + (1+R)(2+R) \binom{R+n+1}{R+3} \quad n \geq 1 \quad (2.47)$$

Let us now return to the expression of $b''(1)$ given by equation (2.41). Using equations (2.30), (2.37), and (2.47), we obtain:

$$b''(1) = (M-R)(M-R-1) \binom{M+1}{R+1} - 2(M-R)(R+1) \binom{M+1}{R+2} \\ + 2(1+R) \binom{M+1}{R+2} + (1+R)(2+R) \binom{M+1}{R+3}$$

which after some algebra yields:

$$b''(1) = (M-R)(M-R-1) \binom{M+1}{R+1} + 2(1+R)(1+R-M) \binom{M+1}{R+2} + (1+R)(2+R) \binom{M+1}{R+3} \quad (2.48)$$

Now, let us return to the computation of $\sigma_{\bar{Y}}^2$; using equation (2.39) and equation (2.40), we obtain:

$$\sigma_{\bar{Y}}^2 = ab(1) + 3ab'(1) + ab''(1) \quad (2.49)$$

where a is given by equation (2.26), $b(1)$ is given by equation (2.30), $b'(1)$ is given by equation (2.38), and $b''(1)$ is given by equation (2.48). Using the facts that

$$\frac{\binom{M+1}{R+2}}{\binom{M+1}{R+1}} = \frac{M-R}{2+R} \quad \text{and} \quad \frac{\binom{M+1}{R+3}}{\binom{M+1}{R+1}} = \frac{(M-R)(M-R-1)}{(2+R)(3+R)}$$

and after some algebra, we obtain:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} r + \lambda \bar{X} r (M-R) \left\{ M-R+2 + \frac{1+R}{2+R} (2R-2M-1) + \frac{1+R}{3+R} (M-R-1) \right\}$$

replacing M and R by their respective values as a function of N and r , we obtain:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} r + \lambda \bar{X} r (N-r) \left\{ N-r+2 + \frac{r-1}{r} (2r-2N-1) + \frac{r-1}{r+1} (N-r-1) \right\} \quad (2.50)$$

Equation (2.50) provides then the variance of the number of occupied processors as a function of both the total number of tasks, N , and the number of levels, r . Moreover, from the previous section, we already know that the upper bound and the lower bound on $\sigma_{\bar{Y}}^2$ are obtained respectively by the process graph having just one level, and the process graph having N levels. Figure 2.6 depicts the variance $\sigma_{\bar{Y}}^2$ for the value $N=10$, and for $\lambda \bar{X}=1$, and as a function of the number of levels $1 \leq r \leq 10$. When $r=1$, we observe that the variance gets its highest value of 100 as expected by equation (2.9). For $r=10$ on the other hand, the variance gets its lowest value of 10. We purposely joined the points in Figure 2.6 by straight lines to shed light on the slope of the decrease in the variance when we move from $r=1$ to $r=10$. We observe that for small value of r , the decrease in the variance is very substantial, and as r approaches the number of tasks N , the decrease in the variance (respectively the increase in the slope) gets smaller.

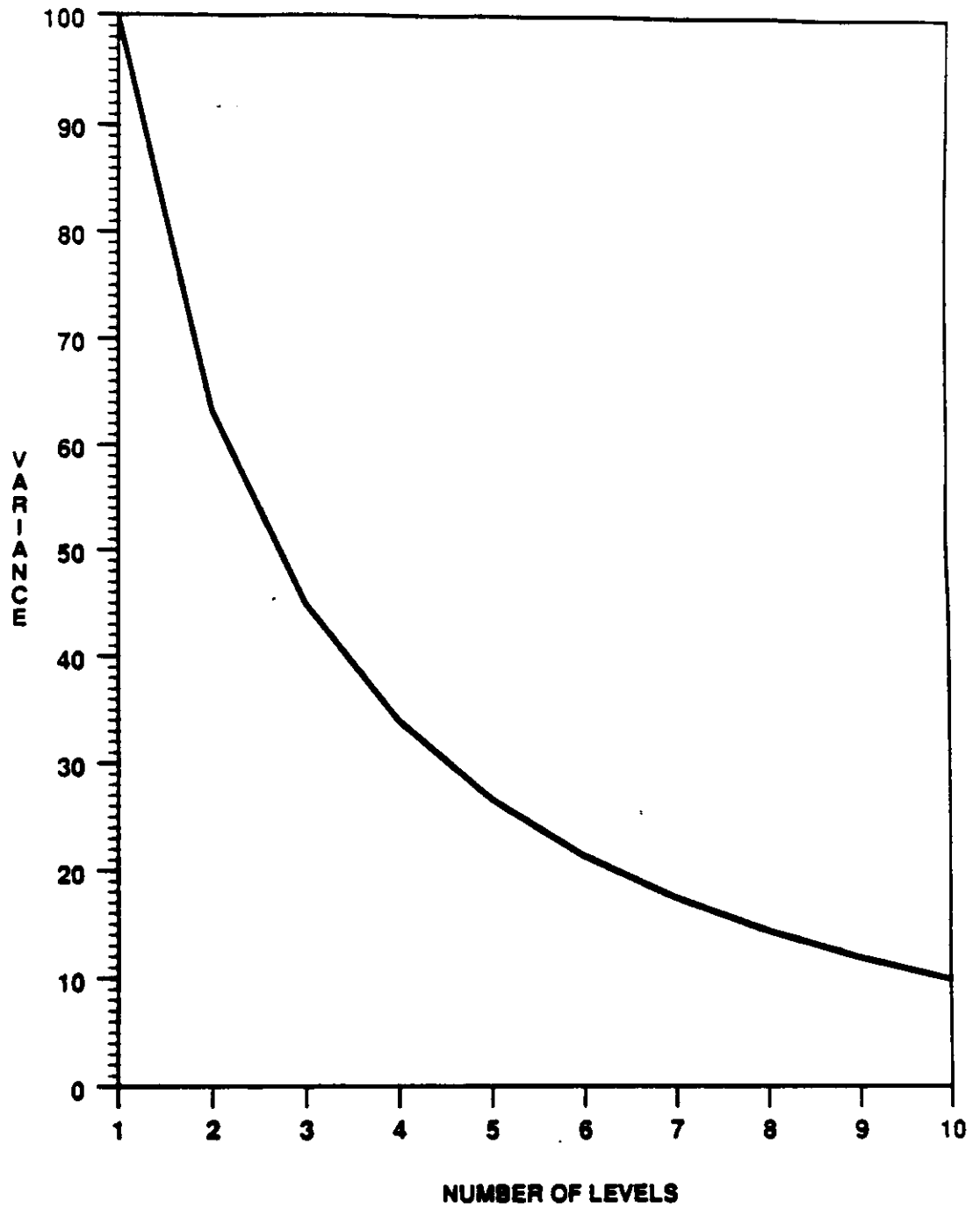


Figure 2.6: Variance of the Number of Occupied Processors versus the Number of Levels; for Semi Random Process Graphs

2.4 Random Process Graphs

We now proceed to analyze the case of random process graphs. Each job has a random process graph with a fixed number of tasks, N , and a random number of levels \bar{r} , $1 \leq \bar{r} \leq N$. Moreover, two jobs having the same number of levels do not necessarily have the same process graph description $J(n_1, \dots, n_i, \dots, n_r)$, where n_i , $i=1, \dots, r$ denotes the number of tasks at level i . We shall first derive a closed form expression of the Z-transform of the number of occupied processors in the system. Closed form expressions for the average and the variance of the number of occupied processors will then be derived.

2.4.1 Distribution of the Number of Occupied Processors

From Proposition 2.2, we know that the probability of an incoming job to have r levels follows the Binomial probability distribution given by :

$$P[\bar{r}=r] = \frac{\binom{N-1}{r-1}}{2^{N-1}} = b(r-1, N-1, \frac{1}{2})$$

Proposition 2.1, on the other hand, gives the probability of having n tasks at a given level k given that the job has r levels and N tasks, where $n=1, \dots, N-r+1$, $k=1, \dots, r$ and $r=1, \dots, N$.

From Figure 2.7, we see that any job that had arrived before time $(t-N\bar{X})$ would not participate (occupy any processor) at time t . On the other hand, a job arriving in the interval $I=(t-N\bar{X}, t)$ will participate if and only if it has enough levels. Let us divide the interval I into N equal slots of duration \bar{X} units of time each, equal to the processing time of one task. We number such slots by $1, 2, \dots, N$ (see Figure 2.7). It follows that a job arriving in slot number i , will occupy some processors at time t if and only if it has at least i levels, where $1 \leq i \leq N$.

Proposition 2.3

The probability of a job arriving in slot i , $1 \leq i \leq N$ to occupy some processors at time t is given by:

$$P[\text{job arriving in slot } i \text{ occupies some processors at time } t] = \frac{\sum_{j=0}^{i-1} \binom{N-1}{j}}{2^{N-1}}$$

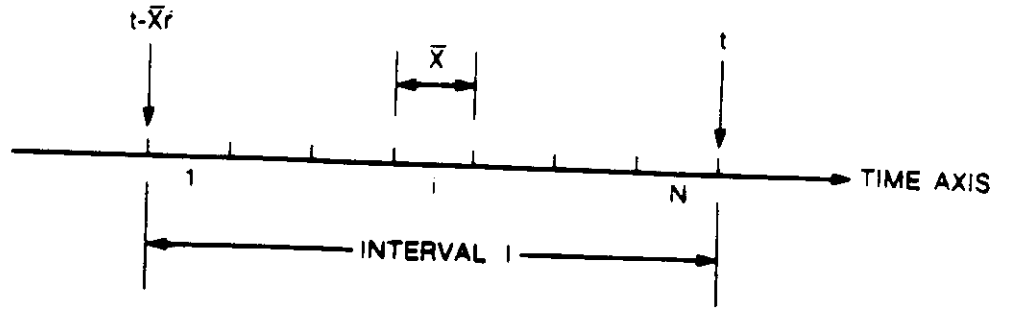


Figure 2.7 Analysis of Random Process Graphs

Proof

A job arriving in slot i , $1 \leq i \leq N$ occupies some processors at time t if and only if it has at least i levels. Hence, from Proposition 2.2 we get:

$$P[\text{job arriving in slot } i \text{ occupies some processors at time } t] = \sum_{r=i}^N \frac{\binom{N-1}{r-1}}{2^{N-1}} = \frac{1}{2^{N-1}} \sum_{j=1}^i \binom{N-1}{N-j}$$

and since $\binom{N-1}{N-j} = \binom{N-1}{j-1}$, we obtain:

$$\sum_{j=1}^i \binom{N-1}{N-j} = \sum_{j=1}^i \binom{N-1}{j-1} = \sum_{j=0}^{i-1} \binom{N-1}{j}$$

Let \bar{Y}_k denote the random variable counting the total number of occupied processors at time t given that k jobs arrived in the interval I , $\bar{Y}_{(k_1, \dots, k_N)}$ denote the random variable counting the total number of occupied processors at time t given that k_i jobs arrived in slot i , $i=1, \dots, N$, $\bar{X}_{i, k}$ denote the random variable counting the number of occupied processors at time t due to jobs that arrived in slot i and given that k_i jobs arrived in slot i , $i=1, \dots, N$, and $\bar{X}_{i, j}$ denote the random variable counting the number of occupied processors by the j th job that arrived in slot i , $i=1, \dots, N$. From these definitions, we readily have :

$$P[\bar{Y} = y] = \sum_{k=0}^{\infty} P[\bar{Y}_k = y] \frac{(\lambda \bar{X} N)^k}{k!} e^{-\lambda \bar{X} N} \quad (2.51)$$

$$P[\bar{Y}_k = y] = \sum_{\substack{N \\ \text{s.t. } \sum_{i=1}^N k_i = k}} P[\bar{Y}_{(k_1, \dots, k_N)} = y] = \frac{k!}{k_1! \dots k_N!} \left[\frac{1}{N} \right]^k \quad (2.52)$$

$$\bar{Y}_{(k_1, \dots, k_N)} = \sum_{i=1}^N \bar{X}_{i, k_i} \quad (2.53)$$

$$\bar{X}_{i, k_i} = \sum_{j=1}^{k_i} \bar{X}_{i, j} \quad (2.54)$$

Also, define the following Z-transforms of the above defined random variables.

$$Y_k(Z) \triangleq \sum_y P[\bar{Y}_k = y] Z^y$$

$$Y_{(k_1, \dots, k_N)}(Z) \triangleq \sum_y P[\bar{Y}_{(k_1, \dots, k_N)} = y] Z^y$$

$$X_{i, k_i}(Z) \triangleq \sum_x P[\bar{X}_{i, k_i} = x] Z^x$$

$$X_{i, j}(Z) \triangleq \sum_x P[\bar{X}_{i, j} = x] Z^x$$

Since the random variables $\bar{X}_{i, j}$'s are independent and identically distributed $\forall j=1, \dots, k_i$, then using equation (2.54), we obtain:

$$X_{i, k_i}(Z) = \prod_{j=1}^{k_i} X_{i, j}(Z)$$

Let us denote by $X_i(Z) \triangleq X_{i, j}(Z)$ since the $\bar{X}_{i, j}$'s are i.i.d. . Thus we get:

$$X_{i, k_i}(Z) = [X_i(Z)]^{k_i}$$

and from equation (2.53), we get:

$$Y_{(k_1, \dots, k_N)}(Z) = \prod_{i=1}^N [X_i(Z)]^{k_i}$$

and from equation (2.52) we obtain:

$$Y_k(Z) = \sum_{\substack{N \\ \text{s.t. } \sum_{i=1}^N k_i = k}} \frac{[X_1(Z)]^{k_1}}{k_1!} \dots \frac{[X_N(Z)]^{k_N}}{k_N!} k! \left[\frac{1}{N} \right]^k = \left[\frac{\sum_{i=1}^N X_i(Z)}{N} \right]^k$$

and finally using equation (2.51), we obtain:

$$Y(Z) = \sum_{k=0}^{\infty} \left[\frac{\sum_{i=0}^N X_i(Z)}{N} \right]^k \frac{(\lambda \bar{X} N)^k}{k!} e^{-\lambda \bar{X} N}$$

which amounts to:

$$Y(Z) = e^{-\lambda \bar{X} N} \cdot e^{\lambda \bar{X} \sum_{i=1}^N X_i(Z)} \quad (2.55)$$

2.4.2 Average Number of Occupied Processors

We now proceed to derive the average number of occupied processors in the case of random process graphs. We have $\bar{Y} = \frac{d}{dz} Y(Z) \Big|_{z=1}$, where the Z-transform $Y(Z)$ of the number of processors is as given by equation (2.55). Hence, we obtain:

$$\bar{Y} = \lambda \bar{X} \sum_{i=1}^N \left\{ \frac{dX_i(Z)}{dz} \Big|_{z=1} \right\} \quad (2.56)$$

We need now to evaluate the Z-transform $X_i(Z)$; concentrating on slot i , we have:

$$P\{\bar{X}_{i,j} = x\} = \sum_{r=1}^N P\{\bar{X}_{i,j} = x / \bar{r} = r\} P\{\bar{r} = r\}$$

where for $i \geq 2$, we have:

$$P\{\bar{X}_{i,j} = x / \bar{r} = r\} = \begin{cases} 0 & \text{if } x \neq 0 \quad 1 \leq r \leq i-1 \\ 1 & \text{if } x = 0 \quad 1 \leq r \leq i-1 \\ \frac{\binom{N-x-1}{r-2}}{\binom{N-1}{r-1}} & \text{if } x \geq 1 \quad i \leq r \leq N \end{cases} \quad (2.57)$$

and for $i=1$, we have:

$$P[\bar{X}_{1j}=x/\bar{r}=r] = \begin{cases} 0 & \text{if } x \neq N \quad r=1 \\ 1 & \text{if } x=N \quad r=1 \\ \frac{\binom{N-x-1}{r-2}}{\binom{N-1}{r-1}} & x \geq 1 \quad r \geq 2 \\ 0 & \text{otherwise} \end{cases} \quad (2.58)$$

Since we know that $P[\bar{r}=r] = \frac{\binom{N-1}{r-1}}{2^{N-1}}$, and by using equation (2.57) and Proposition 2.3, we obtain for the case of $i \geq 2$:

$$P[\bar{X}_{i,j}=x] = \begin{cases} \frac{\sum_{r=1}^{i-1} \binom{N-1}{r-1}}{2^{N-1}} & x=0 \\ \frac{\sum_{r=2}^N \binom{N-x-1}{r-2}}{2^{N-1}} & x \neq 0 \end{cases} \quad i \geq 2 \quad (2.59)$$

Therefore, we obtain:

$$X_{i,j}(Z) = \frac{1}{2^{N-1}} \left\{ \sum_{r=1}^{i-1} \binom{N-1}{r-1} + \sum_{x \geq 1} \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} Z^x \right\} \quad i \geq 2 \quad (2.60)$$

and for the case of $i=1$, and by using equation (2.58), we obtain:

$$P[\bar{X}_{1j}=x] = \begin{cases} 0 & \text{if } x \neq 0 \\ \frac{1}{2^{N-1}} & \text{if } x=N \\ \sum_{r=2}^N \frac{\binom{N-x-1}{r-2}}{2^{N-1}} & \text{if } x \geq 1 \end{cases} \quad (2.61)$$

which then yields:

Since we have $X_{i,j}(Z)|_{z=1} = 1$, we get from equation (2.60):

$$\sum_{x \geq 1} \sum_{r=2}^N \binom{N-x-1}{r-2} = 2^{N-1} - \sum_{r=1}^{i-1} \binom{N-1}{r-1} \quad i \geq 2 \quad (2.62)$$

$$X_{1j}(Z) = \frac{1}{2^{N-1}} \left\{ Z^N + \sum_{x=1}^N \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} Z^x \right\} \quad i=1 \quad (2.63)$$

Now, using equations (2.60) and (2.63), we get:

$$\sum_{i=1}^N \left\{ \frac{dX_i(Z)}{dZ} \Big|_{z=1} \right\} = \sum_{i=2}^N \frac{1}{2^{N-1}} \sum_{x=1}^N x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} + \frac{1}{2^{N-1}} \left\{ N + \sum_{x=1}^N x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} \right\}$$

Define the quantities A and B by the following expressions:

$$A = \sum_{i=2}^N \frac{1}{2^{N-1}} \sum_{x=1}^N x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$$

$$B = \sum_{x=1}^N x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$$

Hence equations (2.56) becomes:

$$\bar{Y} = \lambda \bar{X} \left\{ A + \frac{1}{2^{N-1}} [N + B] \right\} \quad (2.64)$$

The quantities A and B are evaluated in Appendix (B), where A is given by equation (B.9); that is:

$$A = N - 2 + \left[\frac{1}{2} \right]^{N-1}$$

and B is given by equation (B.10); that is:

$$B = 2^N - (N+1)$$

and therefore equation (2.64) becomes:

$$\bar{Y} = \lambda \bar{X} N \quad (2.65)$$

finally, from equation (2.56) and equation (2.65), we deduce that:

$$\sum_{i=1}^N \left\{ \frac{dX_i(Z)}{dZ} \Big|_{z=1} \right\} = N \quad (2.66)$$

2.4.3 Variance of the Number of Occupied Processors

Now, we proceed to derive a closed form expression for the variance of the number of occupied processors, denoted by $\sigma_{\bar{Y}}^2$. We have:

$$\sigma_{\bar{Y}}^2 = \frac{d^2 Y(Z)}{dZ^2} \Big|_{z=1} + \bar{Y} - \bar{Y}^2$$

where \bar{Y} is the average number of occupied processors and is given by equation (2.65). Using equation (2.55), we obtain:

$$\frac{d^2 Y(Z)}{dZ^2} = \frac{dY(Z)}{dZ} \frac{d}{dZ} \left\{ \lambda \bar{X} \sum_{i=1}^N X_i(Z) \right\} + Y(Z) \frac{d^2}{dZ^2} \left\{ \lambda \bar{X} \sum_{i=1}^N X_i(Z) \right\}$$

where $X_i(Z)$ is given by equation (2.60) for $i \geq 2$ and by equation (2.63) for $i=1$. Now, using equation (2.66), the expression of the variance of the number of occupied processors becomes:

$$\sigma_{\bar{Y}}^2 = \bar{Y} + \lambda \bar{X} \sum_{i=1}^N \left\{ \frac{d^2}{dZ^2} X_i(Z) \Big|_{z=1} \right\}$$

and using equations (2.60) and (2.63), we obtain:

$$\sigma_{\bar{Y}}^2 = \bar{Y} + \sum_{i=2}^N \frac{\lambda \bar{X}}{2^{N-1}} \left\{ \sum_{x=21} x(x-1) \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} + \frac{\lambda \bar{X}}{2^{N-1}} \left\{ N(N-1) + \sum_{x=21} \sum_{r=2}^N x(x-1) \binom{N-x-1}{r-2} \right\}$$

Define the quantities C and D by the following expressions:

$$C = \sum_{i=2}^N \frac{1}{2^{N-1}} \sum_{x=21} x^2 \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$$

$$D = \sum_{x=21} x^2 \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$$

It follows then that $\sigma_{\bar{Y}}^2$ becomes:

$$\begin{aligned} \sigma_{\bar{Y}}^2 &= \bar{Y} + \lambda \bar{X} C + \frac{\lambda \bar{X} D}{2^{N-1}} + \frac{\lambda \bar{X} N(N-1)}{2^{N-1}} - \sum_{i=2}^N \frac{\lambda \bar{X}}{2^{N-1}} \left\{ \sum_{x=21} x \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} \\ &\quad - \frac{\lambda \bar{X}}{2^{N-1}} \sum_{x=21} x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\} \\ &= \bar{Y} + \lambda \bar{X} C + \frac{\lambda \bar{X} D}{2^{N-1}} + \frac{\lambda \bar{X} N(N-1)}{2^{N-1}} - \lambda \bar{X} \left\{ \sum_{i=1}^N \left[\frac{dX_i(Z)}{dZ} \Big|_{z=1} \right] - \frac{N}{2^{N-1}} \right\} \end{aligned}$$

using equations (2.56) and (2.66), we obtain:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} \left\{ C + \frac{D}{2^{N-1}} + \frac{N^2}{2^{N-1}} \right\} \quad (2.67)$$

The quantities C and D are evaluated in Appendix (B), where C is given by equation (B.17); that is:

$$C = 3N - 10 + \frac{2N + 5}{2^{N-1}}$$

and D is given by equation (B.18); that is:

$$D = 3 \cdot 2^N - N^2 - 2N - 3$$

Therefore equation (2.67) becomes:

$$\sigma_{\bar{Y}}^2 = \lambda \bar{X} \left\{ 3N - 4 + \left[\frac{1}{2} \right]^{N-2} \right\} \quad (2.68)$$

From the above equation, we observe that for a large value of N (e.g., $N > 5$), the variance of the number of occupied processors is:

$$\sigma_{\bar{Y}}^2 = 3\bar{Y} - 4\lambda\bar{X} \quad \text{for } N \gg 1$$

and finally for the value $N=1$, equation (2.68) verifies that $\sigma_{\bar{Y}}^2 = \lambda\bar{X}$ as provided by equation (2.7) of the fixed process graph case.

2.5 Conclusion

In this chapter, we have analyzed three models of process graphs in the infinite number of processors case. For each model, we have found the distribution and the Z-transform of the number of occupied processors. From the Z-transforms, we were able to derive both the average and the variance of the number of occupied processors.

The important observation is that the average number of occupied processors for the three models is the same, and depends only on the job average arrival rate, the number of tasks in the process graph, and the average service time per task. In the following chapter, we shall extend this result to more general environments of multiprocessor systems and process graph description. For the variance of the number of occupied processors, on the other hand, we found rather easy expressions. For the case of fixed and semi random process graphs, we also determined the process graphs which provide upper and lower bounds on the variance of the number of occupied processors.

CHAPTER 3

AVERAGE NUMBER OF OCCUPIED PROCESSORS

THE GENERAL CASE

In the previous chapter, we have considered the case of multiprocessor systems with infinite number of processors, and have investigated the distribution of the number of occupied processors for three different process graph models. We have assumed that the task service time is constant, the same for all the tasks, that the process graphs have a fixed and prescribed total number of tasks, and that the job arrival process is Poisson. We have found a rather interesting result stating that the average number of occupied processors in the system is only a function of the job average arrival rate, the task constant service time, and the fixed number N of tasks forming the process graph. The question naturally arises as to what extent can this result be generalized.. This is the aim of the current chapter.

Our multiprocessor system can be generalized by relaxing all the assumptions made in the previous chapter. We shall then consider the infinite and the finite number of processors cases, an arbitrary distribution of the number of tasks per job, an arbitrary conditional distribution for the number of levels in the process graph, arbitrary repartitions of the tasks among the levels, arbitrary task service time distribution, perhaps different service requirements for the different tasks, and general job arrival process. Under these rather general conditions, we shall prove that the average number of occupied processors, in both the infinite number of processors case and the finite number of processors case, remains a function of only the job average arrival rate, the task average service requirements, and the average number of tasks per job.

In Section 3.1, we further pursue the case of an infinite number of processors, where we first derive a closed form expression of the task arrival process distribution, its mean and variance, and then investigate the average number of occupied processors under the above mentioned generalized conditions. In Section 3.2, we consider the finite number of processors case, and prove that the average number of processors in the system stays a function of only the job average arrival rate, the task average service requirements, and the average number of tasks per job. In Section 3.3, we discuss and provide a pictorial representation of the job average system time and the average number of occupied processors as a function of the system offered load. For the finite number of processors case, we assume throughout the chapter that the system is in equilibrium.

3.1 Infinite Number of Processors

In this section, we pursue the infinite number of processors case. We shall provide a theorem stating that under the generalized model, the average number of occupied processors in the system is only a function of the job arrival average rate, the task average service times, and the average number of tasks per job. First, we investigate the distribution of the task interarrival times to the system.

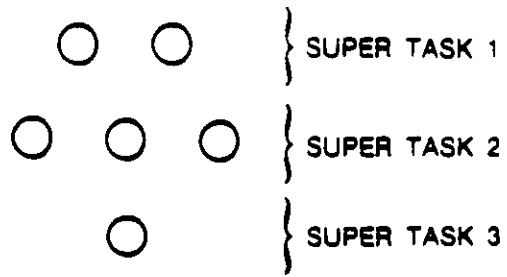
3.1.1 The Task Interarrival Time Process

Since a job is composed of a set of tasks, the task interarrival process is then different from the job arrival process. These two arrival processes are identical only in the case where jobs are composed of only one task. In this section, we derive a closed form expression of the distribution of the task interarrival time process. Throughout the section, we assume that the number of available processors is infinite, the job arrival process is Poisson with average rate λ , and the task service time is constant equal to \bar{X} , the same for all tasks. Jobs are represented by the same process graph with N tasks and $1 \leq r \leq N$ levels. The average and the variance of the task interarrival time will also be derived.

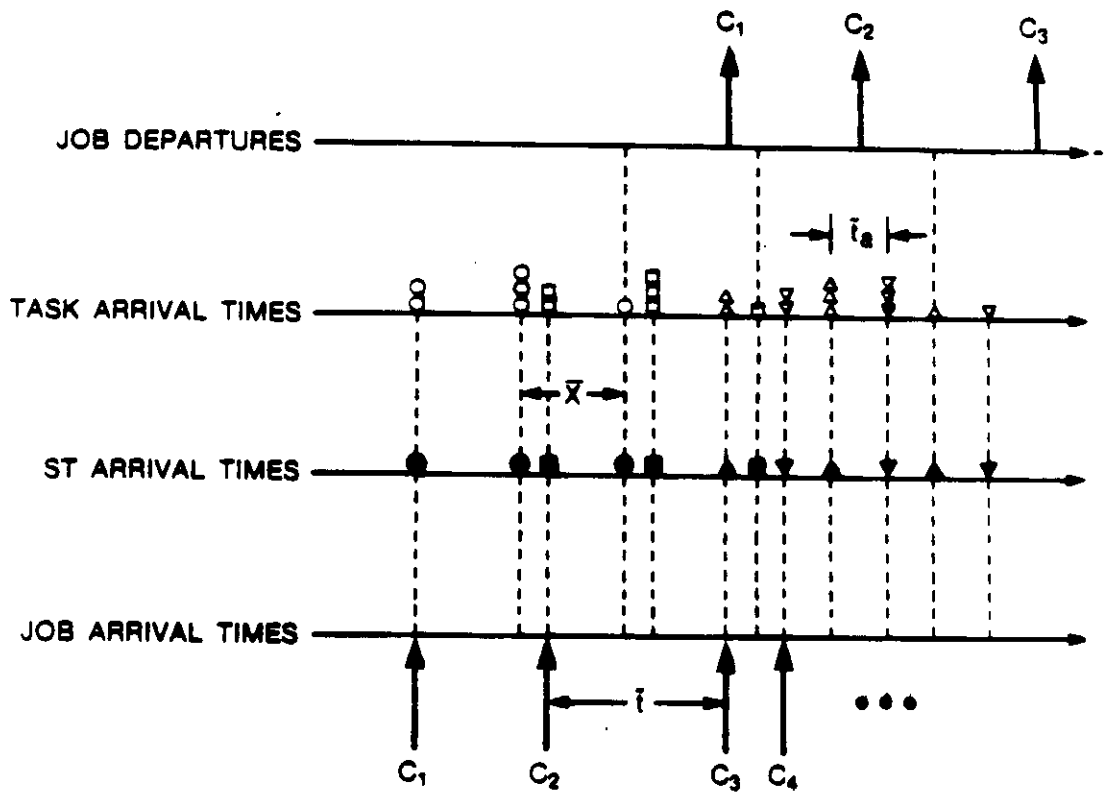
Our multiprocessor system can be viewed as an FCFS queueing system with an infinite number of processors. Let $J = (n_1, n_2, \dots, n_r)$ denote the process graph description, where n_i is the number of tasks at level i , $i=1, \dots, r$, and thus we have $\sum_{i=1}^r n_i = N$. Jobs may thus be regarded as a vertical string of super-tasks (ST); where super-task ST_i , $i=1, \dots, r$ comprises n_i tasks representing the set of tasks at level i in the process graph. Since the number of processors is infinite, the n_i , $i=1, \dots, r$ tasks forming super-task i are then executed in parallel. Upon the arrival of a job to the system, its starting tasks (i.e., the tasks forming its first super-task) are ready-for-service and thus start execution immediately. Upon the completion of its first super-task, the job feeds back all the tasks forming its second level (i.e., its second super-task), which immediately start their execution. Each \bar{X} seconds thereafter and until completion, the job creates all the tasks of its next level as shown in Figure 3.1. Figure 3.1:(b) represents a time diagram of the job arrival process, and the corresponding super-tasks arrival process, the task arrival process, and the job departure process. The process graph description used is depicted in Figure 3.1:(a). Let us first consider the interarrival time of super-tasks to the system.

Proposition 3.1

Provided the task constant service time \bar{X} is strictly positive, no simultaneous super-task arrivals can occur.



(a): Process Graph Description



(b): Arrival Processes Time Diagram

Figure 3.1: Job Arrival and Task Arrival Time Diagram

Proof

Super-task arrivals from the same job are separated exactly by \bar{X} seconds. Since $\bar{X} > 0$, then no simultaneous super-task arrivals from the same job can occur. On the other hand, to have simultaneous super-task arrivals from 2 different jobs, the arrival of these two jobs must be separated exactly by $i\bar{X}$ seconds where $0 \leq i \leq r-1$. But since the job arrival process is Poisson with parameter λ , we have:

$$\begin{aligned} P \left[\text{job arrival in } [t, t+dt] \text{ and job arrival in } [t+i\bar{X}, t+i\bar{X}+dt] \right] \\ = P \left[\text{job arrival in } [t, t+dt] \right] \cdot P \left[\text{job arrival in } [t+i\bar{X}, t+i\bar{X}+dt] \right] \\ = (\lambda dt + O(dt)) \cdot (\lambda dt + O(dt)) = \lambda^2 dt^2 + O(dt) = O(dt) \end{aligned}$$

The above can also be seen by noticing that the probability of having two arrivals separated by exactly $i\bar{X}$ seconds is the same as the probability of having simultaneous arrivals.

■

Proposition 3.1 says that to characterize the distribution of the tasks interarrival time process, we need to find :

1. the distribution of the super-task interarrival times, and then
2. the distribution of the super-task size

First, we proceed to find the distribution of the super-task interarrival times. Recall that jobs are represented by a process graph with r levels, and thus comprising r super-tasks. Let \bar{t} denote the random variable measuring the interarrival time between jobs, and \bar{t}_s denote the random variable measuring the interarrival time between super-tasks. Our objective is to find the probability distribution of the random variable \bar{t}_s ; that is $P[\bar{t}_s \leq t]$. In the sequel, we distinguish two cases depending on the value of the number of levels (i.e., the number of super-tasks) in the process graph.

(1): Case $r=1$

Since each job creates just one super-task and this is exactly upon its arrival to the system, then the distribution of the interarrival time between super-tasks is the same as the job interarrival time distribution. We then have :

$$P[\bar{t}_s \leq t] = 1 - e^{-\lambda t} \quad t \geq 0 \tag{3.1}$$

(2): Case $r \geq 2$

This is the case of two or more super-tasks per job. Depending on whether the interval of time t is less or equal to the task constant service time \bar{X} , we distinguish the two following cases.

(2.1): Case where $0 \leq t < \bar{X}$

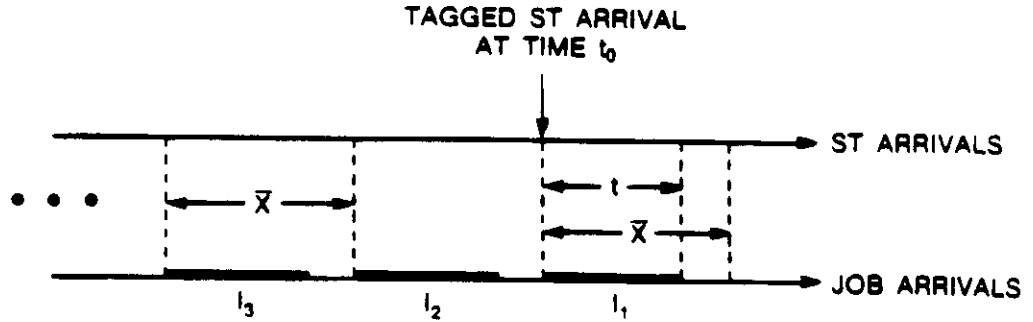


Figure 3.2: Super-task Interarrival Time Diagram, case of $0 \leq t < \bar{X}$

Since $P[\bar{r}_a \leq t] = 1 - P[\bar{r}_a > t]$, let us first compute $P[\bar{r}_a > t]$. As depicted in Figure 2.3, let us place ourselves at the tagged super-task arrival time t_0 , and compute the probability of no super-task arrivals during the interval of time t . The intervals I_i , $i=1, \dots, r$ are of the same time length and are equal to t . Therefore, we have:

$$\begin{aligned}
 P[\bar{r}_a > t] &= P[\text{no ST arrivals during the interval of time } t] \\
 &= P[\text{no job arrivals in the intervals } I_1, I_2, I_3, \dots, I_r] = \prod_{i=1}^r e^{-\lambda t}
 \end{aligned}$$

which yields:

$$P[\bar{r}_a \leq t] = 1 - e^{-\lambda r t} \quad 0 \leq t < \bar{X} \quad (3.2)$$

(2.2): Case where $t \geq \bar{X}$

Since $t \geq \bar{X}$, we must distinguish whether $t = \bar{X}$ or $t > \bar{X}$.

(2.2.1): Case where $t = \bar{X}$

Let us position ourselves at a super-task arrival instant, say t_0 . From Proposition 3.1, we know that the next arriving super-task, if any, must belong to the same job as the tagged super-task. Since $t = \bar{X}$ then:

$$P[\bar{t}_a \leq t] = P[\bar{t}_a \leq \bar{X}] = 1 - P[\bar{t}_a > \bar{X}]$$

on the other hand,

$$P[\bar{t}_a > \bar{X}] = P[\text{no ST arrivals in the interval } [t_0, t_0 + \bar{X}], \text{ and no ST arrival at time } (t_0 + \bar{X})]$$

and since both events are disjoint we obtain:

$$P[\bar{t}_a > \bar{X}] = P[\text{no ST arrivals in the interval } [t_0, t_0 + \bar{X}]] \cdot P[\text{no ST arrivals at time } (t_0 + \bar{X})]$$

Finally, from case (2.1) where $0 \leq t < \bar{X}$, we get from equation (3.2):

$$P[\text{no ST arrivals in the interval } [t_0, t_0 + \bar{X}]] = e^{-\lambda \bar{X}}$$

and by application of Proposition 3.1, we obtain:

$$P[\text{no ST arrivals at time } (t_0 + \bar{X})] = P[\text{ST is the last super-task of its job}]$$

on the other hand, since a job has r super tasks, and each super-task takes \bar{X} seconds of processing time, it follows that:

$$P[\text{job is executing its } i\text{th super-task} \mid \text{job is in the system}] = \frac{1}{r} \quad i=1, \dots, r$$

and therefore, we obtain:

$$P[\bar{t}_a > \bar{X}] = \frac{e^{-\lambda \bar{X}}}{r} \quad t = \bar{X} \quad (3.3)$$

(2.2.2): Case where $t > \bar{X}$

Let us place ourselves at a super-task arrival instant, say t_0 , as indicated in Figure 3.3. Since the interval of time t is strictly larger than \bar{X} , we already know that no job arrivals occur in the interval $(t_0, t_0 + \bar{X})$, and that the system becomes empty at time $t_0 + \bar{X}$. We therefore have:

$$\begin{aligned} P[\bar{t}_a > t] &= P[\text{no ST arrivals during } t] = P[\text{no ST arrivals in } [t_0 + \bar{X}, t_0 + t]] \\ &= P[\text{no job arrivals during } (t - \bar{X})] \end{aligned}$$

which amounts to:

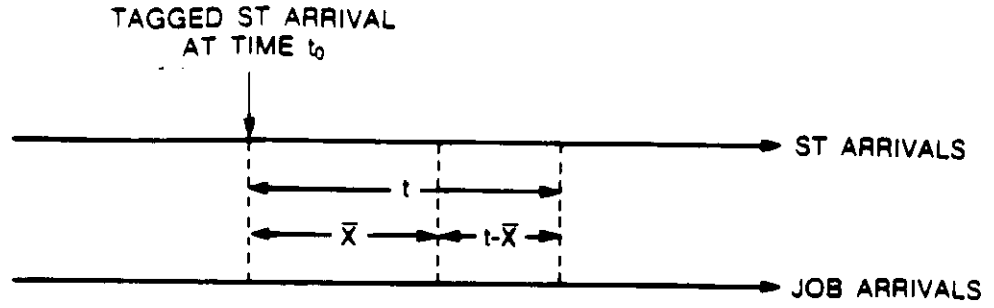


Figure 3.3: Super-task Interarrival Time Diagram, case of $t > \bar{X}$

$$P[\bar{t}_a > t] = e^{-\lambda(t-\bar{X})} \quad t > \bar{X} \quad (3.4)$$

Finally, putting the two cases together, that is for $t \geq \bar{X}$, we get:

$$\begin{aligned} P[\bar{t}_a \leq t] &= 1 - P[\bar{t}_a > t] \\ &= 1 - P[\text{no ST arrivals during } t] \\ &= 1 - P[\text{no ST arrivals during } t \mid \text{no ST arrivals during } \bar{X}] \cdot P[\text{no ST arrivals during } \bar{X}] \\ &\quad - P[\text{no ST arrivals during } t \mid \text{ST arrivals during } \bar{X}] \cdot P[\text{ST arrivals during } \bar{X}] \end{aligned}$$

Since $P[\text{no super-task arrivals during } t \mid \text{super-task arrivals during } \bar{X}] = 0$, then using equation (3.3) and equation (3.4), yields:

$$P[\bar{t}_a \leq t] = 1 - \frac{e^{-\lambda\bar{X}(1-r)}}{r} e^{-\lambda t} \quad t \geq \bar{X} \quad (3.5)$$

Equation (3.2) along with equation (3.5) provide an explicit closed form expression of the probability distribution function of the super-task interarrival time process.

3.1.1.1 Average Interarrival Time Between Super Tasks

Let \bar{t}_a denote the average interarrival time between super-tasks. Therefore:

$$\bar{t}_a = \int_0^{\infty} [1 - P[\bar{t}_a \leq t]] dt$$

and using equations (3.2) and (3.5), we get:

$$\bar{t}_a = \int_0^{\bar{X}} e^{-\lambda t} dt + \int_{\bar{X}}^{\infty} \frac{e^{-\lambda \bar{X}(1-r)}}{r} e^{-\lambda t} dt$$

which amounts to:

$$\bar{t}_a = \frac{1}{\lambda r} \quad r \geq 2$$

since for the case of $r=1$, and from equation (3.1) we have $\bar{t}_a = \frac{1}{\lambda}$, therefore, we obtain:

$$\bar{t}_a = \frac{1}{\lambda r} \quad r \geq 1 \quad (3.6)$$

For any process graph with N tasks and r levels, $1 \leq r \leq N$, and for the case of an infinite number of processors and constant task service time, \bar{X} , the average number of jobs occupying some processors is $\lambda r \bar{X}$. This can be seen by noticing that the only jobs which occupy some processors at any given time t , must have arrived in the interval of time $(t-r\bar{X}, t)$. On the other hand, since the expected number of busy processors in such a case is $\lambda N \bar{X}$, it follows that the jobs which occupy some processors at any given time t , participate on the average by $\frac{N}{r}$ tasks.

3.1.1.2 Variance of the Interarrival Time Between Super Tasks

Let $\sigma_{t_a}^2$ represent the variance of the interarrival times between super-tasks. We have:

$$\sigma_{t_a}^2 = E[\bar{t}_a^2] - E[\bar{t}_a]^2$$

Using equations (3.2) and (3.5), and after some algebra, we obtain:

$$\sigma_{t_a}^2 = \frac{1 - 2(1-r)e^{-\lambda \bar{X} r}}{(\lambda r)^2} \quad r \geq 1 \quad (3.8)$$

Notice that in the special case where $r=1$, equation (3.8) reduces to the variance of the job arrival process (i.e., the variance of the exponential distribution with parameter λ).

In the sequel, we shall find the distribution of the super-task size. Recall that a job is represented by the process graph description $J = (n_1, n_2, \dots, n_r)$, where n_i is the number of tasks at level i , $i=1, \dots, r$. Let \bar{S} denote the random variable representing the size of a super task, and $\delta_k(i)$ denote the binary function, which is equal one if $i=k$ and equal zero otherwise.

Proposition 3.2

the distribution of the size of a super-task, given that all jobs have the same fixed process graph description, is given by :

$$P[\bar{S}=k] = \frac{1}{r} \sum_{j=1}^r \delta_k(n_j) \quad 1 \leq k \leq N-r+1$$

Proof

Consider the arrival process of super-tasks to the system (see Figure 3.1). Take any arrival and call it the tagged arrival. This tagged arrival belongs to a given job, call such a job the tagged job. Hence we have:

$P[\text{tagged arrival is the } j\text{th ST of the tagged job}] =$

$$P[\text{tagged job is executing its } j\text{th level} \mid \text{tagged job is in the system}] = \frac{1}{r}$$

and therefore:

$$P[\bar{S}=k] = \frac{1}{r} \cdot [\text{number of levels having } k \text{ tasks}]$$

the proof is complete by using the binary function $\delta_k(i)$. The restriction on the value of k is due to the fact that the maximum number of tasks at any level cannot exceed $N-r+1$.

111

On the other hand, if jobs are described by semi-random process graphs, the distribution of the super-task size is readily given by Proposition 2.1 of the previous chapter, that is:

$$P[\bar{S}=k] = \frac{\binom{N-k-1}{r-2}}{\binom{N-1}{r-1}} \quad r \geq 2, 1 \leq k \leq N-r+1$$

and,

$$p[\bar{S}=k] = \begin{cases} 0 & \text{if } k \neq N \\ 1 & \text{if } k = N \end{cases} \quad r=1$$

3.1.2 Expected Number of Busy Processors

We have shown in the previous chapter that $\bar{Y} = \lambda N \bar{X}$, where we have assumed for the most general case studied that the job arrival process to the system is Poisson with fixed rate λ , that the total number of tasks per job is fixed to N , and that the task average service time is constant equal to \bar{X} , the same for all the tasks. In this section, we show that such a result still holds for the more general case. If we still assume a Poisson job arrival process and a constant task service time, we can see the multiprocessor system as an $M/G/\infty$ queueing system. For this system, it is readily shown [Klei75], that the probability of having k jobs in the system in steady state is given by:

$$P\{k\} = \frac{(\lambda \bar{X})^k}{k!} e^{-\lambda \bar{X}}$$

Since each job in the system participates on the average by $\frac{\bar{N}}{\bar{r}}$ tasks, it follows that $\bar{Y} = \lambda \bar{N} \bar{X}$. We can further generalize our multiprocessor system by relaxing the Poisson job arrival process assumption. Let \bar{N} denote the average number of tasks per job, \bar{C}_j denote the average concurrency per job over all jobs, \bar{K} denote the average number of jobs present in the system, and T be the average time a job spends in the system.

Theorem 3.1

The expected number of busy processors \bar{Y} in the case of:

1. an infinite number of processors,
2. random service time per task (possibly different service requirement and distribution for each task) with an overall average \bar{X} ,
3. random job arrival process with average arrival rate λ (but independent job arrivals), and
4. random process graph, that is,
 - N random
 - r random, $r=1, \dots, N$
 - random repartition of tasks among levels, and
 - random precedence relationships among levels

is given by:

$$\bar{Y} = \lambda \bar{N} \bar{X}$$

Proof

Since the average number of occupied processors, \bar{Y} , represents the average concurrency in the system, it follows that

$$\bar{Y} = \bar{K} \bar{C}_j$$

Notice that $\overline{KC_j} = \bar{K} \bar{C}_j$, due to the fact that P is infinite. By using Little's formula [Lit61], we have:

$$\bar{K} = \lambda T$$

where the job average system time T can be written as:

$$T = \frac{\bar{NX}}{\bar{C}_j}$$

It then follows that:

$$\bar{Y} = \lambda T \bar{C}_j = \lambda \frac{\bar{NX}}{\bar{C}_j} \bar{C}_j = \lambda \bar{NX}$$

■

3.2 Finite Number of Processors

In this section, the number of processors in the system is finite, say P. We shall prove that the average number of occupied processors, \bar{Y} , is still given by $\bar{Y} = \lambda \bar{NX}$. Throughout this section, we assume that the multiprocessor system is in equilibrium.

Theorem 3.2

If the multiprocessor system is in equilibrium and work-conservative, then the average number of occupied processors \bar{Y} for the case of:

1. finite number of processors, say P,
2. random service time per task (possibly different service requirement and distribution for each task) with overall average \bar{X} ,
3. random job arrival process with average arrival rate λ (but independent job arrivals), and
4. random process graph per job, that is for each job:
 - N random

- r random, $r=1, \dots, N$
- random repartition of tasks among levels, and
- random precedence relationships among levels

is given by:

$$\bar{Y} = \lambda \bar{N} \bar{X}$$

Moreover, if the system is overloaded then

$$\bar{Y} = P$$

Proof

For $p=1, \dots, P$, let \bar{n}_p denote the average number of tasks per job processed by processor p , ρ_p denote the utilization factor of processor p , and ρ be the system total utilization factor. The equilibrium condition is then $\forall p=1, \dots, P \rho_p < 1$ and that $\rho = \sum_{p=1}^P \rho_p < 1$. We have:

$$\rho_p = \lambda \bar{n}_p \bar{X} \quad \text{and} \quad \bar{N} = \sum_{p=1}^P \bar{n}_p$$

$$\bar{Y} = \sum_{p=1}^P \rho_p = \sum_{p=1}^P \lambda \bar{n}_p \bar{X} = \lambda \bar{N} \bar{X}$$

If the system is overloaded (i.e., the system utilization factor ρ is greater than one) then it is easy to see that $\bar{Y} = P$ since all the processors are being used all the time.

□

3.3 Conclusion

In the previous sections, we have proved that the average number of occupied processors in a multiprocessor system with $P=1, 2, 3, \dots$ processors is given by $\bar{Y} = \lambda \bar{N} \bar{X}$, where \bar{N} and \bar{X} represent respectively the average number of tasks per job and the average service time per task. It is interesting to note that the average number of occupied processors does not depend on the jobs description (e.g., the distribution of the number of tasks per job, the distribution of the number of levels in the process graph, the repartition of the tasks among the levels, the precedence relationships among the levels inside the process graph, the distribution of the task service time, the distribution of the job arrival process and the number of processors in the system given that such multiprocessor system is in equilibrium). More importantly, in the case of finite number of processors, the average number of occupied processors is independent of any processor scheduling provided the multiprocessor system is work-conservative.

Figure 3.4 and Figure 3.5 provide a pictorial profile of the system utilization and the average number of occupied processors in the system as a function of the total number of processors. In Figure 3.4, we have $\lambda\bar{N}\bar{X} < 1$, that is the utilization factor of the system, when $P=1$, is less than unity. In Figure 3.5, we have $\lambda\bar{N}\bar{X} \geq 1$, that is the utilization factor of the system, when $P=1$, is greater than unity. Notice that whenever $\rho < 1$, the expected number of busy processors is $\bar{Y} = \lambda\bar{N}\bar{X}$; whereas for $\rho \geq 1$, $\bar{Y} = P$.

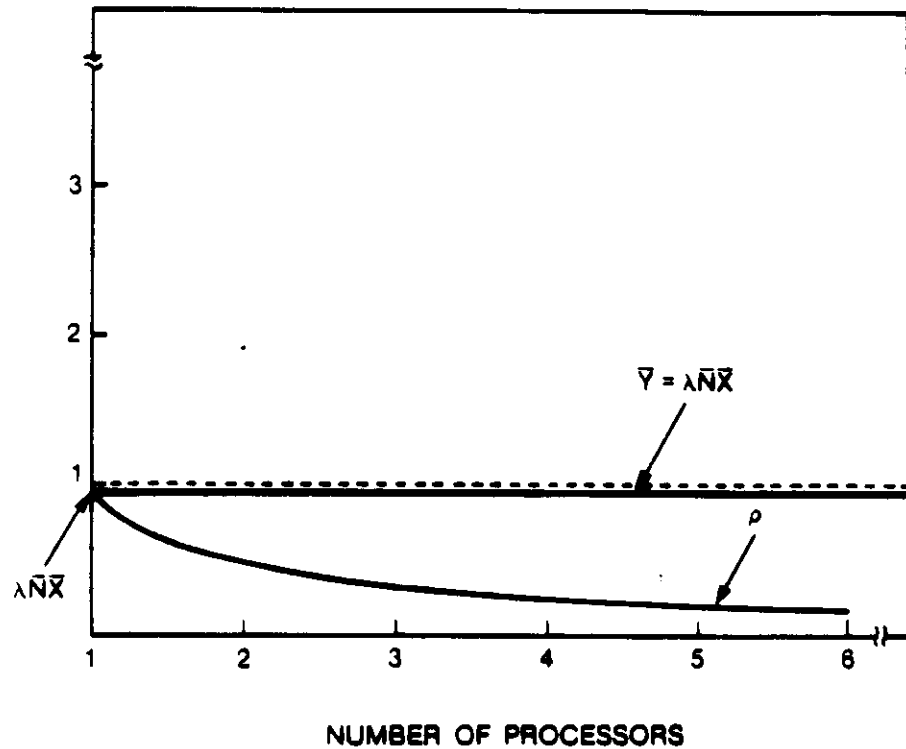


Figure 3.4: System Utilization and Average Number of Occupied Processors versus P , $\lambda\bar{N}\bar{X} < 1$

Figure 3.6 provides a pictorial profile of the average number of occupied processors \bar{Y} and the average system time T as a function of the job arrival rate λ , and for a given number P of processors. In the region where $\rho < 1$, we observe that the expected number of busy processors grows linearly with the number of processors used, and at a constant slope equal to $\bar{N}\bar{X}$. At $\lambda = \frac{P}{\bar{N}\bar{X}}$, the system total utilization factor ρ reaches the value one, which results in an average number of occupied processors equal to $\bar{Y} = P$, and an infinite job average system time.

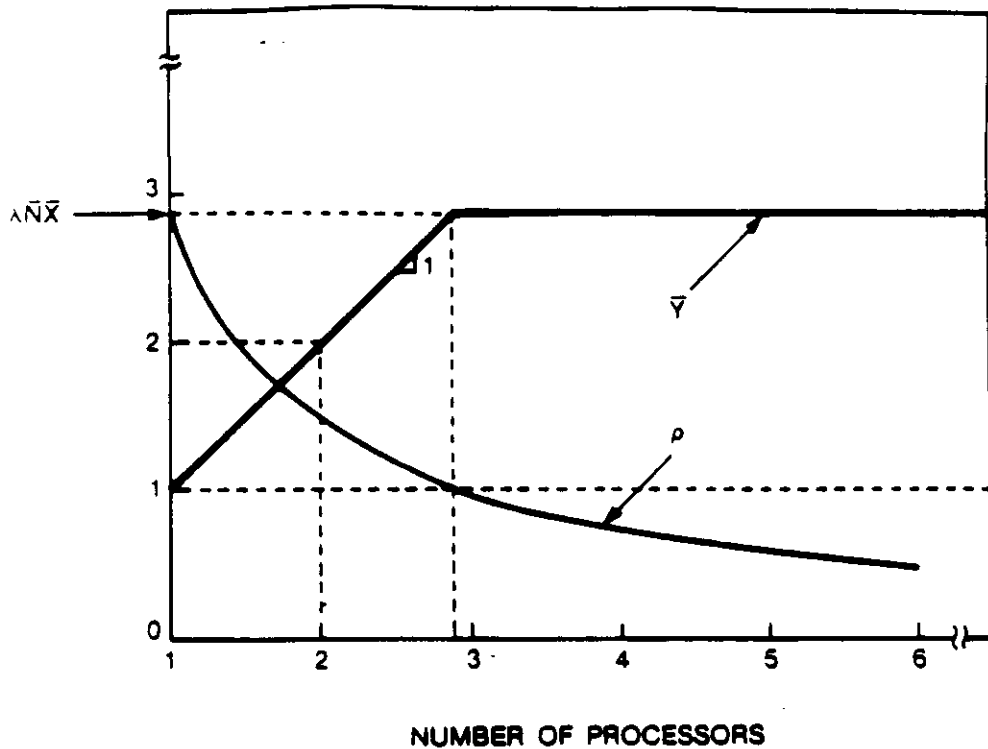


Figure 3.5: System Utilization and Average Number of Occupied Processors versus P , $\lambda \bar{N} \bar{X} \geq 1$

Speedup Factor

One of the major issues in distributed and parallel processing systems is the evaluation of the concurrency. Concurrency is a measure of the achievable parallelism, and can be thought of as the number of busy resources which can be utilized simultaneously. The expected number of busy processors readily ascertain such a measure. The best we can achieve is for the concurrency (equivalently the speedup factor) to grow linearly with P . Indeed, the two previous Theorems witness such a behavior, and prove that for any finite number of processors, the speedup factor is a linear function in P for any value of the system total utilization factor, namely $\bar{Y} = \rho P$.

In practice however, the speedup is much less since some processors are idle at a given time because of conflicts over memory access or communication paths, and inefficient algorithms for properly exploiting the natural parallelism in the computing problems [Mura71, Kuck72, Kuck74, Kuck77, Kuck84].

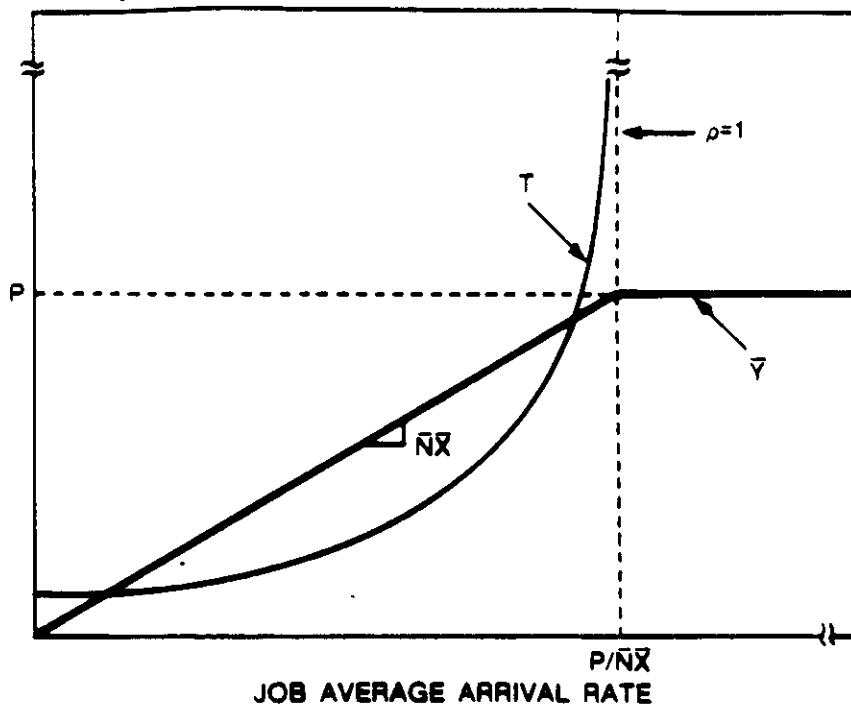


Figure 3.6: Average System Time and Average Number of Occupied Processors versus λ

In the early days of parallel processing, Minsky and Papert [Mins71] provided a depressingly pessimistic form of the speedup factor known as *Minsky's Conjecture*; namely that the speedup factor is equal to the base 2 logarithm of the number of processors used.

Although the expected number of busy processors in a multiprocessor system provides an insight feeling of how much resources can be utilized simultaneously, it does not accurately ascertain how much faster a job can be processed using multiple processors, as opposed to using a single processor. In chapter 6, we shall properly define the speedup measure as a function of the system utilization factor, the number of processors used, and the scheduling strategy, and investigate the parallelism achievable through a parallel processing system and other important related performance issues.

CHAPTER 4

AVERAGE RESPONSE TIME IN PARALLEL PROCESSING SYSTEMS

In the previous chapters, we studied the number of busy processors in multiprocessor systems. In particular, we proved that the expected number of busy processors is a function only of the job average arrival rate, the task average service requirement, and the average number of tasks per job. Although the expected number of busy processors in a multiprocessor system provides some insights as to how much resources can be utilized simultaneously, it does not accurately ascertain the level of parallelism achieved by executing the jobs on a multiprocessor system. This is the aim of the current chapter.

First, we introduce and define a new scheduling policy (i.e., a service discipline) based on a non-egalitarian sharing of the processors capacity among the jobs present in the system. Using this scheduling policy, we convert the process graph describing the jobs into an execution graph which identifies the execution stages assumed by any job throughout its life in the system. In Section 4.2, we consider an infinite number of processors, and we formulate an exact expression, a tight upper bound, and a tight lower bound for the job average response time. In Section 4.3, we consider the uniprocessor case. We first prove that our scheduling policy forms a complete family of scheduling strategies, in the sense that any response time requirement that can be satisfied at all, can be accomplished by a strategy from the family. Then, we prove a conservation law that puts a linear equality constraint on the set of expected system times of the job execution stages. An accurate and yet very simple approximation for the job average response time is then formulated. In Section 4.4, we study the job average response time through a multiprocessor system where all execution stages have the same concurrency degree. This will enable us to formulate a parametric approximation of the job average response time in a multiprocessor system. Simulations are used to validate and prove the excellent accuracy of such an approximation. We conclude the Chapter by studying the achievable parallelism and the efficiency per processor, and by identifying the optimal operating points at which one should operate the multiprocessor system.

4.1 Model Description

We assume that a job may be modeled as a set of N partially ordered tasks, and is represented by a given process graph, the same for all jobs. The processing time of a task is assumed throughout this chapter to be an exponentially distributed random variable. Different tasks in the process graph have independent and perhaps different mean processing requirements. Tasks in the process graph are identified using alphabetical labels. The task identity set, denoted hereafter by Ω , is the set containing the identities of the N tasks. Let \tilde{X}_A , $A \in \Omega$, denote the random variable representing the processing requirement of task A with mean $\frac{1}{\mu_A}$ and a probability density function $b_X(x) = \mu_A e^{-\mu_A x}$ for $x \geq 0$. Jobs arrive to the multiprocessing system according to a Poisson process with an aggregate rate λ . The job process graph is assumed throughout the chapter, unless stated otherwise, to possess only one starting task and only one terminating task. This property of the process graph is required in the design and analysis of the approximations of the job average response time.

Throughout the chapter, we restrict the notion of a scheduling strategy. We consider only strategies which satisfy the following two conditions:

1. They do not explicitly rely on any information about the remaining processing time of any job in the system, and
2. they do not allow processors to be idle when there are jobs waiting to be processed (i.e., work-conserving strategies).

The scheduling strategy (i.e., the service discipline) to be adopted in the case of a finite number of processors will be defined as stated below. For the uniprocessor case however, we shall also consider preemptive and nonpreemptive priority scheduling strategies. In such a case, we consider that the tasks in the process graph are assigned arbitrary but prescribed priority levels. For the infinite number of processors case however, the scheduling strategy vanishes and plays no role.

Let \tilde{n} denote the random variable representing the total number of *ready* (i.e., ready-for-service) tasks from all the jobs present in the system in the steady state. The *Discriminatory Processors Sharing Discipline* for our multiprocessing system is defined as follows:

1. If the total number of ready tasks, \tilde{n} , in the system is less than or equal to the number of processors P , then each ready task is allocated one processor; that is each ready task is processed at a rate of 1 second per second.
2. If the total number of ready tasks, \tilde{n} , in the system is greater than or equal to the number of processors P , then each ready task is served at a rate of $\frac{P}{\tilde{n}}$ seconds per second. The

ready tasks equally share the P processors.

Although at any time the ready tasks share the capacity of the processors in equal proportions, the above defined scheduling discipline divides the total processors capacity in unequal fractions among the jobs present in the system. This is due to the fact that jobs, at any given time, may participate with different numbers of ready tasks. The jobs that possess the largest number of ready tasks will then receive the most preferential treatment at the expense of the others (i.e., at the expense of the jobs having lower numbers of ready tasks). We shall elaborate on this in the sequel, but first let us introduce the notion of an Execution Graph.

An Execution Graph for a given process graph is an acyclic directed graph with nodes representing the state of execution of a job (i.e., the identity of the ready tasks of the job), and edges representing the precedence relationships among the nodes. A node in the execution graph is hereafter called a *stage* of the execution graph. Assuming either an infinite number of processors, or a Processor Sharing service discipline among all the ready tasks, it is not difficult to see that a process graph can always be converted to an execution graph. Consider the process graph given in Figure 4.1, and having 7 tasks identified by the set $\Omega = \{A, B, C, D, E, F, G\}$ with A being the starting task and G being the terminating task. Upon the arrival of a job described by such a process graph, its starting task A immediately acquires the ready-for-service status, and thus the execution stage of the job at its arrival instant comprises only the task A. At the completion time of task A, the job forks into two new tasks; namely task B and task C, which immediately assume the ready-for-service status, and consequently the job execution stage at such an instant comprises both tasks B and C. At this point, both tasks B and C are executed at the same rate. If task B finishes first then tasks D and E assume the ready-for-service status, and the new execution stage at the completion time of task B comprises the three ready tasks; namely C, D and E. Otherwise, if task C finishes first then task F would acquire the ready-for-service status, and consequently at the completion time of task C, the job execution stage comprises the ready tasks B and F. Proceeding in this way, the process graph given by Figure 4.1 results in the execution graph depicted in Figure 4.2, where the stages are represented by circles and are numbered from 1 to 16. Letters inside the circles denote the identities of the ready tasks comprised in such stages.

Generally, a stage in the execution graph represents a specific set of tasks in the job process graph that may be executed in parallel. Formally, let L denote the total number of stages in the execution graph, $\alpha(i), i=1, \dots, L$ identify the set of ready tasks that are executed concurrently when the job is at execution stage i, and $f(i), i=1, \dots, L$ be the number of tasks in stage i (i.e., $f(i)$ denotes the cardinality of the set $\alpha(i), i=1, \dots, L$) also called hereunder the concurrency of stage i. The task identity set Ω is then defined as a function of the $\alpha(i), i=1, \dots, L$ by:

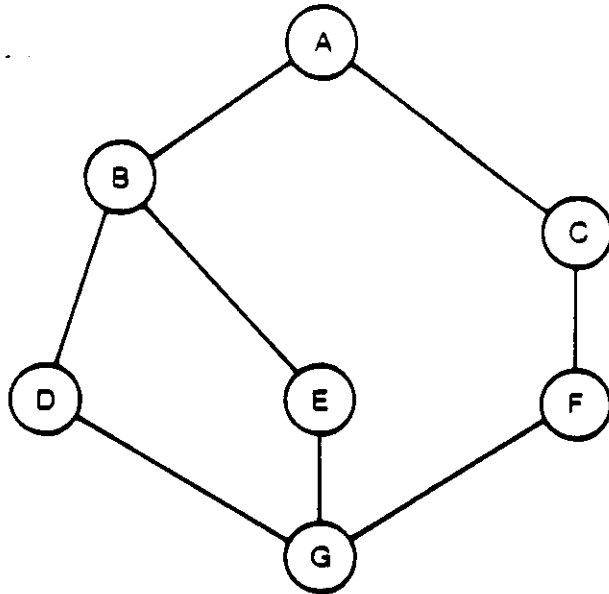


Figure 4.1: Process Graph with Identity Set $\Omega = \{A, B, C, D, E, F, G\}$

$$\Omega = \bigcup_{i=1}^L \alpha(i)$$

From each stage, say stage i , in the execution graph, there are $f(i)$ outgoing edges, each corresponding to the termination of one of the ready tasks being executed in the set $\alpha(i)$. The stages at the end of these edges comprise the set of tasks in $\alpha(i)$ minus the just completed task, plus the new ready tasks, if any, that are activated by the completed task (those which acquire the ready-for-service status upon the completion of the completed task).

To fully describe the job execution graph, we must determine the transition probabilities between the execution stages. Note that the time spent by a job in any given stage is the time needed to finish one of its ready tasks comprised in such a stage, and that upon the completion of the execution of the terminating stage, namely stage L , the job departs the system at once. An execution stage, other than stage L , comprising one ready task, has only one successor stage and consequently the transition probability is one. For execution stages with more than one ready task, the situation is a bit more complicated. Consider execution stage number 3 in the execution graph of Figure 4.2. This stage has 3 ready tasks (namely $\alpha(3) = \{C, D, E\}$) and must then have three successor stages, which in fact are stage 5, stage 6, and stage 7 as depicted in Figure 4.2. The processing times of these tasks are respectively \bar{X}_C , \bar{X}_D , and \bar{X}_E which are exponentially distributed random variables with respective averages $\frac{1}{\mu_C}$, $\frac{1}{\mu_D}$, and $\frac{1}{\mu_E}$. Let

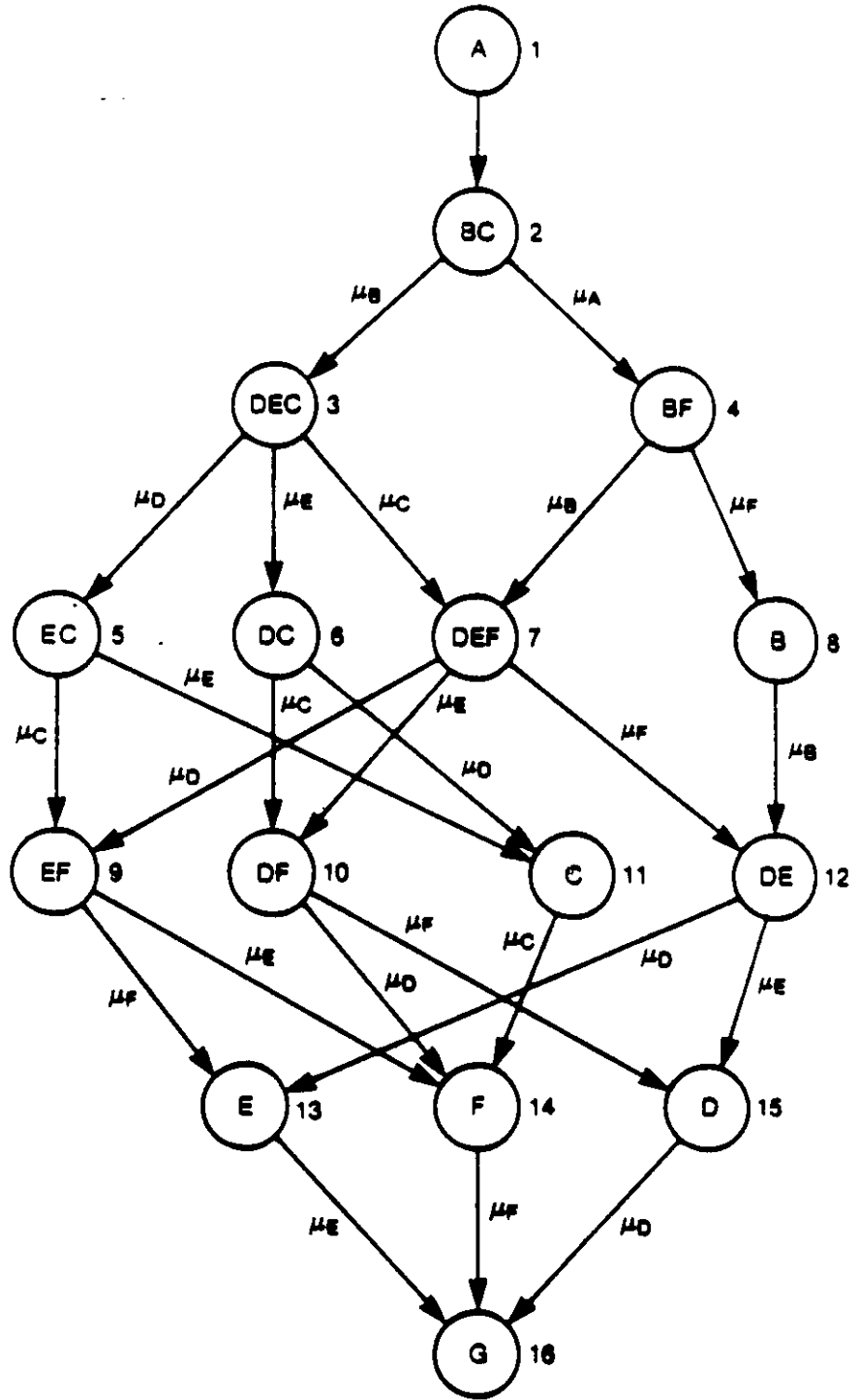


Figure 4.2: The Execution Graph of the Process Graph of Figure 4.1

P_{ij} , $i=1, \dots, L$, $j=1, \dots, L$ be the transition probabilities between stage i and stage j . Due to the memoryless property (i.e., the Markovian property) of the exponential service time distribution, a task in any stage, say stage number i , has the same mean service time regardless of whether it had being processed earlier in another preceding stage. Moreover, if \bar{X} and \bar{Y} are independent and exponentially distributed random variables with respective means $\frac{1}{\mu_X}$ and $\frac{1}{\mu_Y}$, then

$$P\{\bar{X} \leq \bar{Y}\} = \frac{\mu_X}{\mu_X + \mu_Y}. \text{ Hence:}$$

$$P\{\text{task Z completes first} \mid Z \text{ in } \alpha(i)\} = \frac{\mu_Z}{\sum_{\substack{\text{s.t. task A is in } \alpha(i)}} \mu_A} \quad \text{for all } i=1, \dots, L \quad (4.1)$$

Using the above equation, and for our example we obtain:

$$P_{35} = \frac{\mu_D}{\mu_C + \mu_D + \mu_E} \quad P_{36} = \frac{\mu_E}{\mu_C + \mu_D + \mu_E} \quad P_{37} = \frac{\mu_C}{\mu_C + \mu_D + \mu_E}$$

In the case of the same exponential service time distribution for all the N tasks, equation (4.1) becomes:

$$P\{\text{tasks Z completes first} \mid Z \text{ in } \alpha(i)\} = \frac{1}{f(i)} \quad \text{for all } i=1, \dots, L$$

At any time during its sojourn time in the system, a job is fully described by its current execution stage. The global state of the multiprocessing system is thus fully described by the total number of jobs in each stage of the execution graph. It is not difficult to see that the execution graph is a Markovian state transition diagram. Indeed, in Figure 4.2, we indicated, on each directed edge, the instantaneous average rate of exit from the execution stage along that edge for the case of an infinite number of processors. For stage number 3 for example, the rate of exit to stage number 5 is μ_D , the rate of exit to stage number 6 is μ_E , and the rate of exit to stage

In fact:

$$\begin{aligned} P\{\bar{X} \leq \bar{Y}\} &= \int_{y=0}^{\infty} P\{\bar{X} \leq y \mid y \leq \bar{Y} < y+dy\} dP\{\bar{Y} \leq y\} \\ &= \int_{y=0}^{\infty} \left[1 - e^{-\mu_X y}\right] \mu_Y e^{-\mu_Y y} dy \\ &= \int_{y=0}^{\infty} \mu_Y e^{-\mu_Y y} dy - \frac{\mu_Y}{\mu_X + \mu_Y} \int_{y=0}^{\infty} (\mu_X + \mu_Y) e^{-(\mu_X + \mu_Y)y} dy \\ &= 1 - \frac{\mu_Y}{\mu_X + \mu_Y} = \frac{\mu_X}{\mu_X + \mu_Y} \end{aligned}$$

number 7 is μ_C .

Starting from the initial stage in the execution graph, there are many paths a job can traverse before reaching the terminating stage. Since we know the transition probabilities between the stages (i.e., the probability of traversing each edge in the execution graph), the probability that a specific path is to be taking can be calculated. As an example, take the path (1,2,3,5,9,13,16) in the execution graph depicted in Figure 4.2. The probability of taking such a path is:

$$1 \cdot \frac{\mu_B}{\mu_B + \mu_C} \cdot \frac{\mu_D}{\mu_C + \mu_D + \mu_E} \cdot \frac{\mu_C}{\mu_C + \mu_E} \cdot \frac{\mu_F}{\mu_E + \mu_F} \cdot 1 \cdot 1$$

and in the case where all tasks have the same mean service time, the probability of taking such a path becomes $\frac{1}{24}$.

Suppose there are M paths from the initial stage to the terminating stage in the execution graph, and let $P(m)$, $m=1, \dots, M$ represent the probability that a newly arriving job takes path m in the execution graph. Therefore, we may think of our job arrival process as composed of M Poissonian arrival processes, the m th of which has an average rate $\lambda(m) = \lambda P(m)$, $m=1, \dots, M$. Moreover, the number of stages in any given path is equal to the number of tasks, N , in the process graph, and consequently a job is a chain of N specific execution stages and is hereafter regarded as requiring service N times. Upon arrival to the multiprocessing system, a job is at its first execution stage. Upon the completion of this first stage, we may consider that the job immediately and instantaneously feeds back its second execution stage. At the completion of its N th stage, the job departs the system at once. The number of ready tasks a job has at any given time is equal to the concurrency (the number of ready tasks) of the stage the job is in at such a time. During its sojourn time in the system, a job participates with different concurrency levels and hence receives different grades of service. It is for this very reason that our multiprocessing system is hereafter called a *P-dimensional Discriminatory processor sharing With job Feedbacks* and denoted using the P-DPS-WF acronym.

The global state of our P-DPS-WF multiprocessing system is fully described by the vector $S = (n_1, \dots, n_i, \dots, n_L)$ where n_i , $i=1, \dots, L$ represents the number of jobs in the system which are in stage i of their execution. We can think of our multiprocessing system as a single node queueing network with L classes*. For a finite number of processors, the total capacity of the system is allocated to the different classes according to the discriminatory processor sharing discipline defined earlier. If at any given time, the state of the system is $(n_1, \dots, n_i, \dots, n_L)$, then the capacity proportion allocated to class i is:

* A job is of class i , $i=1, \dots, L$ if it is in its i th execution stage; n_i , $i=1, \dots, L$ is also the number of jobs of class i present in the system in steady state.

$$\max_{i=1, \dots, L} \frac{n_i f(i) P}{\left\{ P, \sum_{j=1}^L n_j f(j) \right\}}$$

since a class i job possesses $f(i)$ ready tasks, and $\sum_{j=1}^L n_j f(j)$ is the total number of ready tasks in the system. The total number of jobs in the system, on the other hand, is readily given by $n = \sum_{j=1}^L n_j$.

Let $P[n_1, \dots, n_i, \dots, n_L]$ be the steady state probability density function that the system is in state $(n_1, \dots, n_i, \dots, n_L)$. To obtain these probabilities for all the feasible states, one must find a solution to the global balance equations of the system. From the theory of queueing networks, we know that $P[n_1, \dots, n_i, \dots, n_L]$ has the *Product Form* and is efficiently computable under the following set of assumptions provided by Baskett, Chandy, Muntz and Palacios [Bask75], and subsequently by Chandy, Howard and Towsley: [Chan77]

1. **Allowable Scheduling Disciplines** : the disciplines allowed are: First Come First Served (FCFS), Processor-Sharing (PS), Last Come First Served Preemptive-Resume (LCFS-PR), and Infinite Servers (IS).
2. **Service Time Distribution** : the service times at an FCFS server must be exponentially distributed with the same mean for all classes. The service times at PS, LCFS-PR, and IS can have a general distribution, perhaps with different mean service times for different classes.
3. **State Dependent Service Rates** : the service rate at an FCFS server can depend only on the total queue length of said server. The service rate for a class at a PS, LCFS-PR, and IS servers may also depend on the queue length for that class, but not on the queue length of other classes. Moreover, the overall service rate of a subnetwork may depend on the total number of customers in the subnetwork.
4. **Interarrival time Distribution**: exogenous arrivals for a given class must be Poisson. In particular no bulk arrivals are permitted

From this set of assumptions, we can see that the third one (and perhaps the fourth one too) cannot be satisfied for our multiprocessing system. The third and fourth assumptions are usually referred to as the *Homogeneity Assumption*, which states that the service rate at each server for a particular class does not depend on the state of the system in any way except for the total queue length and the designated class's queue length at that server. This assumption essentially implies the following:

- a. *Single Resource Possession* : a customer may not be present (waiting for service or being served) at more than one server.
- b. *No Blocking* : the server's ability to render service is not controlled by any other servers.
- c. *Independent Customer Behavior* : there should not be any synchronization requirements. Interaction among customers is limited to queueing effects.
- d. *Local Information* : the service rate of any server depends solely on local queue length and not on the state of the rest of the system.
- e. *Fair Service* : if service rates differ by class, the service rate for a class depends only on the queue length of that class at that server, and not on the queue lengths of other classes. The servers may not discriminate against customers in a class depending on queue lengths in other classes.

Nevertheless, there are two cases we can identify where the $P[n_1, \dots, n_i, \dots, n_L]$ has a *Product Form* solution as given by the following Proposition.

Proposition 4.1

If $P[n_1, \dots, n_i, \dots, n_L]$ is the steady state probability density function that the P-DPS-WF multiprocessing system is at state $(n_1, \dots, n_i, \dots, n_L)$, then for the two following cases $P[n_1, \dots, n_i, \dots, n_L]$ has the *Product Form* solution:

1. Infinite number of processors, and
2. finite number of processors with $f(i)=F, i=1, \dots, L$ where F is any positive real constant.

Proof

We shall prove that for these two cases, the *homogeneity assumption* is satisfied. The proportion of the processors capacity, denoted in this proof by C_i , received by class i stages for $i=1, \dots, L$ when the state of the system is $(n_1, \dots, n_i, \dots, n_L)$, is $C_i = \frac{n_i f(i) P}{\max \left\{ P, \sum_{j=1}^L n_j f(j) \right\}}$. For

the case of an infinite number of processors, the proportion C_i becomes $C_i = n_i f(i), i=1, \dots, L$ which depends only on the number of stages of class i , and consequently assures the *homogeneity assumption* for *product form* solutions. For the case of constant concurrency degree, the same for all the stages, the proportion of the processors capacity, $C_i, i=1, \dots, L$, received by class i stages for $i=1, \dots, L$ when the state of the system is $(n_1, \dots, n_i, \dots, n_L)$, is:

$$C_i = \frac{n_i F P}{\max \left\{ P, \sum_{j=1}^L n_j F \right\}} = \frac{n_i P}{\max \left\{ \frac{P}{F}, \sum_{j=1}^L n_j \right\}}$$

the proportion C_i , $i=1, \dots, L$ depends then only on the number of class i stages and the total number of stages in the system; and hence satisfies the *homogeneity assumption* for the system to possess a *product form* solution.

The notion of an execution graph can be extended in a natural way, so that an execution stage i , $i=1, \dots, L$ may have any arbitrary concurrency degree $f(i)$. Two distinguished case are identified in the following definitions.

Definition 4.1

An Abstracted Execution Chain (AEC) is a chain of N execution stages, each of which may have any arbitrary positive real concurrency degree.

Definition 4.2

A Restricted Abstracted Execution Chain (RAEC) is a chain of N execution stages, each of which may have any arbitrary positive and integer concurrency degree. Moreover, the concurrency degree $f(i)$ of stage i , $i=1, \dots, N$ is less than or equal to N , and such that $\sum_{i=1}^N f(i) \leq \frac{N(N+1)}{2}$. An RAEC is said to be feasible if it actually corresponds to a given process graph with one starting task and one terminating task.

In the sequel, we shall use abstracted execution chains to show that the discriminatory processor sharing discipline with feedback forms a complete parameterized family of scheduling strategies. Restricted abstracted execution chains, on the other hand, shall be used to formulate upper and lower bounds on the job expected response time, and ascertain the process graphs that provide such upper and lower bounds.

Note that while an execution graph always corresponds to a given process graph, abstracted execution chains and restricted abstracted execution chains may not necessarily correspond to any real process graph description. Let the stages in the AEC and the RAEC be numbered from 1 to N . Three limiting cases of RAECs may be distinguished from definition (4.2); these are:

Definition 4.3

A Breadth First Execution Chain (BFEC) is an RAEC with $f(i)=N-i+1$ for $i=1,\dots,N$. A Depth First Execution Chain (DFEC) is an RAEC with $f(i)=i$ for $i=1,\dots,N$. An Egalitarian Execution Chain (EEC) is an RAEC with $f(i)=1$ for all $i=1,\dots,N$.

Let ρ denote the total utilization factor of our P-DPS-WF multiprocessing system. We can either express ρ using the process graph description, or its corresponding execution graph description. Using the process graph description, the utilization factor ρ may be expressed as follows:

$$\rho = \frac{\lambda}{P} \sum_{A \in \Omega} \frac{1}{\mu_A} \quad (4.2)$$

Using the execution graph description, we also obtain the above expression of the system utilization factor, for a job takes a given execution path in the execution graph, and in such a path every task appears exactly once. From [Klei75] we know that the stability of the system is maintained as long as ρ is less than unity.

4.2 The Infinite Number of Processors Case

We consider an infinite number of processors. We shall first develop an expression for the job average response time, where jobs may be represented by any given arbitrary process graph comprising N tasks. Then, and by the use of restricted abstracted execution chains, we formulate a tight upper bound and a tight lower bound on the average response time and provide the process graphs, among all possible process graphs comprising N tasks, which achieve these upper and lower bounds.

We now proceed to determine the average input (i.e., arrival) rate to each stage in the job execution graph. Recall that the average arrival rate of jobs to the system is λ , that the number of levels in the execution graph is equal to the number N of tasks in the process graph, and that the number of stages in the execution graph is L , with $L \geq N$. Let λ_i , $i=1,\dots,L$ represent the average arrival rate of class i jobs (i.e., the average input rate to stage i). We readily have $\lambda = \lambda_1 = \lambda_L$, and using the transition probabilities between the stages yields:

$$\lambda_i = \sum_{j=1}^{i-1} P_{ij} \lambda_j \quad i=2,\dots,L \quad (4.3)$$

The sum of the average input rate to all execution stages in any given level of the execution graph is λ . Since there are exactly N levels in the execution graph, we must have:

$$\sum_{i=1}^L \lambda_i = N\lambda$$

An execution stage may belong to several paths in the execution graph. The probability that stage i is visited during a job execution is then given by $\frac{\lambda_i}{\lambda}$. On the other hand, the expected time a job spends in stage i is given by $\frac{1}{\sum_{k \in \alpha(i)} \mu_k}$. Consequently, the average response time,

denoted by T_{∞} , of a job with an arbitrary process graph comprising N tasks is given by:

$$T_{\infty} = \sum_{i=1}^L \frac{\lambda_i}{\lambda \sum_{k \in \alpha(i)} \mu_k} \quad (4.4)$$

For the case of the same average service time, say $\frac{1}{\mu}$, for all the N tasks, equation (4.4) reduces to:

$$T_{\infty} = \frac{1}{\lambda\mu} \sum_{i=1}^L \frac{\lambda_i}{f(i)} \quad (4.5)$$

Let $T_{\infty,UB}$ and $T_{\infty,LB}$ represent, respectively, the upper bound and the lower bound on the expected response time of jobs with an arbitrary process graph comprising N tasks, and in a system with an infinite number of processors. The following two Theorems provide the exact values of $T_{\infty,UB}$ and $T_{\infty,LB}$.

Theorem 4.1

The upper bound $T_{\infty,UB}$ on the expected response time of jobs having any given arbitrary process graph comprising N tasks is given by:

$$T_{\infty,UB} = \sum_{i=1}^N \frac{1}{\mu_i}$$

Proof

From equation (4.4), we obtain the following nonlinear program to solve:

$$\text{Maximize } A = \sum_{i=1}^L \frac{\lambda_i}{\sum_{k \in \alpha(i)} \mu_k} \quad \text{Subject to: } \begin{cases} 1. \sum_{i=1}^L \lambda_i = \lambda N \\ 2. f(i) \geq 1, \forall i=1, \dots, L \\ 3. \lambda_i \leq \lambda \\ 4. L \geq N \end{cases}$$

Consider the i th term in the expression of A . Maximizing $\frac{\lambda_i}{\sum_{k \in \alpha(i)} \mu_k}$ is the same as maximizing

λ_i and minimizing the sum $\sum_{k \in \alpha(i)} \mu_k$, for all $i=1, \dots, L$. The maximum value of $\lambda_i, i=1, \dots, L$ is λ .

The minimum of the sum $\sum_{k \in \alpha(i)} \mu_k, i=1, \dots, L$ is obtained when $f(i)=1$. On the other hand, if $f(i)=1, i=1, \dots, L$ then $L=N$. All the four constraints are also satisfied, and A becomes $A = \lambda \sum_{i=1}^N \frac{1}{\mu_i}$.

It is rather interesting to note that among all possible process graphs comprising N tasks, the process graph $PG(N,N)$ is the one that maximizes the expected job response time; such an average response time is exactly $T_{\infty,UB}$. Also in the special case of the same average service time, say $\frac{1}{\mu}$, for all the N tasks, we have $T_{\infty,UB} = \frac{N}{\mu}$. Although in general it is hard to infer the total number of stages in an execution graph obtained from an arbitrary process graph, the next Lemma identifies the process graphs $PG(N,r)$ comprising N tasks and $r, r=1, \dots, N$ levels which result in the smallest, respectively the largest, number of stages in their corresponding execution graphs $EG(X,N)$.

Lemma 4.1

1. Among all possible process graphs comprising N tasks (i.e., $PG(N,r)$ for $r=1, \dots, N$), the process graph $PG(N,N)$ gives the execution graph with the smallest number of stages; this number of stages is equal to N .
2. Among all possible process graphs comprising N tasks (i.e., $PG(N,r)$ for $r=1, \dots, N$), the process graph $PG(N,1)$ gives the execution graph with the largest number of stages; this number of stages is equal to $(2^N - 1)$.

Proof

First, we prove statement one. Since the execution graph obtained from a PG(N,r) has exactly N levels, and since the minimum number of execution stages per level is one, it follows that the minimum number of stages in an execution graph is also equal to N. We now proceed to prove the second statement. It is not difficult to see that the process graph PG(N,1) gives the execution graph with the largest number of paths; and hence the largest number of stages. On the other hand, the sets $\alpha(i), i=1, \dots, L$ are the *Power set* of the set of tasks, without the empty set. The proof follows since the cardinality of the *Power set* is 2^N .

111

Theorem 4.2

The lower bound $T_{=,LB}$ on the expected response time of jobs having any given arbitrary process graph comprising N tasks, and such that the tasks have the same average service time, $\frac{1}{\mu}$, is given by:

$$T_{=,LB} = \frac{1}{\mu} \sum_{i=1}^N \frac{1}{i}$$

Proof

From equation (4.5), we obtain the following nonlinear program to solve:

$$\text{Minimize } A = \sum_{i=1}^L \frac{\lambda_i}{f(i)} \quad \text{Subject to: } \begin{cases} 1. \sum_{i=1}^L \lambda_i = \lambda N \\ 2. f(i) \geq 1, \forall i=1, \dots, L \\ 3. \lambda_i \leq \lambda \\ 4. L \geq N \end{cases}$$

Since the execution graph has N levels, we can write:

$$A = \sum_{j=1}^N \left\{ \sum_{i \in \text{level } j} \frac{\lambda_i}{f(i)} \right\}$$

Now, let us proceed to minimize level by level under the stated set of constraints. Since for all levels j, $j=1, \dots, L$ we have $\sum_{i \in \text{level } j} \lambda_i = \lambda$, our minimization problem is equivalent to maximizing the number of stages per level and for all levels; hence maximizing the total number of stages in the execution graph. From Lemma 4.1, we already know that the process graph that gives the execution graph with the largest number of stages is the PG(N,1). On the other hand, the

process graph PG(N,1) is the process graph where a job arrives as a bulk of N concurrent tasks.

The service time of such a bulk of N parallel tasks is $\max_{A \in \Omega} \{ \tilde{X}_A \}$, and consequently:

$$T_{\infty, LB} = \int_0^{\infty} [1 - (1 - e^{-\mu t})^N] dt = \frac{1}{\mu} \sum_{i=1}^N \frac{1}{i}$$

III

4.3 The Uniprocessor Case

In the case of a uniprocessor system, we may consider the nodes in the process graph as having assigned priority levels. This system can then be studied by means of the M/G/1 queueing system with job feedback as developed in Chapter 6. However, we are mostly interested in the study of the discriminatory processor sharing service discipline.

Very few studies of the M/G/1 queueing system with the discriminatory processor sharing discipline have appeared in the literature and none to our best knowledge if we also have job feedback. Kleinrock [Klei67] was the first to introduce such a strategy for a single processor system with M job classes and no feedback, and provided an expression for the steady state expected response time of a class k job whose required service time is t. Under a different set of assumptions, and using a different analysis method, O'Donovan [O'Do74] obtained the same expression. More recently, Fayolle, Iasnogorodshi, and Mitrani [Fayo78] presented another solution to the same problem. Their analysis method follows O'Donovan's approach in deriving a system of integro-differential equations for the steady state expected response time of a class k job whose required service time is t. The system of equations was solved, for general distributions of the required service times, by the method of Wiener-Hopf. In the case of exponentially distributed required service times, the authors in [Fayo78] provided a system of linear equations for the unconditional steady state average response times. In this section, we first prove that the 1-DPS-WF family of scheduling strategies forms a complete family in the sense that any response time requirement that can be satisfied at all, can be achieved by a strategy from the family. Then, we proceed to investigate the job average response time in the 1-DPS-WF system, and present an accurate and yet very simple approximation. Finally, we develop and prove a conservation law that puts a linear equality constraint on the set of expected system times of the different stages representing the job execution using the discriminatory processor sharing discipline with feedback.

4.3.1 Completeness of the 1-DPS-WF family of Scheduling Strategies

we now proceed to prove that the 1-DPS-WF with jobs represented by an AEC forms a complete parameterized family of scheduling strategies. A performance requirement stated in terms of the average system times of the different stage types, is said to be achievable if, given the loading conditions on the system (i.e., given the $\rho_i, i=1, \dots, N$), there exists a scheduling strategy which satisfies it. A family of scheduling strategies is said to be complete if every achievable performance requirement can be satisfied by a strategy from the family. Let the performance of the system, given the ρ_i 's, $i=1, \dots, N$, be measured by the vector $T = [T_1, \dots, T_i, \dots, T_N]$. If, for a given scheduling strategy S , the value of the performance vector is T , we say that S achieves T and denote it by $S \Rightarrow T$. A given performance vector T is said to be achievable, if there exists a scheduling strategy S such that $S \Rightarrow T$ (S need not be unique). Denote the set of all achievable performance vectors by H ; we have:

$$H = \left\{ T \mid \exists S : S \Rightarrow T \right\}$$

It is obvious that not all performance vectors (e.g., $T=(0,0,\dots,0)$) are achievable. Let Φ be a family of scheduling strategies, and let H_Φ represent the set of all performance vectors that can be achieved using strategies from the set Φ . That is :

$$H_\Phi = \left\{ T \mid \exists S : S \in \Phi \text{ and } S \Rightarrow T \right\}$$

We say that the family Φ is complete if $H_\Phi = H$. In other words, the family Φ is complete if any performance vector T which can be achieved at all, can be achieved by a strategy from the family Φ . Note that no finite or denumerable family of scheduling strategies can be complete.

Let P_1, P_2, \dots, P_{MP} be the performance vectors of the MP preemptive priority disciplines which can be operated with the N stages. These MP vectors are the vertices or "corners" of the set H . Moreover, the set H is a convex hull defined by these vertices (and is an $(N-1)$ dimensional set because it lies on the hyperplane defined by the generalized conservation law defined and stated in Chapter 6). In other words, the boundary of the convex hull H consists of the performance vectors which correspond to strategies giving one or more stages preemptive priority over the remaining ones. These corners of the set H represent then the extremes of the system performance (best for some stages and worst for others).

Returning to our 1-DPS-WF family of strategies, it is rather easy to see that for any given strategy in such a family (i.e., any given vector $(f(1), \dots, f(i), \dots, f(N))$), multiplying all the $f(i), i=1, \dots, N$ by the same constant does not change the strategy. Therefore one of the $f(i)$'s can be arbitrary fixed; let $f(N)=1$. We have now an $(N-1)$ dimensional parameter set G defined by:

$$G = \left\{ (f(1), \dots, f(i), \dots, f(N-1)) \mid f(i) > 0; i=1, \dots, N-1 \right\}$$

Each point in the set G uniquely determines a 1-DPS-WF strategy, and consequently a performance vector T . Moreover, it is rather easy to see that this correspondence is one-to-one and continuous. Let Ψ denote such a family of strategies. Since the parameter set G is open, it follows that the set H_Ψ of performance vectors achievable by strategies from the family Ψ is also open and therefore $H_\Psi \neq H$; that is, Ψ is not complete. In fact, the performance vector of any preemptive priority discipline cannot be achieved by a 1-DPS-WF strategy because the latter would not allow a stage in the system to be completely deprived of service. However, the family Ψ is *almost complete* in the sense given by the following Theorem:

Theorem 4.3

The set of performance vectors achievable by strategies from the family Ψ , H_Ψ , is equal to the set H of all achievable performance vectors without its boundary. If a performance vector T is an inside point of H , then it can be achieved by a strategy from Ψ ; and if T is on the boundary of H , then it can be approximated as closely as desired by strategies from Ψ .

Proof

Consider the parameter subsets defined by:

$$G_{L,U} = \left\{ (f(1), \dots, f(i), \dots, f(N-1)) \mid L \leq f(i) \leq U; i=1, \dots, N-1 \right\}$$

where L and U are positive real numbers, and let $\Psi_{L,U}$ denote the family of 1-DPS-WF strategies defined over $G_{L,U}$. We then have:

$$G = \lim_{L \rightarrow 0, U \rightarrow \infty} G_{L,U} \quad \text{and} \quad H_\Psi = \lim_{L \rightarrow 0, U \rightarrow \infty} H_{\Psi_{L,U}}$$

The boundary of the set $G_{L,U}$ consists of those points $(f(1), \dots, f(i), \dots, f(N-1))$ for which $f(i)=L$ for at least one i and/or $f(j)=U$ for at least one j , $i=1, \dots, N-1$ and $j=1, \dots, N-1$. Let $B_{L,U}$ be the set of performance vectors which corresponds to these boundary points in $G_{L,U}$. Now, since the set $G_{L,U}$ is compact and there is a one-to-one correspondence between the set $G_{L,U}$ and the set $\Psi_{L,U}$, then the set $H_{\Psi_{L,U}}$ consists of the set $B_{L,U}$ and all performance vectors inside it. Moreover, the set $B_{L,U}$ is a closed and continuous surface since it is the image of a closed and continuous surface by a continuous mapping. Let T be any arbitrary performance vector such that it is an inside point of H . Because $B_{L,U}$ is continuous, then there must exist a sufficiently small L , and a sufficiently large U such that the performance vector T is an inside point of the set $B_{L,U}$. This means that T belongs to $H_{\Psi_{L,U}}$ and hence T belongs to H_Ψ .

□

The above Theorem states the special fact that if a performance vector T is on the boundary of the convex hull H , then it can be approximated as closely as desired by strategies from the family Ψ . In particular, the preemptive priority ordering $(1 > 2 > \dots > N)$ can be achieved by considering the point in the set G defined by $f(N)=1$, $f(i) \rightarrow \infty$ $i=1, \dots, N-1$, and such that $\frac{f(i)}{f(i+1)} \rightarrow \infty$, $i=1, \dots, N-1$. Likewise, the preemptive priority ordering $(1 < 2 < \dots < N)$ is achieved by considering the point in the set G defined by $f(1)=1$, $f(i) \rightarrow \infty$ $i=2, \dots, N$, and such that $\frac{f(i+1)}{f(i)} \rightarrow \infty$, $i=1, \dots, N$.

4.3.2 The Job Average Response Time

We now proceed to investigate the job average response time in the 1-DPS-WF system. Jobs are described according to a given process graph with N tasks. The process graph, in the sequel, is assumed to have only one *starting task* and only one *terminating task*; Figure 4.1 is an example.

We shall first determine, among all possible RAECs with N stages, the RAEC that provides the lowest job average response time, and the RAEC that results in the highest job average response time. We then discuss and present a rather accurate approximation of the job average response time. Simulations are used to back up and validate the accuracy of the approximation.

Theorem 4.4

Among all possible RAECs with N stages, the BFEC is the RAEC which provides the largest job average response time, and the DFEC is the RAEC which provides the lowest job average response time.

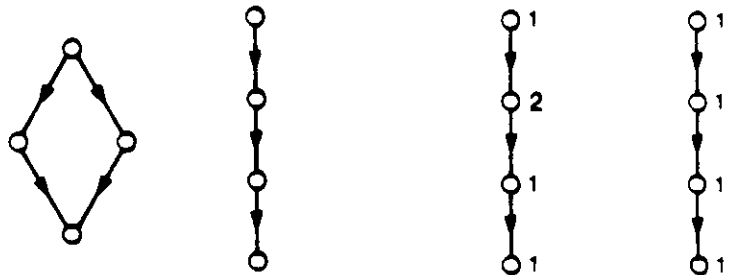
Proof

Among all the RAECs with N stages, the RAEC which results in the earliest job completions is the one that implicitly allocates the highest priority to the jobs having the least remaining processing time; this is the DFEC. The RAEC which results in the furthest job completions on the other hand, is the one that implicitly allocates the highest priority to the jobs having the largest remaining processing time; this is the BFEC. The proof is complete since the unfinished work in the system, at any time, is independent of the way the server capacity is shared among the different jobs present in the system (i.e., independent of the RAEC representing the jobs).

Although the BFEC and the DFEC do not correspond to any given process graph, they present, by means of Theorem 4.4, respectively upper and lower bounds on the job average response time. Many RAECs, among all possible RAECs with N stages, do not actually correspond to a given process graph. Indeed for $N=3$, the only feasible RAEC is the one with $f(1)=f(2)=f(3)=1$; for the number of process graphs with N tasks is one, the process graph represented by a chain of three tasks. There is also a unique process graph for $N=2$, the one corresponding to the RAEC with $f(1)=f(2)=1$. Among all the RAECs with $N=4$ stages, only two are feasible, for there are only two possible process graphs having $N=4$ tasks, and each one of them corresponds to a unique feasible RAEC. These two possible process graphs with $N=4$ are depicted in Figure 4.3:(a) and their correspondent RAECs are depicted in Figure 4.3:(b). For $N=5$, we obtain 4 possible process graphs, each of which corresponds to a unique RAEC. Figure 4.4:(a) represents these 4 possible process graphs, and Figure 4.4:(b) depicts their corresponding RAECs.

From the above, we observe that for any given N , there are only very few feasible RAECs. Moreover, the feasible RAECs are by no means extreme cases. In fact, if there exists a stage i , $i=1, \dots, L$ in the execution graph such that $f(i) \geq 2$, then the stages immediately before the last in the execution graph must have a concurrency degree of one. In other words, any path in the execution graph has the inherent property that $f(1)=f(L-1)=f(L)=1$ for any arbitrary given process graph comprising N tasks. This inherent property of the execution graph assures that the job average response time should be much closer to the average response time given by the EEC with N stages, than to the average response time obtained by using either the BFEC or the DFEC with the same number of stages. The average response time resulting from the EEC may be considered somehow as a median among the average response times given by any feasible RAEC. For large values of N (i.e., $N \geq 5$), the resulting execution graph may have many execution paths. These paths are obviously feasible RAECs with N stages. Due to the inherent property of the execution graph, namely that $f(1)=f(L-1)=f(L)=1$, we may suggest that the majority of the RAECs representing the execution paths provide somehow slightly larger values of the average response time than the one resulting by using the EEC with the same number of stages. Consequently, the EEC represents an optimistic and good approximation of the average response time of jobs having any arbitrary given process graph.

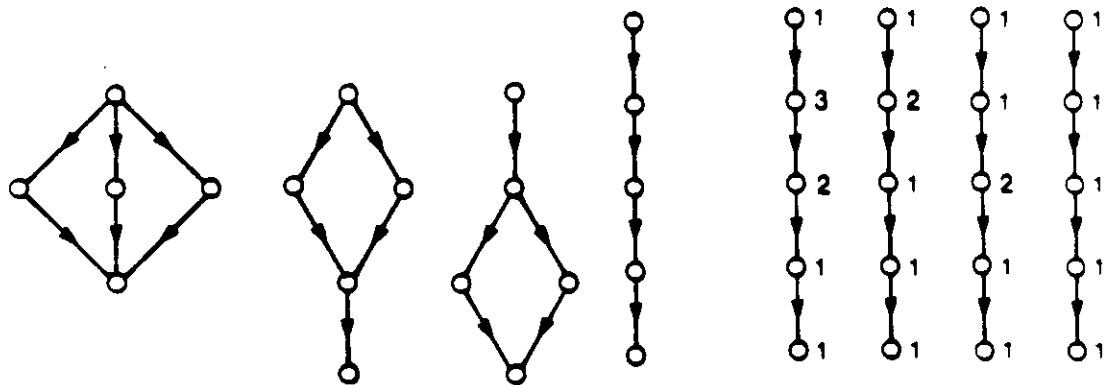
From Proposition 4.1, we know that the steady state probabilities $P[n_1, \dots, n_i, \dots, n_L]$ that the uniprocessor system is in state $(n_1, \dots, n_i, \dots, n_L)$ have a *Product Form* solution when the jobs are represented in the system by their EEC. The uniprocessor system can be seen as a single node BCMP [Bask75, Chan77] queueing network with N classes. A job in the system is of class i , $i=1, \dots, N$ if it is at its i th execution stage. Since the service time of task i , $i=1, \dots, N$ is assumed to be exponentially distributed with an average $\frac{1}{\mu_i}$, and since



(a): Process Graphs

(b): Execution Graphs

Figure 4.3: Process Graphs and their Corresponding Execution Graphs for $N=4$



(a): Process Graphs

(b): Execution Graphs

Figure 4.4: Process Graphs and their Corresponding Execution Graphs for $N=5$

$\lambda_i = \lambda$ for $i=1, \dots, N$, it follows that *:

$$P[n_1, \dots, n_i, \dots, n_N] = P[0, \dots, 0, \dots, 0] \left(\sum_{i=1}^N n_i \right)! \prod_{i=1}^N \frac{(\rho_i)^{n_i}}{n_i!} \quad n_i \geq 0 \quad i=1, \dots, N \quad (4.5)$$

where $\rho_i = \frac{\lambda}{\mu_i}$, $i=1, \dots, N$. Let $n = \sum_{i=1}^N n_i$, and $P[n]$ be the steady state probability that there are n jobs in the system. From equation (4.5), we obtain:

$$P[n] = P[0] \rho^n \quad n \geq 0$$

where $\rho = \sum_{i=1}^N \rho_i = \sum_{i=1}^N \frac{\lambda}{\mu_i}$, and since $\sum_{i=1}^N P[n] = 1$, we get:

$$P[n] = (1-\rho)\rho^n \quad n \geq 0 \quad (4.6)$$

and consequently, if we let \bar{n} denote the average number of jobs in the system in the steady state, we get from equation (4.6) and using the fact that $\bar{n} = \sum_{n=0}^{\infty} nP[n]$:

$$\bar{n} = \frac{\rho}{1-\rho}$$

Since $\lambda = \frac{\rho}{\sum_{i=1}^N \frac{1}{\mu_i}}$, and using Little's result; namely that $\bar{n} = \lambda T(1)$, where $T(1)$ denotes the average response time, we have:

$$T(1) = \frac{\sum_{i=1}^N \frac{1}{\mu_i}}{1-\rho} \quad (4.7)$$

Equation (4.7) represents an approximation (an optimistic approximation) of the job average response time of jobs having any arbitrary given process graph comprising N tasks. To validate this approximation, we simulated the 1-DPS-WF system using the process graph description depicted in Figure 4.1. The method used to estimate the extent of the simulation transient state is *the method of independent replications* [Lave83], and the method used to estimate the statistic $T(1)$ for the 1-DPS-WF system in the steady state is *the method of batch means* [Lave83]. Figure 4.5 depicts the job average response time given by equation (4.7) along with the simulation results represented by the confidence intervals. The confidence intervals are depicted in Figure 4.5 as vertical bars, are obtained from the simulation output via the t -distribution, and are at the 90% level.

* For the sake of clarity and since we are working on the EEC, we are using here numbers instead of letters to identify the different tasks.

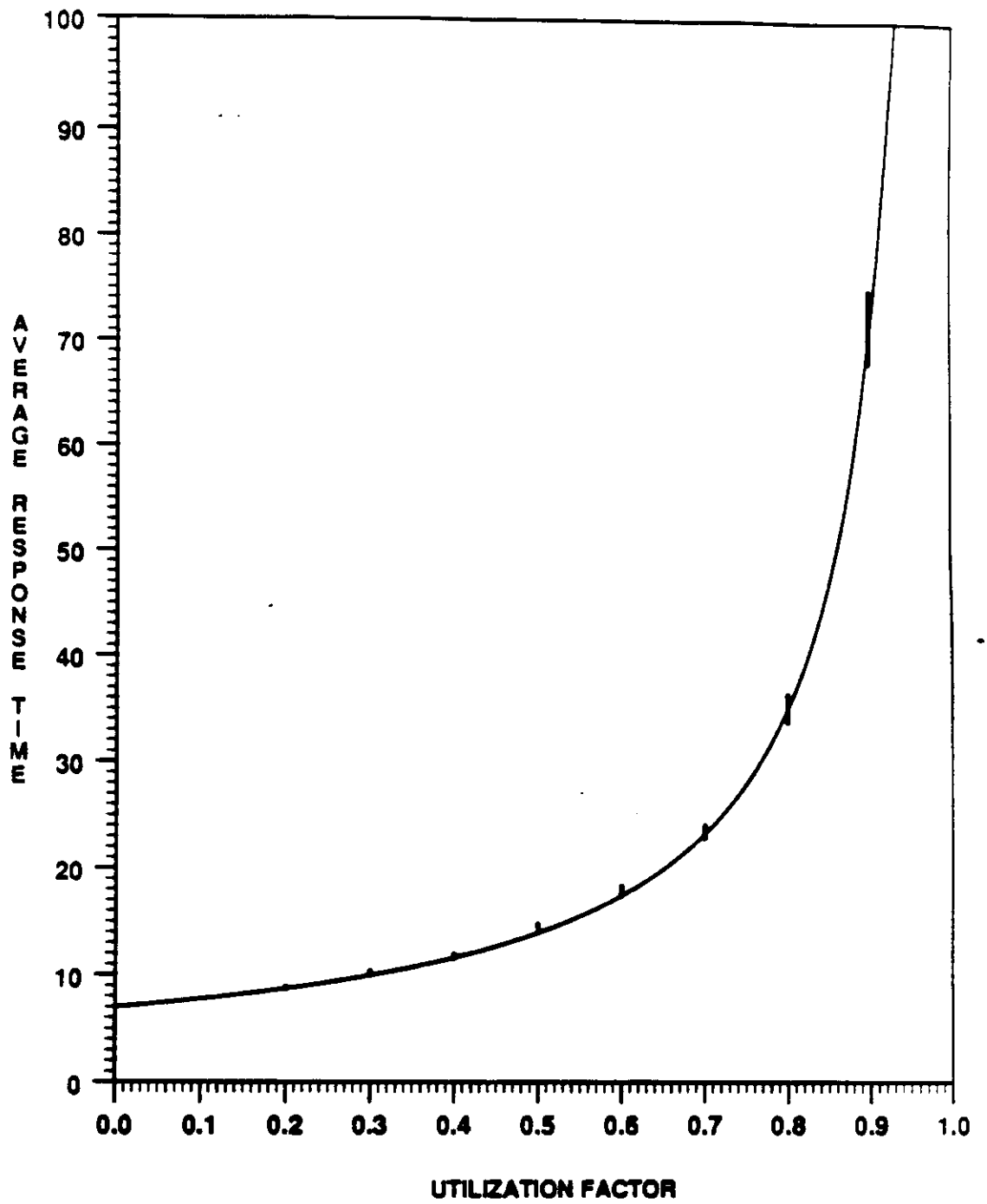


Figure 4.5: Average Response Time in a 1-DPS-WF System

From Figure 4.5, we observe that equation (4.7) represents a very accurate approximation of the job average response time in the 1-DPS-WF system and for jobs having the process graph depicted in Figure 4.1. Over all permissible ranges of the system utilization factor, the approximation is well within the 90% confidence intervals. It is also rather interesting to notice the narrowness of these confidence intervals. The optimistic character of the approximation may be observed on Figure 4.5 for moderate values of the system utilization factor (the average response time curve intersects the confidence intervals at their lower parts). For either small or high values of the utilization factor, such an optimistic behavior is much less noticeable.

4.3.3 The Conservation Law

In this section, we develop a conservation law that puts a linear equality constraint on the set of average system times of the different stages in the abstracted execution chain that represents the job execution in the uniprocessor system using the discriminatory processor sharing discipline with feedback. In [Klei76], Kleinrock established the conservation law for the M/G/1 queueing system and for any non-preemptive work-conserving queueing discipline. In a similar fashion, we shall use the fact that the unfinished work in the system is invariant to the sharing of the server capacity, provided that the sharing discipline does not explicitly rely on information about the remaining processing time of any job in the system. In Section 6.3.5 of Chapter 6, we shall provide a generalization of the conservation law for any M/G/1 queueing system with job feedback, and any non-preemptive work-conserving priority scheduling of the different stages constituting the job.

Let T_i , $i=1,\dots,N$ represent the average time spent in the uniprocessor system by stage i from the time of its arrival to the system until its completion. For stage i , $i=1,\dots,N$, let $\frac{1}{\mu_i}$ be the stage average processing time, and $\rho_i = \frac{\lambda_i}{\mu_i}$ be the uniprocessor utilization due to stages of type i . We consider that all jobs have the same AEC, and hence $\lambda_i = \lambda$, $i=1,\dots,N$, and $\rho_i = \frac{\lambda}{\mu_i}$, $i=1,\dots,N$. Recall that the $f(i)$, $i=1,\dots,N$ are arbitrary and continuous positive real values.

Theorem 4.5 : The 1-DPS-WF Conservation Law

For an exponential work-conserving uniprocessor system using the discriminatory processor sharing discipline with jobs described by an abstracted execution chain, it must be, for any choice of the concurrency degrees $f(i)$, $i=1,\dots,N$, that:

$$\sum_{i=1}^N \left[\sum_{j=i}^N \rho_j \right] T_i = \frac{\rho}{1-\rho} \frac{\sum_{j=1}^N \frac{1}{\mu_j}}{N}$$

where $\rho = \sum_{i=1}^N \rho_i < 1$.

Proof

let us first prove that $\sum_{i=1}^N \left[\sum_{j=i}^N \rho_j \right] T_i = \text{constant}$. Let \bar{U} denote the average unfinished work in the system, and $P[n_1, \dots, n_i, \dots, n_N]$ denote the steady state probability that there are n_i , $i=1, \dots, N$ stages of type i in the system. A stage of type i found in the system participates, in the unfinished work, by its remaining service time plus the service time of its fed back stages; namely stages $i+1, \dots, N$. Therefore, if the state of the system is $(n_1, \dots, n_i, \dots, n_N)$, and by using the Markovian property of the exponential service time distributions, the unfinished work in the system, given such a state, is $\sum_{i=1}^N n_i \sum_{j=i}^N \frac{1}{\mu_j}$. Consequently, the average unfinished work in the system, \bar{U} , is given by:

$$\bar{U} = \sum_{n_1=0}^{\infty} \cdots \sum_{n_i=0}^{\infty} \cdots \sum_{n_N=0}^{\infty} P[n_1, \dots, n_i, \dots, n_N] \left[\sum_{i=1}^N n_i \sum_{j=i}^N \frac{1}{\mu_j} \right]$$

which amounts to:

$$\bar{U} = \sum_{i=1}^N \bar{n}_i \sum_{j=i}^N \frac{1}{\mu_j}$$

where \bar{n}_i is the average number of jobs in execution stage i in the system in steady state. Using Little's result [Litt61], namely that $\bar{n}_i = \lambda_i T_i$, $i=1, \dots, N$ and recalling that the unfinished work in the system is invariant to the sharing discipline of the server capacity among the stages present in the system, completes our first proof. Now let us take the special case of an EEC chain to represent the job description. For this case, we have:

$$T_i = \frac{\sum_{j=i}^N \frac{1}{\mu_j}}{1-\rho} \frac{1}{N} \quad i=1, \dots, N$$

Using these values completes the proof. ■

In the sequel, we shall develop an approximation for the job average response time in the P-DPS-WF multiprocessing system based on the EEC approximation. But first, we shall investigate the P-DPS-WF multiprocessing system where all the stages in the execution graph have the same concurrency degree. Results from such a system will also be used in the P-DPS-WF

average response time approximation.

4.4 The P-DPS-WF Multiprocessing System with the Same Concurrency Degree for all Stages

In this section, we study the P-DPS-WF multiprocessing system where jobs are represented by a given execution graph with stages having the same concurrency degree, denoted hereafter by F (i.e., $f(i)=F, i=1, \dots, L$). A job is therefore described by a chain of N execution stages with the same concurrency degree F . Let the state of the system be $S = (n_1, \dots, n_i, \dots, n_N)$ where $n_i, i=1, \dots, N$ represents the number of jobs which are present in the system and are at their execution stage number i . We shall refer to $n_i, i=1, \dots, N$ as the number of class i jobs. Let $\rho_i, i=1, \dots, N$ denote the utilization factor of the multiprocessing system due to class i jobs, and ρ denote the total utilization factor of the system. Hence we have $\rho = \sum_{i=1}^N \rho_i$. To evaluate the ρ_i 's, $i=1, \dots, N$, recall [Klei75] that the utilization factor is in a fundamental sense the ratio of the rate at which work enters the system to the maximum rate (i.e., capacity) at which the system can perform this work. The work an arriving customer brings to the system equals the number of seconds of service he requires. If $\frac{1}{\mu_i}, i=1, \dots, N$ represents the average service demand of a class i job, we then have $\rho_i = \frac{\lambda}{\mu_i}, i=1, \dots, N$ independently of the value of F . Therefore, we have:

$$\rho = \frac{\lambda}{P} \sum_{i=1}^N \frac{1}{\mu_i} \quad (4.8)$$

Since from Proposition 4.1, we know that the multiprocessing system under investigation has a *product form* solution, we can use the $M \Rightarrow M$ conditions [Bask75, Chan77, Munt73] to determine the steady state probabilities $P(n_1, \dots, n_i, \dots, n_N)$. If the system state is $(n_1, \dots, n_i, \dots, n_N)$, then the departure rate of class i jobs, denoted hereafter by d_i , is given by:

$$d_i = \frac{n_i F P}{\max \left\{ P, \sum_{j=1}^N n_j F \right\}} \mu_i \quad i=1, \dots, N$$

since $\frac{n_i F P}{\max \left\{ P, \sum_{j=1}^N n_j F \right\}}$ is the proportion of the processors capacity allocated to class i jobs.

The above equation may be rewritten as:

$$d_i = \frac{n_i P \mu_i}{\max \left\{ \frac{P}{F}, \sum_{j=1}^N n_j \right\}} \quad i=1, \dots, N \quad (4.9)$$

Now using the $M \Rightarrow M$ conditions [Munt73], we have:

$$\frac{P[n_1, \dots, n_i+1, \dots, n_N] \frac{(n_i+1)P\mu_i}{\max \left\{ \frac{P}{F}, 1 + \sum_{j=1}^N n_j \right\}}}{P[n_1, \dots, n_i, \dots, n_N]} = \lambda_i$$

for all $i=1, \dots, N$ and for all the feasible states. Let χ denote the set of all the feasible states; that is $\chi = \left\{ (n_1, \dots, n_i, \dots, n_N) \mid \forall i=1, \dots, N \ n_i \geq 0 \right\}$. The above equation yields:

$$P[n_1, \dots, n_i+1, \dots, n_N] = \frac{\lambda_i}{\mu_i P} \frac{1}{n_i+1} \max \left\{ \frac{P}{F}, 1 + \sum_{j=1}^N n_j \right\} P[n_1, \dots, n_i, \dots, n_N] \quad (4.10)$$

$$\forall i=1, \dots, N \text{ and } \forall (n_1, \dots, n_i, \dots, n_N) \in \chi$$

Using equation (4.10) repetitively and starting from $P[1,0, \dots, 0]$, we can express the probability of any feasible state as a function of the probability $P[0, \dots, 0]$ of the system being empty. We obtain:

$$P[n_1, \dots, n_i, \dots, n_N] = P[0, \dots, 0] \prod_{i=1}^N \left\{ \left[\frac{\lambda_i}{\mu_i P} \right]^{n_i} \frac{1}{n_i!} \prod_{j=1}^{n_i} \max \left\{ \frac{P}{F}, \sum_{k=1}^{i-1} n_k + j \right\} \right\} \quad (4.11)$$

$$\forall (n_1, \dots, n_i, \dots, n_N) \in \chi$$

Now since we have:

$$\prod_{i=1}^N \left\{ \prod_{j=1}^{n_i} \max \left\{ \frac{P}{F}, \sum_{k=1}^{i-1} n_k + j \right\} \right\} = \prod_{j=1}^{\sum_{i=1}^N n_i} \max \left\{ \frac{P}{F}, j \right\}$$

and by letting $n = \sum_{i=1}^N n_i$, equation (4.11) yields:

$$P[n_1, \dots, n_i, \dots, n_N] = P[0, \dots, 0] \frac{\prod_{j=1}^n \max \left\{ \frac{P}{F}, j \right\}}{P^n} \prod_{i=1}^N \left[\frac{\lambda_i}{\mu_i} \right]^{n_i} \frac{1}{n_i!} \quad (4.12)$$

$$\forall (n_1, \dots, n_i, \dots, n_N) \in \chi$$

Equation (4.12) gives the probability of any feasible state as a function of the probability $P[0, \dots, 0]$ of the system being empty. In the sequel, we develop a closed form expression for the probability $P[n]$ of having a total of n jobs in the system in steady state. Let $m = \lfloor \frac{P}{F} \rfloor$ denoting the largest integer less or equal to $\frac{P}{F}$. We distinguish two cases depending on the value of m .

Case of $n \leq m$

Since $\prod_{j=1}^n \max \left\{ \frac{P}{F}, j \right\} = \left[\frac{P}{F} \right]^n$, equation (4.12) gives:

$$P[n_1, \dots, n_i, \dots, n_N] = P[0, \dots, 0] \frac{1}{F^n} \prod_{i=1}^N \left[\frac{\lambda_i}{\mu_i} \right]^{n_i} \frac{1}{n_i!}$$

$$\forall (n_1, \dots, n_i, \dots, n_N) \in \chi \text{ and } n \leq m$$

Consequently, we have:

$$\begin{aligned} P[n] &= \frac{P[0, \dots, 0]}{F^n} \sum_{\substack{\sum_{i=1}^N n_i = n \\ n_i \geq 0}} \left\{ \prod_{i=1}^N \left[\frac{\lambda_i}{\mu_i} \right]^{n_i} \frac{1}{n_i!} \right\} \\ &= \frac{P[0, \dots, 0]}{F^n n!} \left[\sum_{i=1}^N \frac{\lambda_i}{\mu_i} \right]^n \end{aligned}$$

since $P[0, \dots, 0] = P[0]$, and $\sum_{i=1}^N \frac{\lambda_i}{\mu_i} = P\rho$, the above equation yields:

$$P[n] = P[0] \frac{\left[\frac{P\rho}{F} \right]^n}{n!} \quad \forall n \leq m \quad (4.13)$$

Case of $n > m$

Since $\prod_{j=1}^n \max \left\{ \frac{P}{F}, j \right\} = \left[\frac{P}{F} \right]^m \frac{n!}{m!}$, equation (4.12) yields:

$$P[n_1, \dots, n_i, \dots, n_N] = P[0, \dots, 0] \left[\frac{P}{F} \right]^m \frac{n!}{m!} \frac{1}{F^n} \prod_{i=1}^N \left[\frac{\lambda_i}{\mu_i} \right]^{n_i} \frac{1}{n_i!}$$

$$\forall (n_1, \dots, n_i, \dots, n_N) \in \chi \text{ and } n > m$$

Consequently and after some algebra, we obtain:

$$P[n] = P[0] \frac{\left[\frac{P}{F}\right]^m}{m!} \rho^n \quad \forall n > m \quad (4.14)$$

We now proceed to evaluate the steady state probability $P[0]$ of the system being empty. Using the fact that $\sum_{n=0}^{\infty} P[n] = 1$, and equations (4.13) and (4.14), we have:

$$P[0]^{-1} = \sum_{n=0}^m \left[\frac{P\rho}{F}\right]^n \frac{1}{n!} + \sum_{n=m+1}^{\infty} \frac{\left[\frac{P}{F}\right]^m}{m!} \rho^n$$

which after some algebra yields:

$$P[0]^{-1} = \sum_{n=0}^{m-1} \left[\frac{P\rho}{F}\right]^n \frac{1}{n!} + \frac{\left[\frac{P\rho}{F}\right]^m}{m!} \frac{1}{1-\rho} \quad (4.15)$$

Equations (4.13), (4.14) and (4.15) provide explicit expressions of the steady state probability $P[n]$ of having n jobs in the system. Let \bar{n} denote the steady state average number of jobs in the system. Using equations (4.13), (4.14), and (4.15), we have:

$$\begin{aligned} \bar{n} &= \sum_{n=0}^{\infty} nP[n] \\ &= \sum_{n=0}^m nP[0] \frac{\left[\frac{P\rho}{F}\right]^n}{n!} + \sum_{n=m+1}^{\infty} nP[0] \frac{\left[\frac{P}{F}\right]^m}{m!} \rho^n \end{aligned}$$

which after some algebra, yields:

$$\bar{n} = \frac{P\rho}{F} + P[0] \frac{\left[\frac{P\rho}{F}\right]^m}{m!} \frac{\rho}{(1-\rho)^2} + P[0] \frac{\left[\frac{P\rho}{F}\right]^m}{m!} \left(m - \frac{P}{F}\right) \frac{\rho}{1-\rho} \quad (4.16)$$

Equation (4.16) along with the expression of the probability $P[0]$ given by equation (4.15), explicitly defines the steady state average number of jobs in the P-DPS-WF system where jobs are represented by any given execution graph with stages having the same concurrency degree F . The job average response time, $T(P)$, in such a P-DPS-WF system can then be obtained by using Little's result; namely that $T(P) = \frac{\bar{n}}{\lambda}$, where λ is the aggregate rate of the job Poisson arrival process. Using equation (4.8) and equation (4.16), we thus have:

$$T(P) = \frac{\sum_{i=1}^N \frac{1}{\mu_i}}{F} + P[0] \frac{\left[\frac{P\rho}{F}\right]^m}{m!} \frac{\sum_{i=1}^N \frac{1}{\mu_i}}{P(1-\rho)^2} + P[0] \frac{\left[\frac{P\rho}{F}\right]^m}{m!} \left(m - \frac{P}{F}\right) \frac{\sum_{i=1}^N \frac{1}{\mu_i}}{P(1-\rho)} \quad (4.17)$$

Example

Let us take the case of $F=1$; this is the usual Processor Sharing scheduling strategy. We have $m=P$, and equation (4.16) becomes:

$$\bar{n} = P\rho + P[0] \frac{\left[\frac{P\rho}{F}\right]^P}{P!} \frac{\rho}{(1-\rho)^2}$$

where, by using equation (4.15), $P[0]$ is given by:

$$P[0]^{-1} = \sum_{n=0}^{P-1} \frac{(P\rho)^n}{n!} + \frac{(P\rho)^P}{P!} \frac{1}{1-\rho}$$

This is the known solution of the average number of jobs in the M/M/P FCFS queueing system and the M/G/P Processor Sharing queueing system.

Limiting Behavior of $T(P)$

It is of interest to determine the limiting values $T_0(F, P)$ and $T_1(F, P)$ of the job average response time as the utilization factor ρ approaches respectively zero and one, for any real positive value of the concurrency degree F . As ρ approaches zero, the job average response time approaches the total job average processing time through an empty system. Since the average

processing time of execution stage i , $i=1, \dots, N$ is $\frac{\max\{P, F\}}{FP} \frac{1}{\mu_i}$, we obtain :

$$T_0(F, P) = \frac{\max\{P, F\}}{PF} \sum_{i=1}^N \frac{1}{\mu_i} \quad (4.18)$$

On the other hand, the limiting value $T_1(F, P)$ is specified in the following Theorem.

Theorem 4.6

The limiting value $T_1(F, P)$ of the job average response time through a P-DPS-WF system with a constant concurrency degree F , the same for all the stages, is independent of F and is equal to the average response time in an M/M/1 queueing system having the same utilization factor ρ ; that is:

$$T_1(F, P) = \frac{\rho}{1-\rho} \frac{1}{\lambda}$$

Proof

Let us prove that as ρ approaches one, the average number of jobs in the system as given by equation (4.16) approaches the average number of jobs in an M/M/1 queueing system having the same utilization factor. From equation (4.15), we have:

$$\lim_{\rho \rightarrow 1} P[0] = \frac{m!}{\left[\frac{P\rho}{F}\right]^m (1-\rho)}$$

Replacing $P[0]$ by this limiting behavior in the expression for \bar{n} as given by equation (4.16) yields:

$$\frac{\rho}{\frac{1-\rho}{\bar{n}}} = 1$$

which completes the proof. ■

Let us now return to equation (4.9) providing the departure rate of class i jobs, $i=1, \dots, L$ when the system state is $(n_1, \dots, n_i, \dots, n_L)$. We can rewrite equation (4.9) as follows:

$$d_i = \frac{n_i \frac{P}{F}}{\max \left\{ \frac{P}{F}, \sum_{j=1}^L n_j \right\}} F \mu_i \quad i=1, \dots, N \quad (4.19)$$

We recognize this as the rate of departure of class i jobs from a processor sharing system comprising $\frac{P}{F}$ processors, and where the average service demand of class i jobs is $\frac{1}{F\mu_i}$. This amounts then to a decrease in both the number of processors and the average service demand of class i jobs, $i=1, \dots, N$ when $F > 1$, and to an increase in both the number of processors and the average service time demand of class i jobs, $i=1, \dots, N$ when $F < 1$.

Figure 4.6 depicts the job average response time given by equation (4.17), as a function of the system total utilization factor ρ , and for various values of the constant concurrency degree F . Assume, for example, that we start with $P=20$ processors, $N=1$ and $\mu = 0.05$. Therefore, for $F=1$, we have the usual processor sharing multiprocessor system whose job average response time is depicted by the curve that intersects the y-axis at the value 20. For $F=2$, we obtain the curve that intersects the y-axis at the value 10. This is the curve of the job average response time in a processor sharing system comprising 10 processors and where the job

average service demand is 0.1. For $F=20$, we obtain the curve that intersects the y-axis at the value 1. This is the curve of the job average response time in a processor sharing uniprocessor system, where the job average service demand is 1. Consequently, we may conclude that it is much better to have a system comprising less processors with shorter job service demands than a system comprising more processors but with larger job service demands.

The other average response time curves in Figure 4.6, are obtained using values of F smaller than unity. For $F = \frac{1}{2}$ for example, we obtain the curve that intersects the y-axis at the value 40. This is then the job average response time in a processor sharing system comprising 40 processors, and where the job average service demand is 2.

4.5 Average Response Time in the P-DPS-WF System

We now proceed to investigate the job average response time in a P-DPS-WF multiprocessor system. Jobs are represented by a given arbitrary process graph with N tasks and comprising only one starting task and one terminating task. We shall present and analyze a rather accurate approximation for the job average response time. This approximation is based on the EEC and the P-DPS-WF systems with constant concurrency degree, the same for all execution stages. Simulations are used to validate the approximation. First, we shall deduce the exact value of the average response time of jobs through an empty P-DPS-WF system (i.e., $\rho = 0$), and then we provide a generalization of Theorem 4.4.

4.5.1 Job Average Response Time Through an Empty P-DPS-WF System

The job average response time through an empty system, denoted hereafter by $T_0(P)$, is equal to the average processing time needed by a job when it is alone in the system. Since if \bar{X} and \bar{Y} are independent exponentially distributed random variables with respective means $\frac{1}{\mu_X}$ and $\frac{1}{\mu_Y}$, and if \bar{Z} is the random variable defined by $\bar{Z} = \min(\bar{X}, \bar{Y})$ then $P[\bar{Z} \leq z] = 1 - e^{-(\mu_X + \mu_Y)z}$, $z \geq 0$. It follows that the average of the random variable \bar{Z} is $\bar{Z} = \frac{1}{\mu_X + \mu_Y}$. Consequently, since whenever the first task among the set $\alpha(i)$ completes service, execution stage i terminates, then the average service demand brought by stage i to the system is given by $\frac{1}{\sum_{A \in \alpha(i)} \mu_A}$, $i=1, \dots, L$. On the other hand, the processing rate (i.e., capacity) at which this service demand is serviced depends on whether $f(i)$ is greater than P . If $f(i)$ is less

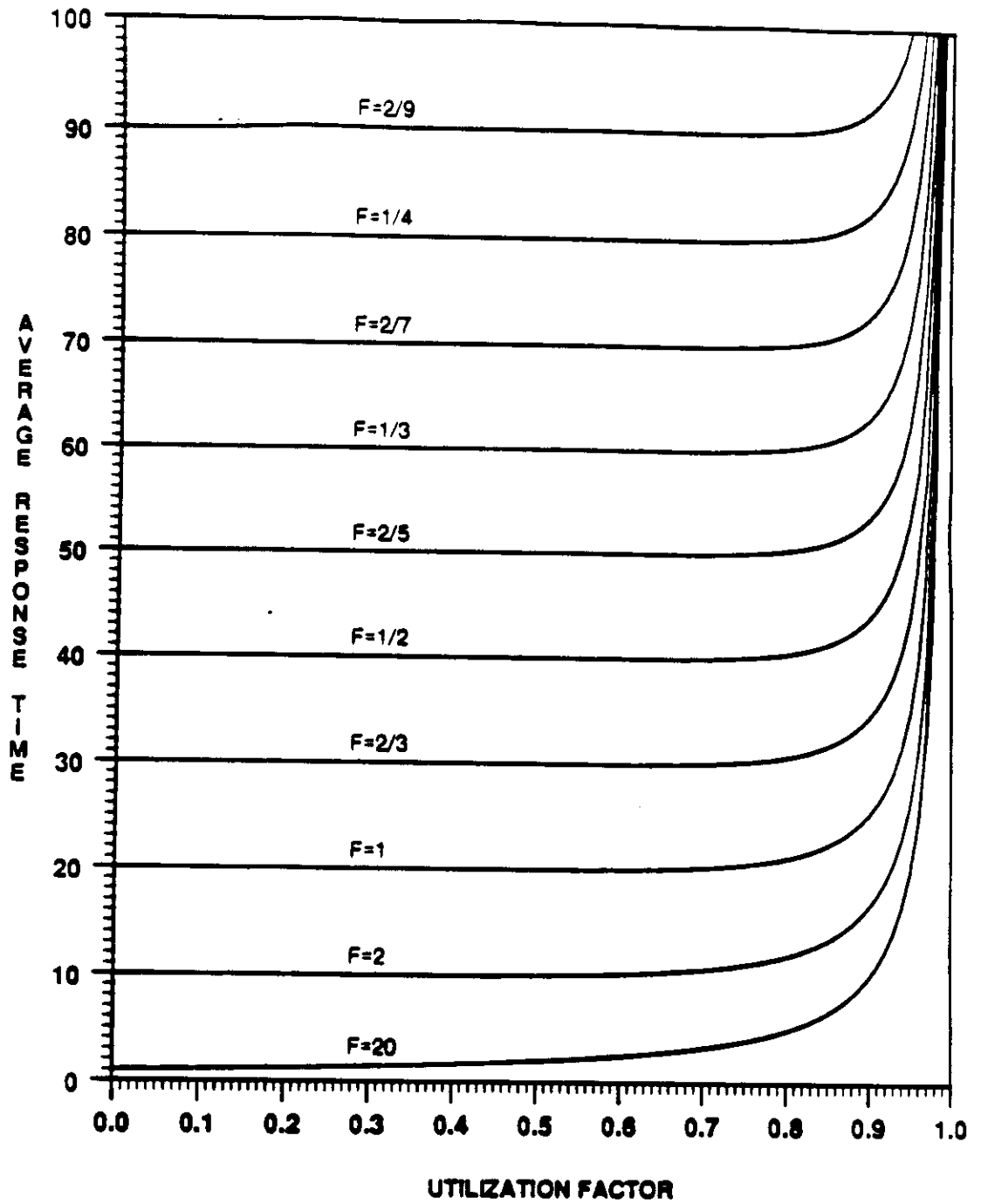


Figure 4.6: Average Response Time in a P-DPS-WF with Constant Concurrency Degree

or equal to P then each task in the set $\alpha(i)$ is processed at a rate of one second per second. If $f(i)$ is greater than P , however, each task in the set $\alpha(i)$ is processed at a rate of $\frac{P}{f(i)}$ seconds per second. If $S_0(i)$ denotes the average processing time of stage i in an empty system, we have:

$$S_0(i) = \frac{\max \left\{ P, f(i) \right\}}{P} \frac{1}{\sum_{A \in \alpha(i)} \mu_A} \quad i=1, \dots, L \quad (4.20)$$

On the other hand, since the probability that stage i is visited during the execution of a job is given by $\frac{\lambda_i}{\lambda}$, $i=1, \dots, L$. We thus obtain:

$$T_0(P) = \sum_{i=1}^L \frac{\lambda_i}{\lambda} S_0(i) \quad (4.21)$$

Equation (4.21) along with equation (4.20), provides the exact value of the total average processing time needed to complete a job through an empty P-DPS-WF system. If $P = \infty$, equation (4.21) reduces to equation (4.4). In the case of $\mu_A = \mu$ for all $A \in \Omega$, equation (4.20) reduces to:

$$S_0(i) = \frac{\max \left\{ P, f(i) \right\}}{P f(i) \mu} \quad i=1, \dots, L$$

and equation (4.21) becomes:

$$T_0(P) = \frac{1}{\lambda P \mu} \sum_{i=1}^L \frac{\lambda_i \max \left\{ P, f(i) \right\}}{f(i)}$$

4.5.2 Approximation of the Job Average Response Time

We now proceed to determine which RAEC, among all possible RAECs with N stages, provides the upper (respectively the lower) bound on the job average response time.

Theorem 4.7

Among all possible RAECs with N execution stages, the DFEC is the RAEC which provides the lowest job average response time. Moreover, there must exist a value ρ^* of the system utilization factor such that for all $\rho \in [0, \rho^*]$, the EEC is the RAEC which provides the largest job average response time, and for $\rho \in (\rho^*, 1)$, the BFEC is the RAEC which provides the largest job average response time.

Proof

The proof is based on a sample path representation of the job arrival and departure patterns. For a given sample of the job arrival process, the RAEC which results in the earliest job departures is the DFEC, for it implicitly and dynamically allocates the highest priority to the jobs nearest to completion. This proves the first statement. For $\rho=0$, the DFEC and the BFEC provide the same average response time which is, on the other hand, smaller than the one provided by the EEC. From Theorem 4.4, we know that the BFEC is the worst RAEC in the 1-DPS-WF system. Since for $\rho \rightarrow 1$, the P-DPS-WF system becomes congested and thus behaves as a 1-DPS-WF system with a total capacity of P seconds per second, it follows that for a sufficiently high value of the utilization factor, the BFEC results in the largest job average response time.

■

Although the BFEC and the DFEC do not actually correspond to any given process graph, they present, by means of the above Theorem, and for certain ranges of the utilization factor, respectively upper and lower bounds on the job average response time. Moreover, given the process graph description, Theorem 4.7 states that there exists a certain value ρ^* such that for all $\rho \in [0, \rho^*]$, the job average response time lies between the one given by the EEC and the one given by the DFEC; and for all $\rho \in [\rho^*, 1)$, the job average response time lies between the one given by the BFEC and the one given by the DFEC.

Recall that among all possible RAECs with N execution stages, there are only a few feasible ones. Moreover, the inherent property of the execution graph, namely that $f(1)=f(L-1)=f(L)=1$, assures that the job average response time for high values of the system utilization factor is much closer to the average response time given by the EEC than to the one obtained by either using the BFEC or the DFEC. The approximation of the average response time in a P-DPS-WF system must satisfy the following:

1. For $\rho = 0$, the approximation should provide the same value as the one given by equation (4.21). Moreover, for small values of the utilization factor, the approximation should result in a curve lying between the curves provided respectively by the EEC and the DFEC.
2. For higher values of the utilization factor (i.e., $\rho > \rho^*$), the approximation should result in a curve that is very close to the curve provided by the EEC.

The above two requirements can be satisfied by using the P-DPS-WF system job average response time given by equation (4.17) with the proper value for the concurrency degree F. Therefore, we have a parametric approximation that depends only on the parameter F. For any given process graph, we start by computing the value of the job average response time $T_0(P)$ through an empty P-DPS-WF using equation (4.21). Now, since the concurrency degree F is

less or equal to P , the number of processors in the system, by equating the just computed value of $T_0(P)$ to $T_0(F,P)$ given by equation (4.18), gives the value of F :

$$F = \frac{\sum_{i=1}^N \frac{1}{\mu_i}}{T_0(P)} = \frac{T_0(1,P)}{T_0(P)}$$

where $T_0(P)$ is given by equation (4.21). Notice that for $P=1$, the value of F is one and, hence our approximation results in the one presented earlier for the uniprocessor case.

We simulated the P-DPS-WF system using the process graph description depicted in Figure 4.1, and for the values $P=2,3,4,5$ and 16. The task average service time is equal to unity and the same for all the seven tasks. The *Method of Independent Replications* [Lave83] is used to estimate the extent of the simulation transient state, and the *Method of Batch means* [Lave83] is used to estimate the job average response time $T(P)$ in the steady state. Figure 4.7 depicts the job average response time as given by equation (4.17), along with the simulation results represented by the confidence intervals. The average response time in an empty system using equation (4.21) is 5.208333 for $P=2$ and 5.055555 for $P \geq 3$. From these values, we get $F=1.344$ for $P=2$ and $F=1.3846$ for $P \geq 3$. The confidence intervals are depicted on Figure 4.7 as vertical bars, are obtained from the simulation using the t-distribution, and are of 90% level.

Over all the permissible range of the utilization factor (i.e., $\rho \in [0,1)$), the approximation is well within the 90% confidence intervals. It is rather interesting to notice the narrowness of these confidence intervals even for very high values of ρ . The optimistic character on the other hand, is less noticeable than in the uniprocessor case.

4.5.3 Achievable Parallelism in a P-DPS-WF System

Significant reductions in the job average response time can be realized by executing a job, described by a given process graph, on a multiprocessor system. This effect is known as the *Speedup factor* (see below), which typically increases with the number of processors composing the multiprocessing system. Along with an increase in the speedup factor, comes a decrease in the efficiency of the processors. As more processors are used, the total amount of processors idle time increases also. While a large speedup factor may appear as a delight for the users, the efficiency of the processors is also very important. It is rather easy to get an efficiency of one, but this system is extremely slow (e.g., the job average response time is too large). This tradeoff is investigated below by the use of the *Power* function as defined in [Klei79, Gail83]. First, we shall investigate the achievable parallelism attained by a multiprocessor system with P processors, and then we shall return to investigate the above tradeoff.

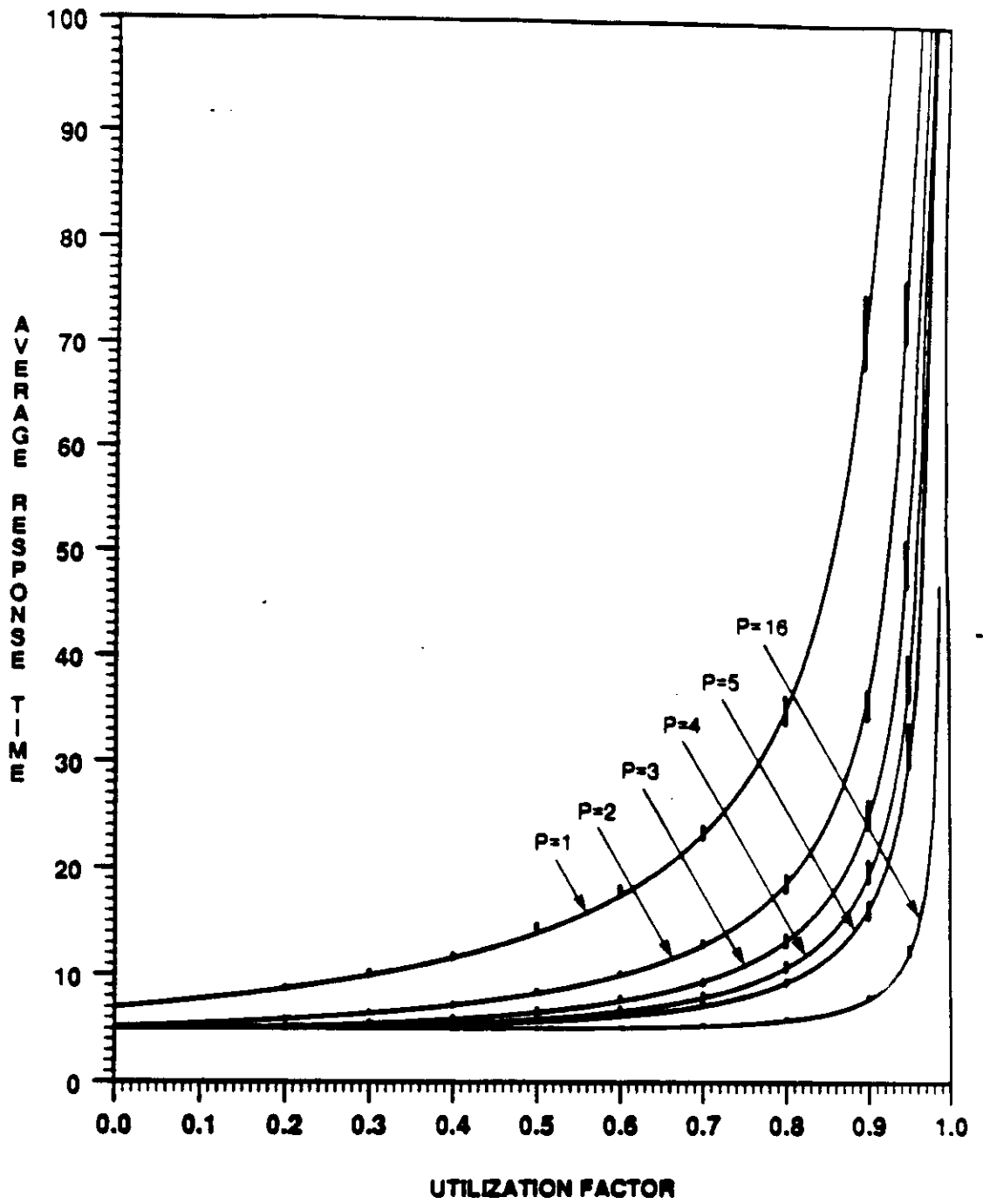


Figure 4.7: Average Response Time in a P-DPS-WF System

Customarily, the speedup factor, denoted by σ , of a parallel processing system is defined as the ratio of the job total processing time through an empty uniprocessor system to the job total processing time through a empty multiprocessor system [Kung84, Hwan84]. This is the same definition as the concurrency degree used previously. It is not difficult to realize that the speedup factor (in the special case of exponentially and identically distributed task service requirements) is bounded above by $\frac{N}{\ln N}$. Indeed for the process graph with N concurrent tasks, and for the case where the service time per task is exponentially distributed with mean $\frac{1}{\mu}$, the same for all tasks, Theorem 4.2 readily gives the lower bound on the average response time using an infinite number of processors. For large N (i.e., $N \gg 1$), $T_{\infty, LB} \cong \frac{\ln N + \Phi}{\mu}$ where Φ is the Euler's constant (i.e., $\Phi = 0.57721\dots$), and the job processing time in a uniprocessor system is $\frac{N}{\mu}$.

Although the speedup factor, as defined above, represents a very useful measure in determining the process graph structural parallelism (i.e., the inherent parallelism within the job process graph), it does not portray the achievable parallelism obtainable by using a multiprocessor system, for it does not incorporate any measure of the queueing effects. It is therefore more interesting for our purposes to redefine the speedup factor as a function of the number P of processors used, the system utilization factor ρ , and the scheduling strategy adopted. For a given scheduling strategy S , we therefore define the speedup factor to be:

$$\sigma(S, P, \rho) = \frac{T_1(S, \rho)}{T_P(S, \rho)} \quad (4.22)$$

where $T_1(S, \rho)$ and $T_P(S, \rho)$ represents the job average response times respectively through a uniprocessor system and a multiprocessor system, for the same scheduling strategy and for the same utilization factor ρ .

The question naturally arises as to which centralized system we are in fact comparing our multiprocessor system. It is not hard to see that indeed we are comparing the multiprocessor system with P processors, to the centralized system composed of P individual noninteracting uniprocessor subsystems (i.e., a multicomputer system as defined in Chapter 1), where the average arrival rate of jobs to each is $\frac{\lambda}{P}$. This indeed constitutes an interesting comparison, for it ascertain the gain achieved by interconnecting the P individual processors. Moreover, the super uniprocessor (i.e., a uniprocessor system having the same capacity as the P -processor system) system is always superior to the multiprocessing system.

For our discriminatory processor sharing scheduling strategy, equation (4.22) becomes:

$$\sigma(DPS-WF, P, \rho) = \frac{T_1(DPS-WF, \rho)}{T_P(DPS-WF, \rho)} \quad (4.23)$$

Using our approximation, we then obtain the speedup function for our process graph of Figure 4.1. Figure 4.8 depicts the P-DPS-WF system speedup as a function of the utilization factor ρ , and for various values of the number of processors P . At $\rho = 0$, we obtain the customary definition of the speedup factor, that is $\sigma(DPS-WF, 1, 0) = 1$, $\sigma(DPS-WF, 2, 0) = 1.344$ and $\sigma(DPS-WF, P, 0) = 1.3846$, for all $P \geq 3$. For $\rho = 1$, we observe that the speedup factor reaches the value P , as stated through Theorem 4.7. Therefore, we may conclude that the speedup factor for a P-DPS-WF system ranges between its lowest value obtained at $\rho = 0$ and which is equal to the value of the concurrency degree F used, to its highest value P obtained at $\rho = 1$. Moreover, all intermediary values of the speedup factor can be obtained by a proper choice of the value of the utilization factor. In Figure 4.8, we also depicted the speedup factor achieved by an infinite number of processors. This speedup factor forms then, for any value of ρ , the upper bound on the achievable parallelism.

Figure 4.9 shows the achievable parallelism (i.e., speedup factor) as a function of the number of processors for different values of ρ . We depict in heavy marks the two limiting cases; namely the case of $\rho = 0$ and the case of $\rho = 1$. When $\rho = 0$, the lowest curve shows that a substantial increase in the speedup factor is only obtained when we move from $P=1$ to $P=2$ and then to $P=3$. For higher values of P , the speedup factor is the same as that achieved by an infinite number of processors. For $\rho = 1$ on the other hand, we have the other heavy marked curve which states that the speedup factor is equal to the number P of processors used. As ρ increases from zero to one, we obtain the other curves. In the next section, we shall determine the optimum operating value of ρ and then deduce the achievable parallelism obtained at such a level.

The efficiency per processor in a multiprocessor system with P processors is the ratio $\frac{\sigma}{P}$. As mentioned earlier, it is also of interest to quantify the efficiency of each processor. Figure 4.10 depicts the efficiency per processor as a function of the utilization factor ρ , and for the process graph of Figure 4.1, and for $P=1, 2, 3, 4, 5, 16$. As expected, we notice that for small values of ρ , the efficiency of each processor is very low, and poorer as P gets larger. As the system utilization factor grows towards one, the efficiency per processor grows rather rapidly and at $\rho = 1$ reaches its maximum value of one.

Recall from our definition of the speedup factor, that we are indeed comparing a P-individual-noninteracting uniprocessors system architecture to a multiprocessing system architecture. Figures 4.8 and 4.9 show how much gain can be achieved by using our P-DPS-WF system architecture compared to the P individual noninteracting 1-DPS-WF systems architecture.

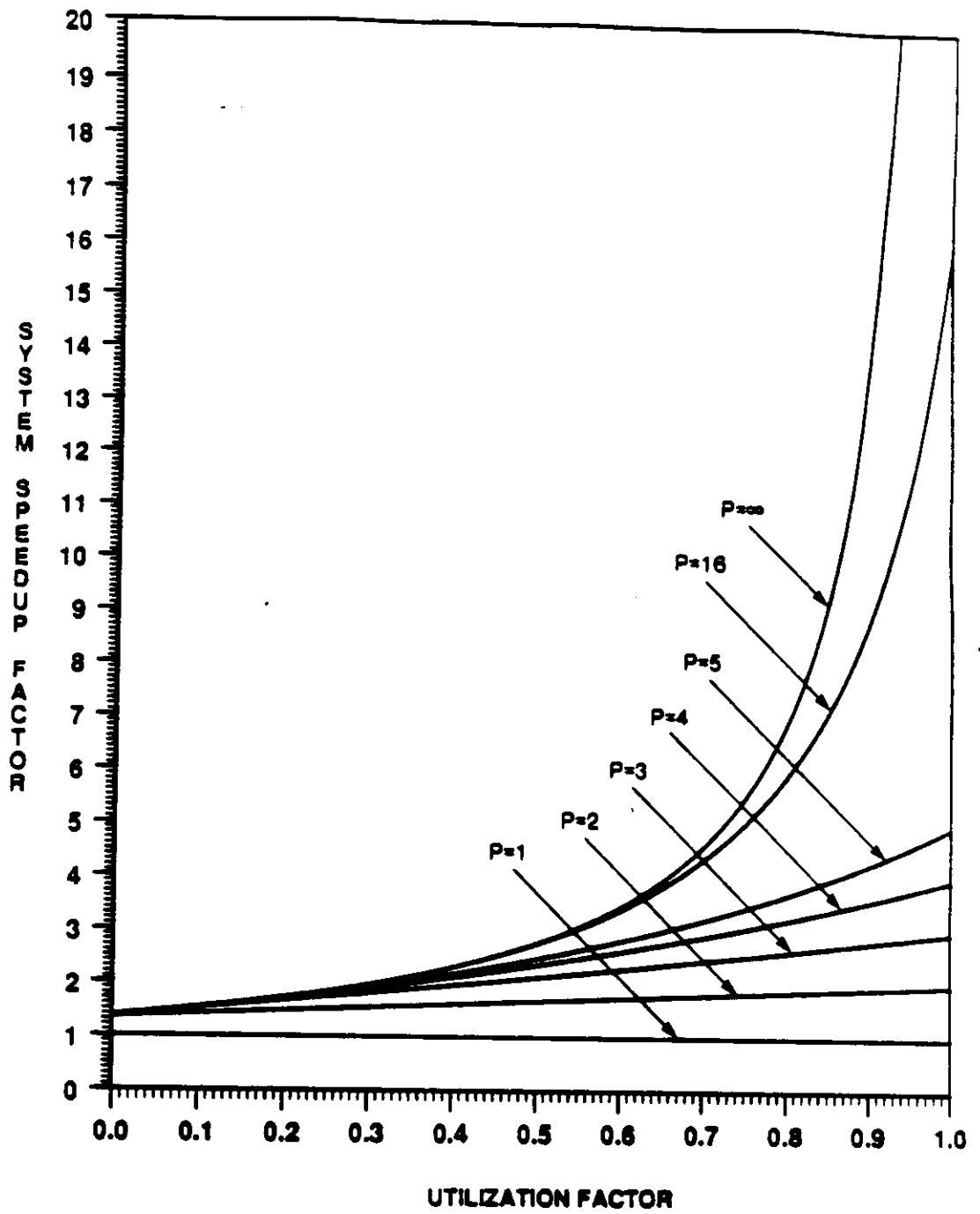


Figure 4.8: Achievable Parallelism in a P-DPS-WF System

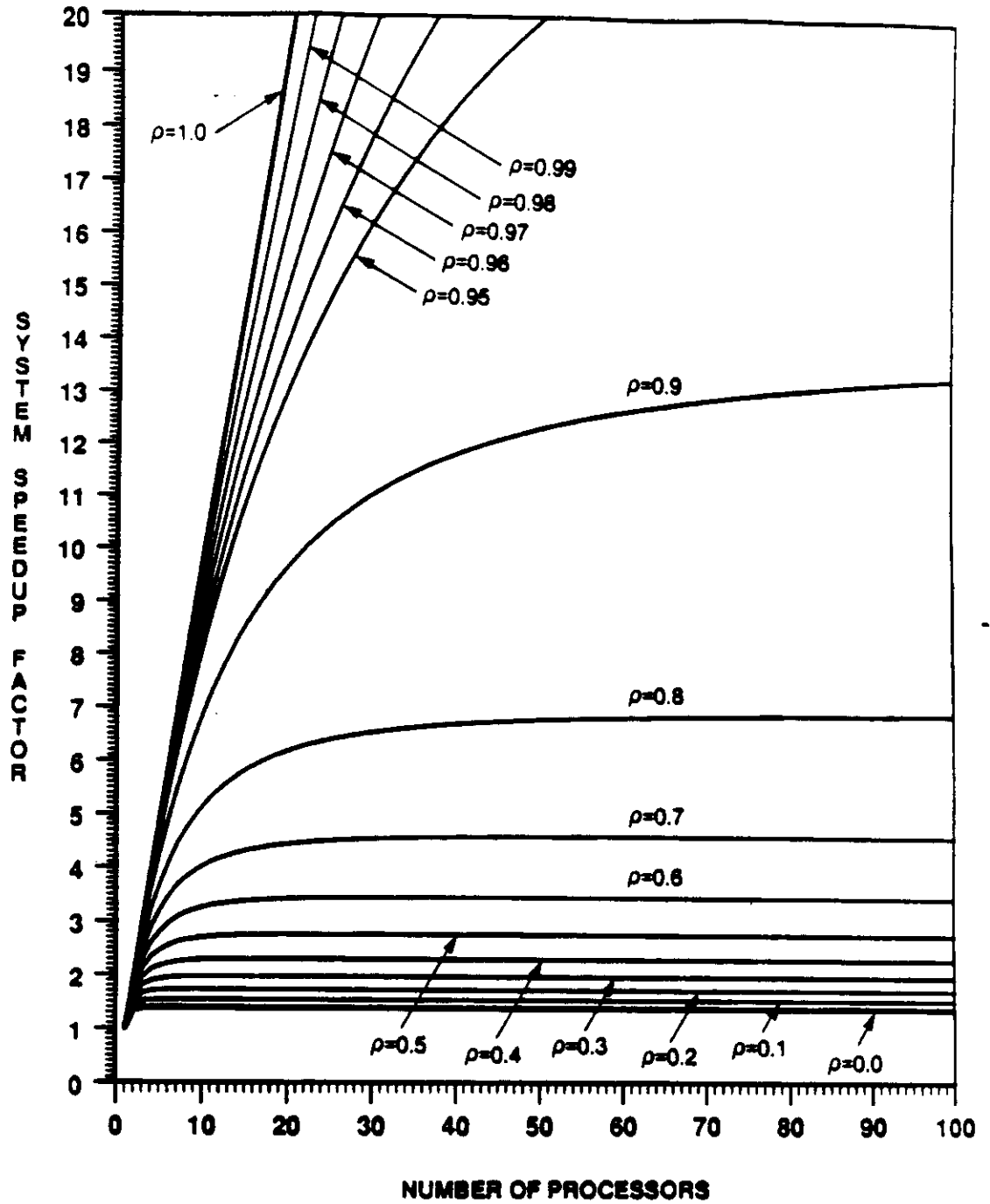


Figure 4.9: Achievable Parallelism versus the Number of Processors

We observe from these figures that the parallel processing architecture is superior (achieves a lower job average response time) to the centralized architecture over all permissible values of ρ . Indeed, this superiority approaches its maximum when the utilization factor approaches one. Nevertheless, at $\rho = 1$, the job average response time in the P-DPS-WF system, while P times less than that in the centralized architecture, is too large to be of any use. The question naturally arises as to which value of the utilization factor should we use for the P-DPS-WF system, and consequently how much parallelism is achieved at this utilization factor point.

Our interest is the tradeoff between throughput and response time involved in choosing a particular system operating point. As the input traffic offered to our multiprocessing system increases, the job average response time increases; see Figure 4.7. On the other hand, since we are operating within the system stability condition; namely $\rho < 1$, then the throughput of the system is equal to its input rate. Hence, the job average response time and the throughput of our multiprocessing system are both increasing functions of the input traffic. A performance measure incorporating throughput and delay into a single function is the notion of *Power* introduced in [Gies78]. It is simply defined as:

$$PW = \frac{\rho}{T_P(\rho)}$$

where ρ is the system utilization factor and $T_P(\rho)$ is the job average response time through the multiprocessing system with P processors. The two contrasting objectives of maximizing throughput and minimizing delay are combined into this single objective function. Other measures of power have appeared in the literature [Yosh77, Klei79].

Note from Figure 4.7 that for small values of ρ , a significant increase in the traffic input (i.e., in the throughput) can be obtained with only a slight increase in the job average response time, motivating us to increase the input traffic in this region. Conversely, for large values of ρ , a large decrease in the job average response time will occur if the input traffic rate is decreased only slightly, motivating us to decrease the input traffic rate in this region. From Figure 4.7, it is not difficult to see that an appropriate operating point for our multiprocessing system would be in the vicinity of the *Knee* of the average response time versus the utilization factor curve. The knee is defined as the point on the curve such that a line through the origin to this point is tangent to the curve. Kleinrock [Klei78a] demonstrated the usefulness of this knee criterion by observing that the value of ρ which maximizes power occurs exactly at the knee. Kleinrock further extended the above argument by noting that the job average response time curve need not be a convex function of ρ . In the cases where more than one tangent line can be found (i.e., more than one knee occurs), maximum power will occur for that tangent line which makes the smallest angle with the horizontal axis. Although it is known that the throughput-response time curves are convex, non convex curves may occur in the case of multiple access protocols which adapt to increasing load. Such a type of behavior was already observed for the URN scheme of Yemini and Kleinrock [Klei78b].

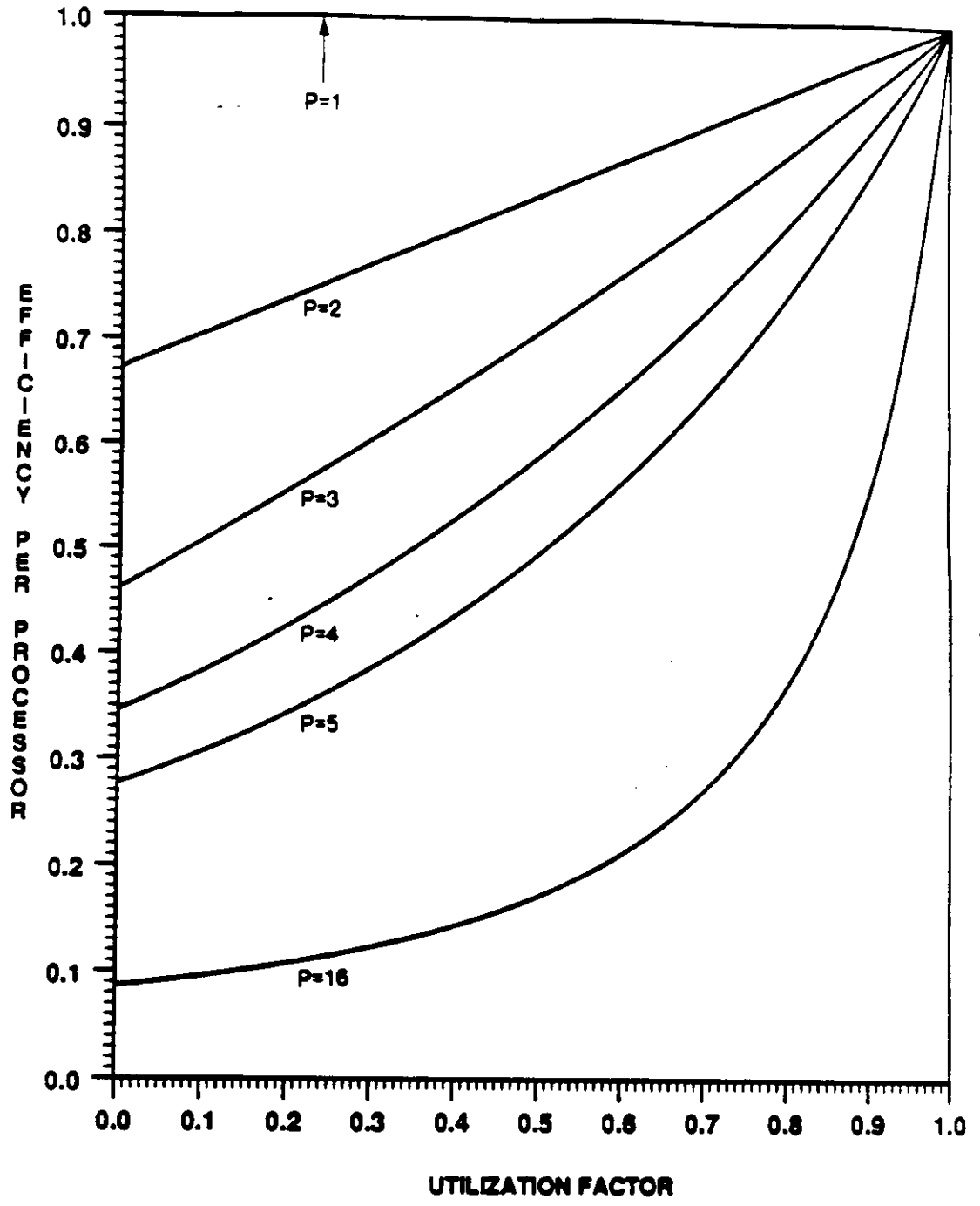


Figure 4.10: Efficiency per Processor in a P-DPS-WF System

Using the process graph description of Figure 4.1, and our approximation of the job average response time through a P-DPS-WF system, we obtain the power profile depicted in Figure 4.11. In this figure, we plot the power of the multiprocessor system as a function of the utilization factor ρ for various values of P . Since for any permissible value of ρ , the minimum average response time is obtained by using an infinite number of processors, it follows that this case provides the upper bound of the power measure. This is then represented on Figure 4.11 by the straight line defined by $PW = \frac{\rho}{T_{\infty}} = \frac{\rho}{5.055555}$. It is rather interesting to notice that the optimal operating point grows with P . Figure 4.12, depicts the relationship between the number of processors P , and the optimal corresponding operating point. For a given value of the utilization factor ρ , Figure 4.12 gives the number of processors to be used in the P-DPS-WF multiprocessor system which behaves optimally at such utilization level. For a given value of P , on the other hand, Figure 4.12 provides the optimal operating point ρ . Notice that in the range $\rho \in [0, 0.5)$ there exists no system that behaves optimally (i.e., at its maximum power), and that for high values of ρ (i.e., $\rho \geq 0.8$), a slight increase in ρ amounts to a large increase in P .

We now contrast our multiprocessing system to the centralized architecture system. Two alternatives may be considered. First, we compare the P-DPS-WF multiprocessor system to the centralized architecture system operating both at their respective optimal operating points. In this case, the achievable parallelism is represented by the lower curve in Figure 4.13. The short dashed line represents the asymptotic behavior of the achievable parallelism as P gets very large. Since the optimal operating point for the centralized architecture is at $\rho = 0.5$ which gives $T(1)=14.0$ for our process graph of Figure 4.1, and since the average response time through an infinite number of processors system is 5.055555, it follows that the asymptotic value is equal to 2.76923. It is also interesting to notice that the achievable parallelism, as a function of P , reaches rather quickly its ultimate asymptotic value. This is mainly due to the fact that after a certain value of P , the P-DPS-WF system behaves as an infinite number of processors system. Second, we compare the P-DPS-WF multiprocessing system to the centralized architecture system operating both at the same value of the utilization factor; the one defined by the operating point of the multiprocessing system. For this case, we obtain the upper curve of Figure 4.13. Although the achievable parallelism of the P-DPS-WF system is less than P , it is monotonically increasing with the number of processors used. This later comparison is more interesting and practical, for it compares the two architecture under the same load conditions.

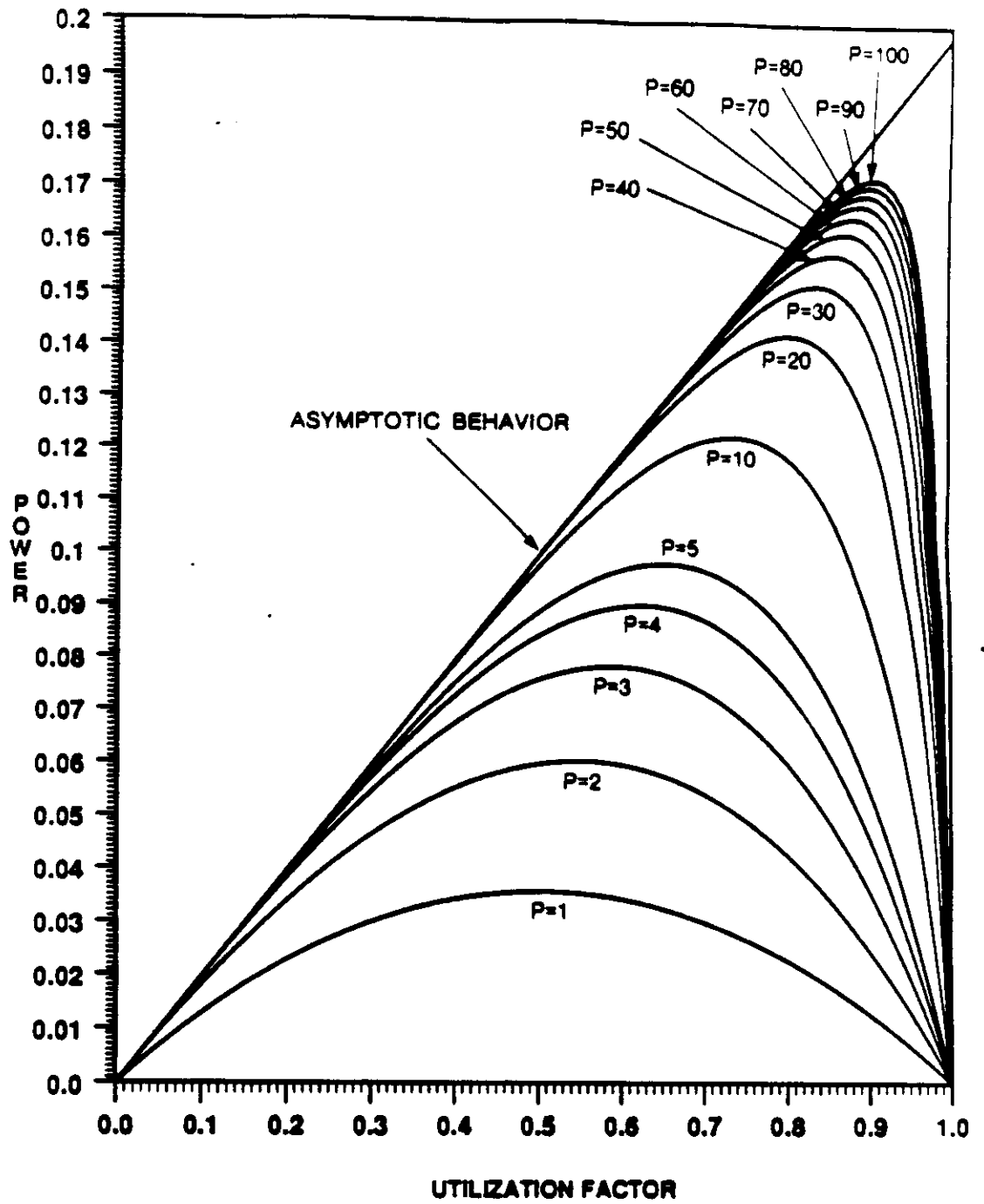


Figure 4.11: Power Function in a P-DPS-WF System

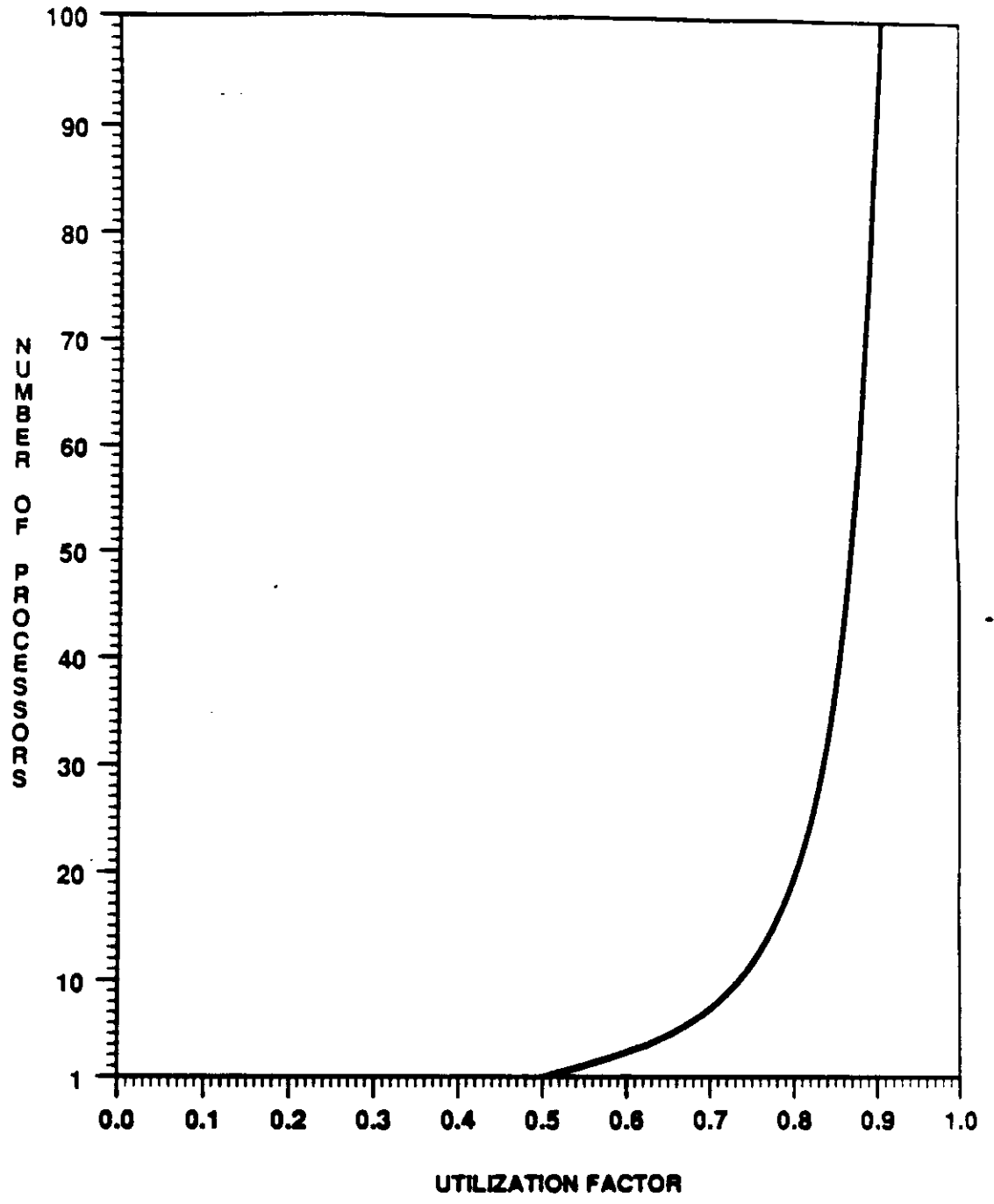


Figure 4.12: Maximum Power in a P-DPS-WF System

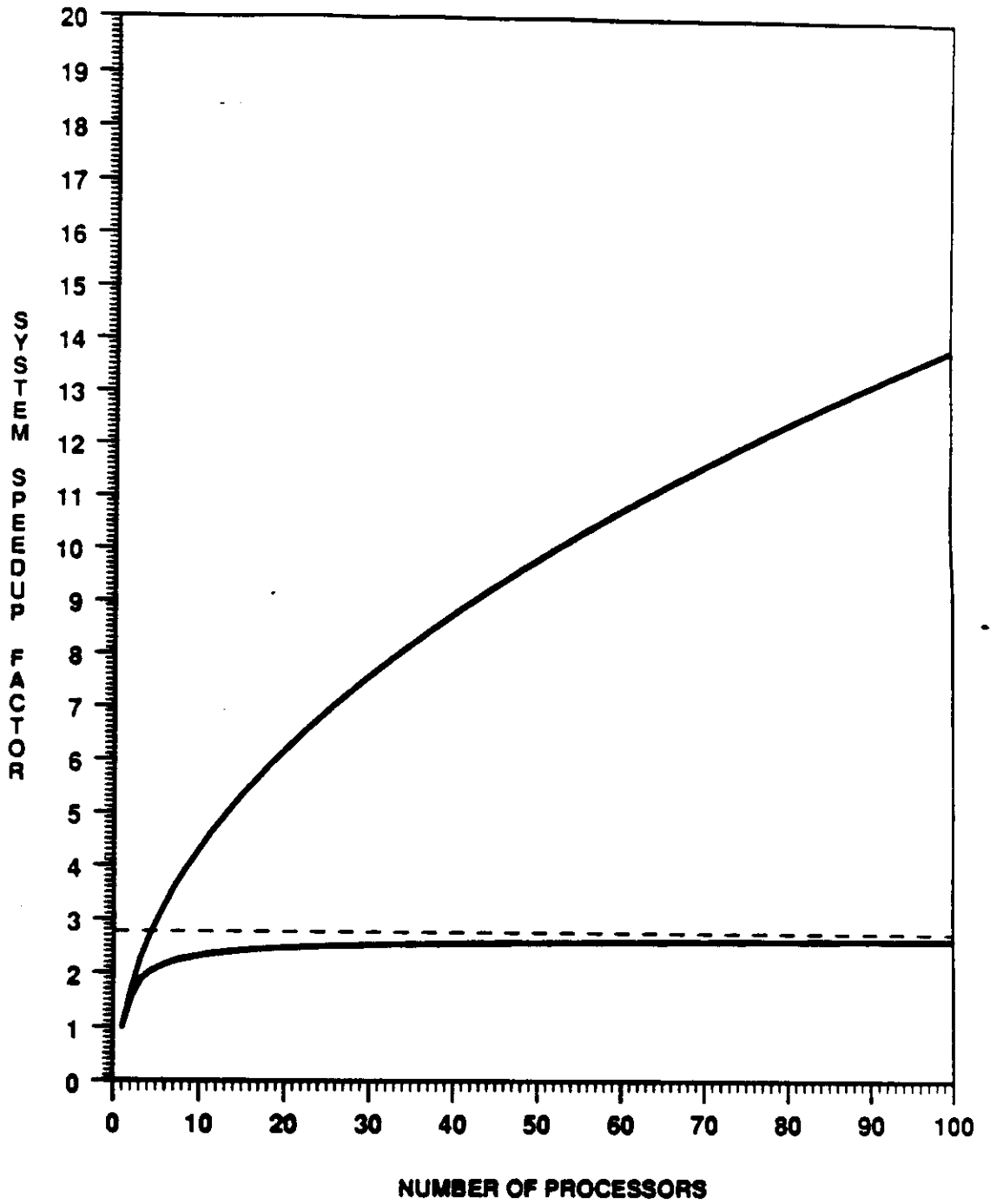


Figure 4.13: Achievable Parallelism at Optimal Operating Points

4.6 Conclusion

Significant reductions in the job average response time can be realized by executing a job, described by a given process graph, on a multiprocessor system. In this chapter, we introduced a new scheduling strategy termed the *Discriminatory Processor Sharing With job Feedback*, which is proved to form a complete family of scheduling strategies in the uniprocessor case. The study of the job average response time assumed a preliminary conversion of the process graph into an execution graph describing the execution stages of a job during its life in the system. By exploiting the execution graph properties, we formulated and proved upper and lower bounds on the job average response time. Accurate and yet very simple approximation for the job average response time are developed and used to quantify the achievable parallelism attained by multiprocessor systems. To portray the level of parallelism achieved, we defined the speed up factor as a function of both the scheduling strategy and the utilization factor of the system.

In the following chapters, we shall study the performance of models of parallel processing systems in which a job, upon arrival, spawns into two or more tasks, each of which must be attended by a separate dedicated processor. These models are further complicated by allowing the jobs to feedback into the system as many times as needed and according to any given arbitrary directed acyclic graph. These models of parallel processing systems are often encountered in many application areas such as concurrent and distributed processing systems, distributed data base systems, and manufacturing and production systems.

CHAPTER 5 PARALLEL PROCESSING WITH CERTAIN SYNCHRONIZATION CONSTRAINTS

In this chapter, we investigate the performance of models of parallel processing systems in which a job, upon arrival, spawns into two or more tasks, each one to be executed independently on a different processor. The job is considered completed upon the termination of the execution of all its spawned tasks. Such models arise in many application areas, such as flexible manufacturing and concurrent and distributed processing systems. In the context of production systems, a customer order can be viewed as a bulk of suborders, each one to be attended by a separate facility or device. In computer systems with parallel architectures, a bulk job can be viewed as a program composed of several concurrent subroutines, each one to be processed on a different processor. In distributed replicated data bases, write requests arriving to a given site must be executed by all the sites to maintain the data base integrity; such a write request is considered to be completed only when all the copies of the data base are updated.

5.1 Introduction and Previous Work

A parallel processing system with P processors in which a customer, upon arrival, subdivides into exactly P tasks, the i th of which must be executed by the i th processor, is known as a P -Dimensional-Fork-Join system. In the context of our previous definition of a process graph, a job is thus a $PG(P,1)$, namely a graph with P tasks and just one level (i.e., a bulk of size P), with the additional constraint that the i th task of a bulk must be executed by the i th processor. Our main purpose in this chapter is to devise ways to determine the response time of a bulk job in such a system.

Flatto and Hahn [Flat84], considered the two dimensional case, namely $P=2$. Their system consists of two heterogeneous processors, each having its own infinite queue. The execution time of a task is exponentially distributed, and the job (i.e., the bulk job) arrival process is Poisson. They obtained, under these Markovian assumptions and via uniformization techniques, a double transform of the system occupancy, and asymptotic formulas as either one of the queues becomes very long. Flatto [Flat85] derived limit laws for the distribution and the expectation of the occupancy of either one of the queues conditioned on the other one. In more

dimensions (i.e., $P \geq 3$) the problem still seems to be completely open.

A more general model is considered by Baccelli and Makowski [Bacc85a] and subsequently the same authors with Schwartz [Bacc85b]. They considered a P-Dimensional-Fork-Join system with heterogeneous servers, general arrival process and general service process. They obtained bounds on the response time of a bulk job in such a system. The key idea they used is to construct two queueing systems, that in the sense of some stochastic ordering, bound the original system and that are more tractable analytically than the original one. This approach is motivated by the fact that increased (respectively, decreased) variability in some of the stochastic components of a queueing system should result in a greater (respectively lower) variability of the waiting times in such a system [Stoy84, Whit84]. Using sample path representation for the quantities of interest (e.g., interarrival times, service times, waiting times), they obtained a lower bound that assumes P mutually independent D/GI/1 parallel queueing systems, and an upper bound that assumes P mutually independent GI/GI/1 parallel queueing systems. However, to establish their upper bound, they additionally assumed that the arrival process to the original system has a divisible distribution. In Section 5.3, we shall prove that such an upper bound still holds when we relax the divisibility assumption.

Nelson and Tantawi [Nels85] considered a Fork-Join system with homogeneous exponential parallel servers and a Poisson arrival process. Using results from [Flat84] they derived exact, asymptotic and approximate expressions for the mean response time of a bulk job for the case $P=2$. For the P-dimensional case they provided a fairly accurate approximation of the mean response time for small P (i.e., $P \leq 32$) based on a scaling technique. However, they also used simulation to determine the scaling function itself.

Other related results have appeared in the literature. Towsley, Chandy and Browne [Tows78] studied computer models in which the CPU and the I/O (or I/O and I/O) activities can be overlapped. Heidelberger and Trivedi [Heid82] and [Heid83] presented approximation methods to study a closed queueing network representing central server computer models where a job spawns into two or more tasks at some point during its execution.

5.2 Model Description

We consider a P-Dimensional-Fork-Join system (denoted hereafter by FJ-P-GI/GI/1) to be a queueing system operated by P parallel heterogeneous processors (i.e., servers) with synchronization constraints on the arrivals and departures. Each server has its own queue of infinite capacity and individually operates according to the First Come First Served (FCFS) discipline. In the next chapter, we generalize this service discipline to accommodate priorities. Upon arrival to the system, a customer (also called hereafter a bulk job), is immediately split into P tasks, each one is allocated to exactly one server (this is called the Fork operation). As soon as

all the P tasks constituting a bulk job have been serviced, the bulk job is immediately and instantaneously recomposed (this is called the Join operation), and leaves the system at once. This synchronization constraint is accomplished by parking the already serviced tasks in an auxiliary waiting area of infinite capacity (this is called hereafter the synchronization box), where they wait to be reunited to the unserviced tasks (i.e., siblings) of the same bulk job. Figure 5.1:(a) gives a pictorial representation of the P -Dimensional-Fork-Join queueing system, where service i , $i=1, \dots, P$, with its queue forms service center i . Figure 5.1:(b) depicts the actual process graph $PG(P,1)$ of a bulk job.

For such a system, we are interested in the determination of the bulk job average response time, denoted hereafter by $T(P)$, and defined as the average delay incurred between the Fork and the Join times. First, let us define the following quantities:

\tilde{X}_i : random variable representing the task service time at processor i , $i=1, \dots, P$, with mean \bar{X}_i and probability density function $b_{X_i}(x)$.

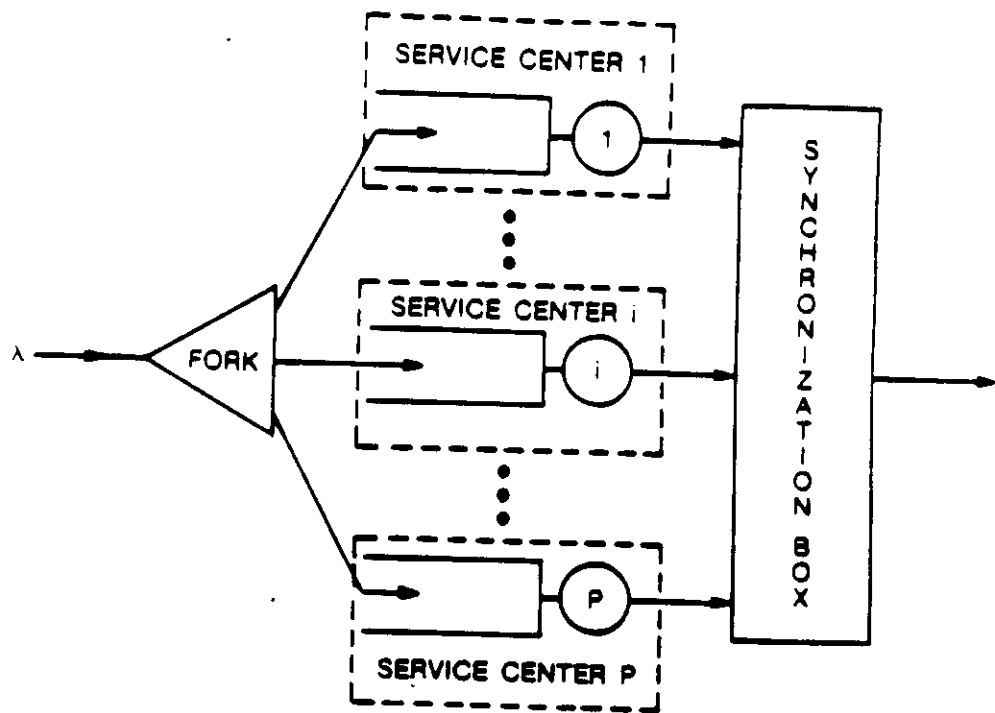
\tilde{t} : random variable representing the interarrival time between bulk jobs to the system, with mean \bar{t} and probability density function $a(t)$.

ρ_i : the utilization factor of processor i , $i=1, \dots, P$.

From the above definitions, we have $\rho_i = \frac{\bar{X}_i}{\bar{t}}$ $i=1, \dots, P$ and for the case where $\bar{X}_i = \bar{X}$

for all $i=1, \dots, P$, we have $\rho_i = \rho = \frac{\bar{X}}{\bar{t}}$. Throughout this chapter, we assume that the random variable \tilde{t} and the random variables \tilde{X}_i , $i=1, \dots, P$, are mutually independent. Therefore, the queueing system associated with any processor operates as a standard GI/GI/1 queueing system. However, the P GI/GI/1 queues constituting our parallel processing system are not independent since they all are driven by the same exact input (i.e., they all have identical inputs). It is this very lack of independence that makes the exact analytical determination of the performance measure $T(P)$ extremely hard. In light of this difficulty, it is natural and relevant to investigate ways of generating bounds and approximations. The purpose of this chapter is to provide easy and yet accurate approximations for the statistic $T(P)$.

The stability condition for our parallel processing system can be easily obtained from standard results on GI/GI/1 queues [Klei75]. Indeed, the system is stable if and only if each service center (see Figure 5.1) in isolation is stable. Therefore, the P -Dimensional-Fork-Join system is stable if and only if $\rho = \max_{1 \leq i \leq P} \{\rho_i\}$ is less than unity. Let us now establish an important result:



(a): The P-Dimensional-Fork-Join System



(b): The Actual Process Graph in a P-Dimensional-Fork-Join System

Figure 5.1: A Representation of the P-Dimensional-Fork-Join System

Proposition 5.1

The bulk jobs depart the system in the same order in which they entered the system.

Proof

Since each bulk job, upon arrival to the system, splits into P tasks and since at each server the tasks are served in an FCFS order, it follows that, at each server, the tasks depart in an FCFS order. Therefore, it is impossible for bulk job $(n+1)$ to depart from the system before the n th bulk job.

□

Let us define the following random variables:

i_s^{n+1} = random variable counting the number of the $(n+1)$ st bulk job siblings that are still in service (i.e., being served) just after the n th bulk job departure from the system.

i_w^{n+1} = random variable counting the number of the $(n+1)$ st bulk job siblings that are still in the server queues just after the n th bulk job departure from the system.

i_b^{n+1} = random variable counting the number of the $(n+1)$ st bulk job siblings that are in the synchronization box just after the n th bulk job departure.

From Proposition 5.1, we readily obtain :

$$1 \leq i_w^{n+1} + i_s^{n+1} \leq P \quad \text{and} \quad 0 \leq i_b^{n+1} \leq P-1 \quad n=1,2,\dots \quad (5.1)$$

Proposition 5.2

For the P -Dimensional-Fork-Join system, it must be that:

$$i_w^{n+1} = 0 \quad n=1,2,\dots$$

and consequently,

$$1 \leq i_s^{n+1} \leq P \quad \text{and} \quad 0 \leq i_b^{n+1} \leq P-1 \quad n=1,2,\dots$$

Proof

We use a proof by contradiction. Without loss of generality, Let the n th bulk job departure be the first instance where we have $i_n^{n+1} \neq 0$. Let ψ denote the set of servers at which the $(n+1)$ st bulk job siblings are waiting in their respective queues. Take a server that is in ψ . This server, having a nonempty queue, must be serving a task, say an m th bulk job sibling. Since this is the first occurrence, m should satisfy $m \geq n+1$. This is a contradiction to Proposition 5.1 and to the hypothesis that each server serves in an FCFS order.

□

Let us define the following quantities for the 2-Dimensional-Fork-Join system:

$P_{i,j}$: the probability that there are i tasks in service center 1 and j tasks in service center 2.

$P[Z,0] \triangleq \sum_{i=0}^{\infty} P_{i,0} Z^i$ the Z -transform of the $P_{i,j}$ conditioned on the fact that service center 2 is empty.

$P[0,Z] \triangleq \sum_{j=0}^{\infty} P_{j,0} Z^j$ the Z -transform of the $P_{i,j}$ conditioned on the fact that service center 1 is empty.

For the case of homogeneous servers, we readily have $P[Z,0]=P[0,Z]$ due to the symmetry between the two servers. With the additional assumption of exponential task service time with mean $\frac{1}{\mu}$, and of a Poisson bulk job arrival process with mean λ , Flatto and Hahn [Flat84] obtained the following closed form expression for this generating function:

$$P[Z,0] = \frac{(1-\rho)^{\frac{3}{2}}}{\sqrt{1-\rho Z}} \quad (5.2)$$

where $\rho = \frac{\lambda}{\mu}$. From this expression, we can readily obtain :

$$P_{0,0} = (1-\rho)^{\frac{3}{2}} \quad (5.3)$$

and,

$$\sum_{i=0}^{\infty} P_{i,0} = 1-\rho \quad (5.4)$$

Equation (5.4) represents the unconditional probability that the second server center is empty. Nelson and Tantawi [Nels85], by inverting equation (5.2), obtained an expression for the boundary probabilities, namely:

$$P_{i,0} = (1-\rho)^{\frac{3}{2}} a_i \rho^i \quad i=0,1,\dots \quad (5.5)$$

where a_i is given by :

$$a_i = \begin{cases} 1 & i=0 \\ \prod_{k=1}^i \left[1 - \frac{1}{2k} \right] & i=1,2,\dots \end{cases}$$

Let $q_k, k \geq 0$ be the probability that the number of tasks in the first service center exceeds the number of tasks in the second service center by exactly k . Thus, we have:

$$q_k = \sum_{i=0}^{\infty} P_{i+k,i} \quad k \geq 0$$

From the balance equations of the system, we get:

$$q_{k+1} = q_k - P_{k,0} \quad k \geq 0$$

which may be written as:

$$q_k = q_0 - \sum_{i=0}^{k-1} P_{i,0} \quad k \geq 1 \quad (5.6)$$

From the stability of the system, we must have $\lim_{k \rightarrow \infty} q_k = 0$, which results in:

$$q_0 = 1 - \rho \quad (5.7)$$

substituting $q_0 = 1 - \rho$ into equation (5.6), yields:

$$q_k = (1 - \rho) - \sum_{i=0}^{k-1} P_{i,0} \quad k \geq 0 \quad (5.8)$$

Using equations (5.5) and (5.8), they obtained the following exact expression for $T(2)$:

$$T(2) = \frac{1}{\mu - \lambda} + \frac{1 - \rho}{\lambda} \sum_{k=1}^{\infty} k \left[1 - \sqrt{1 - \rho} \sum_{i=0}^{k-1} a_i \rho^i \right] \quad (5.9)$$

A closed form asymptotic expression for $T(2)$ as ρ approaches unity is readily obtained, and is given by :

$$T(2) = \frac{1}{\mu - \lambda} + \frac{1}{4\lambda} \left[\frac{\rho - 1}{\ln \rho} \right]^{\frac{3}{2}} \left[1 - \frac{3}{2 \ln \rho} \right], \quad \rho \rightarrow 1 \quad (5.10)$$

In [Nels85] also, a first order (i.e., linear) approximation to $T(2)$ is obtained by examining the behavior of the ratio $\frac{T(1)}{T(2)}$ as ρ either approaches unity or zero, where $T(1)$ is the response time of a job in an M/M/1 queueing system and is given by $T(1) = \frac{1}{\mu - \lambda}$. As ρ approaches zero, the queueing effects may be neglected and hence:

$$\lim_{\rho \rightarrow 0} \frac{T(1)}{T(2)} = \frac{2}{3}$$

As ρ approaches unity, equation (5.10) gives:

$$\lim_{\rho \rightarrow 1} \frac{T(1)}{T(2)} = \frac{8}{11}$$

Their first order approximation to $\frac{T(1)}{T(2)}$ is then:

$$\frac{T(1)}{T(2)} = \frac{2}{3} + \left[\frac{8}{11} - \frac{2}{3} \right] \rho$$

which leads to :

$$T(2) = \frac{33T(1)}{2(11+\rho)} \quad (5.11)$$

5.3 Bounds on the Response Time of a Bulk Job

We now proceed to derive an upper bound for the response time of a bulk job in the FJ-P-GI/GI/1 parallel processing system. Let us define the following random variables:

\tilde{T}_j^i = random variable representing the response time for the i th bulk job through service center j , $j=1, \dots, P$

$\tilde{T}(P, i)$ = random variable representing the response time for the i th bulk job through the FJ-P-GI/GI/1 system.

\tilde{W}_j^i = random variable representing the waiting time of the i th task through service center j , $j=1, \dots, P$.

\tilde{X}_j^i = random variable representing the service time of the i th task at service center j , $j=1, \dots, P$.

\tilde{t}_i = random variable representing the interarrival time between the $(i-1)$ st and the i th bulk jobs.

We assume that all the above quantities have a limiting behavior, namely for $j=1, \dots, P$ the following limits exist,

$$\tilde{T}_j = \lim_{i \rightarrow \infty} \tilde{T}_j^i$$

$$\tilde{T}(P) = \lim_{i \rightarrow \infty} \tilde{T}(P, i)$$

$$\tilde{W}_j = \lim_{i \rightarrow \infty} \tilde{W}_j^i$$

$$\bar{X}_j = \lim_{i \rightarrow \infty} \bar{X}_j^i$$

$$\bar{i} = \lim_{i \rightarrow \infty} \bar{i}_i$$

and that the random variables $\bar{i}_i, \bar{X}_j, \bar{W}_j$, for $j=1, \dots, P$ are mutually independent random variables. A folk Theorem of queuing theory [Haje83, Humb82, Rogo66] states that determinism minimizes waiting times in many queuing systems. It is thus intuitive that the parallel system FJ-P-D/GI/1 has a lower average response time than the system FJ-P-GI/GI/1. Notice that the system FJ-P-D/GI/1 is a system of P independent parallel D/GI/1 queuing systems. Indeed, in [Bacc85a] the authors, by using a sample path representation of the system statistics, proved the above statement. Let $\bar{T}_j^{i,D}$ be a random variable representing the response time of the i th task through the j th D/GI/1 service center, and with a limiting distribution $\bar{T}_j^D = \lim_{i \rightarrow \infty} \bar{T}_j^{i,D}$. It is readily shown [Bacc85a] that:

$$T(P) \geq \int_0^{\infty} \left[1 - \prod_{j=1}^P P\{\bar{T}_j^D \leq x\} \right] dx \quad (5.12)$$

With the additional assumption on the arrival stream $\bar{i}_i, i=1,2,\dots$, namely that the random variables \bar{i}_i are divisible [Fell57], in the sense that they can be represented as a sum of P mutually independent renewal sequences with common probability distribution *, we can construct on the same sample space and family of events of the original system, a new queuing system composed of P parallel GI/GI/1 queues. The key feature of such a new system is that arrivals are no longer synchronized. Again by using a sample path representation of the system statistics, it is readily shown in [Bacc85a] that:

$$T(P) \leq \int_0^{\infty} \left[1 - \prod_{j=1}^P P\{\bar{T}_j^{GI} \leq x\} \right] dx \quad (5.13)$$

where \bar{T}_j^{GI} is the limiting behavior of the random variable representing the response time of the i th task through the j th independent GI/GI/1 system.

Now, let us relax the aforementioned additional assumption of divisibility of the random variables $\bar{i}_i, i=1,2,\dots$, and prove that inequality (5.13) still holds true. Our derivation of this upper bound uses properties of associated random variables [Barl75].

* A typical situation where the notion of divisibility holds is given by the class of P -stage Erlangian renewal processes.

Definition

Random variables $T=(T_1, T_2, \dots, T_n)$ are associated if $\text{cov}[f(T), g(T)] \geq 0$ for all pairs of increasing functions f and g .

Association of random variables satisfies the following multivariate properties, proofs of which can be found in [Bar75].

Properties

- (P1) Any subset of associated random variables are associated.
- (P2) Increasing functions of associated random variables are associated.
- (P3) Independent random variables are associated.
- (P4) If two sets of associated random variables are independent of one another, then their union is a set of associated random variables.
- (P5) If T_1, T_2, \dots, T_n are associated random variables, then :

$$P(T_1 \leq x_1, \dots, T_n \leq x_n) \geq \prod_{i=1}^n P(T_i \leq x_i)$$

We now proceed to show that the random variables representing the response time of the i th task, namely $\tilde{T}_j^i, j=1, \dots, P$ are indeed associated. We first establish by induction that the random variables $\tilde{W}_j^i, j=1, \dots, P$, representing the waiting times of the i th task through service center j , are associated.

Basis step:

For any initial state of the FJ-P-GI/GI/1 system, the random variables $\tilde{W}_j^1, j=1, \dots, P$ are associated by (P3).

Inductive step:

Assume that $\tilde{W}_j^i, j=1, \dots, P$ are associated for $i=1, 2, \dots, m$. From [Klei75], we have the following equality

$$\tilde{W}_j^{i+1} = \left[\tilde{W}_j^i + \tilde{U}_j^i \right]^+, \quad j=1, \dots, P$$

where $\tilde{U}_j^i = \tilde{X}_j^i - \tilde{r}_{i+1}, j=1, \dots, P$ and $[x]^+ = \max(0, x)$. The set of random variables $\tilde{U}_j^i, j=1, \dots, P$ are associated since they are independent random variables (P3). The two sets \tilde{W}_j^i and $\tilde{U}_j^i, j=1, \dots, P$ are independent of one another, hence these sets are associated (P4). Finally since

$\max(0, x)$ is an increasing function of x , it follows that $\bar{W}_j^i, j=1, \dots, P$ are also associated random variables (P2).

Now we can proceed in the same manner to prove that the random variables $\bar{T}_j^i, j=1, \dots, P$ are associated. In fact, we have $\bar{T}_j^i = \bar{W}_j^i + \bar{X}_j^i, j=1, \dots, P$. The random variable $\bar{X}_j^i, j=1, \dots, P$ are associated since they are independent (P3), and the union of the sets \bar{W}_j^i and $\bar{X}_j^i, j=1, \dots, P$ are associated by (P4).

Recall that $\bar{T}(P, i)$ represents the response time of the i th bulk job through the FJ-P-GI/GI/1 parallel processing system. Hence, we have:

$$\bar{T}(P, i) = \max_{1 \leq j \leq P} \left\{ \bar{T}_j^i \right\} \quad i=1, 2, \dots \quad (5.14)$$

Therefore,

$$P[\bar{T}(P, i) \leq x] = P\left[\max_{1 \leq j \leq P} \left\{ \bar{T}_j^i \right\} \leq x \right] = P[\bar{T}_1^i \leq x, \dots, \bar{T}_P^i \leq x] \quad \forall x \geq 0, \forall i \geq 1$$

and since the random variables $\bar{T}_j^i, j=1, \dots, P$ are associated, then using property (P5), we obtain:

$$P[\bar{T}(P, i)] \geq \prod_{j=1}^P P[\bar{T}_j^i \leq x] \quad \forall x \geq 0, \forall i \geq 1 \quad (5.15)$$

Equation (5.15) is valid for any $i=1, 2, \dots$. This equation is hence valid for the transient state of the system, and under the condition of stability, we obtain for the steady state:

$$P[\lim_{i \rightarrow \infty} \bar{T}(P, i) \leq x] \geq \prod_{j=1}^P P[\lim_{i \rightarrow \infty} \bar{T}_j^i \leq x] \quad \forall x \geq 0$$

which amounts to

$$P[\bar{T}(P) \leq x] \geq \prod_{j=1}^P P[\bar{T}_j \leq x] \quad \forall x \geq 0$$

or equivalently,

$$1 - P[\bar{T}(P) > x] \geq \prod_{j=1}^P P[\bar{T}_j \leq x] \quad \forall x \geq 0$$

which finally gives:

$$P[\bar{T}(P) > x] \leq 1 - \prod_{j=1}^P P[\bar{T}_j \leq x] \quad \forall x \geq 0 \quad (5.16)$$

Returning to the computation of the average response time of a bulk job, namely $T(P)$, we have:

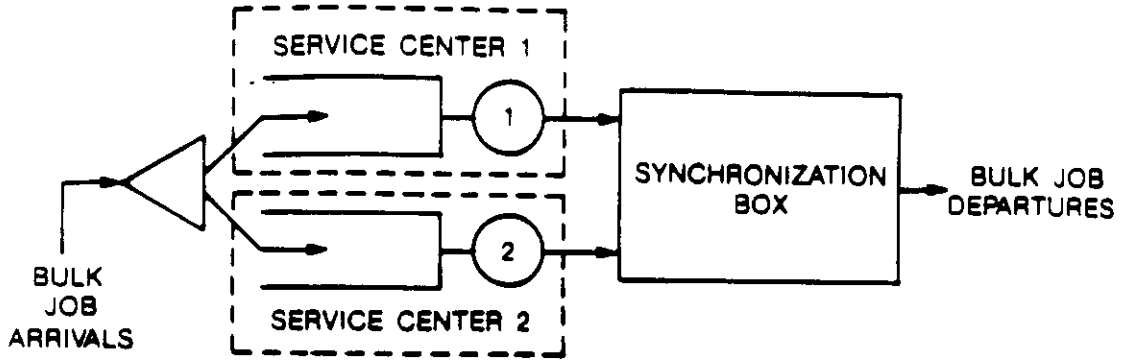
$$T(P) = \int_0^{\infty} P\{\bar{T}(P) > x\} dx \leq \int_0^{\infty} \left[1 - \prod_{j=1}^P P\{\bar{T}_j \leq x\} \right] dx$$

which is exactly inequality (5.13).

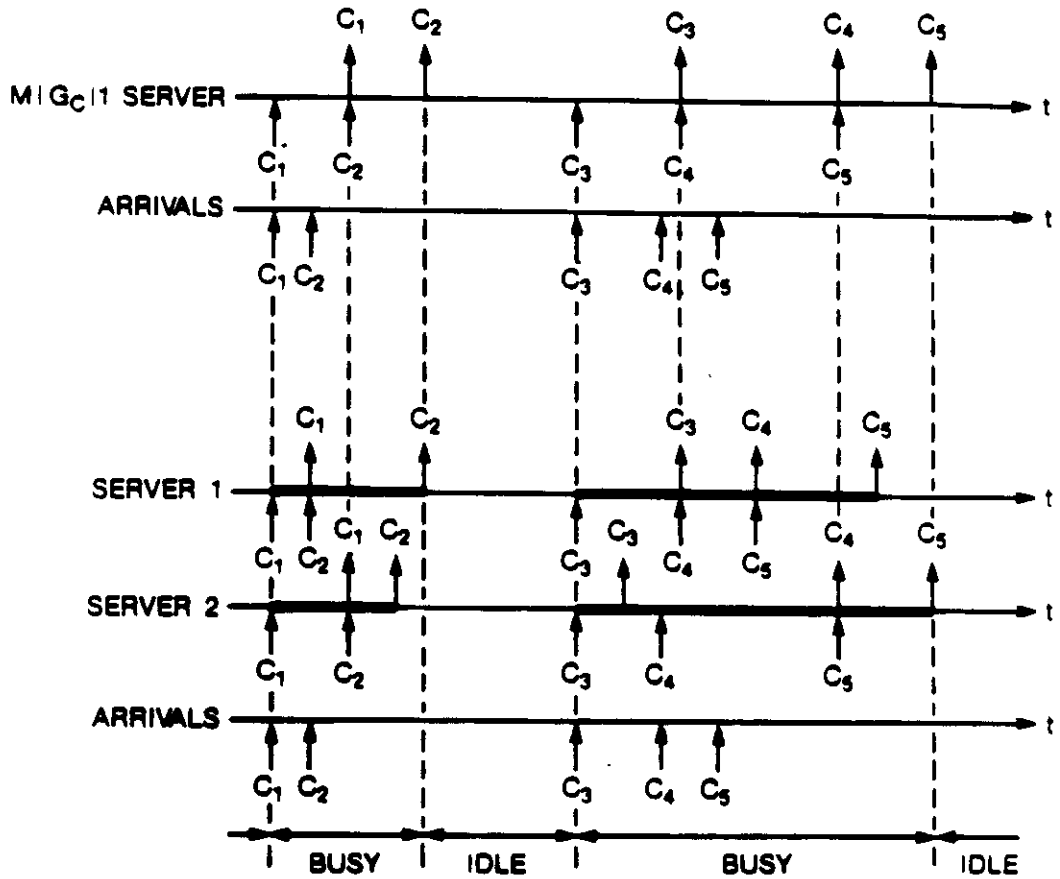
5.4 Approximate Analysis Using M/G/1 Theory

We now proceed to approximate our FJ-P-GI/GI/1 parallel processing system using an M/G/1 representation of the system. In the sequel, we shall restrict our bulk job arrival process to be Poisson, our servers to be homogeneous, and the task service time to be exponentially distributed. Hence, we shall be investigating the FJ-P-M/M/1 parallel processing system. From the previous section, we have seen that one technique to formulate bounds on the response time of a bulk is to compare the system to another parallel processing system with independent service centers (i.e., independent arrival streams to the different service centers). From Property (5.1), we know that the bulk jobs are serviced (i.e., depart the system) in exactly the same order in which they arrived to the system. Property (5.2) says that a bulk job departure from the system leaves all not-yet-serviced siblings of the next bulk job, in service. These observations motivate the analysis of the FJ-P-M/M/1 parallel processing system as an M/G/1 queueing system. Indeed, the bulk job arrival process is Poisson, and the service time of a bulk job is the time needed to complete the service of its remaining siblings. However, as will be discussed later, such bulk job service times are correlated. It is because of this lack of independence that our $M/G_c/1^*$ is not a pure M/G/1 queueing system. In this section, we investigate methods to deal, in an approximate way, with such correlation of subsequent bulk job service times.

Consider the case $P=2$, namely the parallel processing system FJ-2-M/M/1 represented in Figure 5.2:(a). Figure 5.2:(b) depicts some service periods of the FJ-2-M/M/1 parallel processing system, and, for the exact same samples of bulk jobs interarrival times and service times, the corresponding service periods of the equivalent $M/G_c/1$ system. This figure also shows the fashion in which busy periods alternate with idle periods for service center 1, service center 2, and the equivalent $M/G_c/1$ system. Observe that the FJ-2-M/M/1 system, and consequently the equivalent $M/G_c/1$ system, is busy as long as at least one of the two servers is busy. The busy periods for server center 1 and server center 2 are depicted by heavy marks along the corresponding server time axis. In the $M/G_c/1$ system, the departure time of a bulk job corresponds to the completion time of the bulk job latest sibling in the FJ-2-M/M/1 system. Define,



(a): A Representation of the FJ-2-M/M/1 System



(b): A Service Period of the FJ-2-M/M/1 System

Figure 5.2: The FJ-2-M/M/1 Parallel Processing System

\bar{n}_1 = random variable counting the number of tasks in service center 1, in steady state.
 \bar{n}_2 = random variable counting the number of tasks in service center 2, in steady state.
 \bar{n}_b = random variable counting the number of tasks in the synchronization box awaiting for their siblings completion, with average \bar{n}_b .

\bar{X} = random variable representing the service time of a task, with average $\bar{X} = \frac{1}{\mu}$ and distribution $b_X(x) = \mu e^{-\mu x}$ $x \geq 0$.

\bar{t} = random variable representing the interarrival time between successive bulk jobs, with average $\bar{t} = \frac{1}{\lambda}$ and distribution $b_t(t) = \lambda e^{-\lambda t}$ $t \geq 0$.

P_e = the probability, in steady state, that just after a bulk job departure the random variables \bar{n}_1 and \bar{n}_2 are equal.

\bar{n} = random variable counting the number of bulk jobs in the $M/G_c/1$ system in steady state.

\bar{X}_c = random variable representing the service time of a bulk job in the $M/G_c/1$ system, with average \bar{X}_c and distribution $b_{X_c}(x)$.

\bar{X}_c^i = the i th moment of the random variable \bar{X}_c . From the above definitions, we obtain:

$$\bar{n}_b = \left| \bar{n}_1 - \bar{n}_2 \right|$$

where $|x|$ represents the absolute value of x . Note that $P_e \neq P(\bar{n}_b=0)$ since P_e is the probability of having $\bar{n}_b=0$ at bulk job departure times and not for all times. Also, we have:

$$\bar{n} = \max \left\{ \bar{n}_1, \bar{n}_2 \right\}$$

We now proceed to characterize the bulk job service time distribution, $b_{X_c}(x)$, in the $M/G_c/1$ system. From Figure 5.2:(b), we observe that the bulk job service time, \bar{X}_c , is exponentially distributed whenever a bulk job departure leaves $\bar{n}_1 \neq \bar{n}_2$ in the system, and distributed as the maximum of two exponentials whenever a bulk job departure leaves $\bar{n}_1 = \bar{n}_2$, namely $\bar{n}_b=0$, in the system. Consequently, we may define :

$$\bar{X}_c = \begin{cases} \max(\bar{X}_1, \bar{X}_2) & \text{with probability } P_e \\ \bar{X} & \text{with probability } (1-P_e) \end{cases} \quad (5.17)$$

where the random variables \bar{X}_1 and \bar{X}_2 are independent and distributed identically to \bar{X} . Nevertheless, consecutive service times are correlated. Consider a service time immediately following a very long bulk job service; this service time is most probably distributed exponentially (as opposed to a maximum of two exponentials), since the other sibling of the bulk job had a

* The subscript c is to indicate that it is an $M/G/1$ system with correlated service times

good chance of already being served. In fact, all points corresponding to bulk job departures leaving the system with $\bar{n}_b=0$ are renewal points in the sense that future service times do not depend on any history previous to such points. Another way to observe this correlation among successive bulk job service times, is to realize that a bulk service time depends on the actual values of \bar{n}_1 and \bar{n}_2 at the beginning of service. On the other hand, the values of \bar{n}_1 and \bar{n}_2 depend on the length of the previous bulk job service time (provided that the system has not gone idle meanwhile). Equation (5.17) is an approximate representation of the actual bulk job service time. It assumes that the service times are independent and identically distributed, hence enabling us to use the pure M/GI/1 theory. Equation (5.17) stands for any particular distribution of the random variable \bar{X} . Here, \bar{X} is exponentially distributed and hence, $b_{X_c}(x)$ is a maximum of two exponentials with probability P_e and an exponential with probability $(1-P_e)$. First, we determine the distribution of the maximum of two independent identically distributed random variables. Let $\bar{Y}=\max(\bar{X}_1, \bar{X}_2)$, hence we have:

$$\begin{aligned} P(\bar{Y} \leq y) &= P(\max(\bar{X}_1, \bar{X}_2) \leq y) \\ &= P(\bar{X}_1 \leq y, \bar{X}_2 \leq y) \\ &= P(\bar{X}_1 \leq y)P(\bar{X}_2 \leq y) \end{aligned}$$

and since we have $P(\bar{X}_1 \leq y) = P(\bar{X}_2 \leq y) = 1 - e^{-\mu y}$ $y \geq 0$, we obtain:

$$P(\bar{Y} \leq y) = [1 - e^{-\mu y}]^2 \quad y \geq 0$$

Therefore, we finally obtain:

$$b_{X_c}(x) = [2\mu e^{-\mu x} - 2\mu e^{-2\mu x}]P_e + \mu e^{-\mu x} [1 - P_e] \quad x \geq 0$$

or,

$$b_{X_c}(x) = \mu [1 + P_e] e^{-\mu x} - 2\mu P_e e^{-2\mu x} \quad x \geq 0 \quad (5.18)$$

Its Laplace transform, denoted by $B_c^*(s)$, is :

$$B_c^*(s) = \frac{\mu [1 + P_e]}{\mu + s} - \frac{2\mu P_e}{2\mu + s} \quad (5.19)$$

From the above equation and using $\bar{X}_c = -\frac{dB_c^*(s)}{ds} \Big|_{s=0}$, and $\bar{X}_c^2 = \frac{d^2 B_c^*(s)}{ds^2} \Big|_{s=0}$, we get:

$$\bar{X}_c = \frac{2 + P_e}{2\mu} \quad (5.20)$$

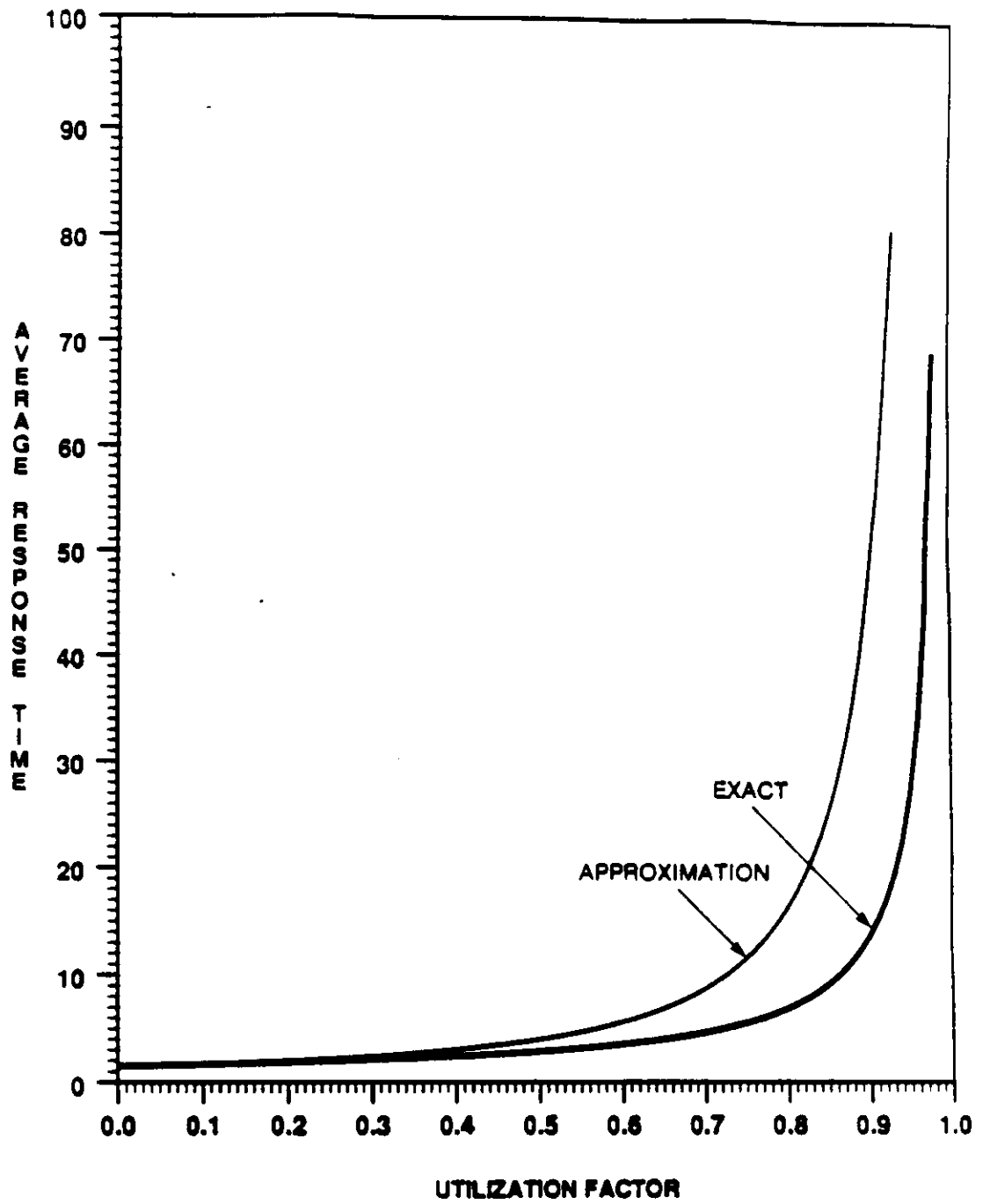


Figure 5.3: Approximation Using an $M/G_c/1$ Representation of the System

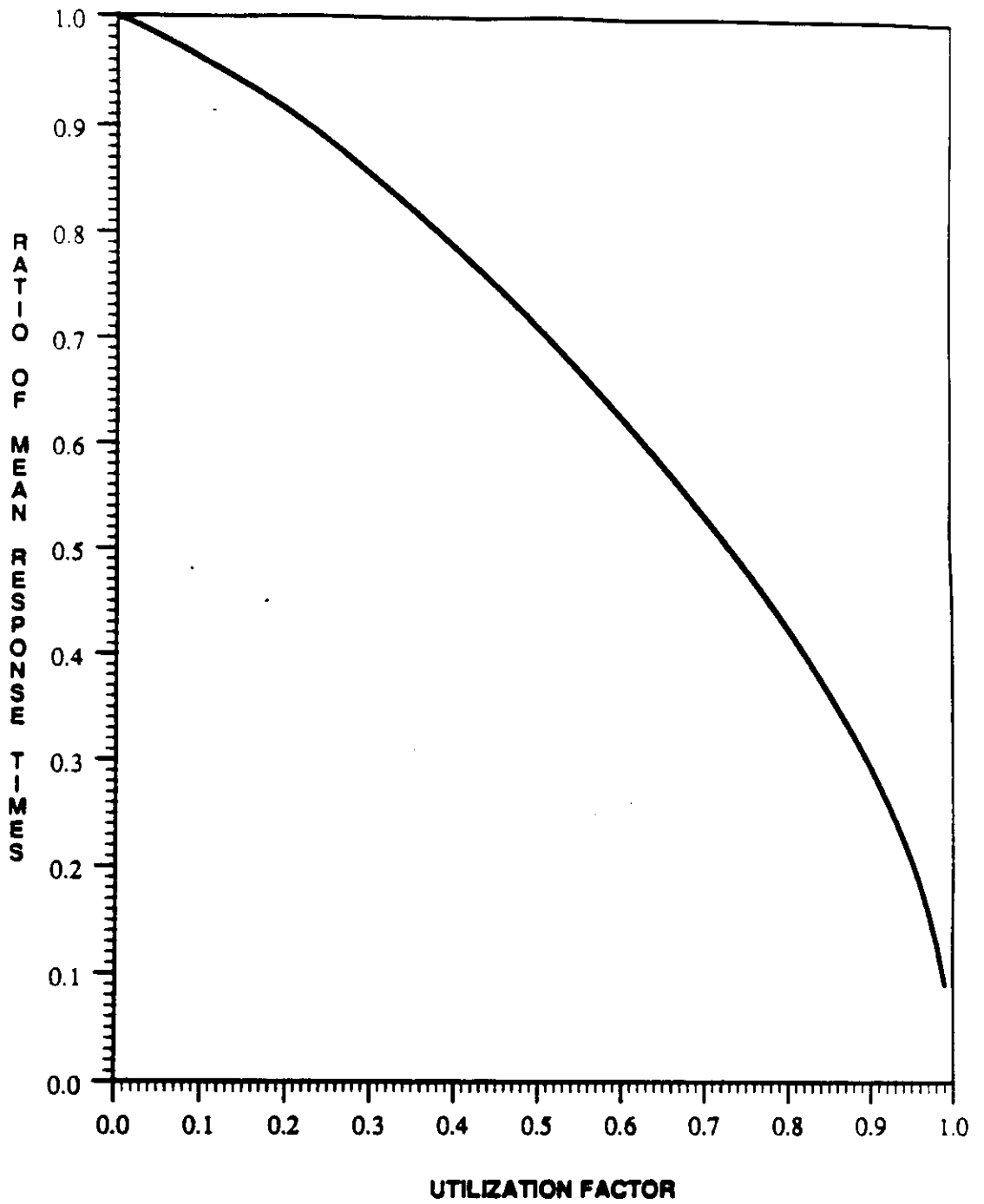


Figure 5.4: Discrepancy Between the Average Response Times Given by Formulae (5.9) and (5.28)

5.5 Analysis of the Modified $M/G_c/1$ System

From the last section we have seen that the main point in analyzing the $M/G_c/1$ system is the way to approximate the bulk job service times to render successive service times independent and identically distributed. In this section, we proceed in a similar manner, namely to represent the bulk job service time distribution with a geometric behavior. Nevertheless, we wish to decrease the variance of its distribution.

The interpretation of equation (5.27) is that, under heavy traffic load conditions, a bulk job departure almost always leaves just one sibling of the next bulk job in service (the other sibling must be at the synchronization box according to Proposition 5.2). This is the well known *Arc Sine Law* [Fell66], in which one considers a particle that moves along the positive direction of the x -axis, and at each unit of time it jumps upward with probability one half and downward with probability one half. This is similar to the FJ-P-M/M/1 system where a downward jump represents server 1 finishing its current task first, and an upward jump represents server 2 finishing its current task first, provided that we have a large number of tasks queued at each server (which is the case since $\rho \rightarrow 1$). The *Arc Sine Law* says that enormously many trials are usually required before the particle returns to a position on the x -axis. From equation (5.27), one first order (i.e., linear) approximation to the probability P_e is:

$$P_e = 1 - \tau \quad (5.29)$$

Using the above expression into equation (5.18), we obtain:

$$b_{X_c}(x) = (2 - \tau)\mu e^{-\mu x} - 2(1 - \tau)\mu e^{-2\mu x} \quad x \geq 0 \quad (5.30)$$

which yields:

$$\bar{X}_c = \frac{3 - \tau}{2\mu}, \quad \bar{X}_c^2 = \frac{7 - 3\tau}{2\mu^2}$$

and since $\tau = \lambda \bar{X}_c$, we get:

$$\tau = \frac{3\rho}{2 + \rho} \quad (5.31)$$

which in turn yields:

$$\bar{X}_c = \frac{3}{\mu(2 + \rho)} \quad (5.32)$$

$$\bar{X}_c^2 = \frac{7 - \rho}{\mu^2(2 + \rho)} \quad (5.33)$$

Now let us see if we have succeeded in decreasing the variance of the bulk job service time distribution by this approximation. The variance is given by equation (5.22) as a function of the probability P_e . Let σ_1^2 be the variance obtained by using the P_e given by equation (5.24), and

σ_2^2 be the variance obtain by using the P_n given by equation (5.29). After some algebra, we get:

$$\sigma_1^2 = \frac{\rho + 2(1-\rho)(1-\sqrt{1-\rho}) \left[\rho - 2(1-\rho)(1-\sqrt{1-\rho}) \right]}{\mu^2 \rho} \quad (5.34)$$

$$\sigma_2^2 = \frac{5+5\rho-\rho^2}{4(2+\rho)^2} \quad (5.35)$$

Figure 5.5 portrays the behavior of the ratio $\frac{\sigma_1^2}{\sigma_2^2}$ versus ρ . Notice that in the limiting cases (i.e., $\rho \rightarrow 0$, $\rho \rightarrow 1$), both variances are equal due to the fact that in both cases the probability P_n satisfies the limiting behavior of equation (5.27).

Now we proceed to the determination of the average response time of a bulk job. The average waiting time of a customer in the M/G/1 system is given by $W(2) = \frac{\lambda \bar{X}_c^2}{2(1-\tau)}$, hence by using equations (5.33) and (5.31) we have:

$$W(2) = \frac{3\rho}{2\mu(1-\rho)} + \frac{\rho}{4\mu} \quad (5.36)$$

Notice that $\frac{\rho}{\mu(1-\rho)}$ is just the average waiting time of a job in an M/M/1 queueing system having the same utilization factor ρ . Thus, we may write equation (5.36) as:

$$W(2) = \frac{3}{2} W_{M/M/1} + \frac{\rho}{4\mu}$$

by using $T(2) = W(2) + \bar{X}_c$, we get:

$$T(2) = \frac{3}{2} T_{M/M/1} + \frac{\rho(\rho-4)}{4\mu(2+\rho)} \quad (5.37)$$

where $T_{M/M/1}$ is the average response time of a job in an M/M/1 queueing system having the same utilization factor ρ ; namely $T_{M/M/1} = \frac{1}{\mu(1-\rho)}$. Finally, by using Little's result [Lit61], we get the following expression for the average number of bulk jobs in the M/G_c/1 system:

$$\bar{N} = \frac{3}{2} \bar{N}_{M/M/1} + \frac{\rho^2(\rho-4)}{4(2+\rho)}$$

where $\bar{N}_{M/M/1}$ is the average number of jobs in an M/M/1 queueing system having the same utilization factor ρ ; namely $\bar{N}_{M/M/1} = \frac{\rho}{1-\rho}$. Figure 5.6 depicts the average response times of a bulk job given respectively by equation (5.9) and equation (5.37), versus the utilization factor ρ . These average response times are very close to each other, and show the applicability and the accuracy of the modified M/G_c/1 system approximation. In fact, for $\rho \in [0, 0.7]$ both average response times are almost identical, and for higher values of the utilization factor ρ , the average

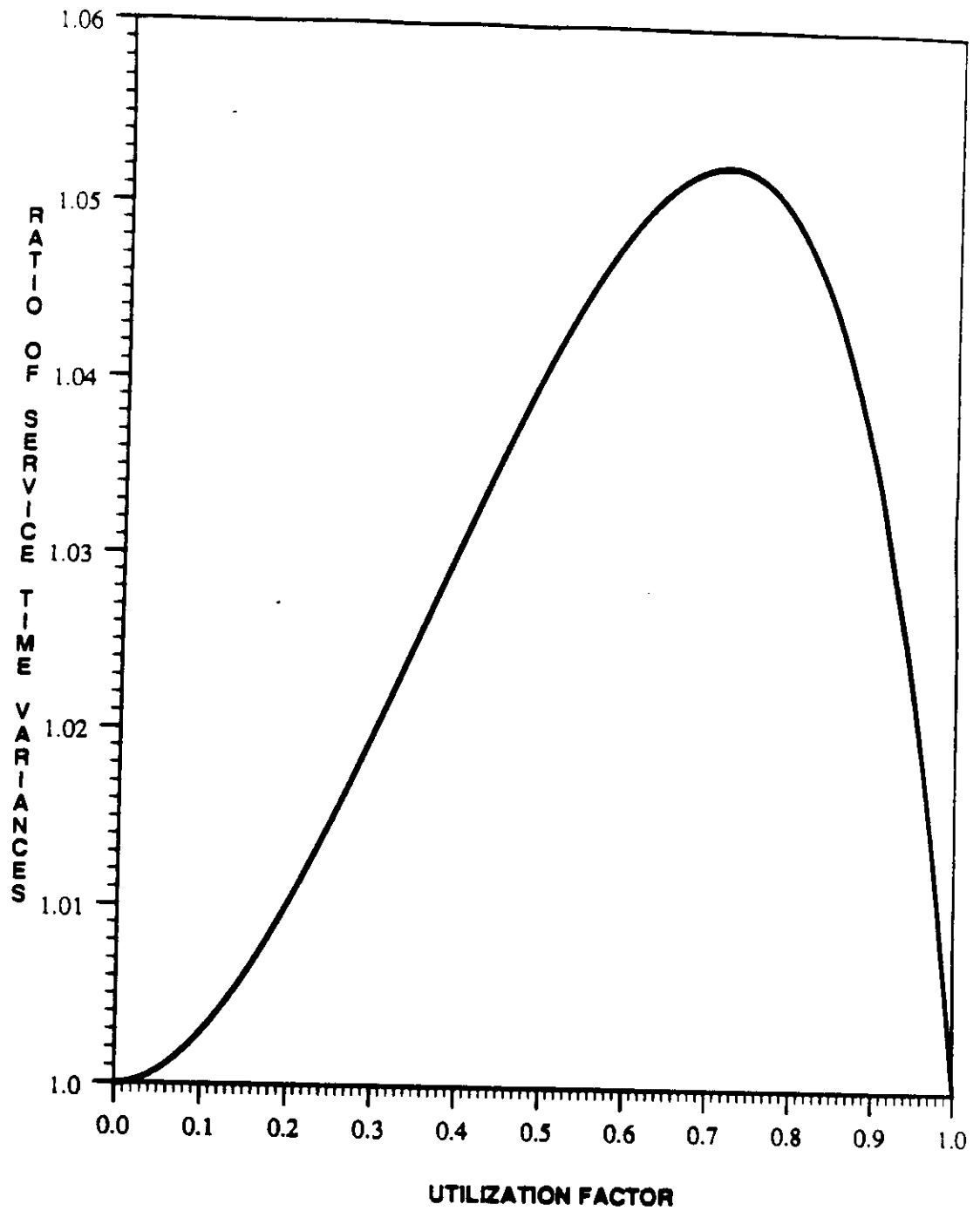


Figure 5.5: The Ratio of the Service Time Variances given by Equations (5.34) and (5.35)

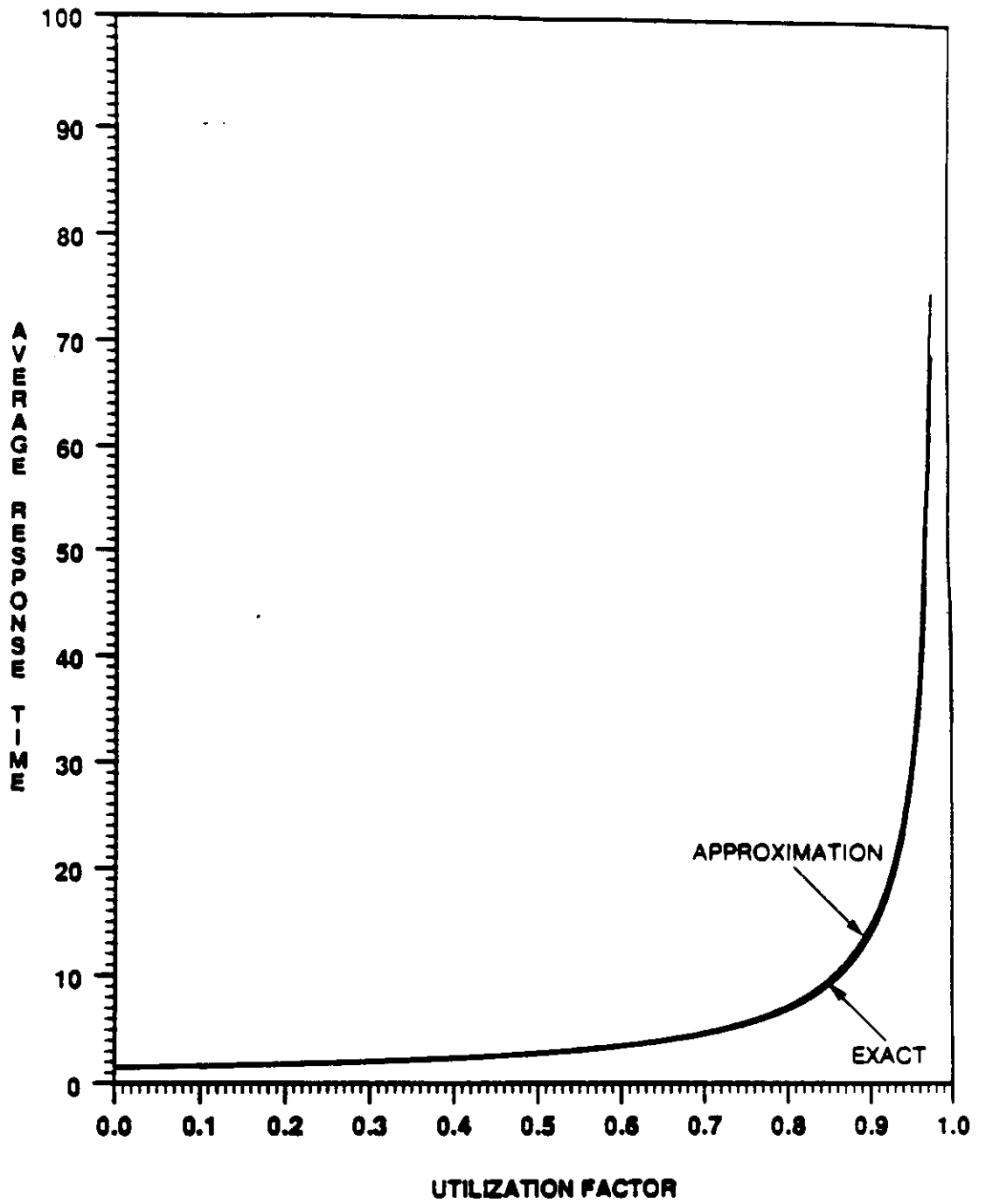


Figure 5.6: The Average Response Time Using the Modified $M/G_C/1$ Analysis

response time given by (5.37) is slightly higher. The average relative error between the two average response times is less than 5% for values of ρ in the range (0,0.95).

5.6 Analysis of the $M/G_c/1$ System Using a Load Adjustable Service Time Distribution

In the previous two sections, we analyzed the $M/G_c/1$ system by considering a bulk job service time distribution having a geometric behavior. In this section, we proceed to study the queueing system using a different representation of the bulk job service time distribution. From the previous analysis we know that, for very light traffic (i.e., $\rho \rightarrow 0$), the service time of a bulk job is distributed as the maximum of two exponentials. Under heavy traffic conditions (i.e., $\rho \rightarrow 1$), we found that the service time tends to be exponentially distributed. Therefore, we may consider the service time of a bulk job as a weighted sum of two independent random variables. Namely, let:

\tilde{X}_1 = random variable exponentially distributed with mean $\frac{1}{\mu}$ and distribution
 $b_{\tilde{X}_1}(x) = \mu e^{-\mu x} \quad x \geq 0.$

\tilde{X}_2 = random variable distributed as the minimum of two independent exponentially distributed random variables, namely $b_{\tilde{X}_2}(x) = 2\mu e^{-2\mu x} \quad x \geq 0.$

Now we represent the service time of bulk jobs as a weighted sum of the random variables \tilde{X}_1 and \tilde{X}_2 :

$$\tilde{X}_c = \alpha(\tau)\tilde{X}_1 + (1-\alpha(\tau))\tilde{X}_2$$

To determine the weighting function $\alpha(\tau)$, notice that the limiting behavior of this function must be:

$$\lim_{\tau \rightarrow 0} \alpha(\tau) = 1 \quad \lim_{\tau \rightarrow 1} \alpha(\tau) = 0$$

This leads us to consider a first order approximation to $\alpha(\tau)$, namely:

$$\alpha(\tau) = 1 - \tau$$

which gives:

$$\tilde{X}_c = \tilde{X}_1 + (1-\tau)\tilde{X}_2 \tag{5.38}$$

Using the definitions of \tilde{X}_1 and \tilde{X}_2 , we get:

$$\bar{X}_c = \frac{3-\tau}{2\mu} \quad , \quad \bar{X}_c^2 = \frac{4+(1-\tau)(3-\tau)}{2\mu^2}$$

where $\tau = \lambda \bar{X}_c$, that is:

$$\tau = \frac{3\rho}{2+\rho} \quad (5.39)$$

Notice that the above expression of the utilization factor is exactly the same as the one given by equation (5.31) of the previous section. This is quite natural, since for the average service time of a bulk job, \bar{X}_c , P_e and $\alpha(\tau)$ play the same role. Nevertheless, the second and higher moments of these service times have different expressions. Using equation (5.39) in the expression for \bar{X}_c and \bar{X}_c^2 , yields:

$$\bar{X}_c = \frac{3}{\mu(2+\rho)} \quad (5.40)$$

$$\bar{X}_c^2 = \frac{2}{\mu^2} + \frac{6(1-\rho)}{\mu^2(2+\rho)^2} \quad (5.41)$$

Now we proceed to find the average response time of a bulk job using this new approximation. First, let us determine the average waiting time of a bulk job. Using $W(2) = \frac{\lambda \bar{X}_c^2}{2(1-\tau)}$, along with equations (5.41) and (5.39), yields:

$$W(2) = \frac{\rho(2+\rho)}{2\mu(1-\rho)} + \frac{3\rho}{2\mu(2+\rho)} \quad (5.42)$$

Since $T(2) = W(2) + \bar{X}_c$, we obtain from equations (5.40) and (5.42):

$$T(2) = \frac{3}{2\mu} + \frac{\rho(2+\rho)}{2\mu(1-\rho)} \quad (5.43)$$

Finally, the average number of bulk jobs in the system is obtained by using Little's formula, and is given by:

$$N = \frac{3\rho}{2} + \frac{\rho}{1-\rho} \frac{\rho(2+\rho)}{2}$$

Figure 5.7 depicts the average response time of a bulk job given by equation (5.9) along with the one given by equation (5.43), versus the utilization factor ρ . This figure shows the high accuracy of the present approximation. For small to moderate values of the utilization factor ρ (i.e., $\rho \in [0, 0.7]$), both average response times are almost identical. For higher values of ρ , however, the average response time given by equation (5.43) is slightly higher. For the latter range of the utilization factor ρ , the average relative error is less than 4%.

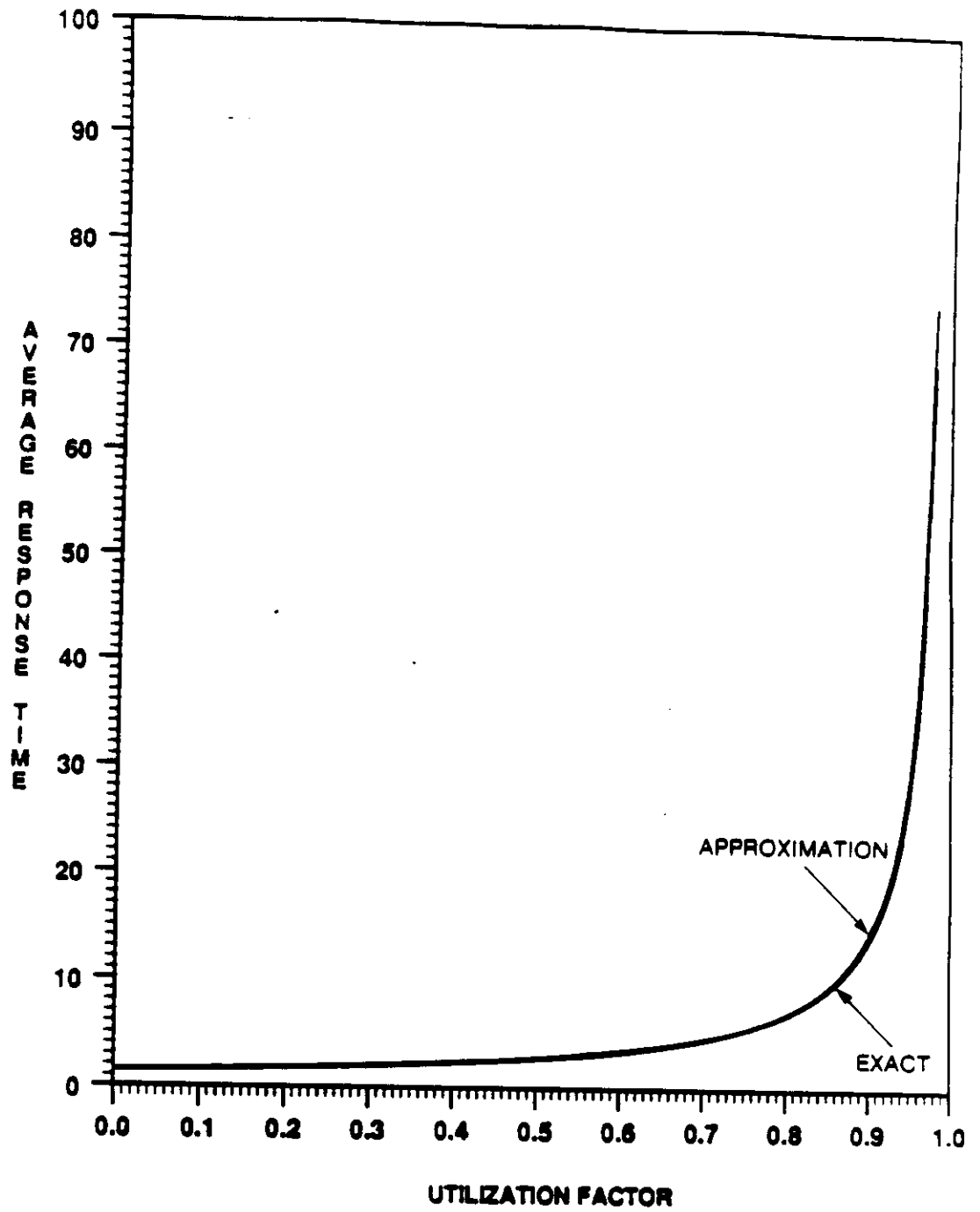


Figure 5.7: Approximation Using a Load Adjustable Service Time Distribution

5.7 Analysis of the $M/G_c/1$ System Using Independent Start-up and Finishing-up Delays

In the preceding sections, we investigated ways to analyze the FJ-2-M/M/1 parallel processing system using an $M/G_c/1$ representation. In such an $M/G_c/1$ system, successive bulk job service times are correlated. A close look at the system revealed that whenever the synchronization box is empty, the current (if any) bulk job service time is distributed as the maximum of two independent identically distributed exponentials; otherwise the current bulk job service time is exponentially distributed. On the other hand, from equation (5.7), we know that the probability of having an empty synchronization box is $(1-\rho)$ which is exactly the probability of having one of the M/M/1 service centers idle unconditioned on the other one. This rather important observation leads us to consider the analysis of the $M/G_c/1$ system as an M/G/1 queueing system obtained from an M/M/1 queueing system in which an additional delay is added to the first customer of each busy period. In Section 5.7.1, we analyze the response time in such queueing systems but in a more general context; namely the response time of an M/G/1 queueing system obtained from another M/G/1 queueing system in which an independent additional delay is added to each customer starting a busy period. The former M/G/1 system is hereafter called *the modified M/G/1 system with start-up periods*.

Section 5.7.2 provides a response time analysis for an M/G/1 queueing system with finishing-up delays; namely a response time analysis of an M/G/1 queueing system obtained from another M/G/1 system in which an independent additional delay is added whenever a busy period ends in the latter system. The former queueing system is hereafter called *the M/G/1 system with finishing-up periods*.

In Section 5.7.3, we compare the response time of both the M/G/1 system with start-up periods and the M/G/1 system with finishing-up periods. We then apply the derived results to obtain a very accurate approximation of the average response time of bulk jobs in the FJ-2-M/M/1 parallel processing system.

Queueing systems where a special treatment is considered whenever they become idle have appeared often in the literature. Miller [Mill64] analyzed a queueing system where the server goes on a vacation "rest period" of a random length whenever it becomes idle. He also investigated a queueing system where the first customer arriving to an empty system is given a special service time. Scholl [Scho76], and subsequently Scholl and Kleinrock [Scho83] analyzed a server with "rest periods" using a different approach. These types of queueing systems were also reported in several other studies including Cooper [Coop70], Heyman [Heym77], Levy and Yechiali [Levy75], Shanthikumar [Shan80], Avi-Itzhak, Maxwell and Miller [Avi-65], Van Der Duyn Schouten [Scho78], and Levy [Levy84]. In particular, in [Levy84] the author considered an M/G/1 system where the system is "turned off" whenever it becomes idle. When a customer arrives to an idle system, it cannot be served immediately; rather an independent random amount of time, called starter, is required to start the empty "

cold " system before such a customer can be served. The author also considered the case where the " cold start " depends on either the amount of work arriving to the system at the beginning of the start-up period, or on the length of the idle period preceding the start-up operation. He showed that the delay distribution in such a queueing system with a " starter " is composed of the direct sum of two independent random variables, namely

1. the delay in the equivalent queueing system without the " starter ", and
2. the additional delay suffered due to the " starter " presence

This decomposition property of the delay, has also been reported by Fuhrmann [Fuhr83], and Doshi [Dosh83]. However, the methods used to derive this property are rather different.

5.7.1 Analysis of the Modified System with Start-up Periods

As stated earlier, and in contrast to the aforementioned studies of M/G/1 systems with special treatments whenever they become idle, the emphasis here is on queueing systems where such a special treatment may, in addition, be needed even though the system is not idle. Specifically, we are interested in the study of the response time in such a modified M/G/1 system (called hereafter the modified M/G/1 system). The modified system is obtained from a pure M/G/1 queueing system (called hereafter the pure system), but where each customer starting a busy period in the pure system is given a special treatment in the modified system. Our objective is to determine the response time distribution of a customer in the modified system. To proceed let us define, for the pure system, the following quantities:

$U(t)$ = the unfinished work in the system at time t .

C_n = the n th customer.

τ_n = the arrival time of customer C_n .

$t_n = \tau_n - \tau_{n-1}$ = the interarrival time between customer C_{n-1} and customer C_n .

x_n = the service time of customer C_n .

Y_i = the length of the i th busy period.

Z_i = the length of the i th idle period, with probability distribution function $Z_i(t) \triangleq P(Z_i \leq t)$, probability density function $z_i(t) \triangleq \frac{dZ_i(t)}{dt}$, and Laplace transform $Z_i^*(s) \triangleq \int_0^{\infty} e^{-st} z_i(t) dt$.

R_n = the response time (i.e., total time spent in the system) of customer C_n , with probability distribution function $R_n(t) \triangleq P(R_n \leq t)$, probability density function $r_n(t) \triangleq \frac{dR_n(t)}{dt}$, and Laplace transform $R_n^*(s) \triangleq \int_0^{\infty} e^{-st} r_n(t) dt$.

Also let us define the following quantities for the modified system:

S_i = the start-up delay suffered in the modified system by the first customer of the i th busy period of the pure system, with probability distribution function $S_i(t) \triangleq P(S_i \leq t)$, probability density function $s_i(t) \triangleq \frac{dS_i(t)}{dt}$, and Laplace transform $S_i^*(s) \triangleq \int_0^{\infty} e^{-st} s_i(t) dt$.

D_i = the total additional delay suffered in the modified system by the first customer of the i th busy period of the pure system, with probability distribution function $D_i(t) \triangleq P(D_i \leq t)$, probability density function $d_i(t) \triangleq \frac{dD_i(t)}{dt}$, and Laplace transform $D_i^*(s) \triangleq \int_0^{\infty} e^{-st} d_i(t) dt$.

B_i = the *propagated* delay from the previous busy period that the first customer of the i th busy period of the pure system suffers, with probability distribution function $B_i(t) \triangleq P(B_i \leq t)$, probability density function $b_i(t) \triangleq \frac{dB_i(t)}{dt}$, and Laplace transform $B_i^*(s) \triangleq \int_0^{\infty} e^{-st} b_i(t) dt$.

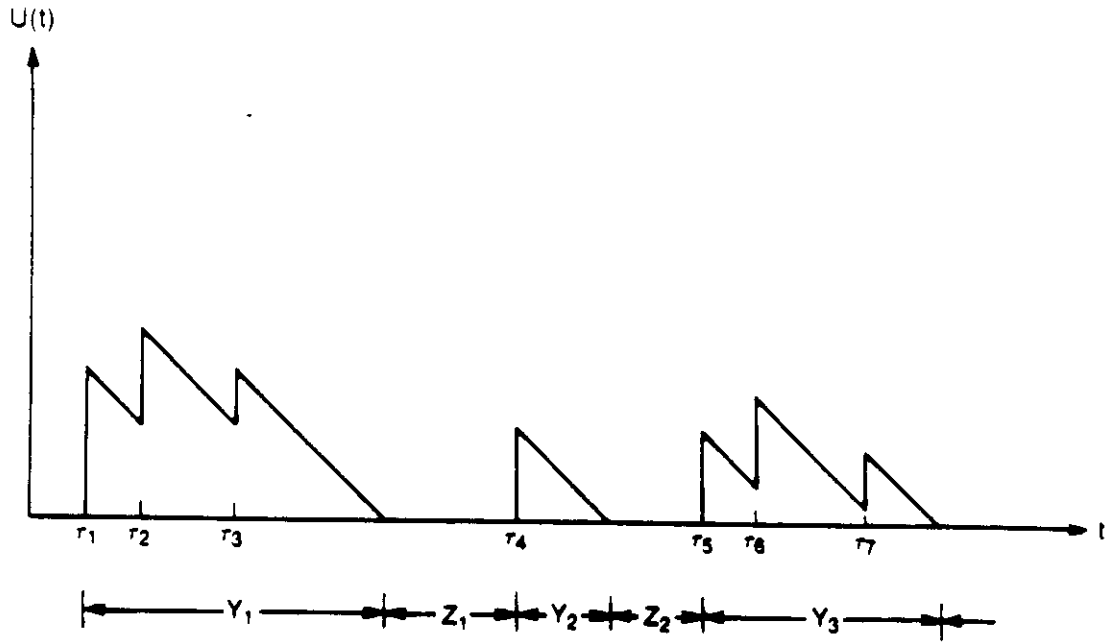
RM_n = the response time (i.e., total time spent in the modified system) of customer C_n , with probability distribution function $RM_n(t) \triangleq P(RM_n(t) \leq t)$, probability density function $rm_n(t) \triangleq \frac{dRM_n(t)}{dt}$, and Laplace transform $RM_n^*(s) \triangleq \int_0^{\infty} e^{-st} rm_n(t) dt$.

From the above definitions, we observe that the additional delay suffered in the modified system by the first customer of the i th busy period of the pure system is the sum of the start-up delay and the propagated delay. In the sequel, we assume the following:

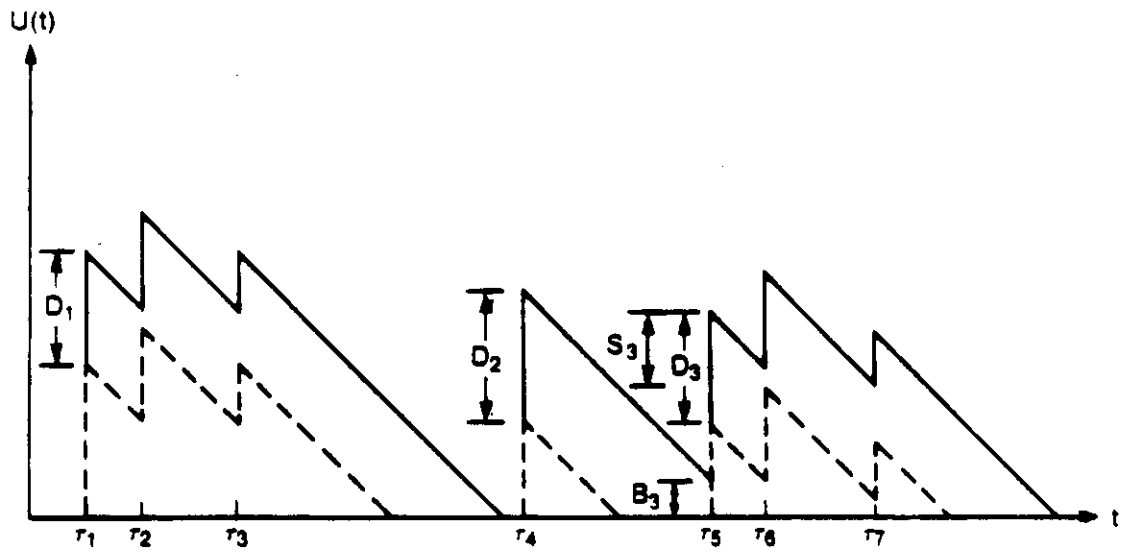
1. the random variables representing the start-up delays, S_i , $i=1,2,\dots$, are independent and identically distributed, and
2. the sequences S_i , $i=1,2,\dots$, t_n , $n=1,2,\dots$ and x_n , $n=1,2,\dots$ are mutually independent sequences

Figure 5.8:(a) depicts the behavior of the unfinished work $U(t)$ in the pure $M/G/1$ system. For the exact same sample of arrivals (i.e., the sequence τ_1, τ_2, \dots) and the same sample of service times (i.e., the sequence x_1, x_2, \dots), we can induce the behavior of the unfinished work $U(t)$ in the modified system. Figure 5.8:(b) depicts this $U(t)$, where the dashed line represents the pure system of Figure 5.8:(a), and the solid line represents the modified system.

From Figure 5.8:(a), we observe that customer C_1 , arriving to an empty system, suffers an additional delay equal to a start-up delay D_1 distributed as S_1 . Customers C_2 and C_3 suffer exactly the same additional delay. Customer C_4 , arriving to an empty system in both the pure and the modified systems, suffers the additional delay of a second start-up delay S_2 , which is independent and identically distributed as S_1 . However, customer C_5 , arriving to a busy modified system (see Figure 5.8:(b)), and an empty pure system (see Figure 5.8:(a)), must suffer



(a): Event Time Diagram for the Pure M/G/1 System



(b): Event Time Diagram for the Modified M/G/1 System

Figure 5.8: Behavior of the Unfinished Work in the System With and Without Start-up Delays

both a start-up delay and the propagated delay ($D_2 - Z_2$). Customers C_6 and C_7 suffer the same additional delay as customer C_5 .

We now proceed to analyze the modified M/G/1 system. The additional delay suffered by the first customer of busy period i of the pure system, can be recursively calculated from the following recursion equation:

$$\begin{cases} D_1 = S_1 \\ D_{i+1} = \begin{cases} S_{i+1} & \text{if } D_i \leq Z_i \\ D_i - Z_i + S_{i+1} & \text{if } D_i \geq Z_i \end{cases} \end{cases} \quad (5.44)$$

The first line in the above recursion equation represents the initial condition. Indeed both the pure and the modified systems are assumed to be idle when the first customer arrived. Thus the additional delay amounts to a start-up delay period. The second line represents the case where the arriving customer (i.e., the first customer of busy period i) finds both the pure system and the modified system idle. Thus only a start-up period is needed. Line 3 of the recursion equation represents the case where the customer finds the modified system busy; thus the additional delay for such customer is the sum of the propagated delay from the previous busy period plus an independent start-up period.

Our main interest is to evaluate the additional delay suffered by an arbitrary customer. From Figure 5.8, we observe that such a delay has the same distribution as the additional delay suffered by the first customer of the pure system's busy period. This is stated in the following Theorem.

Theorem 5.7.1

Customers belonging to the same busy period in the pure system, suffer the same additional delay in the modified system with start-up periods.

□

Also from the recursion equation (5.44), we observe that the additional delay D_i is independent of the length of the i th idle period of the pure system, $i=1,2,\dots$. In fact, the random variables Z_i and S_i , $i=1,2,\dots$ are mutually independent, and that D_i is only a function of Z_1, Z_2, \dots, Z_{i-1} and S_1, S_2, \dots, S_{i-1} .

Now we proceed to show that the response time of a customer in the modified system possesses the aforementioned decomposition property, in the sense that this response time is the direct sum of the additional delay suffered in the modified system plus the response time in the pure system.

Theorem 5.7.2

The additional delay suffered by an arbitrary customer in the modified system with start-up periods is independent of the response time such a customer has in the pure system.

Proof

The response time of an arbitrary customer in the pure system is only a function of the behavior of the system from the start of the busy period. On the other hand, the additional delay suffered by such a customer in the modified system is equal to the additional delay of the first customer starting the same busy period. The proof is thus complete by noticing that the additional delay of the first customer is only a function of what had happened prior to the busy period.

□

Similarly, we can show that the additional delay D_i , $i=1,2,\dots$ is independent of the number of customers served in the i th busy period. Consequently, Theorem 5.7.1 results in the following Corollary.

Corollary 5.1

The limiting distribution of the additional delay suffered by an arbitrary customer in the modified system with start-up periods is identical to the limiting distribution of D_i .

□

Now we proceed to determine the response time of an arbitrary customer in the modified system. Since the job arrival process is Poisson with parameter λ , it follows that the interarrival times as well as the length of the idle periods are exponentially distributed with parameter λ . Hence we have:

$$Z_i(t) = 1 - e^{-\lambda t} \quad , \quad z(t) = z_i(t) = \lambda e^{-\lambda t} \quad , \quad Z^*(s) = Z_i^*(s) = \frac{\lambda}{\lambda + s}$$

where $z(t)$ and $Z^*(s)$ are respectively the limiting behavior of $z_i(t)$ and $Z_i^*(s)$ as i approaches infinity. From the definition of the propagated delay, B_i , we have:

$$B_{i+1} = \begin{cases} D_i - Z_i & \text{if } D_i \geq Z_i \\ 0 & \text{if } D_i < Z_i \end{cases} \quad i \geq 1$$

Combining the above equation with the recurrence equation (5.44), we obtain the following recurrence equation:

$$D_{i+1} = B_{i+1} + S_{i+1} \quad i \geq 1$$

Since the start-up delay S_{i+1} and the propagated delay B_{i+1} are independent, we get:

$$D_{i+1}^*(s) = S_{i+1}^*(s) \cdot B_{i+1}^*(s) \quad i \geq 1$$

and as i approaches the infinity, we get the following limiting behavior:

$$D^*(s) = S^*(s) \cdot B^*(s) \quad (5.45)$$

From our definitions, the probability density function of the propagated delay is thus given by:

$$b_{i+1}(t) = \int_{r=0}^{\infty} z(r) d_i(r+t) dr + \mu_0(t) \left[\int_{r=0}^{\infty} d_i(r) dr \int_{u=0}^{\infty} z(u) du \right]$$

Using $z(r) = \lambda e^{-\lambda r}$, and passing to the Laplace Transform of $b_{i+1}(t)$, we get:

$$\begin{aligned} B_{i+1}^*(s) &= \int_{t=0}^{\infty} e^{-st} \left[\int_{r=0}^{\infty} \lambda e^{-\lambda r} d_i(r+t) dr \right] dt \\ &\quad + \int_{t=0}^{\infty} e^{-st} \mu_0(t) \left[\int_{r=0}^{\infty} d_i(r) dr \int_{u=0}^{\infty} \lambda e^{-\lambda u} du \right] dt \\ &= \int_{t=0}^{\infty} e^{-st} \left[\int_{r=0}^{\infty} \lambda e^{-\lambda r} d_i(r+t) dr \right] dt + D_i^*(\lambda) \\ &= \int_{r=0}^{\infty} \lambda e^{-\lambda r} \left[\int_{y=r}^{\infty} e^{-sy} d_i(y) dy \right] dr + D_i^*(\lambda) \\ &= \frac{\lambda}{s-\lambda} \int_{y=0}^{\infty} e^{-sy} d_i(y) (e^{-\lambda y} - 1) dy + D_i^*(\lambda) \\ &= \frac{\lambda(D_i^*(\lambda) - D_i^*(s))}{s-\lambda} + D_i^*(\lambda) \end{aligned}$$

which finally gives:

$$B_{i+1}^*(s) = \frac{sD_i^*(\lambda) - \lambda D_i^*(s)}{s-\lambda} \quad i \geq 1$$

this gives us the following limiting behavior :

$$B^*(s) = \frac{sD^*(\lambda) - \lambda D^*(s)}{s-\lambda}$$

Using the above equation along with equation (5.45), yields:

$$D^*(s) = \frac{sS^*(s)D^*(\lambda)}{s-\lambda+\lambda S^*(s)}$$

Using the fact that $D^*(s=0) = 1$, we get :

$$D^*(\lambda) = 1-\lambda\bar{S}$$

where \bar{S} is the average start-up delay. Hence we finally obtain:

$$D^*(s) = S^*(s) \frac{s(1-\lambda\bar{S})}{s-\lambda+\lambda S^*(s)} \quad (5.46)$$

Therefore, by using $\bar{D} = -\left. \frac{dD^*(s)}{ds} \right|_{s=0}$, the average additional delay is thus given by:

$$\bar{D} = \frac{\lambda\bar{S}^2}{2(1-\lambda\bar{S})} + \bar{S} \quad (5.47)$$

From the decomposition property, Theorem 5.7.1, the average response time of an arbitrary customer in the modified M/G/1 system with start-up periods is then :

$$RM_i = R_i + D_i \quad i \geq 1$$

and since the additional delay suffered by an arbitrary customer in the modified system is independent of the response time such a customer has in the pure system (Theorem 5.7.2), we have:

$$RM_n^*(s) = R_n^*(s) \cdot D_n^*(s) \quad n \geq 1$$

and as n approaches the infinity, we obtain the following limiting behavior:

$$RM^*(s) = R^*(s)D^*(s) \quad (5.48)$$

Finally using equation (5.47), the average response time of a customer in the modified system with start-up periods in the steady state, denoted by \bar{RM} is given by:

$$\bar{RM} = \bar{R} + \frac{\lambda\bar{S}^2}{2(1-\lambda\bar{S})} + \bar{S} \quad (5.49)$$

where \bar{R} is the average response time of a customer in the pure M/G/1 system in the steady state.

5.7.2 Analysis of the Modified System with Finishing-up Periods

Consider the modified M/G/1 queueing system obtained from another pure M/G/1 system where an independent additional delay period is added in the modified system whenever a busy period ends in the pure system. This can be thought of as the server in the modified system taking a rest period whenever the pure system ends a busy period. In this section, we provide a response time analysis for such a modified M/G/1 queueing system. The approach we use is similar to the one used in the previous section, in particular, we use the same notation as for the pure system. For the modified system, we define the following quantities:

F_i = the finishing-up delay (i.e., the server rest period) incorporated in the modified system upon the termination of the i th busy period of the pure system; with probability distribution function $F_i(t) \triangleq P(F_i \leq t)$, probability density function $f_i(t) \triangleq \frac{dF_i(t)}{dt}$, and Laplace transform

$$F_i^*(s) \triangleq \int_0^{\infty} e^{-st} f_i(t) dt.$$

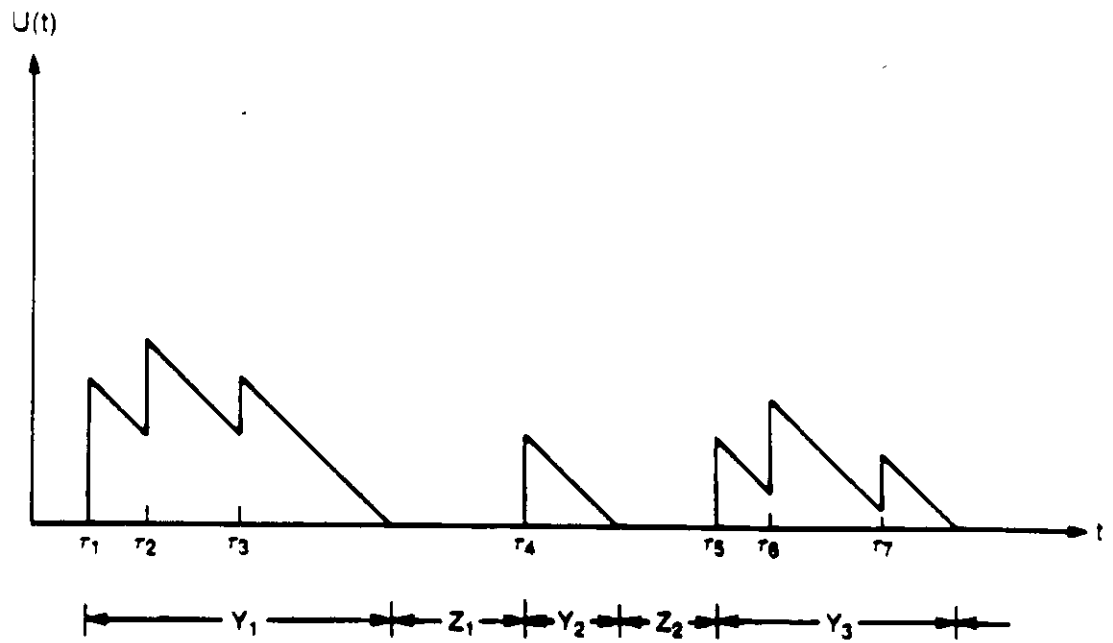
D_i = the total additional delay suffered in the modified system by the first customer of the i th busy period of the pure system, with probability distribution function $D_i(t) \triangleq P(D_i \leq t)$, probability density function $d_i(t) \triangleq \frac{dD_i(t)}{dt}$, and Laplace transform $D_i^*(s) \triangleq \int_0^{\infty} e^{-st} d_i(t) dt$.

In the sequel, we assume the following:

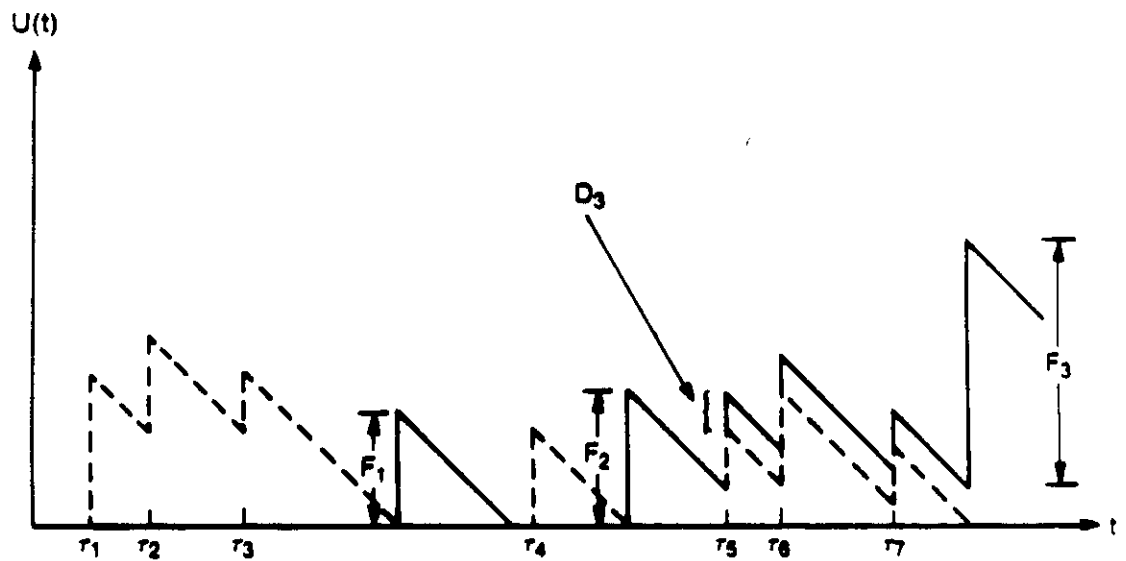
1. the random variables representing the finishing-up delay periods, F_i $i=1,2,\dots$, are independent and identically distributed, and
2. the sequences F_i $i=1,2,\dots$ and t_n $n=1,2,\dots$ and x_n $n=1,2,\dots$ are mutually independent sequences.

Figure 5.9:(a) depicts the behavior of the unfinished work $U(t)$ in the pure M/G/1 system. For the exact same sample of arrivals (i.e., the sequence τ_1, τ_2, \dots) and the same sample of service times (i.e., x_1, x_2, \dots), we can induce the behavior of the unfinished work $U(t)$ in the modified system. Figure 5.9:(b) depicts such $U(t)$, where the dashed line represents the pure system of Figure 5.9:(a), and the solid line represents the modified M/G/1 queueing system.

From Figure 5.9, we observe that upon the termination of the first busy period, an additional finishing-up delay is added in the modified system. Customer C_4 arrives to an empty system in Figure 5.9:(a), and since the additional delay is smaller than the next idle period, it also finds the modified system empty. At the end of C_4 service time, the pure system becomes idle and thus another independent additional finishing-up delay is added in the modified system. However, customer C_5 arrives and finds the pure system empty and the modified system busy.



(a): Event Time Diagram for the Pure M/G/1 System



(b): Event Time Diagram for the Modified M/G/1 System

Figure 5.9: Behavior of the Unfinished Work in the System With and Without Finishing-up Delays

and consequently must wait the propagated delay which is the remaining unfinished work of the modified system.

We now proceed to analyze the modified M/G/1 system. The total additional delay suffered by the first customer of the i th busy period of the pure system can be recursively calculated from the following recursion equation:

$$\begin{cases} D_1 = 0 \\ D_{i+1} = \begin{cases} D_i + F_i - Z_i & \text{if } D_i + F_i \geq Z_i \\ 0 & \text{if } D_i + F_i \leq Z_i \end{cases} \end{cases} \quad (5.50)$$

The first line in the above recursion equation represents the initial condition of the system being empty. In the case where $D_i + F_i \leq Z_i$, no delay is suffered by the first customer of the $(i+1)$ st busy period of the pure system since the length of the i th idle period of the pure system is larger than the accumulated delay $D_i + F_i$; this is represented in line 3 of the above recursion equation. Finally, line 2 represents the case where the accumulated delay is larger than the length of the i th idle period of the pure system. Our objective is to evaluate the total additional delay suffered by an arbitrary customer. From Figure 5.9, we observe that such a delay has the same distribution as the additional delay suffered by the first customer of the pure system's busy period. This is stated in the following Theorem.

Theorem 5.7.3

Customers belonging to the same busy period in the pure M/G/1 system suffer the same additional delay in the modified M/G/1 system with finishing-up periods.

■

From the recursion equation (5.50), we observe that the additional delay D_i is independent of the length of the i th idle period of the pure system, $i=1,2,\dots$. Indeed the random variables Z_i and F_i , $i=1,2,\dots$ are mutually independent, and the additional delay D_i is only a function of Z_1, Z_2, \dots, Z_{i-1} and F_1, F_2, \dots, F_{i-1} . The response time of a customer in the modified system has the decomposition property in the sense that the response time is the direct sum of the additional delay suffered in the modified system plus the response time in the pure system.

Theorem 5.7.4

The additional delay suffered by an arbitrary customer in the modified M/G/1 system with finishing-up periods is independent of the response time such a customer has in the pure system.

■

The proof of the above Theorem is similar to the proof of Theorem (5.7.2). Also, since the additional delay D_i , $i=1,2,\dots$ is independent of the number of customers served in the i th busy period, we obtain the following Corollary.

Corollary 5.2

The limiting distribution of the additional delay suffered by an arbitrary customer in the modified M/G/1 system with finishing-up periods is identical to the limiting distribution of D_i .

411

Now we proceed to determine the response time of an arbitrary customer in the modified system. Let $B_i = D_i + F_i$ with probability distribution $B_i(t) = P(B_i \leq t)$, probability density function $b_i(t) = \frac{dB_i(t)}{dt}$, and Laplace transform $B_i^*(s) = \int_0^\infty e^{-st} b_i(t) dt$. From (5.50), we get the following equivalent recursion equation:

$$D_{i+1} = \begin{cases} B_i + Z_i & \text{if } B_i \geq Z_i \\ 0 & \text{if } B_i \leq Z_i \end{cases} \quad i \geq 1 \quad (5.51)$$

Since the finishing-up delay F_i and the additional delay D_i are independent, we have:

$$B_i^*(s) = D_i^*(s) F_i^*(s) \quad i \geq 1$$

and as i approaches the infinity, we get the following limiting behavior:

$$B^*(s) = D^*(s) \cdot F^*(s)$$

From our definition, the probability density function of the additional delay in the M/G/1 system with finishing-up periods is thus given by:

$$d_{i+1}(t) = \int_{r=0}^{\infty} z(r) b_i(r+t) dr + \mu_0(t) \int_{r=0}^{\infty} \left[b_i(r) \int_{u=r}^{\infty} z(u) du \right] dr \quad (5.52)$$

Since the job arrival process is Poisson with parameter λ , it follows that the interarrival times as well as the length of the idle periods are exponentially distributed with the same parameter λ . Hence equation (5.52) yields:

$$D_{i+1}^*(s) = \int_{t=0}^{\infty} e^{-st} \left[\int_{r=0}^{\infty} \lambda e^{-\lambda r} b_i(r+t) dr \right] dt + \int_{t=0}^{\infty} e^{-st} \left[\mu_0(t) \int_{r=0}^{\infty} b_i(r) \int_{u=0}^{\infty} z(u) du dr \right] dt$$

$$\begin{aligned}
&= \int_{t=0^+}^{\infty} e^{-st} \left[\int_{r=0}^{\infty} \lambda e^{-\lambda r} b_i(r+t) dr \right] dt + B_i^*(\lambda) \\
&= \int_{r=0}^{\infty} \lambda e^{-(\lambda-s)r} \left[\int_{y=r}^{\infty} e^{-sy} b_i(y) dy \right] dr + B_i^*(\lambda) \\
&= \frac{\lambda[B_i^*(\lambda) - B_i^*(s)]}{s-\lambda} + B_i^*(\lambda)
\end{aligned}$$

which finally gives:

$$D_{i+1}^*(s) = \frac{sD_i^*(\lambda)F_i^*(\lambda)}{s-\lambda+\lambda F_i^*(s)} \quad i \geq 1$$

this gives us the following limiting behavior:

$$D^*(s) = \frac{sD^*(\lambda)F^*(\lambda)}{s-\lambda+\lambda F^*(s)} \quad (5.53)$$

Using the fact that $D^*(s=0) = 1$, equation (5.53) yields:

$$D^*(\lambda)F^*(\lambda) = 1 - \lambda \bar{F}$$

where \bar{F} is the average finishing-up delay. Hence, we finally obtain:

$$D^*(s) = \frac{s(1-\lambda \bar{F})}{s-\lambda+\lambda F^*(s)} \quad (5.54)$$

Therefore, by using $\bar{D} = -\frac{dD^*(s)}{ds} \Big|_{s=0}$, the average additional delay is thus given by:

$$\bar{D} = \frac{\lambda \bar{F}^2}{2(1-\lambda \bar{F})} \quad (5.55)$$

From the decomposition property, Theorem 5.7.3, the average response time of an arbitrary customer in the modified M/G/1 system with finishing-up periods is then :

$$RM_i = R_i + D_i \quad i \geq 1$$

and since the additional delay suffered by an arbitrary customer in the modified system is independent of the response time such a customer has in the pure system (Theorem 5.7.4), we have:

$$RM_n^*(s) = R_n^*(s)D_n^*(s) \quad n \geq 1$$

As n approaches the infinity, we obtain the following limiting behavior:

$$RM^*(s) = R^*(s) \cdot D^*(s) \quad (5.56)$$

Finally using equation (5.55), the average response time of a customer in the modified system with finishing-up periods in the steady state, denoted by \overline{RM} , is given by:

$$\overline{RM} = \overline{R} + \frac{\lambda \overline{F^2}}{2(1-\lambda \overline{F})} \quad (5.57)$$

where \overline{R} is the average response time of a customer in the pure M/G/1 system in the steady state.

5.7.3 Comparison and Application to the Study of the FJ-2-M/M/1 System

Now we proceed to interpret the results obtained in the last two sections, and to provide an alternative way to derive such results; specifically equation (5.46) and equation (5.54). Then we shall apply such results to determine the response time of a bulk job in the FJ-2-M/M/1 parallel processing system.

The Laplace transform of the additional delay of an arbitrary customer in the modified M/G/1 system with start-up periods, given by equation (5.46), has exactly the same form as the Pollaczek-Khinchin (P-K) transform equation [Klei75] of the distribution of the total time (i.e., the response time) spent in the M/G/1 queueing system that has for service time distribution the start-up delay distribution $S(t)$. Let us return to the recursion equation (5.44). Let such a recursion equation represent the functional equation of an M/G/1 system where S_i is the service time of the i th customer, D_i the response time of the i th customer, and Z_i the interarrival time between the i th and the $(i+1)$ st customers. Indeed, it is an M/G/1 system since the variables Z_i , $i=1,2,\dots$ are assumed to be exponentially distributed. Figure 5.10 represents an event time diagram of such an M/G/1 system. From Figure 5.10:(a), we observe that a customer, say C_{i+1} , finding the system empty is equivalent to have $D_i \leq Z_i$. In such a case, the response time D_{i+1} for customer C_{i+1} is equal to its service time S_{i+1} . On the other hand, if customer C_{i+1} finds the system busy, that is $D_i \geq Z_i$ (see Figure 5.10:(b)), its response time is the sum of its waiting time, $D_i - Z_i$, plus its service time S_{i+1} .

The Laplace transform of the additional delay of an arbitrary customer in the modified M/G/1 system with finishing-up periods, given by equation (5.54), has exactly the same form as the (P-K) transform equation [Klei75] of the distribution of the waiting time in an M/G/1 queueing system having the finishing-up delay distribution as its service time distribution. As in the previous case, we see the recursion equation of the additional delay in such a modified system, equation (5.54), is the functional equation of an M/G/1 system where F_i is the service time of the i th customer, D_i the waiting time of the i th customer, and Z_i the interarrival time between the i th and the $(i+1)$ st customers. Figure 5.11 represents an event time diagram of such an M/G/1 system. The waiting time of C_{i+1} is zero if, upon arrival, he finds the system idle, that is

$D_i + F_i \leq Z_i$ (see Figure 5.11:(b)). On the other hand, if upon arrival, customer C_{i+1} finds the system busy, that is $D_i + F_i \geq Z_i$ (see Figure 5.11:(a)), then he must wait $(D_i + F_i) - Z_i$. Notice that $D_i + F_i$ is the response time of customer C_i .

We know from Theorem 5.7.2 that the response time of a job in the modified M/G/1 system with start-up periods is the direct sum of the additional delay suffered by such a job plus the response time it would have in the pure M/G/1 system. This leads us to represent the FJ-2-M/M/1 parallel processing system as *Two queueing systems in Tandem*.

Figure 5.12 depicts such a representation. In the sequel, we shall use the term *queueing system* to refer to system (S1) or system (S2) of Figure 5.12, and the term *service center* to refer to service center 1 or service center 2 of Figure 5.2:(a).

From the definition of the FJ-2-M/M/1 system, we know that a task leaving one of the service centers either waits in the synchronization box for its sibling to complete, or finds its sibling already in the synchronization box and thus a Join operation is immediately performed and the corresponding bulk job departs the system at once. Let us call the tasks that must wait in the synchronization box *the effective tasks*. In Figure 5.12, we are only interested in these effective tasks.

We now proceed to characterize queueing system (S1) and queueing system (S2) of Figure 5.12. The arrival process of effective tasks to the Tandem system is Poisson with aggregate λ . The service time of an effective task in (S2) is exponentially distributed with mean $\frac{1}{\mu}$, since the oldest effective task in the synchronization box waits an amount of time equal to the remaining time to complete the service of its sibling at one of the service centers. To characterize the service time of effective tasks in (S1), notice that:

- a. whenever $\bar{n}_b = 0$ (i.e., $\bar{n}_1 = \bar{n}_2$), the interdeparture time between effective tasks from the service centers of the FJ-2-M/M/1 system are distributed as the minimum of two exponentials with the same mean $\frac{1}{\mu}$.
- b. whenever $\bar{n}_1 \neq \bar{n}_2 \neq 0$, the interdeparture time between effective tasks from the service centers of the FJ-2-M/M/1 system is exponentially distributed with mean $\frac{1}{\mu}$, since only task departures from the service center with the lowest occupancy are accounted for.

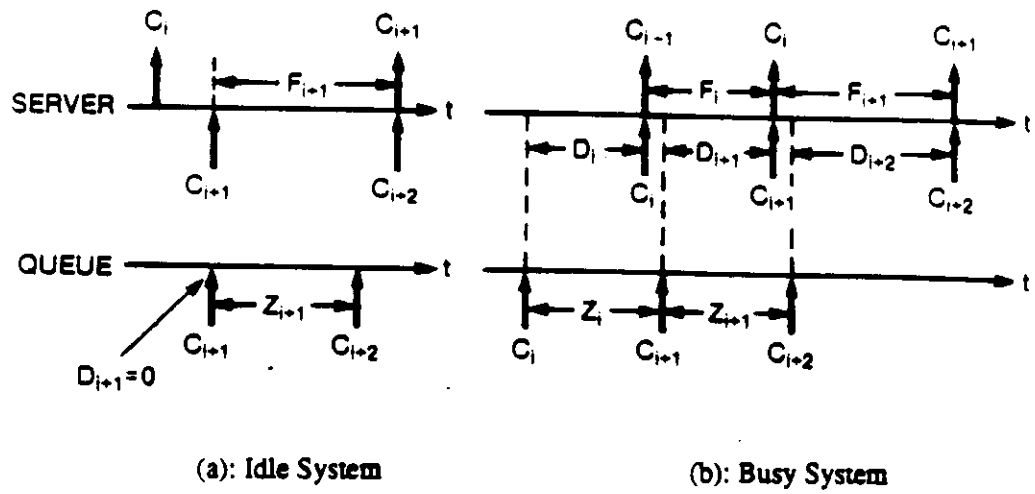


Figure 5.11: An Event Time Diagram for the System With Finishing-up Periods

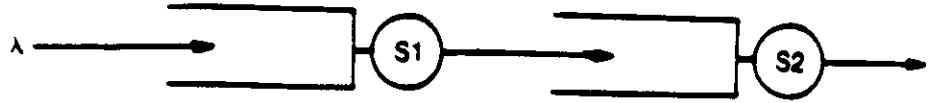


Figure 5.12: Representation of the FJ-2-M/M/1 System as Two Queues in Tandem

Let \bar{X} be the random variable representing the service time of an effective task at the queuing system (S1), with probability density function $b_X(x)$, average \bar{X} and second moment \bar{X}^2 . From the above, we have:

$$b_X(x) = \mu e^{-\mu x} \left[1 - \sum_{i=1}^{\infty} P(\bar{n}_1=i, \bar{n}_2=0) - \sum_{j=1}^{\infty} P(\bar{n}_1=0, \bar{n}_2=j) - \sum_{i=0}^{\infty} P(\bar{n}_1=i, \bar{n}_2=i) \right] \\ + 2\mu e^{-2\mu x} \sum_{i=0}^{\infty} P(\bar{n}_1=i, \bar{n}_2=i)$$

Using equations (5.3), (5.4), and (5.7), we obtain:

$$b_X(x) = \mu e^{-\mu x} \left[1 - 3(1-\rho) + 2(1-\rho)^{\frac{3}{2}} \right] + 2\mu e^{-2\mu x} (1-\rho) \quad (5.58)$$

From the above expression of the service time distribution of (S1), we get:

$$\bar{X} = \frac{5\rho - 3 + 4(1-\rho)^{\frac{3}{2}}}{2\mu}$$

$$\bar{X}^2 = \frac{11\rho - 7 + 8(1-\rho)^{\frac{3}{2}}}{2\mu^2}$$

Using the above equations, the response time of an effective task through the queuing system (S1), denoted by T_{S1} is thus given by:

$$T_{S1} = \frac{\lambda \bar{X}^2}{2(1-\lambda \bar{X})} + \bar{X} \\ = \frac{\rho(11\rho - 7 + 8(1-\rho)^{\frac{3}{2}})}{2\mu \left[2 - \rho(5\rho - 3 + 4(1-\rho)^{\frac{3}{2}}) \right]} + \frac{5\rho - 3 + 4(1-\rho)^{\frac{3}{2}}}{2\mu}$$

The queueing system (S2) is a G/M/1 queueing system. In the sequel, we shall approximate the response time of this system by an M/M/1 system response time with the same arrival rate. In fact, we may consider the service probability density distribution $b_X(x)$ to represent the probability density distribution of the start-up delay in the modified M/G/1 system. Hence an approximation to the response time of a bulk job through the FJ-2-M/M/1 parallel processing system may be formulated as:

$$T(2) = \frac{\rho}{1-\rho} + T_{S1}$$

Finally by replacing T_{S1} by its value, we obtain:

$$T(2) = \frac{\rho}{1-\rho} + \frac{\rho(11\rho-7+8(1-\rho)^{\frac{3}{2}})}{2\mu[2-\rho(5\rho-3+4(1-\rho)^{\frac{3}{2}})]} + \frac{5\rho-3+4(1-\rho)^{\frac{3}{2}}}{2\mu} \quad (5.59)$$

Figure 5.13 depicts the average response time given by equation (5.9) along with the average response time given by equation (5.59), as a function of the utilization factor ρ . This figure shows the excellent accuracy of the present approximation. In fact, both curves are almost superimposed, and the relative error is less than 2% overall the permissible values of the utilization factor ρ .

5.8 Approximate Analysis of the FJ-P-M/M/1 System

We now return to the P processors case, and proceed to provide an approximate solution for the measure $T(P)$ through the parallel processing system FJ-P-M/M/1 where $P \geq 1$. In Section 5.3, we provided upper and lower bounds for the more general system FJ-P-GI/GU/1, given respectively by inequalities (5.12) and (5.13). Since we have identical service centers, we may define $P(\bar{T} \leq x) = P(\bar{T}_j \leq x) \quad j=1, \dots, P$. Recall that $\bar{T}_j, j=1, \dots, P$ is the random variable representing the response time of tasks through service center j . Consequently, inequality (5.13) reduces to:

$$T(P) \leq \int_0^{\infty} [1 - (P(\bar{T} \leq x))^P] dx$$

From [Klei75], we know that $P(\bar{T} \leq x) = 1 - e^{-\mu(1-\rho)x} \quad x \geq 0$ where $\rho = \frac{\lambda}{\mu}$, the utilization factor of a service center. Therefore, the above inequality gives:

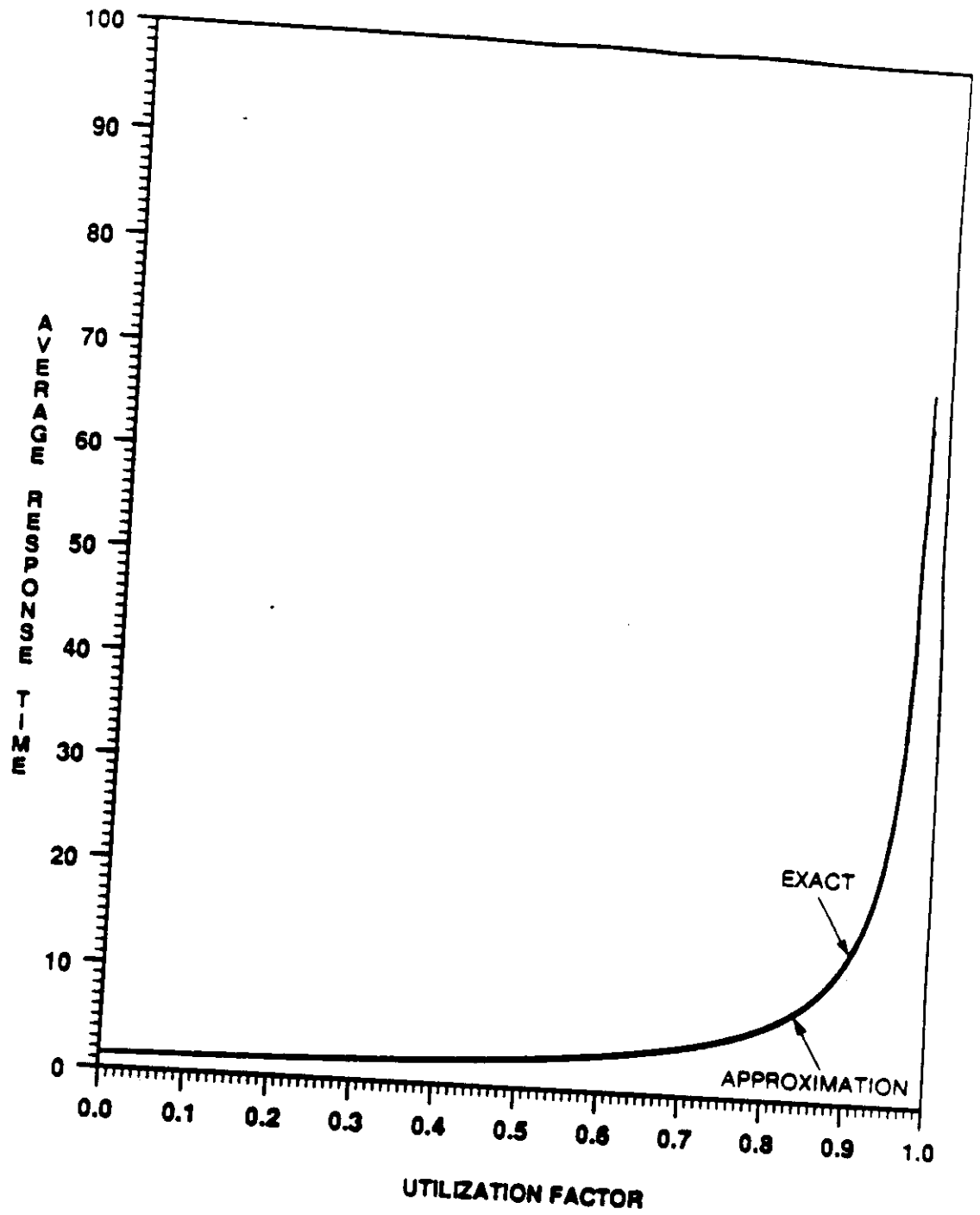


Figure 5.13: Approximation Using a Tandem Representation of the FJ-2-M/M/1 System

$$T(P) \leq \int_0^{\infty} \left[1 - \left[1 - e^{-\mu(1-\rho)x} \right]^P \right] dx$$

which amounts to :

$$T(P) \leq \frac{1}{\mu(1-\rho)} \int_0^{\infty} \left[1 - (1 - e^{-x})^P \right] dx$$

this finally gives:

$$T(P) \leq T_{M/M/1} H(P) \quad P \geq 1 \quad (5.60)$$

where $T_{M/M/1} = \frac{1}{\mu(1-\rho)}$ which is the average response time of an M/M/1 queueing system with utilization factor ρ , and $H(P) = \sum_{i=1}^P \frac{1}{i}$ the harmonic series. On the other hand, a very simple lower bound for the FJ-P-M/M/1 parallel processing system may be obtained by neglecting the queueing effects, that is :

$$T(P) \geq \frac{1}{\mu} H(P) \quad P \geq 1 \quad (5.61)$$

From inequalities (5.60) and (5.61), we observe that both bounds grow at the same rate $H(P)$. The tightness of such bounds is of no concern here. Since $\lim_{P \rightarrow \infty} T(P) \rightarrow \infty$, then it must be the case that $T(P)$ also grows at the same rate. Consequently, knowing the value of $T(1) = T_{M/M/1} = \frac{1}{\mu(1-\rho)}$, we may write* :

$$T(P) = G_P(\rho) T_{M/M/1} \quad P \geq 1 \quad (5.62)$$

where the function $G_P(\rho)$ is a scaling factor that grows at the rate $H(P)$. A first order approximation to $G_P(\rho)$ may be formulated as:

$$G_P(\rho) = a(\rho) + b(\rho)H(P) \quad P \geq 1 \quad (5.63)$$

From equation (5.62), we have $G_1(\rho) = 1$, which along with equation (5.63) yields:

$$b(\rho) = 1 - a(\rho)$$

Thus substituting $b(\rho)$ in equation (5.63) yields:

$$G_P(\rho) = a(\rho) \left[1 - H(P) \right] + H(P) \quad P \geq 1 \quad (5.64)$$

To determine the unknown function $a(\rho)$, we use the expression for $T(2)$ given by equation (5.11), that is:

* This approximation technique is referred to as the *Scaling Approximation* [Nels85].

$$T(2) = \frac{33}{2(11+\rho)} T_{M/M/1} = \left[a(\rho) \left[1-H(2) \right] + H(2) \right] T_{M/M/1}$$

which gives:

$$a(\rho) = \frac{3\rho}{11+\rho} \quad (5.65)$$

Substituting $a(\rho)$ from equation (5.65) into equations (5.64) and (5.63), leads to the following approximate expression of the performance measure $T(P)$:

$$T(P) = \left[H(P) - \frac{3\rho}{11+\rho} \left[H(P) - 1 \right] \right] T_{M/M/1} \quad P \geq 1 \quad (5.66)$$

Validation of The Approximation

Figure 5.14 depicts the average response time $T(P)$ through the FJ-P-M/M/1 system for the values $P=4,8,16,32$ and 64 . To validate our approximation, we simulated the FJ-P-M/M/1 system for these values of P . The method used to estimate the extent of the transient state is "The Method of Independent Replications" [Lave83], and the method used to estimate the statistic $T(P)$ in steady state is "The Method of Batch Means" [Lave83]. The confidence intervals, represented as vertical bars in Figure 5.14, are obtained via the t-distribution, and are of 90% level.

From Figure 5.14, we observe that for small to moderate values of the utilization factor ρ (i.e., $\rho \in [0,0.6]$), the approximation is very accurate in the sense that the confidence intervals are very small. For larger values of ρ , the widths of the confidence intervals are larger, however the approximation is still accurate and within the imposed accuracy (i.e., the 90% confidence level). Nevertheless, it is necessary to realize that for large values of the utilization factor ρ , the variability in the queuing measures increases, and thus has a side effect on the width of these confidence intervals.

For the value $P=4$ (the bottom curve), the approximation results in very accurate values of the statistic $T(4)$; indeed the mean relative error (i.e., relative to the middle of the confidence interval) is less than 3%. As P gets larger, we observe that the width of the confidence intervals get larger also, thus increasing the mean relative error. For $P=64$ (the highest curve in Figure 5.14), the mean relative error is about 5% for $\rho \in [0.6,0.95]$.

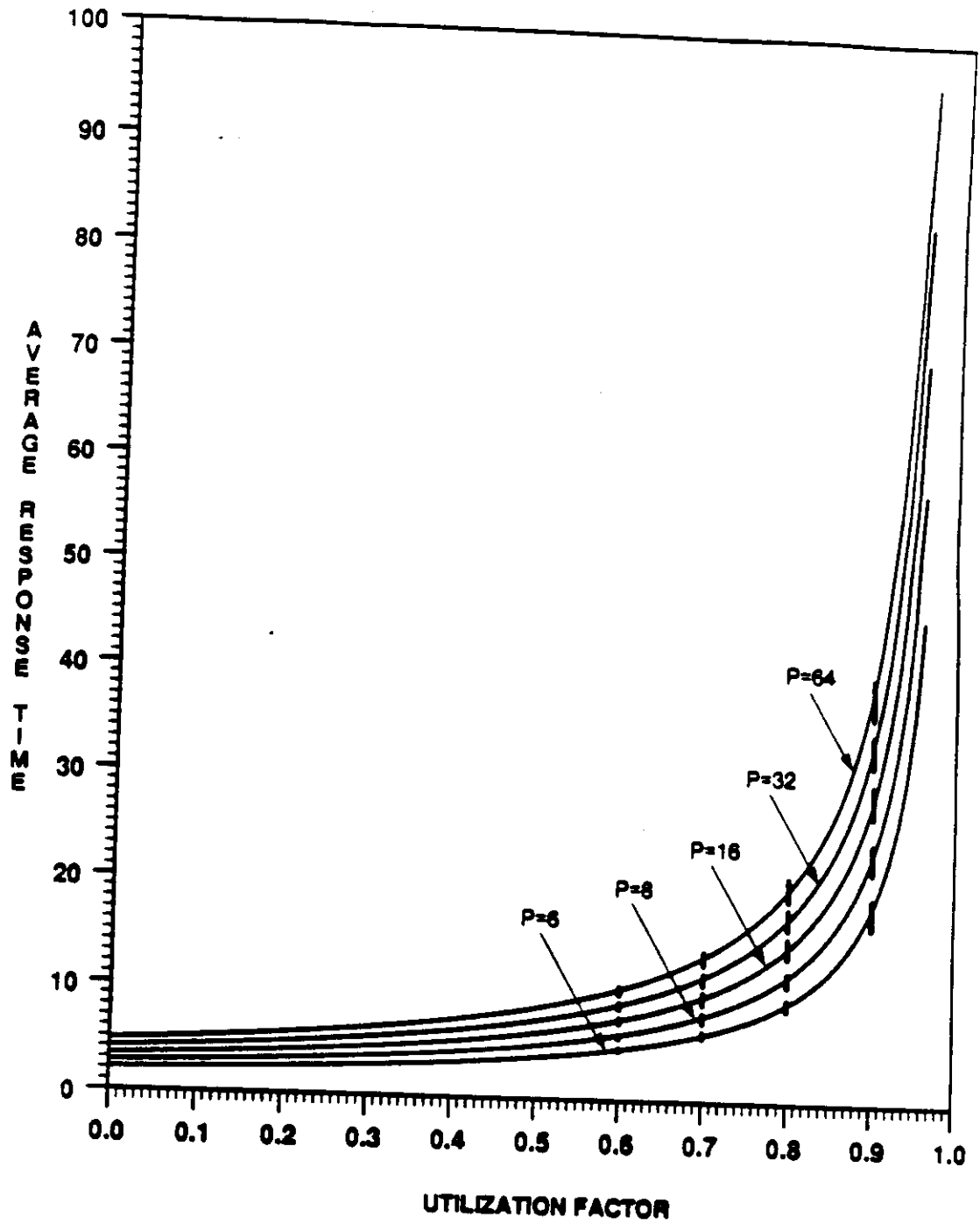


Figure 5.14: Validation of the Approximation of the FJ-P-M/M/1 System

5.9 Conclusion

In this chapter, we construct models and methodologies to analyze the job average response time through a Fork-Join parallel processing system. In Section 5.3, we dealt with the P heterogeneous processors case, where using a folk Theorem in queueing theory, we provided a lower bound on the job average response time, and using associated random variables, we were able to formulate an upper bound on the job average response time. In Section 5.4, we started investigating the Fork-Join parallel processing system comprising two identical and exponential servers. We regarded this parallel processing system as an $M/G/1$ queueing system with correlated consecutive job service times. The service time of a bulk job in a such a $M/G/1$ system is proved to be exponentially distributed whenever the previous bulk job leaves only one of the next job siblings in service, and distributed as the maximum of two exponentials whenever the previous bulk job leaves both siblings of the next job in service. Using the probabilities of these occurrences, we were able to provide, in Section 5.5, a rather accurate approximation of the job average response time. In Section 5.6 however, we considered a load adjustable bulk service time distribution, namely that the bulk job service time depends on the current system load. Using the load adjustable distribution and the $M/G/1$ theory, we obtained a very good approximation of the job average response time. This latter approximation being simple and yet very accurate, shall be used in the next chapter, where we investigate and analyze more general Fork-Join parallel processing systems.

In Section 5.7, using some inherent properties of the Fork-Join parallel processing system, we pursue to study some modified $M/G/1$ queueing systems. In Section 5.7.1, we studied a modified $M/G/1$ queueing system obtained from another pure $M/G/1$ system, in which an additional start-up period is added whenever a busy periods starts in the pure system. In Section 5.7.2 however, we studied a modified $M/G/1$ queueing system derived from another pure $M/G/1$ system, in which an additional finishing-up period is added whenever a busy period ends in the pure system.

For both the modified $M/G/1$ system with start-up periods and the modified $M/G/1$ system with finishing-up periods, we proved a decomposition property stating that the job response time distribution in such queueing systems is composed of the direct sum of the response time in the pure system and the additional delay suffered in the modified system due either to the start-up periods or to the finishing-up periods presence. Results from these modified systems analysis are then used to construct an excellent approximation of the bulk job average response time through the Fork-Join parallel processing system comprising two homogeneous and exponential processors.

In Section 5.8, we returned to the more general case of $P > 2$ processors, and provided a very good approximation of the bulk job average response time. This approximation is based on a scaling technique stating that the lower bound, the upper bound and the average response time itself grow, as a function of the number of processors P , at the same rate. Simulations were used to validate the accuracy of such an approximation.

CHAPTER 6

PARALLEL PROCESSING WITH SYNCHRONIZATION CONSTRAINTS AND JOB FEEDBACK

In the previous chapter, we investigated ways to analyze the performance of models of parallel processing systems with P processors, in which a job, upon arrival, subdivides into exactly P tasks, the i th of which must be attended by the i th processor. The process graph of such a bulk job was composed of a bulk arrival of P concurrent (i.e., independent) tasks; namely a $PG(P,1)$. As soon as all the P tasks constituting the bulk job were serviced, the bulk job was immediately and instantaneously recomposed and departed the system at once. In this chapter, we investigate the performance of the same model of parallel processing systems with P processors and synchronization constraints, but we allow the bulk jobs to have a more general process graph. Jobs arrive to the Fork-Join parallel processing system $FJ-P-M/M/1$ according to a Poisson process. Each job consists of a set of M stages which must be performed in a specified order. A job is thus represented by a Directed Acyclic Graph, called hereafter a *stage graph*. A node in the stage graph represents a given stage of the job, an edge (i,j) between node i and node j denotes the precedence relationship between stage i and stage j , and a stage is composed of P concurrent tasks, the i th of which must be executed by the i th processor. Edge (i,j) is used to prevent the start of stage j execution unless stage i execution has been completed. Any number of stages may be executed concurrently if and only if, every predecessor of any one stage does not include any of the other stage. Moreover, stages have prescribed, but arbitrary, priority levels. All P tasks forming a given stage are assigned the same priority level, that of their stage. When a job enters the $FJ-P-M/M/1$ parallel processing system, its first stage has a ready for service status. Upon completion of any stage, its successor stages (if any) immediately achieve the ready-for-service status. Among the ready-for-service tasks at any one of the P processors, next service is provided to the task in the highest priority level in the order in which they achieved the ready-for-service status. The task service time is assumed to be exponentially distributed. A job is considered completed when all its stages are executed. The most important performance measure for the $FJ-2-M/M/1$ parallel processing system with jobs having an arbitrary stage graph, is the job sojourn time defined as the difference between the job completion time and the job arrival time to the system.

The intent in this chapter is to devise ways to determine, in an approximate way, the average sojourn time of a job in the FJ-2-M/M/1 system where, at any one of the processors, service is allocated to the ready-for-service tasks on an FCFS basis. The methodology used, however, could be extended to investigate more advanced service disciplines.

From the previous chapter, we have seen that the FJ-2-M/M/1 system with jobs having a PG(2,1) as a process graph, can be analyzed by an $M/G_c/1$ queueing system representation. For the current system, we shall provide analogous Propositions to Proposition 4.1 and Proposition 4.2, and thus our approach is to represent the FJ-2-M/M/1 parallel processing system with jobs having an arbitrary stage graph as an $M/G_c/1$ queueing system where the service time of consecutive stages are correlated. We shall approximate the service time distribution of a stage in the same manner we did in the previous chapter in Section 4.6; namely by using a load adjustable service time distribution. In the study of the $M/G/1$ queueing system, we shall allow the stages to have prescribed but arbitrary priority levels. The approach taken to study such an $M/G/1$ queueing system with jobs having an arbitrary stage graph draws upon some basic results from renewal theory and queueing theory. This approach is primarily motivated by the work of Wolff [Wol82], in which he proved the long-suspected fact that a Poisson arrival views the system exactly as a random observer would*, and by the work of Cobham [Cobh54], in which he analyzed a particular nonpreemptive priority queueing system with Poisson arrivals and general service times. In [Cobh54], the author obtained a system of linear equations for the mean waiting time of each priority level that could be solved in closed form. Priority queues have been extensively studied in the literature. A comprehensive treatment of some of the earlier work in this area, including references to the most important papers, is given in Jaiswal's book [Jais68]. The survey paper by Kobayashi and Konheim [Koba77] exhaustively reviews the queueing models that have been used in the design of communication systems. Gay and Seaman [Gay75] introduced slight modifications and extensions to Cobham's paper [Cobh54], to allow for preemptive and nonpreemptive priorities. A good discussion of Cobham's technique and a list of references can be found in Kleinrock's book [Klei76]. In a more recent work, Daigle and Houstis [Daig81] and Simon [Simo84] provided an average analysis of a task oriented multipriority queueing system where jobs arrive at the system according to a Poisson process, and each job consists of a set of tasks to be accomplished in a prescribed order.

Although a chain is a special case of a stage graph, we distinguish the two cases, and shall study the case of a chain shaped stage graph first. This study will then serve as a basis for the analysis of the more general case of an arbitrary stage graph. As mentioned earlier, our methodology in studying the FJ-2-M/M/1 system with an arbitrary stage graph, is to represent it as an $M/G_c/1$ queueing system. This $M/G_c/1$ queueing system is converted to a pure $M/G/1$

* This property is usually called Poisson Arrivals See Time Averages (PASTA). Wolff [Wol82] proved this property under a basic *lack of anticipation assumption*, which says that the process being observed (e.g., the state of a queueing system) cannot anticipate the future jumps of the Poisson process.

system by using a load adjustable service time distribution. With respect to the pure M/G/1 system, the results of this chapter present a generalization of the work on nonpreemptive M/G/1 queueing systems with priorities as described above. We also present simulation results to validate the model.

6.1 Model Description

We consider an FJ-2-M/M/1 parallel processing system with 2 identical processors and jobs represented by an arbitrary stage graph with M nodes (i.e., stages). Let $f(i)$, $i=1, \dots, M$ be the priority level of stage of type i (i.e., stage number i) in the sense that higher priority levels are served first. Without loss of generality, we assume that the stages are numbered in an increasing order level by level, such that the nodes on any given level of the graph are numbered in an increasing order from the left to the right. Let $CHILD[i]$, $i=1, \dots, M$, be the binary column vector identifying the immediate descendents (i.e., the children) of node i in the tree stage graph, such that:

$$CHILD[i, k] = \begin{cases} 1 & \text{if stage } k \text{ is a child of stage } i & i=1, \dots, M \\ 0 & \text{otherwise} & k=1, \dots, M \end{cases}$$

Let stage l be a child of stage i (i.e., $CHILD[i, l] = 1$), we define by $LCHILD[i, l]$ the binary column vector identifying the children of stage i that are to the left of stage l in the tree stage graph. Similarly, let $RCHILD[i, l]$ denote the binary column vector identifying the children of stage i that are to the right of stage l in the tree stage graph. Let e_l be the column vector having its l th component equal to 1, and all the other components set to zero. From the above definitions, we readily have:

$$CHILD[i] = LCHILD[i, l] + RCHILD[i, l] + e_l \quad \forall l, \text{ s.t. } CHILD[i, l] = 1$$

Without loss of generality, and in addition to our ordering of the stages, we assume that for all i , n and l such that $CHILD[i, n] = 1$ and $CHILD[i, l] = 1$, we have:

- if $f(n) < f(l)$ then $n > l$, and hence upon the completion of stage i execution, stage l acquires the ready-for-service status first.
- if $f(n) > f(l)$ then $n < l$, and hence upon the completion of stage i execution, stage n acquires the ready-for-service status first.
- if $f(n) = f(l)$ then $n < l$ if stage n is to acquire the ready-for-service first upon the completion of stage i execution.

Consequently, upon the completion of a stage i execution, its children achieve the ready-for-service status with the lowest type and the highest priority level first.

Since the FJ-2-M/M/1 system is converted to a single server $M/G_C/1$ queueing system, we shall first transform our stage graph into a tree shaped stage graph. This is stated in the following Proposition; its proof on the other hand states how such a transformation is accomplished.

Proposition 6.1

An arbitrary stage graph, with nodes numbered in the manner described above, can always be converted into an equivalent tree shaped stage graph with the same precedence relationships among the stages.

Proof

Take any node in the stage graph, say node i , that has two or more incoming edges. Let Ψ_i denote the set of the source nodes of these edges. That is:

$$\Psi_i = \left\{ j \mid 1 \leq j \leq i-1 \text{ and } (j,i) \text{ is an edge in the stage graph} \right\}$$

Let k be the highest number in the set Ψ_i . Eliminate all edges (j,i) from the stage graph, where $j \in \Psi_i - \{k\}$. Once this is done for all stages having two or more incoming edges in the stage graph, we obtain a tree shaped stage graph. Moreover, this resulting tree shaped stage graph assumes the same precedence relationships among the stages, in the sense that the stages acquire the ready-for-service status in the same exact order as they did in the original stage graph.

■

In the sequel, we consider that a preliminary transformation of the stage graph into a tree shaped stage graph is already performed. Jobs arrive to the system according to a Poisson process with aggregate rate λ , and the service time of a task at any one of the processors is exponentially distributed with average $\frac{1}{\mu}$. Let $\lambda_i, i=1, \dots, M$ denote the average input rate of stages of type i to the system; hence $\lambda_i = \lambda$ for $i=1, \dots, M$. Let $\rho_i, i=1, \dots, M$ denote the utilization factor of a processor due to stages of type i , namely $\rho_i = \frac{\lambda_i}{\mu}$ for $i=1, \dots, M$. The stability of our parallel processing system is maintained as long as each one of the processors composing it is stable. Let $\rho, = \sum_{i=1}^M \rho_i = \frac{\lambda M}{\mu}$ denote the total utilization factor of a processor. The stability condition of any one of the processors, and consequently of the whole system, is then $\rho, < 1$. The node (i.e., stages) of the stage graph are numbered in an increasing order, level by level, such that the nodes on any given level are numbered in an increasing order from the left to the right. Figure 6.1:(a) depicts a chain shaped stage graph with $M=5$ and Figures 6.1:(b) and 6.1:(c)

depict two samples of tree shaped stage graphs with $M=5$. The circles represent the stages, and the integers to their sides denote their type numbers. Each circle is composed of two tasks, each of which is to be allocated to a different processor.

Although a chain is a special case of a tree, we shall distinguish the two cases and study the chain shaped stage graph case first. Let $T_i, i=1, \dots, M$ denote the expected total amount of time a stage of type i spends in the FJ-2-M/M/1-system; namely the time from when it enters the queues until it completes service. We shall then find $T_i, i=1, \dots, M$, and then deduce the expected sojourn time $T(2)$ of a job in the system (i.e., the expected total time spent in the system).

A chain shaped stage graph with M stages is hereafter regarded as a job that requires service M times. Upon arrival of such a job to the system, its first stage is ready-for-service and thus is immediately split into two tasks each of which is allocated to a different processor. Upon the join of these tasks (i.e., at the time both tasks are completed), the stage is immediately and instantaneously recomposed and fed back to the queues. This stage is then immediately split into two tasks again each of which must be served by a different processor. In this manner, we can consider the job as feeding back $(M-1)$ times before completion. The time of the i th feedback of a given job is the time at which its $(i+1)$ st stage achieves the ready-for-service status. It follows that when a chain shaped stage graph job arrives to the system, its priority level is $f(1)$, the priority of its first stage, and when it feeds back the i th time, its priority is $f(i+1)$, the priority of its $(i+1)$ st stage. The service discipline at any one of the processors is a nonpreemptive priority discipline in the sense that the next service is given to the task in the highest priority group in the order in which they achieved the ready-for-service status (i.e., entered the queue).

A tree shaped stage graph with M stages can also be regarded as a job that requires service M times. Upon arrival of such a job to the system, its first stage (i.e., the root of its tree) is ready-for-service and thus immediately and instantaneously splits into two tasks, each of which is allocated to a different processor. Upon the completion of both tasks, the stage is immediately recomposed and fed back to the queues as several new stages, each with a possibly different priority level. In order to avoid the confusing situation where a stage feeds back several stages of the same priority level, we have assumed that the lowest numbered stage is scheduled for service first. This is the case where in Figure 6.1:(b), stage number 2 and stage number 3 have the same assigned priority level, that is $f(2)=f(3)$. In this case, we assume that stage number 2 will be scheduled for service before stage number 3 at both processors. If, for such a tree shaped stage graph, we prefer that stage number 3 is to be scheduled first (i.e., achieves the ready-for-service status first), then we represent the graph as depicted in Figure 6.1:(c).

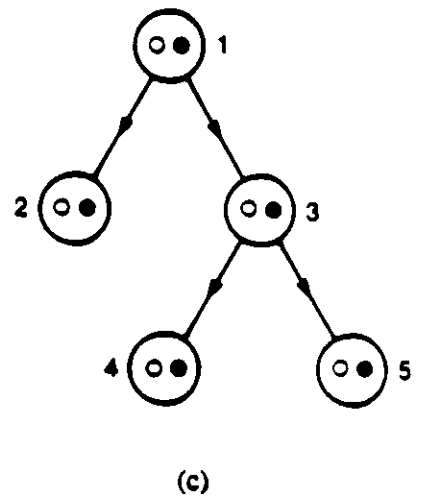
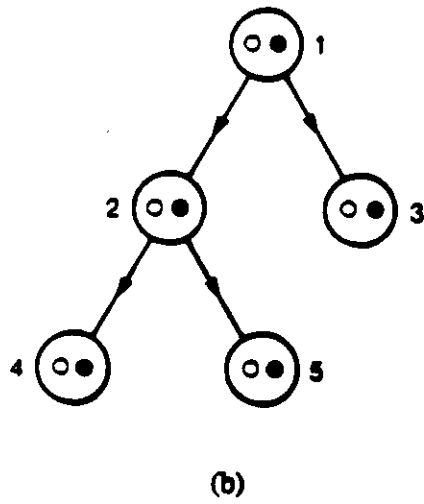


Figure 6.1: Examples of Chain and Tree Shaped Stage Graphs

Throughout this chapter, we assume that a stage feedback is instantaneous and performed before the start of the service of the next stage to be served. At stage feedback instants (i.e., arrival instants of stages of type j , $j=2,\dots,M$) the state of the system is thus fully described by the number of stages of each type present in the queue. On the other hand, the state of the system at type 1 stage arrivals has a distribution which is the same as the steady state distribution since such arrivals are Poisson.

Let $T_0(2)$ denote the expected sojourn time of a job in an empty FJ-2-M/M/1 system. In the case of a chain shaped stage graph, it is easy to see that $T_0(2) = \frac{3M}{2\mu}$ since each one of the M stages takes an average service time of $\frac{3}{2\mu}$. The case of an arbitrary tree shaped stage graph is a bit more complicated. Figure 6.2 sketches the computation of $T_0(2)$ for the tree shaped stage graph of Figure 6.1:(b), assuming the same priority level for all stages. Upon the arrival of a job, its first stage (i.e., stage number 1) is ready-for-service and thus immediately and instantaneously splits into two tasks, each of which is allocated to a different processor. This first stage takes an average service time equal to $\frac{3}{2\mu}$, and then feeds back stage number 2 and stage number 3. After an average amount of time equal to $\frac{1}{2\mu}$, one of the tasks composing stage number 2 completes service, and after another average amount of time equal to $\frac{1}{2\mu}$ another task finishes. At this point, if stage number 2 is completed (which happens with probability $\frac{1}{2}$), it feeds back stage number 4 and stage number 5, otherwise the just completed task belongs to stage number 3 and hence we must wait another average amount of time equal to $\frac{1}{\mu}$ to complete stage number 2. This scenario continues in the same manner until the whole job is completed. Figure 6.3 represents a similar sketch for the computation of $T_0(2)$ for the case of the tree shaped stage graph depicted in Figure 6.1:(c), assuming the same priority level for all the stages. In Figures 6.2 and 6.3, we represent the state of the system by dots which represent the number of tasks queued at each processor, without depicting on which processor these tasks will be processed. The number to the side of the system states denotes the average amount of time that we spend in such a state. For states with two outgoing edges, the probability of following a given edge is $\frac{1}{2}$. From Figure 6.2, we observe that for the tree shaped stage graph of Figure 6.1:(b), we have $T_0(2) = 6.625$ and from Figure 6.3, we observe that for the tree shaped stage graph of Figure 6.1:(c), we obtain $T_0(2) = 7$. Notice that the two tree shaped stage graphs differ only in the way in which the stages achieve the ready-for-service status. The tree shaped stage graph of Figure 6.1:(c) gives a larger $T_0(2)$ since in that graph, stage number 2 achieves the ready-for-service status before stage number 3 does, and stage number 2 possesses two descendants.

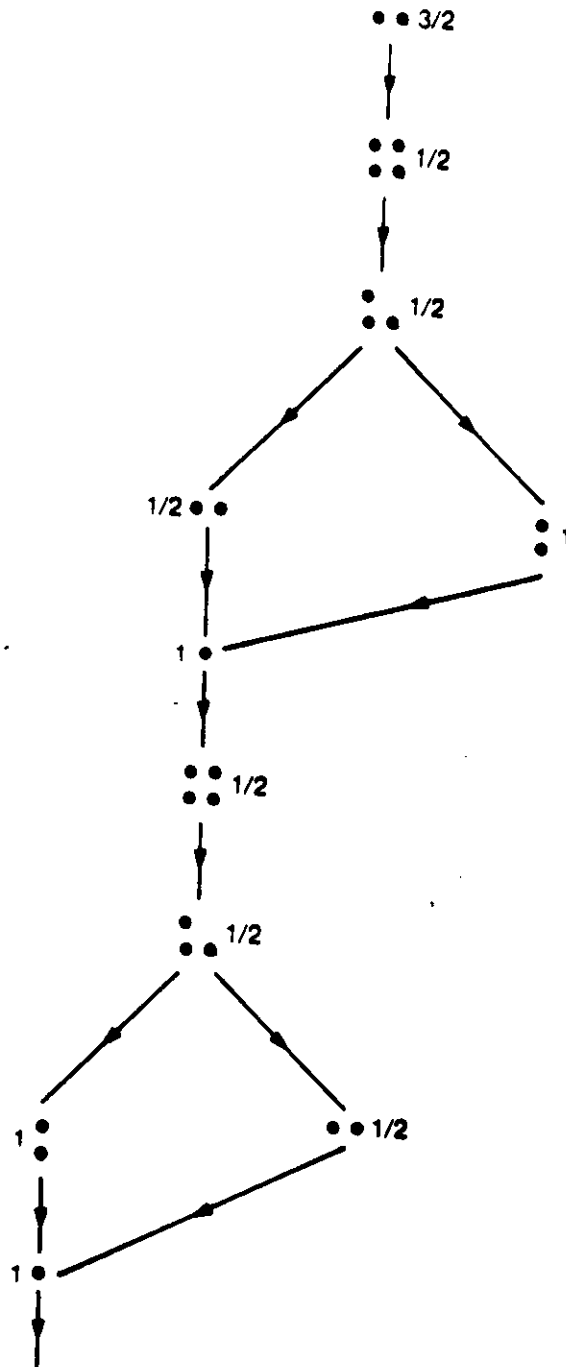


Figure 6.3: Computation of the Expected Sojourn Time in an Empty System for the Tree Shaped Stage Graph of Figure 6.1:(c)

Proposition 6.2

For any given priority assignment to the M stages, jobs depart the system in the same order in which they arrived.

|||

Proposition 6.3

A stage departure from the system leaves all unserved siblings (i.e., one or the two siblings) of the next stage to be served (if any) in service.

|||

The proofs of the above Propositions are omitted due to their similarity to the proofs of Proposition 5.1 and Proposition 5.2 of the previous chapter. Propositions 6.2 and 6.3 motivate the analysis of the FJ-2-M/M/1 parallel processing system with jobs having an arbitrary stage graph as an $M/G_c/1$ system with job feedback. In fact, the job arrival process is Poisson, and the service time of a stage is the needed time to complete the service of its remaining siblings. However, as stated in the previous chapter, the service time of consecutive serviced stages are correlated. We shall use our approximation technique developed in Section 4.6 of the previous chapter to approximately analyze the parallel processing system at hand. The study of the equivalent $M/G_c/1$ system is accomplished in two steps. The first step is the study of the $M/G/1$ queueing system with feedbacks dictated by an arbitrary stage graph, and the second step is the characterization of the stage service time.

The analysis of the $M/G/1$ system with tree feedback is based upon the following facts:

1. A Poisson arrival views the system exactly as a random observer would; this is the PASTA property [Wolf82],
2. There is a simple relationship between average queue sizes and average waiting times in the steady state, namely Little's Theorem [Litt61],
3. The expected number of stages of each type at the points in time where some stage feeds back to the system can be obtained by a linear transformation of the steady state expected number of stages of each type, and
4. The expected total time spent in the system by a stage entering the system exogenously (i.e., a stage of type 1) or endogenously (i.e., stage of type i , $i=2,\dots,M$) is linear in the number of stages of each type present in the system.

We, therefore, shall obtain a system of linear equations to solve for the steady state expected stages population and system times, and consequently we will obtain virtually anything of

interest by various linear transformations.

6.2 Chain Shaped Stage Graphs

We consider an FJ-2-M/M/1 parallel processing system and jobs having chain shaped stage graphs with M stages. Jobs are regarded as requiring service M times by feeding back to the system (M-1) times before completion. The time of the *i*th feedback of a given job is thus the time at which its (*i*+1)st stage achieves the ready-for-service status. Recall that $T_i, i=1, \dots, M$ is the expected total amount of time stage *i* spends in the FJ-2-M/M/1 system, namely the time from when it enters the queues (i.e., it achieves the ready-for-service status) until it completes service. The expected sojourn time (e.g., the expected total time spent in the system) of a job denoted hereafter by $T(2)$, will be:

$$T(2) = \sum_{i=1}^M T_i \quad (6.1)$$

Let \bar{X}_i be the average service time of a stage of type *i* in the $M/G_c/1$ queueing system, and t_i be the expected time the server will remain serving a type *i* stage as seen by a Poisson arrival in steady state. Let $D_j(i)$ be the expected total delay that a type *i* stage found in the queue causes our entering type *j* stage. If $f(i) < f(j)$, the type *i* stage does not delay our type *j* stage. If $f(i) \geq f(j)$, our type *j* stage will not only have to wait for *i*'s service, but also for (*i*+1)'s service if $f(i+1) > f(j)$, etc. It follows that the functions $D_j(i), j=1, \dots, M, i=1, \dots, M$ are most easily computed recursively as follows:

$$D_j(i) = \begin{cases} 0 & \text{if } f(i) < f(j) \\ \bar{X}_i + D_j(i+1) & \text{if } f(i) \geq f(j) \end{cases} \quad j=1, \dots, M, i=1, \dots, M \quad (6.2)$$

where

$$D_j(i+1) /_{f(i+1) > f(j)} = \begin{cases} D_j(i+1) & \text{if } f(i+1) > f(j) \\ 0 & \text{if } f(i+1) \leq f(j) \end{cases} \quad j=1, \dots, M, i=1, \dots, M-1$$

and the boundary condition

$$D_j(M+1) = 0 \quad j=1, \dots, M \quad (6.3)$$

Suppose that there are exactly v_i stages of type *i* in the system upon the arrival of a stage of type *j*, $j=1, \dots, M$. The system state vector (v_1, v_2, \dots, v_M) does not include the entering stage. Let $T_j(v_1, \dots, v_M), j=1, \dots, M$ be the expected amount of time a job will spend in the system as a type *j* stage; namely the expected amount of time a stage of type *j* spends in the system. Taking into account the previously mentioned facts on which our analysis is based, we write

$T_j(v_1, \dots, v_M)$ for $j=2, \dots, M$ as a sum of three terms:

1. The total expected time waiting for higher and equal priority stages already in the system upon our type j stage arrival or feedback, denoted by $T_j^1(v_1, \dots, v_M)$
2. The total expected time spent waiting for higher priority stages that arrive while our type j stage waits, denoted by $T_j^2(v_1, \dots, v_M)$, and
3. Our type j stage expected service time, namely \bar{X}_j .

Since an arbitrary type 1 stage finds the server in the middle of serving some stage (unless the system is empty), the computation of $T_1(v_1, \dots, v_M)$ must, in addition to the above three terms, account for the waiting time incurred by the stage that is being served.

Let $Q = (Q_1, \dots, Q_i, \dots, Q_M)^T$ be the column vector describing the expected system state* as seen by a Poisson arrival; namely as seen by an exogenous stage arrival (i.e., a stage of type 1), where $Q_i, i=1, \dots, M$ denotes the steady state average number of stages of type i in the system. Let $\rho = (\rho_1, \dots, \rho_i, \dots, \rho_M)^T$ be a column vector where $\rho_i, i=1, \dots, M$ denotes the utilization factor of a processor due to stages of type i . Consequently, the column vector $(Q-\rho)$ describes the queue steady state average occupancy. In the sequel, we shall first derive the expected total time spent in the system by a stage conditioned on the state of the system found upon its arrival. Then, we shall derive the average occupancy of the system found by a type i stage, $i=1, \dots, M$, and consequently uncondition to derive the system times in steady state.

6.2.1 Conditional Expected Total Time Spent in the System for Endogenous Stages

In this section, we derive the expected total time spent in the system by a stage that has just fed back (i.e., a stage of type $j, j=2, \dots, M$), conditioned on the the state of the system found at such a point in time. Recall that at feedback times, the state of the system is fully described by the number of stages of each type in the queue. Suppose there are exactly v_i stages of type i in the system (i.e., queued up) when a stage of type j enters. As mentioned earlier, the state of the system at such feedback times does not include our entering type j stage. Using equation (6.2) and equation (6.3), we obtain:

$$T_j^1(v_1, \dots, v_M) = \sum_{f(i) \geq f(j)} v_i D_j(i) \quad j=2, \dots, M \quad (6.4)$$

To compute $T_j^2(v_1, \dots, v_M)$, the total expected time spent in the system by the tagged j stage waiting for higher priority stages that arrive while it is waiting, we need to know the expected number of exogenous stages (i.e., type 1 stages or simply jobs) that arrive during our tagged

* By abuse of notations, we hereunder refer to the expected system state as the system state.

type j stage total waiting time, and consequently their expected contributions to j 's waiting time, as a function of the system state (v_1, \dots, v_M) found by the tagged type j stage upon entering the system. We claim that the expected number of stages of type 1 that arrive during j 's waiting time is $\lambda [T_j(v_1, \dots, v_M) - \bar{X}_j]$ where $[T_j(v_1, \dots, v_M) - \bar{X}_j]$ is j 's expected total waiting time given that the system is found at state (v_1, \dots, v_M) . In the same manner, we claim that the expected total delay these type 1 stages cause our type j stage is $\lambda [T_j(v_1, \dots, v_M) - \bar{X}_j] D_j(1)/f(1) > f(j)$. Appendix (C) provides a proof of the above claims. Therefore, we can write

$$T_j^2(v_1, \dots, v_M) = \lambda [T_j(v_1, \dots, v_M) - \bar{X}_j] D_j(1)/f(1) > f(j) \quad j=2, \dots, M \quad (6.5)$$

Using equations (6.4) and (6.5), we get:

$$T_j(v_1, \dots, v_M) = \sum_{f(i) \geq f(j)} v_i D_j(i) + \lambda [T_j(v_1, \dots, v_M) - \bar{X}_j] D_j(1)/f(1) > f(j) + \bar{X}_j \quad j=2, \dots, M$$

whose solution is:

$$T_j(v_1, \dots, v_M) [1 - \lambda D_j(1)/f(1) > f(j)] = \sum_{f(i) \geq f(j)} v_i D_j(i) - \lambda \bar{X}_j D_j(1)/f(1) > f(j) + \bar{X}_j$$

equivalently:

$$T_j(v_1, \dots, v_M) = \frac{\sum_{f(i) \geq f(j)} v_i D_j(i)}{1 - \lambda D_j(1)/f(1) > f(j)} + \bar{X}_j \quad j=2, \dots, M \quad (6.6)$$

It is interesting to notice that the expected total time spent in the system by a type j stage, $j=2, \dots, M$, is linear in the v_i 's. Thus $T_j(v_1, \dots, v_M)$ depends only on their means. Therefore, by a state of the system at stage feedback instants, we shall mean a column vector $S = (s_1, \dots, s_i, \dots, s_M)^T$ where $s_i, i=1, \dots, M$ is the expected number of stages of type i found in the queue. Such a vector is indeed a sufficient specification of the system state for our purposes as indicated by equation (6.6).

Let $S_j = (s_1^j, \dots, s_i^j, \dots, s_M^j)^T$ be a column vector where $s_i^j, i=1, \dots, M$, is the expected number of stages of type i found in the system when stage j enters as distinguished from Q which is the column vector describing the expected system state as seen by a Poisson arrival. From equation (6.6), we can write:

$$T_j(S_j) = F_j \cdot S_j + r_j \quad j=2, \dots, M \quad (6.7)$$

where F_j is the row vector whose components are:

$$F_j(i) = \begin{cases} \frac{D_j(i)}{1 - \lambda D_j(1)/f(1) > f(i)} & f(i) \geq f(j) \\ 0 & \text{otherwise} \end{cases} \quad \begin{matrix} j=2, \dots, M \\ i=1, \dots, M \end{matrix} \quad (6.8)$$

and the quantity r_j is defined by the following expression:

$$r_j = \bar{X}_j \quad j=2, \dots, M \quad (6.9)$$

6.2.2 Conditional Expected Total Time in System for Exogenous Stages

In this section, we derive the expected total time spent in the system by a stage of type 1, conditioned on the state of the system seen upon such arrival. An arriving type 1 stage sees the server in the middle of serving some stage unless the system is empty. We can then decompose the expected total time spent in the system by a stage of type 1 into two parts.

$$T_1(S_1) = T_{11}(S_1) + T_{12}(S_1)$$

where $T_{11}(S_1)$ is the expected time spent waiting for the stage that is being served when our type 1 stage enters, and $T_{12}(S_1)$ is the expected value of the rest of its expected total time given the system state S_1 . Since $\rho_i, i=1, \dots, M$ is the steady state probability that the server is serving a stage of type i , we have:

$$T_{11}(S_1) = \sum_{i=1}^M \rho_i \left[t_i + D_1(i+1)/f(i+1) > f(1) \right] \quad (6.10)$$

where t_i is the service residual life of stage i that is being served. On the other hand, and since the type 1 stages arrive to the system according to a Poisson process with aggregate rate λ , the state of the system seen upon such an arrival is thus the steady state average number of stages of each type; namely $S_1 = Q$. Therefore, we obtain:

$$T_{12}(S_1) = \sum_{f(i) \geq f(1)} (Q_i - \rho_i) D_1(i) + \bar{X}_1 \quad (6.11)$$

Using Little's formula [Litt61], namely that $Q = \Lambda T$, where Λ is the $(M \times M)$ diagonal matrix with $\Lambda_{ii} = \lambda_i$, for $i=1, \dots, M$ and T is the column vector $(T_1, \dots, T_i, \dots, T_M)$, we get:

$$T_1(S_1) = T_{11}(S_1) + \sum_{f(i) \geq f(1)} \lambda_i T_i(S_1) D_1(i) - \sum_{f(i) \geq f(1)} \rho_i D_1(i) + \bar{X}_1$$

which can be written in a vector notation as:

$$T_1(S_1) = F_1 \cdot (S_1 - \rho) + r_1 \quad (6.12)$$

where F_1 is the row vector defined by :

$$F_1(i)_{i=1,\dots,M} = \begin{cases} D_1(i) & f(i) \geq f(1) \\ 0 & \text{otherwise} \end{cases} \quad (6.13)$$

and the quantity r_1 is defined by the following expression:

$$r_1 = \bar{X}_1 + T_{11}(S_1) \quad (6.14)$$

6.2.3 Conditional System States at Stage Feedback Times

In order to solve for the T_j , $j=1,\dots,M$, we need to know the state of the system when a stage of type j enters the queue. For stages of type 1, the state of the system is simply the steady state Q . For $j=2,\dots,M$, this is no longer true. Let us define the following indicator functions for $i=1,\dots,M$, $j=1,\dots,M$ and any priority level k :

$$I_i^k(j) = \begin{cases} 1 & \left\{ \begin{array}{l} \text{if a type } j \text{ stage, which is already in the system} \\ \text{when a stage with priority } k \text{ enters the queue,} \\ \text{will be a type } i \text{ stage when the priority } k \text{ stage} \\ \text{completes service} \end{array} \right. \\ 0 & \text{otherwise} \end{cases}$$

which is equivalent to:

$$I_i^k(j) = \begin{cases} 1 & \text{if } i=j \text{ and } f(i) < k \\ 1 & \text{if } \begin{cases} i > j \text{ and} \\ f(j) \geq k \text{ and} \\ f(i) \leq k \text{ and} \\ \forall j < n < i, f(n) > k \end{cases} \quad \begin{array}{l} i=1,\dots,M \\ j=1,\dots,M \end{array} \\ 0 & \text{otherwise} \end{cases} \quad (6.15)$$

and the indicator function

$$I_i^k(0) = \begin{cases} 1 & \text{If a stage of type } i \text{ which enters the system after} \\ & \text{a priority } k \text{ stage, will be a type } i \text{ stage when the} \\ & \text{priority } k \text{ stage completes its service} \\ 0 & \text{otherwise} \end{cases}$$

which is equivalent to:

$$I_i^k(0) = \begin{cases} 1 & \text{if } \begin{cases} f(i) \leq k \text{ and} \\ \forall n < i, f(n) > k \end{cases} \\ 0 & \text{otherwise} \end{cases} \quad i=2, \dots, M \quad (6.16)$$

and for the special case of $i=1$, we have:

$$I_1^k(0) = \begin{cases} 1 & f(1) \leq k \\ 0 & f(1) > k \end{cases} \quad (6.17)$$

Say a stage of type j , $j=1, \dots, M$, enters the queue and sees the system in state $S_j = (s_1^j, \dots, s_i^j, \dots, s_M^j)^T$, where s_i^j , $i=1, \dots, M$, is the average number of stages of type i found in the system by such a type j stage. What will the the system state $Y_j = (y_1^j, \dots, y_i^j, \dots, y_M^j)^T$ when its service completes, where y_i^j , $i=1, \dots, M$ is the average number of stages of type i left in the queue when this type j stage finishes its service. We shall distinguish two cases; namely the case of exogenous stages (i.e., $j=1$) and the case of endogenous stages (i.e., $j=2, \dots, M$).

6.2.3.1 Conditional System States at Completion Times of Endogenous Stages

In this section, we proceed to evaluate the system state at the completion time of an endogenous stage, that is at the completion time of a stage of type j , $j=2, \dots, M$, conditioned on the state of the system found when this stage type entered the queue (i.e., fed back into the queue). Using the above defined indicator functions, we can express y_i^j , $i=1, \dots, M$ in terms of the vector S_j as:

$$y_i^j = \sum_{k=1}^i I_i^{(j)}(k) s_k^j + \lambda_1 \left[T_j(S_j) - \bar{X}_j \right] I_i^{(j)}(0) + \lambda_1 \bar{X}_j / \mu_i \quad i=1, \dots, M \quad (6.18)$$

Here, the first part of the right hand side represents the tracking of all those stages found in the system (when our type j stage entered) which will be stages of type i when our type j stage

completes its service. The second part of the right hand side represents the tracking of the new exogenous arrivals while our type j stage waits, and which will be type i stages when our type j stage completes service. Finally, the third part of the right hand side of the above equation represents the number of type 1 stages that arrive to the system while our type j stage is being served. This third part is non null only for $i=1$, that is

$$\lambda_1 \bar{X}_{j/i=i=1} = \begin{cases} \lambda_1 \bar{X}_j & i=1 \\ 0 & \text{otherwise} \end{cases}$$

Using equation (6.7), equation (6.18) yields:

$$y_i^j = \sum_{k=1}^i I_i^{f(j)}(k) s_k^j + \lambda_1 F_j S_j I_i^{f(j)}(0) + \lambda_1 \bar{X}_{j/i=i=1} \quad i=1, \dots, M$$

Now, since $I_i^{f(j)}(k)=0$ for the values $k=i+1, \dots, M$, we may write the system state vector Y_j using vector and matrix notation as:

$$Y_j = A_j \cdot S_j + B_j \quad j=2, \dots, M \quad (6.19)$$

where A_j is an $(M \times M)$ matrix defined by:

$$A_j(i, k) = \begin{cases} I_i^{f(j)}(k) + \lambda_1 F_j(k) I_i^{f(j)}(0) & \text{if } f(i) \leq f(j) \\ 0 & \text{otherwise} \end{cases}$$

and since for $f(i) > f(j)$, we have $I_i^{f(j)}(k) = I_i^{f(j)}(0) = 0$ for all $j=2, \dots, M$, $i=1, \dots, M$ and $k=1, \dots, M$, we get:

$$A_j(i, k) = I_i^{f(j)}(k) + \lambda_1 F_j(k) I_i^{f(j)}(0) \quad \begin{matrix} j=2, \dots, M \\ i=1, \dots, M \\ k=1, \dots, M \end{matrix} \quad (6.20)$$

B_j is the column vector with its i th component defined by:

$$B_j(i) = \begin{cases} \lambda_1 r_j & \text{if } i=1 \\ 0 & \text{if } i=2, \dots, M \end{cases} \quad j=2, \dots, M \quad (6.21)$$

6.2.3.2 Conditional System States at Completion Times of Exogenous Stages

Since the job arrival process is Poisson, it follows that $S_1 = Q = \Lambda T$, and that with probability p_i such an exogenous arrival finds the server in the middle of serving a stage of type i , $i=1, \dots, M$. Let $Y_1 = (y_1^1, \dots, y_1^M)^T$ be the state of the system upon our type 1 stage service completion, conditioned on the system state S_1 . The evaluation of the column vector Y_1 follows the same steps of the previous section with the additional term for the tracking of the stage being served at the time of our type 1 stage arrival. Thus we have:

$$y_i^1 = \sum_{k=1}^i I_i^{f(1)}(k)(s_k^1 - \rho_k) + \lambda_1 \left[T_1(S_1) - \bar{X}_1 \right] I_i^{f(1)}(0) + \lambda_1 \bar{X}_1 / \text{if } i=1 \quad (6.22)$$

$$+ \sum_{k=1}^{i-1} \rho_k I_i^{f(1)}(k+1) / \left\{ [f(k+1) > f(1)] \text{ or } [k+1=i \text{ and } f(k+1) \leq f(1)] \right\} \quad i=1, \dots, M$$

The fourth term of the right hand side of the above equation represents the tracking of the stage being served upon our exogenous stage arrival, and can be rewritten as:

$$\sum_{k=1}^{i-1} \rho_k \left\{ I_i^{f(1)}(k+1) / \text{if } f(k+1) > f(1) + 1 / \text{if } k+1=i \text{ and } f(k+1) \leq f(1) \right\}$$

Now, by using equation (6.17), we may write equation (6.22) as:

$$y_i^1 = \sum_{k=1}^i I_i^{f(1)}(k)(s_k^1 - \rho_k) + \lambda_1 T_1(S_1) I_i^{f(1)}(0)$$

$$+ \sum_{k=1}^{i-1} \rho_k \left\{ I_i^{f(1)}(k+1) / \text{if } f(k+1) > f(1) + 1 / \text{if } k+1=i \text{ and } f(k+1) \leq f(1) \right\} \quad i=1, \dots, M$$

which, by using equation (6.12) yields:

$$y_i^1 = \sum_{k=1}^i I_i^{f(1)}(k)(s_k^1 - \rho_k) + \lambda_1 F_1 S_1 I_i^{f(1)}(0) + \lambda_1 r_1 I_i^{f(1)}(0)$$

$$+ \sum_{k=1}^{i-1} \rho_k \left\{ I_i^{f(1)}(k+1) / \text{if } f(k+1) > f(1) + 1 / \text{if } k+1=i \text{ and } f(k+1) \leq f(1) \right\} \quad i=1, \dots, M$$

Finally, since $S_1 = Q$ and $I_i^{f(1)}(k) = 0$ for $k=i+1, \dots, M$, and since for all $i=1, \dots, M$ and $k=1, \dots, M$, we have $I_i^{f(1)}(k) = I_i^{f(1)}(0) = 0$ if $f(i) > f(1)$, we can write the column vector Y_1 using vector and matrix notation as:

$$Y_1 = A_1 \cdot (Q - \rho) + B_1 + B \quad (6.23)$$

where A_1 is an $(M \times M)$ matrix defined by:

$$A_1(i, k) = I_i^{f(1)}(k) + \lambda_1 F_1(k) I_i^{f(1)}(0) \quad \begin{matrix} i=1, \dots, M \\ k=1, \dots, M \end{matrix} \quad (6.24)$$

and B_1 is a column vector with its i th component defined by:

$$B_1(i) = \begin{cases} \lambda_1 r_1 & \text{if } i=1 \\ 0 & \text{if } i=2, \dots, M \end{cases} \quad (6.25)$$

and B is a column vector with its i th component defined by:

$$B_i^{\dagger} = \sum_{k=1}^{i-1} \rho_k \left\{ I_i^{f(1)}(k+1) /_{if f(k+1) > f(1)} + 1 /_{if k+1 = \text{and } f(k+1) \leq f(1)} \right\} \quad i=1, \dots, M \quad (6.26)$$

6.2.4 System States at Service Completion Times

Equations (6.19) and (6.23) provide explicit expressions for the state of the system at service completion times of a stage of type j , $j=1, \dots, M$, conditioned on the state of the system found by such a type j stage upon its entering the queue. Equation (6.23) says that when a type 1 stage feeds back to the queue as a type 2 stage, the state of the system is:

$$Y_1 = A_1(Q - \rho) + B_1 + B_1^{\dagger}$$

This is then the state of the queue found by an entering type 2 stage, that is $S_2 = Y_1$. When such a type 2 stage completes service, and then feeds back as a type 3 stage, the state of the system is obtained using equation (6.19); namely:

$$Y_2 = A_2 Y_1 + B_2$$

This is then the state of the queue found by an entering type 3 stage; that is $S_3 = Y_2$. In the same manner, we have for any $l \geq 2$:

$$S_l = \left[\prod_{i=1}^{l-1} A_i \right] (Q - \rho) + \left[\prod_{i=2}^{l-1} A_i \right] B_1^{\dagger} + \sum_{k=1}^{l-1} \left[\prod_{j=k+1}^{l-1} A_j \right] B_k \quad l=1, \dots, M \quad (6.27)$$

where the matrix product is taken by definition to be:

$$\prod_{i=k}^n A_i \triangleq \begin{cases} A_k A_{k-1} \cdots A_n & n \geq k \\ I & k = n+1 \\ 0 & k \geq n+2 \end{cases}$$

where I is the identity matrix. The state of the system found by an entering type l , $l=2, \dots, M$ is $S_l = Y_{l-1}$, and $S_1 = Q$ the steady state average number of stages of each type in the system.

6.2.5 Steady State Equations for the Expected Total Time in System

In Sections 6.2.1 and 6.2.2, we formulated explicit expressions for the expected total time spent in the system respectively by endogenous stages (i.e., stages of type j , $j=2, \dots, M$) and exogenous stages (i.e., stages of type 1) conditioned on the state of the system found upon arrival. We now proceed to uncondition and evaluate the expected system times in steady state. Since in the steady state we have $Q = \Lambda T$, then using equations (6.7) and (6.12) along with equation (6.27) and the fact that $S_l = Y_{l-1}$ for $l=2, \dots, M$ and $S_1 = Q$, we get the following system

of linear equations for the expected total time spent in the system by each stage type $j, j=1, \dots, M$.

$$T_j = F_j \left[\prod_{i=1}^{j-1} A_i \right] \lambda T - F_j \left[\prod_{i=1}^{j-1} A_i \right] \rho + F_j \left[\prod_{i=2}^{j-1} A_i \right] B_1 + F_j \sum_{k=1}^{j-1} \left[\prod_{i=k+1}^{j-1} A_i \right] B_k + r_j \quad (6.28)$$

Equation (6.28) gives M linear equations for the expected total time spent in the system by each stage type T_1, T_2, \dots, T_M . After solving this linear system, we compute the expected sojourn time of a job in the parallel processing system FI-2-M/M/1 with jobs having a chain shaped stage graph by using equation (6.1), and the appropriate expressions for \bar{X}_i and $t_i, i=1, \dots, M$ defined in the following section. The following are examples to demonstrate the use of the above established formulae for the computation of the expected total time spent in a single server system.

Example 1

Consider an M/G/1 queueing system with jobs having a single stage process graph; that is $M=1$. From equation (6.28), we get:

$$T_1 = F_1 \lambda T_1 - F_1 \rho + r_1$$

which gives:

$$T_1 = \frac{r_1 - F_1 \rho}{1 - \lambda F_1}$$

From equation (6.8), we get $F_1 = D_1(1) = \bar{X}$, and from equation (6.15), we get $r_1 = \bar{X} + \rho t_1 = \bar{X} + \lambda \frac{\bar{X}^2}{2}$. Putting all the terms together, we obtain:

$$T_1 = \frac{\lambda \bar{X}^2}{2(1-\rho)} + \bar{X}$$

The first term of the right hand side of the above expression denotes the average waiting time, and is the well known (P-K) formula for the mean waiting time in an M/G/1 queueing system [Klei75].

Example 2

Consider an M/G/1 system with jobs having a stage graph composed of a chain of two consecutive stages with the first stage (i.e., the stage of type 1) having a priority level $f(1)$ lower than that of the second stage (i.e., the stage of type 2); that is $f(1) < f(2)$. Jobs arrive to the system according to a Poisson process with aggregate rate λ , and the service time of a stage independent of its type, is exponentially distributed with mean $\frac{1}{\mu}$; namely $\bar{X}_1 = \bar{X}_2 = \frac{1}{\mu}$, and

$\rho_1 = \rho_2 = \frac{\lambda}{\mu}$. Using our system of linear equations given by (6.28), we get:

$$\begin{aligned} T_1 &= F_1 \Lambda T - F_1 \rho + r_1 \\ T_2 &= F_2 A_1 \Lambda T - F_2 A_1 \rho + F_2 B_1^1 + F_2 B_1 + r_2 \end{aligned} \quad (6.29)$$

where

$$\Lambda = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}, \quad \rho = \left(\frac{\lambda}{\mu}, \frac{\lambda}{\mu} \right)^T, \quad T = (T_1, T_2)^T, \quad F_1 = \left(\frac{2}{\mu}, \frac{1}{\mu} \right), \quad F_2 = \left(0, \frac{1}{\mu} \right)$$

$$A_1 = \begin{bmatrix} \frac{2\lambda}{\mu} & \frac{\lambda}{\mu} \\ 0 & 0 \end{bmatrix}, \quad r_1 = \frac{1}{\mu} \left(1 + \frac{3\lambda}{\mu} \right), \quad r_2 = \frac{1}{\mu}, \quad B_1 = \left(\frac{\lambda}{\mu} \left(1 + \frac{3\lambda}{\mu} \right), 0 \right)^T, \quad B_1^1 = (0, 0)^T$$

Solving (6.29) yields:

$$T_1 = \frac{\lambda + \mu}{\mu(\mu - 2\lambda)} \quad \text{and} \quad T_2 = \frac{1}{\mu}$$

Finally, using equation (6.1) yields:

$$T(2) = \frac{2\mu - \lambda}{\mu(\mu - 2\lambda)}$$

Notice that this queueing system is equivalent to an M/G/1 queue where the service time distribution is the convolution of two exponentials; namely a two stage Erlang distribution with mean $\frac{2}{\mu}$ and second moment $\frac{6}{\mu^2}$. Using the known (P-K) formula for the mean waiting time in an M/G/1 queueing system [Klei75], yields the exact same result.

Example 3

Consider an M/G/1 system with jobs having a stage graph composed of a chain of two consecutive stages with the first stage (i.e., the stage of type 1) having a priority level $f(1)$ higher than that of the second stage (i.e., the stage of type 2); that is $f(1) > f(2)$. As in the previous example, jobs arrive to the system according to a Poisson process with aggregate rate λ , and the service time of a stage independently of its type is exponentially distributed with mean $\frac{1}{\mu}$; namely $\bar{X}_1 = \bar{X}_2 = \frac{1}{\mu}$, and $\rho_1 = \rho_2 = \frac{\lambda}{\mu}$. Using our system of linear equations given by (6.28), we get:

$$\begin{aligned} T_1 &= F_1 \Lambda T - F_1 \rho + r_1 \\ T_2 &= F_2 A_1 \Lambda T - F_2 A_1 \rho + F_2 B_1^1 + F_2 B_1 + r_2 \end{aligned} \quad (6.30)$$

where

$$\Lambda = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}, \rho = \left(\frac{\lambda}{\mu}, \frac{\lambda}{\mu} \right)^T, T = (T_1, T_2)^T, F_1 = \left(\frac{1}{\mu}, 0 \right), F_2 = \frac{1}{\mu-\lambda} (1, 1)$$

$$A_1 = \begin{bmatrix} \frac{\lambda}{\mu} & 0 \\ 1 & 1 \end{bmatrix}, r_1 = \frac{1}{\mu} \left(1 + \frac{2\lambda}{\mu} \right), r_2 = \frac{1}{\mu}, B_1 = \left(\frac{\lambda}{\mu} \left(1 + \frac{2\lambda}{\mu} \right), 0 \right)^T, B_2 = \left(0, \frac{\lambda}{\mu} \right)^T$$

Solving (6.30) yields:

$$T_1 = \frac{\lambda + \mu}{\mu(\mu - \lambda)} \quad \text{and} \quad T_2 = \frac{(\mu - 2\lambda)^2 + 3\lambda\mu}{\mu(\mu - \lambda)(\mu - 2\lambda)}$$

Finally, using equation (6.1) yields:

$$T(2) = \frac{(\mu - 2\lambda)(2\mu - \lambda) + 3\lambda\mu}{\mu(\mu - \lambda)(\mu - 2\lambda)}$$

■

6.2.6 Job Average Sojourn Time

Our main objective is to analyze the FJ-2-M/M/1 system with $f(i)=f(1)$ for all $i=1, \dots, M$. Tasks at any one of the processors are then scheduled for service on an FCFS basis. We represent the service time distribution of a stage in the equivalent $M/G_c/1$ queueing system as a load adjustable distribution. Namely, let:

\bar{Z}_1 = random variable exponentially distributed with mean $\frac{1}{\mu}$,

\bar{Z}_2 = random variable exponentially distributed with mean $\frac{1}{2\mu}$,

\bar{X}_c = random variable representing the service time of a stage in the $M/G_c/1$ system, with mean \bar{X}_c and second moment \bar{X}_c^2 ,

τ = the utilization factor of the $M/G_c/1$ system,

\bar{X}_i = the average service time in the $M/G_c/1$ system of a stage of type i , $i=1, \dots, M$, and

t_i = the expected amount of time the server will remain serving a type i , $i=1, \dots, M$, stage as seen by a Poisson arrival in the steady state.

For our case of $f(i)=f(1)$ for $i=1, \dots, M$, we represent the service time of a stage as a weighted sum of the random variables \bar{Z}_1 and \bar{Z}_2 :

$$\bar{X}_c = \bar{Z}_1 + \alpha(\tau)\bar{Z}_2$$

Here $\alpha(\tau)$ is a weighting function that depends on the utilization factor, τ , of the system. To determine this weighting function, we note that under very light traffic conditions (i.e., $\rho_s \rightarrow 0$ or equivalently $\tau \rightarrow 0$), the service time of a stage, \bar{X}_c , is distributed as the maximum of two independent exponentials of the same mean $\frac{1}{\mu}$, and thus we must have $\lim_{\tau \rightarrow 0} \alpha_{\tau \rightarrow 0}(\tau) = 1$. Under

heavy traffic conditions, namely $\rho \rightarrow 1$, the average service time of a stage, \bar{X}_c , should approach $\frac{1}{\mu}$ since the two stability conditions $\rho < 1$ and $\tau < 1$ are equivalent. We therefore have $\lim_{\alpha \rightarrow 1} \alpha(\tau) = 0$. As in the previous chapter, we consider a first order approximation, namely $\alpha(\tau) = 1 - \tau$. We obtain:

$$\bar{X}_c = \bar{Z}_1 + (1 - \tau)\bar{Z}_2$$

Using the definitions of \bar{Z}_1 and \bar{Z}_2 , we get:

$$\bar{X}_c = \frac{3 - \tau}{2\mu} \quad \text{and} \quad \bar{X}_c^2 = \frac{4 + (1 - \tau)(3 - \tau)}{2\mu^2}$$

and since $\tau = \lambda M \bar{X}_c$, that is :

$$\tau = \frac{3\rho_s}{2 + \rho_s}$$

we get:

$$\bar{X}_c = \frac{3}{\mu(2 + \rho_s)} \quad \text{and} \quad \bar{X}_c^2 = \frac{2}{\mu^2} + \frac{6(1 - \rho_s)}{\mu^2(2 + \rho_s)^2}$$

and finally, for our system with $f(i) = f(1)$, $i = 1, \dots, M$, we have:

$$\bar{X}_i = \bar{X}_c = \frac{3}{\mu(2 + \rho_s)} \quad i = 1, \dots, M \quad (6.31)$$

$$t_i = \frac{\bar{X}_c^2}{2\bar{X}_c} = \frac{5 + \rho_s}{\mu(2 + \rho_s)} \quad i = 1, \dots, M \quad (6.32)$$

Validation of the Approximation

Figure 6.4 depicts the expected sojourn time of the chain shaped stage graph represented in Figure 6.1:(a), as a function of the utilization factor ρ_s . The expected times in system T_1, T_2, T_3, T_4 , and T_5 of the 5 stages composing the chain graph are obtained by solving the system of linear equations given by (6.28), where the stage average service times, \bar{X}_i , $i = 1, \dots, M$ is given by (6.31), and the average residual life of a stage i service time, $i = 1, \dots, M$ are given by (6.32). The expected sojourn time of a job is thus obtained using equation (6.1).

To validate our approximation, namely the representation of the stage service time by a load adjustable distribution, we simulated the FJ-2-M/M/1 parallel processing system for the given chain shaped stage graph. The transient state is eliminated using short independent replications [Lave83]. The method used to estimate the statistic measure $T(2)$ in the steady state is *The Method of Batch Means*. The confidence intervals are set to a 90% level, are generated via the t-distribution, and are represented in Figure 6.4 as vertical bars.

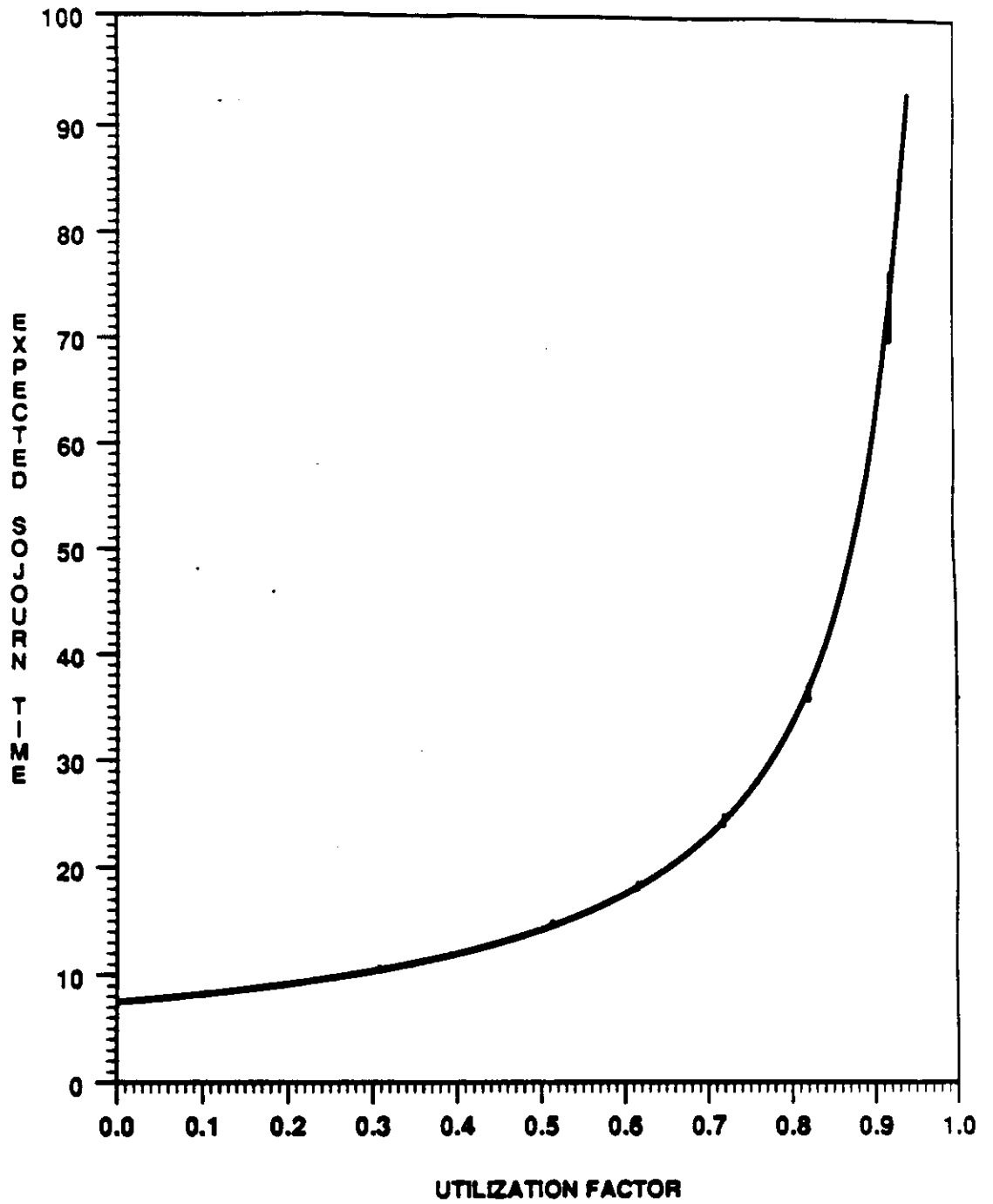


Figure 6.4: Expected Sojourn Time of the Chain Shaped Stage Graph of Figure 6.1:(a)

From Figure 6.4, we observe that the load adjustable distribution approximation is very accurate; indeed as depicted in the figure, and for $\rho_s \in (0,0.7)$, the confidence interval widths are very small indicating the rather good accuracy of the approximation for this range of the utilization factor. For higher values of ρ_s , while the confidence interval width is somewhat larger, the approximation is still within the prescribed accuracy. Recall that high values of ρ_s have the side effect of increasing the variability of the system statistics, and hence increasing the confidence intervals width. The mean relative error of the approximation is less than 1% for $\rho_s \in [0,0.7]$, and less than 3% for $\rho_s \in [0.71,0.95]$.

6.3 Tree Shaped Stage Graph

We now proceed to determine the average sojourn time of a job having a tree shaped stage graph (indeed an arbitrary stage graph as stated through Proposition 6.1) through the FJ-2-M/M/1 parallel processing system. Jobs are regarded as requiring service M times. Upon a job arrival to the system, its first stage is ready-for-service and thus immediately split into two tasks, each of which is directed to a different processor. Upon the completion of both tasks, the stage is immediately recomposed and fed back to the queues as several new stages according to the structure of the tree stage graph.

As in the case of the chain shaped stage graph, and due to Proposition 6.1, Proposition 6.2 and Proposition 6.3, we shall approximate the FJ-2-M/M/1 parallel processing system with jobs having a tree shaped stage graph as an $M/G_c/1$ system with tree feedback. The study of such an equivalent $M/G_c/1$ system is accomplished in two steps. The first step is the analysis of the $M/G/1$ queueing system with tree feedback, and the second step is the characterization of the stage service time. As before, let T_i 's, $i=1, \dots, M$, denote the expected total amount of time a stage of type i spends in the system from when it achieves the ready-for-service status until it completes service, and $T(2)$ denote the expected sojourn time of a job through the FJ-2-M/M/1 system. Unlike the case of the chain shaped stage graph where $T(2)$ is the sum of all the T_i 's as given by equation (6.1), the expected sojourn time is now the sum of only some of the T_i 's. Assuming the same priority level for all the stages, the average sojourn time of a job having the tree shaped stage graph of Figure 6.1:(b) is given by:

$$T(2) = T_1 + T_2 + T_5 \quad (6.33)$$

and the expected sojourn time of a job having the tree shaped stage graph of Figure 6.1:(c) is given by:

$$T(2) = T_1 + T_3 + T_5 \quad (6.34)$$

Let \bar{X}_i , $i=1, \dots, M$ be the average service time of a stage of type i in the equivalent $M/G_c/1$ queueing system, and $D_j(i)$, $i=1, \dots, M$, $j=1, \dots, M$ be the expected total delay that a type i stage found in the queue causes an entering type j stage. Note that if $f(i) < f(j)$, then the type i stage does not delay the entering type j stage. On the other hand, if $f(i) \geq f(j)$, the entering type j stage will not only have to wait for i 's service, but also for the service of i 's children that have a priority level greater than $f(j)$. As in the case of the chain shaped stage graph, the functions $D_j(i)$'s, $i=1, \dots, M$, $j=1, \dots, M$ are most easily computed recursively as follows:

$$D_j(i) = \begin{cases} 0 & \text{if } f(i) < f(j) \\ \bar{X}_i + \sum_{k, s.t. CHILD(i,k)=1} D_j(k) / f(k) > f(j) & \text{if } f(i) \geq f(j) \end{cases} \quad (6.35)$$

and the boundary conditions (6.36)

$$D_j(M+1) = 0 \quad j=1, \dots, M$$

6.3.1 Conditional Expected Total Time in System

Let $S_j = (s_1^j, \dots, s_i^j, \dots, s_M^j)^T$ be the column vector representing the state of the system, such that s_i^j , $i=1, \dots, M$ is the expected number of stages of type i found in the system when stage j enters, $j=1, \dots, M$. The expected total time spent in the system by an endogenous stage that has just fed back, conditioned on the state of the system found at this feedback instant, is derived in the same manner as in Section 6.2.1, where the functions $D_j(i)$'s are now replaced by the ones given in equation (6.35). Namely, the $T_j(S_j)$, $j=1, \dots, M$ are readily given by equations (6.7), (6.8) and (6.9) where the $D_j(i)$'s are defined by equation (6.35). For exogenous stages, an arriving type 1 stage sees the server in the middle of serving some stage unless the system is empty. Similar to the case of the chain shaped stage graph, we decompose the expected total time spent in the system by a stage of type 1 into two parts:

$$T_1(S_1) = T_{11}(S_1) + T_{12}(S_1)$$

where $T_{11}(S_1)$ is the expected amount of time spent waiting for the stage that is being served when our type 1 stage enters, and $T_{12}(S_1)$ is the expected value of the rest of its expected total time given that the system is in the state S_1 . Since the exogenous stages arrive to the system according to a Poisson process, the state of the system seen upon such arrivals is thus the steady state average number of stages of each type; namely $S_1 = Q$. Consequently, $T_{12}(S_1)$ is readily given by equation (6.11) where the functions $D_j(i)$'s are now given by equation (6.35). On the other hand, and since p_i , $i=1, \dots, M$ is the steady state probability that the server is serving a stage of type i , we have:

$$T_{11}(S_1) = \sum_{i=1}^M \rho_i \left[t_i + \sum_{k: s.t. CHILD(i,k)=1} D_1(k)/f(k) > f(i) \right] \quad (6.37)$$

Here $t_i, i=1, \dots, M$ is the service residual life of stage i that is being served upon the exogenous stage arrival. The conditional expected total time spent in the system by an exogenous stage is therefore readily given by equations (6.12), (6.13) and (6.14) where the functions $D_1(i), i=1, \dots, M$ are now given by equation (6.35) and the function $T_{11}(S_1)$ is given by equation (6.37).

6.3.2 Conditional System States at Stage Feedback Times

We now proceed to determine the states of the system at feedback instants. For exogenous stages, the state of the system is simply the steady state $S_1 = Q$. For endogenous stages, this is no longer true. For any given two stages, say i and $j, i=1, \dots, M, j=1, \dots, M$, we define the row vector denoted by $PATH[k][i,j], k=0, \dots, PATH[0][i,j]$, where $PATH[0][i,j]$ denotes the number of stages in the path (i,j) including stage i and stage j . $PATH[k][i,j], k=0, \dots, PATH[0][i,j]$ is thus the k th stage type on the path (i,j) in the tree stage graph. Let also define by $DESC[j], j=1, \dots, M$ the row binary vector identifying the descendents of stage j , such that:

$$DESC[j,i] = \begin{cases} 1 & \text{if stage } i \text{ is a descendent of stage } j \\ 0 & \text{otherwise} \end{cases} \quad i=1, \dots, M$$

In particular, note that $DESC[i,i]=1$. Now, we redefine the indicator functions $I_i^k(j)$ and $I_i^k(0)$ as follows:

$$I_i^k(j) = \begin{cases} 1 & \text{if } i=j \text{ and } f(i) < f(k) \\ 1 & \text{if } \begin{cases} i > j \\ DESC[j,i] = 1 \text{ and} \\ f(j) \geq k \text{ and} \\ f(i) \leq k \text{ and} \\ \forall n \text{ satisfying } (n \neq j, n \neq i \text{ and} \\ \exists m \neq 0, \text{ such that } PATH[m][i,j]=n), f(n) > k \end{cases} \\ 0 & \text{otherwise} \end{cases} \quad \begin{matrix} i=1, \dots, M \\ j=1, \dots, M \end{matrix} \quad (6.38)$$

$$I_i^k(0) = \begin{cases} 1 & \text{if } \begin{cases} i > 1 \text{ and} \\ f(i) \leq k \text{ and} \\ \forall n, \text{ such that } (n \neq i \text{ and } \exists m \neq 0 \\ \text{such that } \text{PATH}[m][1,i]=n), f(k) > n \end{cases} \\ 1 & \text{if } i=1 \text{ and } f(1) \leq k \\ 0 & \text{otherwise} \end{cases} \quad i=1, \dots, M \quad (6.39)$$

Say a stage of type j , $j=1, \dots, M$ enters the queue and sees the system at state $S_j = (s_1^j, \dots, s_i^j, \dots, s_M^j)^T$, where s_i^j , $i=1, \dots, M$ is the average number of stages of type i found in the system by this type j stage. What will be the system state $Y_j = (y_1^j, \dots, y_i^j, \dots, y_M^j)^T$, when its service completes, where y_i^j , $i=1, \dots, M$ is the average number of stages of type i left in the queue when such type j stage finishes its service. We shall find the system states at stage completion instants conditioned on the state of the system found upon arrival to the queue. For endogenous stages, these conditional system states are readily given by equations (6.19), (6.20) and (6.21) where now the indicator functions are defined by equations (6.38) and (6.39) and the functions $D_j(i)$'s are defined by equation (6.35). For the exogenous stages, the additional term for the tracking of the stage being served at the time of the type 1 stage arrival is a bit more complicated.

$$y_i^j = \sum_{k=1}^i I_i^{f(1)}(k)(s_k^j - \rho_k) + \lambda_1 \left[T_1(S_1) - \bar{X}_1 \right] I_i^1(0) + \lambda_1 \bar{X}_1 / \text{if } i=1 \\ + \sum_{k=1}^{i-1} \rho_k \left\{ \left[\sum_{j, \text{ s.t. } \text{CHILD}(k,j)=1} I_i^{f(1)}(j) / f(j) > f(1) \right] + 1 / \text{CHILD}(k,i)=1 \text{ and } f(i) \leq f(1) \right\} \quad (6.40) \\ i=1, \dots, M$$

and since $\forall i \neq 1$ we have $I_i^{f(1)}(0) = 0$, and using equation (6.12), equation (6.40), we get:

$$y_i^j = \sum_{k=1}^i I_i^{f(1)}(k)(s_k^j - \rho_k) + \lambda_1 F_1 S_1 I_i^{f(1)}(0) + \lambda_1 r_1 I_i^{f(1)}(0) \\ + \sum_{k=1}^{i-1} \rho_k \left\{ \left[\sum_{j, \text{ s.t. } \text{CHILD}(k,j)=1} I_i^{f(1)}(j) / f(j) > f(1) \right] + 1 / \text{CHILD}(k,i)=1 \text{ and } f(i) \leq f(1) \right\} \\ i=1, \dots, M$$

where r_1 is defined by equation (6.14) with T_{11} given by equation (6.37). Finally, since $S_1 = Q$ and $I_i^{f(1)}(k) = 0$ for $k=i+1, \dots, M$, and since for all $i=1, \dots, M$ and $k=1, \dots, M$, we have $I_i^{f(1)}(k) = I_i^{f(1)}(0) = 0$ if $f(i) > f(1)$, the system state column vector Y_1 can be written using vector and matrix notation as given by equation (6.23) where A_1 is an $(M \times M)$ matrix defined by equation (6.24), B_1 is the column vector defined by equation (6.25), and B_1^i is the column vector with its i th component defined by:

$$B_i^1 = \sum_{k=1}^{i-1} \rho_k \left\{ \left[\sum_{j: \text{CHILD}(k,j)=1} I_i^{(1)}(j) / f(j) > f(i) \right] + 1 / \text{CHILD}(k,i)=1 \text{ and } f(i) \leq f(i) \right\} \quad (6.41)$$

$$i=1, \dots, M$$

6.3.3 System States at Service Completion Times

So far, we have formulated explicit expressions for the state of the system at service completion times of a stage of type j , $j=1, \dots, M$, conditioned of the state of the system found by such a type j stage upon its arrival to the queue. Upon the arrival of an exogenous stage, the state of the system is the steady state average number of each stage type in the system, and thus the state of the queue seen by such an arrival is $(S_1 - \rho) = (Q - \rho)$. When an exogenous stage completes its service, the state of the queue is given by equation (6.23); namely:

$$Y_1 = A_1(Q - \rho) + B_1 + B_1^1$$

where A_1 is defined by equation (6.24), B_1 is defined by equation (6.25), and B_1^1 is defined by equation (6.41). Such a type 1 stage may feedback several stages possibly of different priority levels. Let L_1 be the number of stages fed back by a type 1 stage; namely:

$$L_1 = \sum_{i=1}^M \text{CHILD}(1, i)$$

According to our assumed ordering scheme, type 1 stage children are then numbered in an increasing order (i.e., from 2 to $L_1 + 1$) and decreasing priority level from the left to the right in the job tree structure. Consequently, the state of the system seen by a type 2 stage is Y_1 ; namely $S_2 = Y_1$, and for any l , $2 \leq l \leq L_1 + 1$, we have:

$$S_l = Y_1 + L \text{CHILD}(1, l) \quad 2 \leq l \leq L_1 + 1$$

On the other hand, the state of the system upon the departure of a type 2 stage is then obtained from equation (6.19):

$$Y_2 = (A_2 S_2 + B_2) + R \text{CHILD}(1, 2)$$

The second term of the right hand side of the above equation accounts for the rest of stage 1 children that are not counted in S_2 . In the same manner, we have for any child of an exogenous stage:

$$Y_l = (A_l S_l + B_l) + R \text{CHILD}(1, l) \quad 2 \leq l \leq L_1 + 1$$

For endogenous stages, when a stage of type k , $k=2, \dots, M$ completes its service, the state of the queue left at such a point in time is given by:

$$Y_k = A_k S_k + B_k + RCHILD(i, k) \quad k=2, \dots, M$$

where the type i stage is the parent of stage k ; namely $CHILD(i, k)=1$. Furthermore, for any type l stage such that $CHILD(k, l)=1$, we have:

$$S_l = Y_k + LCHILD(k, l)$$

$$Y_l = A_l S_l + B_l + RCHILD(k, l)$$

Consequently, to evaluate the state of the system found by a type l stage upon its arrival to the queue, namely S_l , and the state of the queue left upon the completion of its service, namely Y_l , we need to know the state of the queue at the service completion time of the stage l parent, namely Y_k such that $CHILD(k, l)=1$. Knowing the state of the system upon the arrival of an exogenous stage, we can then deduce in the manner described above all S_i , $i=1, \dots, M$. We therefore obtain the following expression for the state of the queue found upon the arrival of a stage of type l , $l=1, \dots, M$:

$$\begin{aligned} S_l = & \left[\prod_{i=1}^{PATH(0)[1,l]-1} A_{PATH(i)[1,l]} \right] \cdot (Q - \rho) + \left[\prod_{i=2}^{PATH(0)[1,l]-1} A_{PATH(i)[1,l]} \right] \cdot B \quad (6.42) \\ & + \sum_{k=1}^{PATH(0)[1,l]-1} \left[\prod_{j=k+1}^{PATH(0)[1,l]-1} A_{PATH(j)[1,l]} \right] \\ & \cdot \left[B_{PATH(k)[1,l]} + LCHILD \left[PATH(k)[1,l], PATH(k+1)[1,l] \right] \right] \\ & + \sum_{k=2}^{PATH(0)[1,l]-1} \left[\prod_{j=k+1}^{PATH(0)[1,l]-1} A_{PATH(j)[1,l]} \right] \\ & \cdot RCHILD \left[PATH(k-1)[1,l], PATH(k)[1,l] \right] \end{aligned}$$

where again, the matrix product is taken by definition to be :

$$\prod_{i=k}^n A_i \triangleq \begin{cases} A_k A_{k-1} \cdots A_n & n \geq k \\ I & k = n+1 \\ 0 & k \geq n+2 \end{cases}$$

6.3.4 Steady State Equations for the Expected Total Time in System

In Section 6:3:1, we provided explicit expressions for the expected total time spent in the system respectively by endogenous and exogenous stages, conditioned on the system states found upon the arrivals of such stages. We now proceed to uncondition and evaluate these expected system times in the steady state, using the expression of the system states found upon arrivals given by equation (6.42). Since in the steady state we have $Q = \Lambda T$, then using equations (6.7) and (6.12) along with equation (6.42), we obtain the following system of linear equations for the expected total time spent in the system by stage type $j, j=1, \dots, M$.

$$\begin{aligned}
 T_j = & F_j \cdot \left[\prod_{i=1}^{PATH[0][1,j]-1} A_{PATH[i][1,j]} \right] \cdot \Lambda T \\
 & - F_j \cdot \left[\prod_{i=1}^{PATH[0][1,j]-1} A_{PATH[i][1,j]} \right] \cdot \rho \\
 & + F_j \cdot \left[\prod_{i=2}^{PATH[0][1,j]-1} A_{PATH[i][1,j]} \right] \cdot B_1 \\
 & + F_j \cdot \sum_{k=1}^{PATH[0][1,j]-1} \left[\prod_{i=k+1}^{PATH[0][1,j]-1} A_{PATH[i][1,j]} \right] \\
 & \quad \cdot \left[B_{PATH[k][1,j]} + LCHILD \left[PATH[k][1,j], PATH[k+1][1,j] \right] \right] \\
 & + F_j \cdot \sum_{k=2}^{PATH[0][1,j]-1} \left[\prod_{i=k+1}^{PATH[0][1,j]-1} A_{PATH[i][1,j]} \right] \\
 & \quad \cdot RCHILD \left[PATH[k-1][1,j], PATH[k][1,j] \right] \\
 & + r_j
 \end{aligned} \tag{6.43}$$

Equation (6.43) gives M linear equations for the expected total time spent in the system by each stage type, namely T_1, T_2, \dots, T_M . The expected sojourn time of a job having a tree shaped stage graph through the FJ-2-M/M/1 parallel processing system may now be computed using the proper formulation of $T(2)$ and the appropriate expressions for \bar{X}_i and $t_i, i=1, \dots, M$ defined in the following section. The following example demonstrates the use of the above established formulae for the computation of the expected total time spent in the system by each stage type in the tree shaped stage graph, in the case of a single server M/G/1 system.

Example

Consider an M/G/1 queueing system with jobs having a tree shaped stage graph composed of three stages. The root (i.e., the stage of type 1) has a priority level $f(1)=1$, and two children numbered 2 and 3. The type 2 stage is the left child and has a priority level $f(2)=3$, and the type 3 stage is the right child and has a priority level $f(3)=2$. Jobs arrive to the system according to a Poisson process with aggregate rate λ , and the service time of a stage independently of its type is exponentially distributed with mean $\frac{1}{\mu}$, namely $\bar{X}_1 = \bar{X}_2 = \bar{X}_3 = \frac{1}{\mu}$, and $\rho_1 = \rho_2 = \rho_3 = \frac{\lambda}{\mu}$. Using our system of linear equations given by (6.43), we get:

$$\begin{aligned} T_1 &= F_1 \Lambda T - F_1 \rho + r_1 \\ T_2 &= F_2 A_1 \Lambda T - F_2 A_1 \rho + F_2 B_1 + F_2 [B_1 + LCHILD(1,2)] + r_2 \\ T_3 &= F_3 A_1 \Lambda T - F_3 A_1 \rho + F_3 B_1 + F_3 [B_1 + LCHILD(1,3)] + r_3 \end{aligned} \quad (6.44)$$

where

$$\Lambda = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}, \quad \rho = \left(\frac{\lambda}{\mu}, \frac{\lambda}{\mu}, \frac{\lambda}{\mu} \right)^T, \quad T = (T_1, T_2, T_3)^T, \quad F_1 = \frac{1}{\mu} (3, 1, 1), \quad F_2 = \left(0, \frac{1}{\mu}, 0 \right)$$

$$F_3 = \left(0, \frac{1}{\mu}, \frac{1}{\mu} \right), \quad r_1 = \frac{5\lambda}{\mu^2} + \frac{1}{\mu}, \quad r_2 = \frac{1}{\mu}, \quad r_3 = \frac{1}{\mu}, \quad B_1 = \left(\frac{5\lambda^2}{\mu^2} + \frac{\lambda}{\mu} \right)^T$$

$$B_1 = (0, 0, 0)^T, \quad LCHILD(1,2) = (0, 0, 0)^T, \quad LCHILD(1,3) = (0, 1, 0)^T, \quad A_1 = \begin{bmatrix} \frac{3\lambda}{\mu} & \frac{\lambda}{\mu} & \frac{\lambda}{\mu} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Solving (6.44) yields:

$$T_1 = \frac{3\lambda + \mu}{\mu(\mu - 3\lambda)}, \quad T_2 = \frac{1}{\mu} \quad \text{and} \quad T_3 = \frac{2}{\mu}$$

The expected sojourn time of the defined tree shaped stage graph is given by $T(1) = T_1 + T_3$, which yields :

$$T(1) = \frac{3(\mu - \lambda)}{\mu(\mu - 3\lambda)}$$

Notice that $T_2 = \frac{1}{\mu}$ since a type 2 stage, having a priority level greater than that of the type 1 stage, does not undergo any waiting time. $T_3 = \frac{2}{\mu}$ since a type 3 stage must wait for its sibling

type 2 stage service time. On the other hand, this queueing system is equivalent to an M/G/1 queue where the service time distribution is the convolution of three exponentials; namely a three stage Erlang distribution with mean $\frac{3}{\mu}$ and second moment $\frac{12}{\mu^2}$. Using the known (P-K) formula of the mean waiting time in an M/G/1 queueing system [Klei76], yields the same exact result.

■

6.3.5 Generalized Conservation Law

We now proceed to develop a conservation law that puts a linear equality constraint on the set $T_i, i=1, \dots, M$ of the expected sojourn times of the different stages of the stage graph. In [Klei76], Kleinrock established the conservation law for the M/G/1 queueing system and for any non-preemptive work-conserving queueing discipline. Note that Kleinrock's conservation law also holds true for any M/M/1 queueing system and any preemptive priority work-conserving queueing discipline. In a similar fashion, we use the fact that the unfinished work in the system is invariant to the order of service provided that the discipline does not explicitly rely on any information about the remaining processing time of any stage and/or job in the system.

Recall that $\bar{X}_i, i=1, \dots, M$ denotes the random variable representing stage i service time in the M/G/1 system with average \bar{X}_i , and that $T_i, i=1, \dots, M$ is the expected amount of time stage i spends in the system from the time of its acquisition of the ready-for-service status until its service completion. Let $W_i, i=1, \dots, M$ be the expected amount of time stage i spends in the system from the time of its acquisition of the ready-for-service status until the start of its execution; that is $W_i = T_i - \bar{X}_i, i=1, \dots, M$. Let $\bar{Y} = \sum_{i=1}^M \bar{X}_i$ denote the random variable representing the total service requirement per job, with average \bar{Y} and second moment \bar{Y}^2 . The following Theorem states a generalized version of the conservation law.

Theorem 6.1: The M/G/1 with feedback Conservation Law

For any M/G/1 queueing system with jobs represented by a given stage graph and stages having prescribed priorities, and for any non-preemptive work-conserving service scheduling of the stages, it must be that:

$$\sum_{i=1}^M \left\{ \sum_{j=i}^M \text{DESC}(i,j) \bar{X}_j \right\} W_i = \frac{\rho}{1-\rho} \frac{\sum_{i=1}^M \bar{X}_i^2 + \sum_{i=1}^M \sum_{j \neq i}^M \bar{X}_i \bar{X}_j}{2}$$

Proof

Let $(n_1, \dots, n_i, \dots, n_M)$ be the state of the system where $n_i, i=1, \dots, M$ represents the number of stages of type i in the system in steady state, and let $U(n_1, \dots, n_i, \dots, n_M)$ denote the unfinished work in the system given the state $(n_1, \dots, n_i, \dots, n_M)$, with average \bar{U} . Therefore the unfinished work in the system, given such a state, is $\sum_{i=1}^M n_i \sum_{j=i}^M DESC[i, j] \bar{X}_j$, plus the remaining service time needed by the stage being executed. Let $\bar{t}_i, i=1, \dots, M$ represent the remaining service time of the stage being served as seen by a Poisson arrival, with average $\bar{t}_i = \frac{\bar{X}_i^2}{2\bar{X}_i}$. The probability that the service is serving a stage of type $i, i=1, \dots, M$ is simply ρ_i . Let $P(n_1, \dots, n_i, \dots, n_M)$ denote the steady state probability that the system is in state $(n_1, \dots, n_i, \dots, n_M)$. Consequently, the average unfinished work in the system, \bar{U} , is given by:

$$\bar{U} = \sum_{n_1=0}^{\infty} \dots \sum_{n_i=0}^{\infty} \dots \sum_{n_M=0}^{\infty} P(n_1, \dots, n_i, \dots, n_M) \left[\sum_{i=1}^M n_i \sum_{j=i}^M DESC[i, j] \bar{X}_j \right] + \sum_{i=1}^M \rho_i \bar{t}_i$$

which amounts to:

$$\bar{U} = \sum_{i=1}^M \rho_i \bar{t}_i + \sum_{i=1}^M \bar{n}_i \sum_{j=i}^M DESC[i, j] \bar{X}_j$$

Using Little's formula: namely that $\bar{n}_i = \lambda W_i, i=1, \dots, M$ we therefore obtain:

$$\sum_{i=1}^M W_i \sum_{j=i}^M DESC[i, j] \bar{X}_j = \text{constant} \tag{6.45}$$

to compute the value of the constant, we consider * the special tree shaped stage graph comprising M stages and such that for all $i=1, \dots, M-1$, we have $f(i+1) \geq f(i)$. For such a tree, we can see our queueing system as an $M/G/1$ system where jobs have a continuous service given by the random variable \tilde{Y} . Using the *Pollaczek-Khinchin* mean waiting time formula [Klei75], we have:

$$W_1 = \frac{\frac{\lambda \bar{Y}^2}{2}}{1-\rho} \quad \text{and} \quad W_i = 0 \quad i=2, \dots, M$$

since $\sum_{j=1}^M DESC[1, j] = \frac{\rho}{\lambda}$, and since \bar{Y}^2 is given by:

* It is not difficult to see that any tree with an arbitrary numbering and priority assignment of stages, corresponds a tree shaped stage graph with the required ordering of stages and their priority levels.

$$\overline{Y^2} = \sum_{i=1}^M \overline{X_i^2} = \sum_{i=1}^M \sum_{j \neq i}^M \overline{X_i X_j}$$

then using these values of W_i , $i=1, \dots, M$ and $\overline{Y^2}$ into equation (6.45) completes the proof of our conservation law.

from this conservation law, we see that any attempt to modify the stages service scheduling so as to reduce one of the W_i will amount to an increase in some of the other W_i ; however, this need not to be an even trade since in general the weighting factors for the different W_i 's are different. □

6.3.6 Job Average Sojourn Time

Recall that our stated objective is to analyze the FJ-2-M/M/1 system with $f(i)=f(1)$ for all $i=1, \dots, M$. Tasks at any one of the processors are then scheduled for service on an FCFS basis. The use of a load adjustable distribution in the equivalent queuing system is based on the fact that in an FJ-2-M/M/1, the more queued-up stages the closer is the stage average service time to $\frac{1}{\mu}$. Under light traffic conditions, we have seen that the sojourn time of a job having the tree shaped stage graph of Figure 6.1:(b) and Figure 6.1:(c) gives a lower $T_0(2)$ than the one having the chain shaped stage graph of Figure 6.1:(a). Indeed, it is always true that a tree shaped stage graph gives a lower $T_0(2)$ than a chain stage graph having the same number of stages. This statement is due to the inherent property of the FJ-2-M/M/1 parallel processing system in which the more queued-up stages, the faster is the stage service time. On the other hand, under very heavy traffic conditions, the number of queued-up stages is high whether the job has a tree shaped stage graph or a chain shaped stage graph. Consequently, the stage service time of a job having a tree shaped stage graph approaches from below the service time of a stage in a chain shaped stage graph. Let β denote the ratio of $T_0(2)$ given by the tree shaped stage graph, and $T_0(2)$ given by the chain shaped stage graph having the same number of stages. Define the load adjustable weighting function $\gamma(\tau)$ by :

$$\gamma(\tau) = (1-\beta)\tau + \beta$$

where τ is the utilization factor of our $M/G_c/1$ system. Let \bar{X}_T represent the service time of a stage of the tree stage graph, with mean \bar{X}_T and second moment \bar{X}_T^2 . Hence, we have $\tau = \lambda M \bar{X}_T$. We define the random variable \bar{X}_T as a weighted function of the random variable \bar{X}_C representing the service time of a stage in the chain stage graph and defined in Section 6.2.5.

$$\bar{X}_T = \gamma(\tau) \bar{X}_C \tag{6.46}$$

The weighting function $\gamma(\tau)$ satisfies $\lim_{\tau \rightarrow 0} \gamma(\tau) = \beta$, and $\lim_{\tau \rightarrow 1} \gamma(\tau) = 1$. From the above definitions, we readily obtain $\bar{X}_T = \gamma(\tau) \bar{X}_C$. Using the expression for \bar{X}_C given by equation (6.31), we get:

$$\tau = \frac{3\rho_s\beta}{2(1-\rho_s) + 3\rho_s\beta} \quad (6.47)$$

Notice that the utilization factor τ satisfies $\lim_{\rho_s \rightarrow 0} \tau = 0$ and $\lim_{\rho_s \rightarrow 1} \tau = 1$. From equation (6.47), and since $\tau = \lambda M \bar{X}_T = \mu \rho_s \bar{X}_T$, we have :

$$\bar{X}_T = \frac{3\beta}{\mu [2(1-\rho_s) + 3\rho_s\beta]} \quad (6.48)$$

Also from equation (6.46), we have $\bar{X}_T^2 = \gamma(\tau)^2 \bar{X}_C^2$, and finally for our system with $f(i)=f(1)$, $i=1, \dots, M$, we obtain:

$$\bar{X}_i = \bar{X}_T = \frac{3\beta}{\mu [2(1-\rho_s) + 3\rho_s\beta]} \quad i=1, \dots, M \quad (6.49)$$

$$t_i = \frac{\bar{X}_T^2}{2\bar{X}_T} = \frac{2\rho_s(2+\rho_s)^2 + 6(1-\rho_s)\rho_s}{3\mu [2(1-\rho_s) + 3\rho_s\beta]} \quad i=1, \dots, M \quad (6.50)$$

Validation of the Approximation

Figure 6.5 depicts the expected sojourn time of the tree shaped stage graph represented in Figure 6.1:(b), as a function of the utilization factor ρ_s . The expected system times T_1, T_2, T_3, T_4 , and T_5 of the 5 stages composing the tree graph are obtained by solving the system of linear equations given by (6.43), where the stage average service times, $\bar{X}_i, i=1, \dots, M$ are given by (6.49), and the stage service time residual life $t_i, i=1, \dots, M$ is given by (6.50). The expected sojourn time of such a job is obtained using equation (6.33).

Figure 6.6 depicts the expected sojourn time of a job having the tree shaped stage graph represented in Figure 6.1:(c), as a function of the utilization factor ρ_s . T_1, T_2, T_3, T_4 , and T_5 for the graph are obtained by solving the system of linear equations given by (6.43), where the \bar{X}_i , and the $t_i, i=1, \dots, M$ are given respectively by (6.49) and (6.50). The expected sojourn time of such a job is obtained using equation (6.34).

To validate our approximation, we simulated the FJ-2-M/M/1 parallel processing system for the given tree shaped stage graphs. Again, the extent of the transient state is estimated using short independent replications, and the method used to estimate the statistic $T(2)$ in the steady state is *the Method of Batch Means*. All confidence intervals are set to a 90% level, and are generated via the t-distribution.

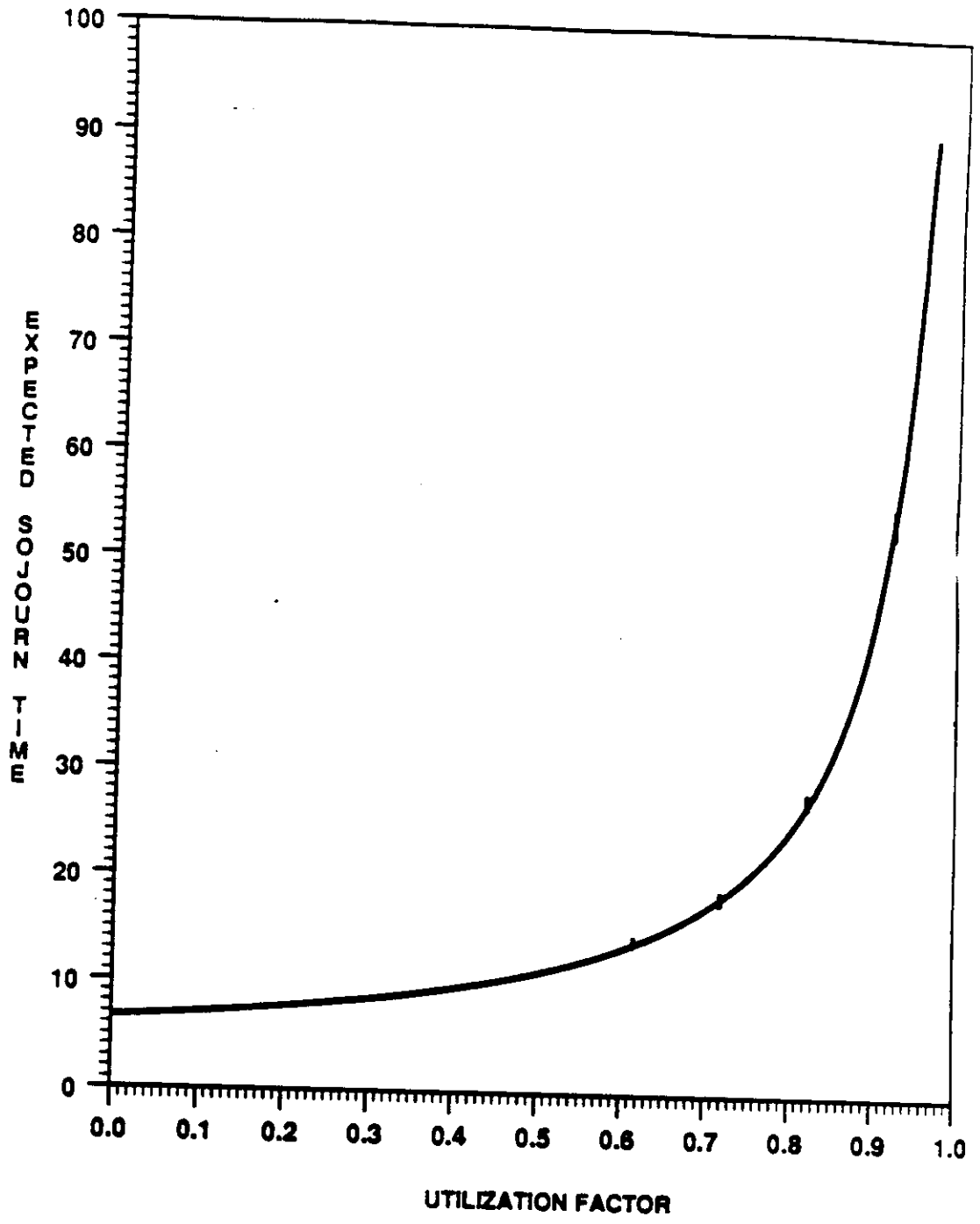


Figure 6.5: Expected Sojourn Time of the Tree Shaped Stage Graph of Figure 6.1:(b)

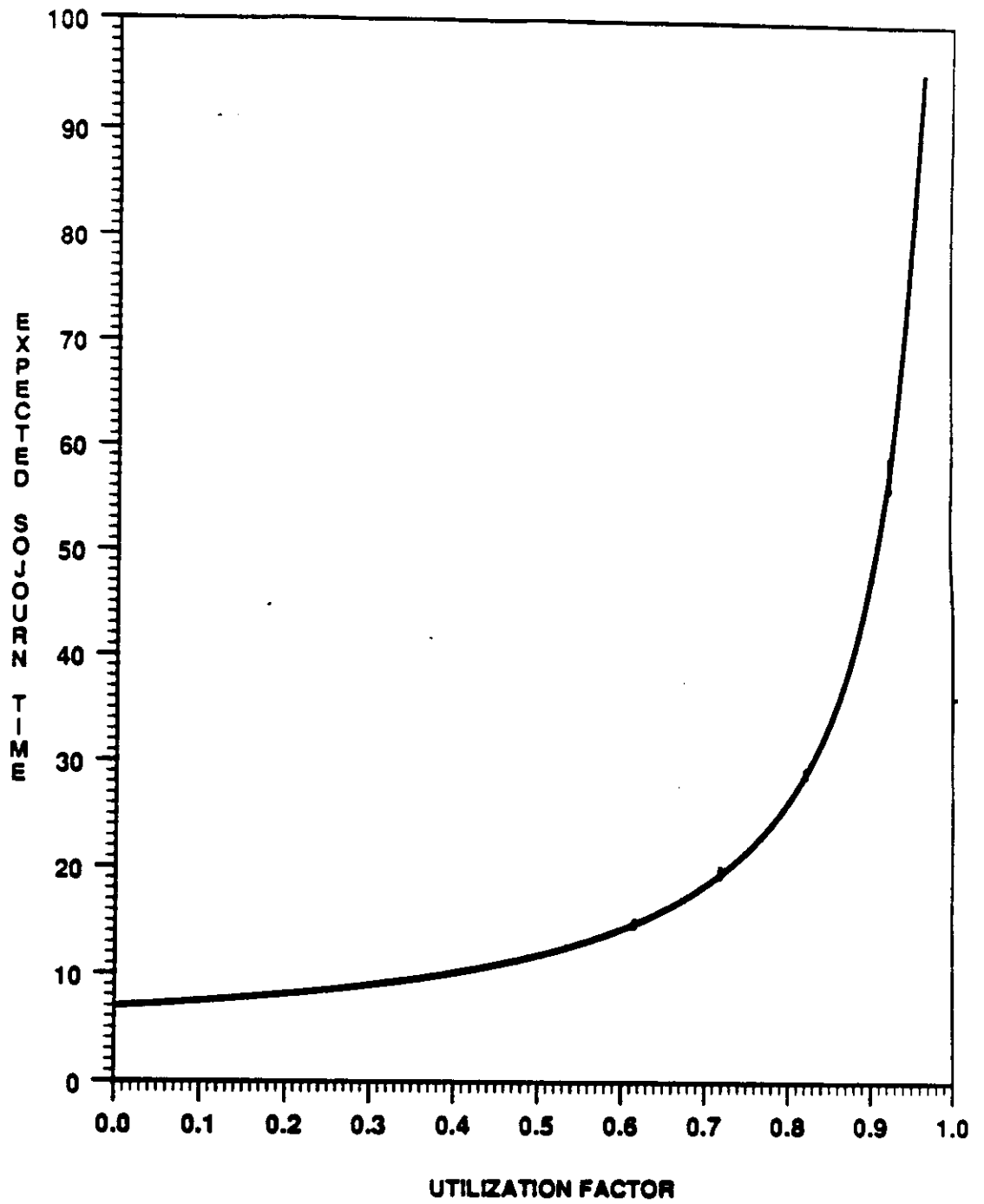


Figure 6.6: Expected Sojourn Time of the Tree Shaped Stage Graph of Figure 6.1:(c)

From both figures, we observe that the approximation results in very accurate values of the expected sojourn time. The confidence interval widths are very small almost over all the permissible range of the utilization factor ρ_j . The mean relative error is in the order of 1.5% over the full range of ρ_j . Relative to the case of the chain shaped stage graph whose expected sojourn time is depicted on Figure 6.4, we observe that the approximation is somewhat more accurate in the sense that it results in smaller confidence intervals. Figure 6.7 depicts the expected sojourn time of a job having respectively the chain shaped graph of Figure 6.1:(a), the tree shaped stage graph of Figure 6.1:(b), and the tree shaped stage graph of Figure 6.1:(c). The lowest curve represents the $T(2)$ of Figure 6.1:(b), and the highest curve represents the $T(2)$ of Figure 6.1:(a).

6.4 Conclusion

In this chapter, we have investigated ways to analyze the performance of models of parallel processing systems, in which jobs are represented by a given arbitrary directed acyclic stage graph. Each stage is composed of two tasks which must be attended by different processors. Our methodology consisted of representing the parallel processing system as an $M/G_C/1$ queueing system with correlated consecutive stage service times. It has been shown that such a methodology results in excellent approximations of the job expected sojourn time through the parallel processing system.

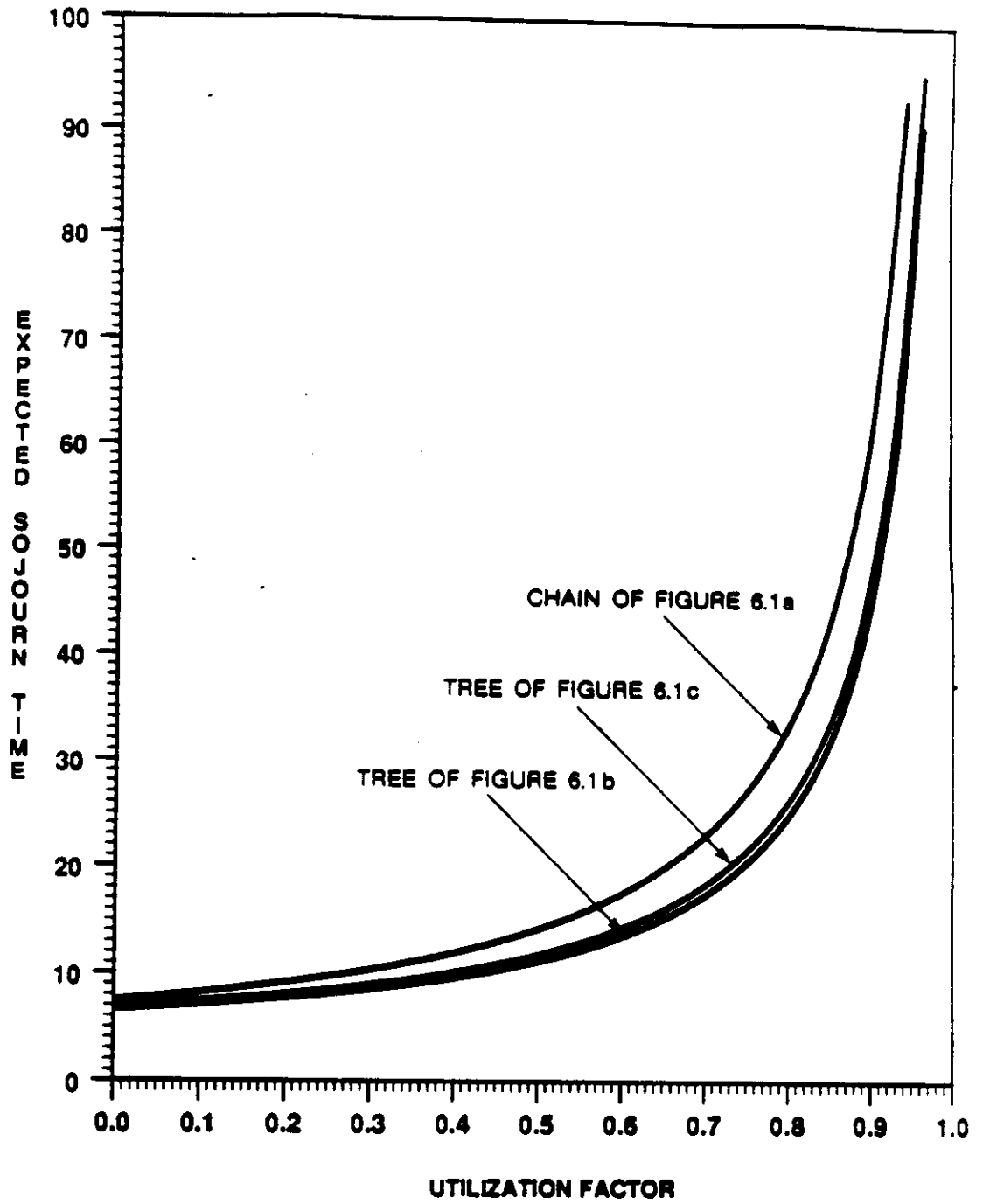


Figure 6.7: Comparison of the Expected Sojourn Times

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

In the previous chapters, we have seen that the main difficulty in accurately analyzing parallel processing systems is the internal and yet inherent parallelism within jobs. At any given time, a job may need and consequently may hold more than one processor. Unfortunately, this system characteristic of simultaneous resource possession precludes a *Product Form* solution for the model. Coupled, dependent systems like these are very difficult to analyze and exact numerical solution is usually not feasible. Our methodology undertook, in some fashion, a means to understand and then control the underlying interdependencies. Hence, based on the underlying stochastic processes of interest, we devised accurate and yet very simple approximate solutions whenever exact analysis failed.

In Chapter 2, we essentially ignored the dependency by assuming an infinite number of processors and a constant service time per task. We derived the probability distribution of the number of occupied processors, the generating function of this distribution and its first two moments, for different job structures. The study of this simple model resulted in two fundamental facts. The first concerns the large body of algebra involved in determining the distribution of the number of occupied processors and its first two moments. The second fact is that the average number of busy processors is independent of the different job structures studied, and depends only on the job average arrival rate, the average number of tasks per job, and the task average service requirement. The question naturally arises as to what extent can this latter result be generalized. Chapter 3 is devoted to answering such a question. The generalization was stated through Theorem 3.1 for the infinite number of processors case, and through Theorem 3.2 for the finite number of processors case. Although the expected number of busy processors in a multiprocessing system sheds light on how much of the resources can (on the average) be utilized simultaneously, it does not accurately ascertain the degree of parallelism achieved by executing the jobs on a multiprocessor system (the achievable gain in the average response time). We introduced a new service scheduling strategy based on a non-egalitarian processor sharing among the jobs present in the system. We formulated upper and lower bounds on the job average response time, and devised a rather accurate and yet very simple parametric approximation based on the speedup achieved by executing a job through an empty multiprocessor system. Using this approximation, we proceeded to evaluate the achievable parallelism, the efficiency of

the processors, and consequently the optimal operating points at which one should operate the multiprocessor system. Chapter 5 and Chapter 6 are devoted to the analysis of dedicated processors, in which a job, upon arrival, splits into several clones, each of which is attended by a separate dedicated processor. In Chapter 5, we proved an upper and a lower bound on the job expected sojourn time, defined as the expected time spent in the system between a job arrival time and the execution completion of all its clones. Based on the underlying stochastic processes of interest, our approach undertook a means to control the interdependencies by converting the system into an $M/G/1$ queueing system with correlated consecutive service times. We presented ways and methodologies to deal in an approximate way with these correlations. In Chapter 6, we pursued the same model of a multiprocessing system with synchronized arrivals, and we further permitted jobs to feed back into the system according to any prespecified acyclic feedback structure. We first extended the results on $M/G/1$ queueing systems with priorities and job feedback, and then used results from Chapter 5 to analyze the job expected sojourn time in such multiprocessing systems with synchronized arrivals and job feedback.

The potential for extensions and generalizations of this dissertation is present in almost every chapter. We believe, however, that the most valuable and urgently needed extensions would be the study of the communication delays incurred when exchanging data between the different processes and processors, a refinement of the parallel processing system analysis, and the analysis of the run time behavior of concurrent programs.

Communication Cost

Throughout the dissertation, we have assumed that the cost of exchanging data between processors is free and is achieved instantaneously. In reality, there is always some delay incurred when communicating between processors. The communication delays are rather crucial components which must be considered in conjunction with the processing delays to better ascertain the parallelism achieved by multiprocessor systems, and to properly compare multiprocessor systems to multicomputer systems. A simple model to account for such communication delays would incorporate, in the job process graph, some communication nodes representing both the communication times involved between parallel tasks, and the data transfers required between consecutive tasks.

A New Class of Queueing Systems

We have seen that a job may need and consequently may hold more than one processor at a time. Consider the process graph of Figure 4.1. Let us assume that no more than one processor may work on a given task. When task A is being processed, the job can proceed at a maximum rate of 1 second per second. When task A is completed, tasks B and C may begin, and the job can proceed at a maximum rate of 2 seconds per second, assuming a multiprocessor system comprising two or more processors, etc. Consequently, the maximum rate at which a

job can absorb work depends upon where it is in its processing cycle (i.e., which tasks are ready), and on the number of processors composing the multiprocessor system. This forms a new class of queueing systems where the maximum rate at which a job can proceed varies with elapsed time, as compared to the classic queueing model, which assumes that the maximum rate at which a job can absorb work is constant. In Chapter 6, we adopted the discriminatory processor sharing of the processors' total capacity among the jobs present in the system, and thus we solved (exactly in the infinite number of processors case, and approximately in the finite processors case) for the job average response time in such a system. Further research is undoubtedly needed in this direction.

Concurrent Programs Run-time Behavior

In Chapter 1, we distinguished two categories of relationships among the set of tasks composing a given job. The precedence relationships (type 1) between tasks specify the order of acquisition of the ready-for-service-status. The parallel relationships (type 2) specify the dynamic interactions that govern the actual execution of the active tasks within a job. In this dissertation, we represented jobs by process graphs comprising thus only relationships of type 1. The study of parallel relationships among active tasks is also crucial in understanding the run-time behavior of parallel and distributed algorithms. Moreover, a third type of relationship must also be considered: this concerns the intercommunications and synchronizations required between active tasks of different concurrent jobs. Further research in this direction is necessary to better understand the dynamics of parallel programs during their execution. It is the author's opinion that alternate modeling techniques are needed to investigate and understand the run-time behavior of programs incorporating both type 2 and type 3 parallel relationships. Most of the relevant mathematics used to analyze dependent systems of interacting processes, such as queueing theory, stochastic processes and probability theory, are only tractable for systems in which the random variables are assumed to be independent. A fairly new approach consists of using a *Geometric Concurrency Model* similar to Dijkstra's *Process Graph* for the characterization of deadlocks in multiprocessing systems. The Geometric Model has been studied by H.T. Kung, W. Lipski, C.H. Papadimitriou, and M. Yannakakis [Yann79, Lips81, Papa83]. The model was later used by Carson [Cars84] to prove liveness properties of concurrent programs and by M. Abrams [Abra86] to ascertain the exact run-time behavior of two interacting processes executed asynchronously on separate processors.

APPENDIX A

Derivation of $\sum_{n=0}^{\infty} \binom{n+k}{k} x^n$

Let the bivariate function $a_k(n)$ be defined as:

$$a_k(n) = \binom{n+k}{k} x^n \quad \forall k \geq 0, n \geq 0 \quad (\text{A.1})$$

where the absolute value of x is less than unity. From the above equation, we obtain the following recurrence relation on the bivariate function $a_k(n)$:

$$a_k(n) = \frac{n+k}{k} a_{k-1}(n) \quad k \geq 1 \quad (\text{A.2})$$

with the following boundary condition for $k=0$:

$$a_0(n) = x^n \quad n \geq 0 \quad (\text{A.3})$$

Define the normal generating function of $a_k(n)$ by $A_k(Z)$, $\forall k \geq 0$; that is $A_k(Z) \triangleq \sum_{n=0}^{\infty} a_k(n) Z^n$.

Thus equation (A.2) yields:

$$A_k(Z) = \frac{Z}{k} \frac{d}{dZ} A_{k-1}(Z) + A_{k-1}(Z) \quad k \geq 1 \quad (\text{A.4})$$

Now, we proceed to show, by induction on the index k , that $A_k(Z) = [A_0(Z)]^{k+1}$.

Basis step

From equation (A.2), we have $a_0(n) = x^n$, and consequently its generating function $A_0(Z)$ is given by: $A_0(Z) = \frac{1}{1-xZ}$. On the other hand, for the value of $k=1$, we have

$a_1(n) = (n+1)x^n$, and hence its generating function $A_1(Z)$ is given by: $A_1(Z) = \frac{1}{(1-xZ)^2}$.

Therefore we readily have: $A_1(Z) = [A_0(Z)]^2$.

Inductive Step

Now, suppose that we have:

$$A_j(Z) = [A_0(Z)]^{j+1} \quad 1 \leq j \leq k-1$$

and let us show it for the value k . We have:

$$\frac{d}{dZ} A_{k-1}(Z) = xk [A_0(Z)]^k$$

thus, using equation (A.4), we indeed obtain $A_{k+1}(Z) = [A_0(Z)]^{k+1}$.

Finally, since $A_0(Z) = \frac{1}{1-xZ}$, and putting $Z=1$, we obtain:

$$\sum_{n=0}^{\infty} \binom{n+k}{n} = \left[\frac{1}{1-x} \right]^{k+1} \quad k \geq 0 \quad (\text{A.5})$$

■

APPENDIX B

Derivation of $A = \sum_{i=2}^N \frac{1}{2^{N-1}} \sum_{x=1}^i x \left\{ \sum_{r=i}^N \binom{N-x-1}{r-2} \right\}$

To evaluate the quantity A, notice that for any given value of i, $i=2, \dots, N$ correspond a range of values for x, and a range of values for r. For $i=j$ for example, we have $r=j, j+1, \dots, N$, and $x=1, 2, \dots, N-j+1$. This value of i accounts with the following in the expression of the quantity A:

$$1. \left\{ \binom{N-2}{j-2} + \dots + \binom{N-2}{N-2} \right\} + 2. \left\{ \binom{N-3}{j-2} + \dots + \binom{N-3}{N-3} \right\} + \dots + (N-j+1). \left\{ \binom{j-2}{j-2} \right\}$$

In the above expression, we purposely factored out the values of x. Let A_x denote the participation of the value x in the quantity A. Therefore, for any value of x, $x=1, \dots, N-1$, we obtain:

$$A_x = x \left\{ \binom{N-x-1}{0} + 2 \binom{N-x-1}{1} + 3 \binom{N-x-1}{2} + \dots + (i+1) \binom{N-x-1}{i} + \dots + (N-x) \binom{N-x-1}{N-x-1} \right\}$$

Which amounts to:

$$A_x = x \sum_{i=0}^{N-x-1} (i+1) \binom{N-x-1}{i} \quad x=1, \dots, N \tag{B.1}$$

since x varies from 1 to N-1, it follows that:

$$A = \sum_{x=1}^{N-1} A_x$$

and hence by using equation (B.1), we obtain:

$$A = \frac{1}{2^{N-1}} \sum_{x=1}^{N-1} x \sum_{i=0}^{N-x-1} (i+1) \binom{N-x-1}{i} \tag{B.2}$$

on the other hand, we have:

$$\sum_{i=0}^{N-x-1} (i+1) \binom{N-x-1}{i} = \sum_{i=0}^{N-x-1} \binom{N-x-1}{i} + \sum_{i=0}^{N-x-1} i \binom{N-x-1}{i}$$

and since by using the binomial theorem [Liu68], we have:

$$\sum_{i=0}^{N-x-1} \binom{N-x-1}{i} = 2^{N-x-1} \quad \text{and} \quad \sum_{i=0}^{N-x-1} i \binom{N-x-1}{i} = (N-x-1) 2^{N-x-2}$$

therefore equation (B.2) becomes:

$$A = \frac{1}{2^{N-1}} \sum_{x=1}^{N-1} x (N-x+1) 2^{N-x-2} \quad (\text{B.3})$$

Now, define the quantities A1 and A2 by:

$$A1 = \sum_{x=1}^{N-1} x \left[\frac{1}{2} \right]^{x-1}$$

$$A2 = \sum_{x=1}^{N-1} x (x-1) \left[\frac{1}{2} \right]^{x-2}$$

Therefore equation (B.3) becomes:

$$A = \frac{1}{8} \left\{ 2NA1 - A1 \right\} \quad (\text{B.4})$$

now, we proceed to evaluate the expressions of A1 and A2. Let $y = \frac{1}{2}$; we have:

$$A1(y) = \sum_{x=1}^N x y^{x-1} = \frac{d}{dy} \sum_{x=1}^{N-1} y^x = \frac{d}{dy} \left[y \frac{1-y^{N-1}}{1-y} \right]$$

which amounts then to:

$$A1(y) = \frac{\left[1 - Ny^{N-1} \right] (1-y) + y - y^N}{(1-y)^2} \quad (\text{B.5})$$

replacing the dummy variable y by its value $\frac{1}{2}$, we obtain:

$$A1 = 4 - (N+1) \left[\frac{1}{2} \right]^{N-2} \quad (\text{B.6})$$

Now let us evaluate A2; we have:

$$A2(y) = \sum_{x=1}^{N-1} x (x-1) y^{x-2} = \frac{d^2}{dy^2} \sum_{x=1}^{N-1} y^x$$

which amounts then to:

$$A_2(y) = \frac{N(N-1)y^{N-2}}{y-1} + \frac{2}{1-y}A_1(y) \quad (\text{B.7})$$

replacing the dummy variable y by its value $\frac{1}{2}$, we obtain:

$$A_2 = 16 - \left[N^2 + N + 2 \right] \left[\frac{1}{2} \right]^{N-3} \quad (\text{B.8})$$

Finally, by using equations (B.4), (B.6) and (B.8), we obtain:

$$A = N - 2 + \left[\frac{1}{2} \right]^{N-1} \quad (\text{B.9})$$

III

Evaluation of $B = \sum_{x \geq 1} x \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$

As before, let A_x denote the total participation of the value x , $x=1, \dots, N-1$ in the expression of the quantity B . For the value $x=i$ for example, we have:

$$A_i = i \binom{N-i-1}{0} + i \binom{N-i-1}{1} + \dots + i \binom{N-i-1}{N-i-1} = i 2^{N-i-1}$$

it follows then that the expression of B becomes:

$$B = \sum_{x=1}^{N-1} x 2^{N-x-1} = 2^{N-2} \sum_{x=1}^{N-1} x \left[\frac{1}{2} \right]^{x-1} = 2^{N-2} A_1(y)$$

and by using equation (B.6) and replacing the dummy variable y by its value $\frac{1}{2}$, we get:

$$B = 2^N - (N+1) \quad (\text{B.10})$$

III

Derivation of $C = \sum_{i=2}^N \frac{1}{2^{N-1}} \sum_{x \geq 1} x^2 \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$

From the evaluation of the quantity A earlier in this Appendix, and by following the same exact steps, it is not hard to see that the quantity C may be written as:

$$C = \frac{1}{2^{N-1}} \sum_{x=1}^{N-1} x^2 (N-x+1) 2^{N-x-2} \quad (\text{B.11})$$

Now, define the quantities A3 and A4 by:

$$A3 = \sum_{x=1}^{N-1} x^2 \left[\frac{1}{2} \right]^{x-2}$$

$$A4 = \sum_{x=1}^{N-1} x^3 \left[\frac{1}{2} \right]^{x-3}$$

Therefore equation (B.11) becomes:

$$C = \frac{1}{16} \left\{ 2(N+1)A3 - A4 \right\} \quad (\text{B.12})$$

Now, we proceed to evaluate the expressions of A3 and A4. To evaluate A4, we need first to evaluate the following expression:

$$A5 = \sum_{x=1}^{N-1} x(x-1)(x-2) \left[\frac{1}{2} \right]^{x-3}$$

Let $y = \frac{1}{2}$, the quantity A5 can then be written as:

$$A5(y) = \sum_{x=1}^{N-1} x(x-1)(x-2)y^{x-3} = \frac{d^3}{dy^3} \left\{ \sum_{x=1}^{N-1} y^x \right\} = \frac{d}{dy} A2(y)$$

using the expression of A2(y) as given by equation (B.7), we obtain:

$$A5(y) = \frac{d}{dy} \left\{ \frac{N(N-1)y^{N-2}}{y-1} + \frac{2}{1-y} A1(y) \right\}$$

where the expression of the quantity A1(y) is given by equation (B.5). Define the quantities A6(y) and A7(y) by:

$$A6(y) = \frac{d}{dy} \left\{ \frac{N(N-1)y^{N-2}}{y-1} \right\}$$

$$A7(y) = \frac{d}{dy} \left\{ \frac{2A1(y)}{1-y} \right\}$$

Therefore the quantity A5(y) can be rewritten as:

$$A5(y) = A6(y) + A7(y) \quad (\text{B.13})$$

After derivation and some algebra and by replacing the dummy variable y by its value $\frac{1}{2}$, we obtain:

$$A6 = -N(N-1)^2 \left[\frac{1}{2} \right]^{N-4} \quad (\text{B.14})$$

Now, we proceed to evaluate the quantity A7(y); we have:

$$A7(y) = \frac{d}{dy} \left\{ \frac{2A1(y)}{1-y} \right\} = \frac{2}{1-y} \frac{d}{dy} A1(y) + \frac{2}{(1-y)^2} A1(y)$$

Since from the definitions of the quantities A1(y) and A2(y), we have $\frac{d}{dy} A1(y) = A2(y)$, and after some algebra and replacing the dummy variable y by its value $\frac{1}{2}$, we obtain:

$$A7 = 96 - (N^2 + 2N + 3) \left[\frac{1}{2} \right]^{N-5} \quad (\text{B.15})$$

Returning now to the expression of the quantity A5, and using equations (B.13), (B.14), and (B.15), we obtain:

$$A5 = 96 - (N^3 + 5N + 6) \left[\frac{1}{2} \right]^{N-4} \quad (\text{B.16})$$

Let us now return to the evaluation of the quantity C. Since:

$$A4 = A5 + 3 \sum_{x=1}^{N-1} x^2 \left[\frac{1}{2} \right]^{x-3} - 2 \sum_{x=1}^{N-1} x \left[\frac{1}{2} \right]^{x-3}$$

using equations (B.12), (B.13), (B.14), (B.15) and (B.16), and after some algebra, we obtain:

$$\begin{aligned} C &= \frac{1}{16} \left\{ 2(N-2) \sum_{x=1}^{N-1} x(x-1) \left[\frac{1}{2} \right]^{x-2} + 4N \sum_{x=1}^{N-1} x \left[\frac{1}{2} \right]^{x-1} - A5 \right\} \\ &= \frac{1}{16} \left\{ 2(N-2)A2 + 4NA1 - A5 \right\} \end{aligned}$$

and therefore by using the expressions of the quantities A1, A2, and A5, which are given respectively by equations (B.6), (B.8), and (B.16), we obtain:

$$C = 3N - 10 + \frac{2N + 5}{2^{N-1}} \quad (\text{B.17})$$

Evaluation of $D = \sum_{x \geq 1} x^2 \left\{ \sum_{r=2}^N \binom{N-x-1}{r-2} \right\}$

From the evaluation of the quantity B earlier in this Appendix, and by following the same exact steps, it is not hard to see that the quantity D may be written as:

$$\begin{aligned}
 D &= \sum_{x=1}^{N-1} x^2 2^{N-x-1} = 2^{N-3} \sum_{x=1}^{N-1} x^2 \left[\frac{1}{2} \right]^{x-2} \\
 &= 2^{N-3} \left\{ \sum_{x=1}^{N-1} x(x-1) \left[\frac{1}{2} \right]^{x-2} + 2 \sum_{x=1}^{N-1} x \left[\frac{1}{2} \right]^{x-1} \right\} \\
 &= 2^{N-3} \left\{ A_2 + 2A_1 \right\}
 \end{aligned}$$

and therefore using the expressions of the quantities A1 and A2 given respectively by equations (B.6) and (B.8), and after some algebra, we obtain:

$$D = 3 \cdot 2^N - N^2 - 2N - 3 \quad (B.18)$$

■

APPENDIX C

Expected Number of Exogenous Arrivals During a Stage Waiting Time

We solve for the expected number of exogenous stages (i.e., jobs) that arrive during a type j , $j=1, \dots, M$ stage total waiting time, and consequently their expected contribution to j 's total waiting time, as a function of the system state S_j found upon such a type j stage arrival to the system. Since the waiting time process is not independent of the arrival and the service processes, Wald's Lemma cannot be applied in a straight forward manner.

Let $S_j = (s_1^j, \dots, s_i^j, \dots, s_M^j)$ be the vector representing the state of the system such that s_i^j , $i=1, \dots, M$ is the expected number of stages of type i found in the system when stage j enters. Recall that $T_j^1(S_j)$ is the total expected time the type j stage spends waiting for higher and equal priority stages already in the system upon its arrival, and is given by (6.4). Let $W_j^1(S_j)$ be the total expected time, the type j stage spends waiting for higher priority stages that arrive during the time $W_j^{i-1}(S_j)$, $i=1, 2, \dots$ with $W_j^0(S_j) = T_j^1(S_j)$. Let a_i , $i=1, 2, \dots$ be the expected number of exogenous arrivals occurring during the time $W_j^{i-1}(S_j)$. Since the arrival process is Poisson with aggregate rate λ , we have:

$$a_i = \lambda W_j^{i-1}(S_j) \quad i=1, 2, \dots \quad (C.1)$$

Lemma

There exists an $i \geq 1$ such that for $k \geq i$, we have $a_k = 0$

Proof

The proof follows directly from the definition of the a_i , $i=1, 2, \dots$ and the stability of the system. The total expected number of exogenous arrivals during the total expected waiting time of the type j stage is: $\sum_{i=1}^{\infty} a_i$. On the other hand, if for k , $k \geq 1$, we have $a_k = 0$ then $W_j^k(S_j) = 0$ and all

the subsequent a_l and $W_j^l(S_j)$, $l > k$, are equal to zero.

The total expected waiting time of the type j stage is :

$$W_j = \sum_{i=0}^{\infty} W_j^i$$

Using (C.1), yields:

$$\sum_{i=0}^{\infty} a_i = \lambda W_j$$

References

- [Abra86] M. Abrams, "Performance Analysis of Unconditionally Synchronized Distributed Computer Programs Using the Geometric Concurrency Model," *Ph.D. Dissertation (Chapter 5), Technical Report No. 1696, University of Maryland, Department of Computer Science*, August 1986.
- [Acke82] W.B. Ackerman, "Data Flow Languages," *IEEE Computer Magazine*, Vol. 15(2), february 1982, pp. 15-25.
- [Ada81] *Ada, The Programming Language Ada. Reference Manual. Lecture Notes in Computer Science*: Springer-Verlag, 1981.
- [Agra84] A. Agrawala, E. Coffman, M. Garey, and S. Tripathi, "A Stochastic Optimization Algorithm Minimizing Expected Flow Times on Uniform Processors," *IEEE Transactions on Computers*, Vol. c-33, No. 4, April, 1984, pp. 351-356.
- [Alex83] P. Alexander, "Array Processors in Medical Imaging," *IEEE Computer Magazine*, Vol. 16, No. 6, June 1983, pp. 17-31.
- [Avi-65] B. Avi-Itzhak, W.L. Maxwell, and L.W. Miller, "Queueing with Alternating Priorities," *Operations Research*, Vol. 13-(2), April-May, 1965, pp. 306-318.
- [Bacc85a] F. Baccelli and A.M. Makowski, "Simple Computable Bounds for the Fork-Join Queue," *The John Hopkins Conf. on Inf. Sc., John Hopkins Univ.*, 1985.
- [Bacc85b] F. Baccelli, A.M. Makowski, and A. Schwartz, "Simple Computable Bounds and Approximations for the Fork-Join queue," *Proc. Int. Seminar on Computer Networking and Performance Evaluation, Held at Tojo Kaikan, Tokyo, September 18-20, 1985*.
- [Bari75] R.E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*: Holt, Reinhart and Winston, Inc., 1975.

- [Bam68] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes, "The ILLIAC IV Computer," *IEEE Transactions on Computers*, Vol. C-17, No. 8, August 1968, pp. 746-757.
- [Bask75] F. Baskett, K.M. Chandy, R.R. Muntz, and F. Palacios-Gomez, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *JACM*, Vol. 22, No. 2, April 1975.
- [Bask77] F. Baskett and T.W. Keller, *An Evaluation of the CRAY-1 Computer (High Speed Computer and Algorithm Organization, Kuck et al., Editors)*, New York: Academic Press, 1977.
- [Belg85] A. Belghith and L. Kleinrock, "Analysis of the Number of Occupied Processors in a Multiprocessing system," *Technical Report No. UCLA-CSD-850027, School of Engineering and Applied Science, Computer Science department, University of California, Los Angeles, August, 1985.*
- [Boot83] W.C. Booth, "Approaches to Radar Signal Processing," *IEEE Computer Magazine*, Vol. 16, No. 6, June 1983, pp. 32-43.
- [Boud85] J. Le Boudec, "A BCMP Extension to Multiserver Stations with Concurrent Classes of Customers," *Rapport de recherche No. 390, INRIA, France, March 1985.*
- [Bren83] T.A. Brengle, B.I. Cohen, and M.E. Stewart, "Simulating Field-Reversed Magnetic-Mirror Fusion Devices," *IEEE Computer Magazine*, Vol. 16, No. 6, June 1983, pp. 43-5.
- [Cars84] S.D. Carson, "Geometric Models of Concurrent Programs," *Ph.D. Dissertation, Technical Report No. 84-05, Departments of Applied Mathematics and Computer Science, University of VA, Charlottesville, VA, 1984.*
- [Case78] R.P. Case and A. Pageds, "Architecture of the IBM System 370," *Communications of the ACM*, Vol. 21, No. 1, 1978, pp. 73-96.
- [Chan75a] K.M. Chandy, U. Herzog, and L. Woo, "Parametric Analysis of Queueing Networks," *IBM Journal of Research and Developments*, Vol. 19, No. 1, January 1975, pp. 36-42.
- [Chan75b] K.M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queueing Networks," *IBM Journal of Research and Development*, Vol. 19, No. 1, January 1975, pp. 43-49.

- [Chan77] K.M. Chandy, J.H. Howard, and D.F. Towsley, "Product Form and Local Balance in Queueing Networks," *JACM*, Vol. 24, No. 2, April 1977, pp. 250-263.
- [Chen83] S.C. Chen, *Large-Scale and High-Speed Multiprocessor System for Scientific Applications: CRAY X-MP Series*, Julich, West Germany: Springer Verlag, June 20-22, 1983.
- [Chen80] T.C. Chen, "Overlap and Pipeline Processing," *Introduction to Computer Architecture. Chapter 9 (Stone Editor)*, 1980, pp. 427-486.
- [Chu80] W.W. Chu, L.J. Holloway, M.T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *IEEE Computer Magazine*, Vol. 13, November 1980, pp. 57-69.
- [Cobh54] A. Cobham, "Priority Assignments in Waiting Line Problems.," *Operations Research*, Vol. 2, 1954, pp. 70-76.
- [Coop70] R.B. Cooper, "Queues Served in Cyclic Order: Waiting Times," *Bell System Technical Journal*, Vol. 49, 1970, pp. 399-413.
- [Corp73] Control Data Corporation, "Control Data STAR-100 Features Manual," *Publication Number 60425500*, October 1973.
- [Corp80] Control Data Corporation, *CDC Cyber 200/Model 205 Technical Description*, November 1980.
- [Corp76] IBM Corporation, "IBM 3838 Array Processor Functional Characteristics," *Publication Number 6A24-3639-0, File Number S370-08*, October 1976.
- [Daig81] J.N. Daigle and C.E. Houstis, "Analysis of a Task Oriented Multipriority Queueing System," *IEEE Transactions On Communications*, Vol. COM-29, No. 11, November, 1981, pp. 1669-1677.
- [Dick82] A. Dickinson, "Optimizing Numerical Weather Forecasting Models for the CRAY-1 and CYBER-205 Computers," *Computer Physics Communications*, Vol. 26, 1982, pp. 459-468.
- [Dorr78] F.W. Dorr, "The Cray-1 at Los Alamos," *Datamation*, October 1978, pp. 113-120.

- [Dosh83] B.T. Doshi, "Stochastic Decomposition in a M/G/1 Queue with Vacations," *Bell Laboratories*, 1983.
- [Efe82] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed systems," *IEEE Computer Magazine*, Vol. 15, June 1982, pp. 50-56.
- [Faro83] R.T Farouki, S.L. Shapiro, and S.A. Teukolsky, "Computational Astrophysics on the Array Processor," *IEEE Computer Magazine*, Vol. 16, No. 6, June 1983, pp. 73-84.
- [Fayo78] G. Fayolle, I. Iasnogorodski, and I Mitrani, "On the Sharing of a Processor Among Many Job Classes," *IRIA-Laboria, Domaine de Voluceau, 78150 Le Chesnay, France*, 1978.
- [Fell57] W. Feller, *An Introduction to Probability Theory and its Applications, Vol 1 (second Edition)*: John Wiley & Sons, Inc., 1957.
- [Fell66] W. Feller, *An Introduction to Probability Theory and its Applications, Vol 2 (second Edition)*: John Wiley & Sons, Inc., 1966.
- [Feng77] T. Feng, "Editor's Introduction, Special Issue on Parallel Processors and Processing," *Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 1-2.
- [Feng74] T.Y. Feng, "Data Manipulation Functions in Parallel Processors and Their Implementations," *IEEE Transactions on Computers*, Vol. C-23, No. 3, March 1974, pp. 309-318.
- [Flat84] L. Flatto and S. Hahn , "Two Parallel Queues Created by Arrivals with Two Demands I," *SIAM J. Appl. Math.*, Vol. 44, No. 5, October 1984, pp. 1041-1053.
- [Flat85] L. Flatto, "Two Parallel Queues Created by Arrivals with Two Demands II," *SIAM J. Appl. Math.*, Vol. 45, No. 5, October 1985, pp. 861-878.
- [Flynn66] M.J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, Vol. 54, 1966, pp. 1901-1909.
- [Fors83] K.S. Forsstrom, "Array Processors in Real-Time Flight Simulation," *IEEE Computer Magazine*, Vol. 16, No. 6, June 1983, pp. 62-72.
- [Fuhr83] S.W. Fuhrmann, "A Note on the M/G/1 Queue with Server Vacations," *Bell Laboratories*, 1983.

- [Gail83] R. Gail, "On the Optimization of Computer Network Power," *PhD Dissertation, School of Engineering and Applied Science, Computer Science Department, University of California, Los Angeles*, September 1983.
- [Gay75] T.N. Gay and P.H. Seaman, "Composite Priority Queue," *IBM Journal Research and Development*, Vol. 19, 1975, pp. 78-81.
- [Gies78] A. Giessler, J. Hanle, A. Konig, and E. Pade, "Free Buffer Allocation - An Investigation by Simulation," *Computer Networks*, Vol. 1 (3), July 1978, pp. 191-204.
- [Grah70] W.R. Graham, "The Parallel and the Pipeline Computers," *Datamation*, April 1970, pp. 68-71.
- [Gree80] L. Green, "A Queueing System in Which Customers Require a Random Number of Servers," *Operations Research*, Vol. 28, No. 6, November-December 1980.
- [Haje83] B. Hajek, "The Proof of a Folk Theorem on Queueing Delay with Applications to Routing in Networks," *Journal of the ACM*, Vol. 30, 1983, pp. 834-851.
- [Hall72] T.G. Hallin and M.J. Flynn, "Pipelining of Arithmetic Functions," *IEEE Transactions on Computers*, August 1972, pp. 880-886.
- [Hand77] W. Handler, "The Impact of Classification Schemes On Computer Architecture," *Proceedings of the International Conference On Parallel Processing*, 1977, pp. 7-15.
- [Hans75] P.B. Hansen, "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 199-207.
- [Hayn82a] L.S. Haynes, "Highly Parallel Computing," *IEEE Computer Magazine*, Vol. 15, No. 1, January 1982, pp. 7-8.
- [Hayn82b] L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell, "A Survey of Highly Parallel Computing," *IEEE Computer Magazine*, Vol. 15, No. 1, January 1982, pp. 9-24.
- [Heid82] P. Heidelberger and K.S. Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks," *IEEE Transactions on Computers*, Vol. C-31, No. 11, November 1982, pp. 1099-1108.

- [Heid83] P. Heidelberger and S.K. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency," *IEEE Transactions on Communications*, Vol. 32, No. 1, January 1983, pp. 73-82.
- [Heym77] D.P. Heyman, "The T-Policy for the M/G/1 Queue," *Management Science*, Vol. 23, 1977, pp. 775-778.
- [Hill85] W.D. Hillis, *The Connection Machine*, Cambridge, Massachusetts USA: The MIT Press, 1985.
- [Hint72] R.G. Hintz and D.P. Tate, "Control Data STAR-100 Processor Design," *Proceedings of the COMPCON*, September 1972, pp. 1-4.
- [Hoar78] C.A.R. Hoare, "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21(8), August 1978.
- [Humb82] P. Humblet, "Determinism Minimizes Waiting Times in Queues," *Technical Report, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*, 1982.
- [Hwan84] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*: McGraw Hill (Series in Computer Organization and Architecture), 1984.
- [IEEE83a] IEEE, "Special Issue On Computerized Tomography," *Proceedings of the IEEE*, Vol. 71, No. 3, March 1983, pp. 289-448.
- [IEEE83b] IEEE, "Tomorrow Computers: Next Generation," *IEEE Spectrum Magazine*, Vol. 20, No. 11, November 1983.
- [IEEE84] IEEE, "Special Issue On Seismic Signal Processing," *Proceedings of the IEEE*, Vol. 72, No. 10, October 1984, pp. 1233-1424.
- [Jack63] J.R. Jackson, "Jobshop-like Queueing Systems," *Management Science* 10, 1963, pp. 131-142.
- [Jaco83] P.A. Jacobson and E.D. Lazowska, "The Method of Surrogate Delays - Simultaneous Resource Possession in Analytic Models of Computer Systems," *ACM Sigmetrics conf. on Measurement and Modeling of Computer Systems Minneapolis, Minnesota*, August 29-31, 1983.

- [Jais68] N.K. Jaiswal, *Priority Queues*, New York: Academic Press, 1968.
- [Kana85] H. Kanakia and F.A. Tobagi, "Theoretical Results on Distributed Processing with Limited Computing Resources," *Stanford University, SEL Technical Report No. 85-273*, April 1985.
- [Klei67] L. Kleinrock, "Time-Shared Systems : A Theoretical Treatment," *Journal of the Association of Computer Machinery (JACM)*, Vol. 14, 1967, pp. 242-261.
- [Klei75] L. Kleinrock, *Queueing Systems, Vol 1: Theory*, New York: Wiley-Interscience, 1975.
- [Klei76] L. Kleinrock, *Queueing Systems, Vol 2: Computer Applications*, New York: Wiley-Interscience, 1976.
- [Klei78a] L. Kleinrock, "On Flow Control in Computer Networks," *Conference Records, International Conference on Communications*, Vol. 2, June 1978, pp. 27.2.1-27.2.5.
- [Klei78b] L. Kleinrock and Y. Yemini, "On Optimal Adaptive Scheme for Multiple Access Broadcast Communication," *Conference Records, International Conference on Communications*, June 1978, pp. 7.2.1-7.2.5.
- [Klei79] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," *Conference Record, International Conference on Communications*, June 1979, pp. 43.1.1-43.1.10.
- [Klei84] L. Kleinrock, "On The Theory of Distributed Processing," *Twenty-second Annual Allerton Conference on Communication, Control, and Computing*, October 3, 1984.
- [Koba77] H. Kobayashi and A.C. Konheim, "Queueing Models for Computer Communications systems analysis," *IEEE Transactions on Communications*, Vol. COM-25, January, 1977, pp. 2-29.
- [Kowa85] J.S. Kowalik, *Parallel MIMD Computation: HEP Supercomputer and its Applications*, Cambridge, Massachusetts London England: The MIT Press, 1985.
- [Kuck68] D.J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Transactions on Computers*, Vol. C-17, No. 8, August 1968, pp. 758-770.

- [Kuck72] D.J. Kuck, Y. Muraoka, and S.C. Chen, "On the Number of Operations Simultaneously Executable in Fortran-Like Programs and Their Resulting Speedup," *IEEE Transactions on Computers*, Vol. C-21, No. 12, December 1972, pp. 1293-1310.
- [Kuck74] D.J. Kuck, P.P. Budnik, S.C. Chen, D.H. Lawrie, R.A. Towle, R.E. Strebendt, E.W. Davis, Jr., J. Han, P.W. Kraska, and Y. Muraoka, "Measurements of Parallism in Ordinary FORTRAN Programs," *IEEE Computer Magazine*, January 1974, pp. 37-45.
- [Kuck77] D.J. Kuck, "A Survey of Parallel Machine Organization and Programming," *ACM Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 29-59.
- [Kuck82] D.J. Kuck and R.A. Stokes, "The Burroughs Scientific Processor (BSP)," *IEEE Transactions on Computers*, Vol. C-31, No. 5, May 1982, pp. 363-376.
- [Kuck84] D.J. Kuck and et al., "The Effects of Program Restructuring, Algorithm Change and Architecture Choice on Programs," *Proceeding of the International Conference on Parallel Processing*, 1984.
- [Kung84] K.C. Kung, "Concurrency in Parallel Processing Systems," *Ph.D. Dissertation, School of Engineering and Applied Science, Computer Science Department, University of California, Los Angeles*, 1984.
- [Lave83] S.S. Lavenberg, *Computer Performance Modeling Handbook*, New York: Academic Press, 1983.
- [Levy84] H. Levy, *Non-Uniform Structures and Synchronization Patterns in Shared-Channel Communication Networks*: Ph.D. Dissertation, UCLA Computer Science Department, UCLA Los Angeles, August 1984.
- [Levy75] Y. Levy and U. Yechiali, "Utilization of the Idle Time in an M/G/1 Queueing System," *Management Science*, Vol. 22, 1975, pp. 202-211.
- [Lips81] W. Lipski and C.H. Papadimitriou, "A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems," *Journal on Algorithms*, Vol. 2-3, September 1981, pp. 211-226.
- [Litt61] J. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research*, Vol. 9, No. 2, March 1961, pp. 383-387.

- [Liu68] C.L. Liu, *Introduction to Combinatorial Mathematics*, New York: McGraw Hill, 1968.
- [McGr80] J.R. McGraw, "Data Flow Computing - Software Development," *IEEE Transactions on Computers*, December 1980, pp. 1095-1103.
- [McGr82] J.R. McGraw, "The VAL language: Description and Analysis," *ACM Transactions on Programming Languages and Systems*, Vol. 4(1), January 1982, pp. 44-82.
- [Mill64] L. Miller, "Alternating Priorities in Multi-class Queues," *Ph.D. Dissertation, Cornell University, Uthaca, New York*, 1964.
- [Mins71] M. Minsky and S. Papert, *On Some Associative, Parallel and Analog Computations: Associative Information Technologies*, (Elsevier North Holland, New York), 1971.
- [Miur83] K. Mijura and K. Uchida, *FACOM Vector Processor VP-100/VP-200*, Julich, West Germany: Proceedings of the NATO Advanced Research Workshop on High-Speed Computing, (J. Kawalik Editor) Spriger Verlag, June 20-22, 1983.
- [Moll81] M.K. Molloy, "On the Integration of Delay and Throughput Measures in Distributed Processing Models," *Ph.D. Dissertation, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles*, 1981.
- [Moll82] M.K. Molloy, "Performance Analysis Using Stochastic Petri Nets," *IEEE Transactions on Computers*, September 1982.
- [Munt73] R.R. Muntz, "Poisson Departure Processes and Queueing Networks," *Proceedings of the 7th Annual Princeton Conference on Information Science, Princeton University*, 1973, pp. 435-440.
- [Mura71] Y. Muraoka, "Parallelism Exposure and Exploitation in Programs," *Ph.D. Dissertation Technical Report No. 71-424, Department of Computer Science, University of Illinois at Urbana-Champaign*, February 1971.
- [Nels85] R. Nelson and A.N. Tantawi, "Approximate Analysis of Fork/Join Synchronisation in Parallel Queues," *IBM T.J. Watson Research Center, Yorktown Heights, NY 10598*, Vol. RC 11481 (No. 51563), October 30, 1985.

- [O'Do74] T.M. O'Donovan, "Direct Solutions of M/G/1 Processor-Sharing Models," *Operations Research*, Vol. 22, 1974, pp. 570-575.
- [Papa83] C.H. Papadimitriou, "Concurrency Control by Locking," *SIAM Journal on Computing*, Vol. 12-2, May 1983, pp. 215-226.
- [Pete81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, New Jersey: Prentice-Hall, Englewood Cliffs, 1981.
- [Pinc84] S. Pincus, "A Responsive Queueing Model for Attendant Servers," *Bell Comm. Research Inc. West Long Branch, New Jersey*, 1984.
- [Pott83] J.L. Potter, "Image Processing on the Massively Parallel Processor," *IEEE Computer Magazine*, Vol. 16, No. 1, January 1983, pp. 62-67.
- [Rama77] C.V. Ramamourthy and H.F. Li, "Overlap and Pipeline Processing," *ACM Computing Surveys*, Vol. 9, No. 1, Introduction to Computer Architecture, Chapter 9, March 1977, pp. 61-102.
- [Rodr80] G. Rodrigue, E.D. Giroux, and M. Pratt, "Perspectives on Large-Scale Scientific Computation," *IEEE Computer Magazine*, October 1980, pp. 65-80.
- [Rodr82] G. Rodrigue, *Parallel Computations*, New York, London: Academic Press, 1982.
- [Rogo66] B.A. Rogozin, "Some Extremal Problems in Queueing Theory," *Theory of Applied Probability*, Vol. 11, 1966, pp. 144-151.
- [Rose83] A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays," *IEEE Computer Magazine*, Vol. 16, No. 1, January 1983, pp. 14-20.
- [Russ78] R.M. Russell, "The Cray-1 Computer System," *Communications of the ACM*, January 1978, pp. 63-72.
- [Saty80] M. Satyanarayanan, *Multiprocessors: A Comparative Study*, New Jersey: Prentice Hall, Englewood Cliffs, 1980.
- [Sauc81] C.H. Sauer, "Approximate Solution of Queueing Networks with Simultaneous Resource Possession," *IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598*, Vol. RC 8679 (No. 37903), October 30, 1981.

- [Scho76] M. Scholl. "Multiplexing Techniques for Data Transmission of Packet-Switched Radio Systems." *Ph.D. Dissertation Report No. UCLA-ENG-76123, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, December 1976.*
- [Scho83] M. Scholl and L. Kleinrock, "On The M/G/1 Queue with Rest Periods and Certain Service-Independent Queueing Disciplines," *Operations Research*, Vol. 31-(4), July-August, 1983, pp. 705-719.
- [Scho78] F.A. Van der Duyn Schouten, "An M/G/1 Queueing Model with Vacation Times," *Zeitschrift Operations Research*, 1978, pp. 95-105.
- [Seil84] A.F. Seila, "On Waiting Times for a Queue in Which Customers Require Simultaneous Service from a Random Number of Servers," *Operations Research*, Vol. 32, No. 5, September-October 1984.
- [Shan80] J.G. Shanthikumar, "Some Analysis of the Control of Queues Using Level Crossing of Regenerative Processes," *Journal of Applied Probability*, Vol. 17, 1980, pp. 814-821.
- [Simo84] B. Simon, "Priority Queues With Feedback," *Association for Computing Machinery (ACM)*, Vol. 31, No 1, January, 1984, pp. 134-149.
- [Ston78] H.S. Stone and S.H. Bokhari, "Control of Distributed Processes," *IEEE Computer Magazine*, Vol. 11, July 1978, pp. 97-106.
- [Stot82] P.D. Stotts, Jr., "A Comparative Survey of Concurrent Programming Languages," *SIGPLAN Notices*, 17(9), September 1982, pp. 76-87.
- [Stoy84] D. Stoyan, *Comparison Methods for Queues and Other Stochastic Models (D.J.Daley Editor)*, New York: John Wiley & Sons, 1984.
- [Suga80] R. Sugarman, "Superpower Computer," *IEEE Spectrum Magazine*, April 1980, pp. 28-34.
- [Suga83] R. Sugarman and P. Wallich, "The Limits to Simulation," *IEEE Spectrum Magazine*, Vol. 20, No. 4, April 1983, pp. 36-41.
- [Taka83] N. Takahashi and M. Amamiya, "A Dataflow Processor Array System: Design and Analysis," *Proceedings of the 10th International Symposium on Computer Architecture*, June 13-17, 1983, pp. 243-251.

$$\bar{X}_c^2 = \frac{4+3P_e}{2\mu^2} \quad (5.21)$$

$$VAR(\bar{X}_c) = \bar{X}_c^2 - (\bar{X}_c)^2 = \frac{4+P_e}{4\mu^2} [2-P_e] \quad (5.22)$$

We define the utilization of the system $M/G_c/1$, denoted hereafter by τ , to be $\tau = \lambda \bar{X}_c$, and hence by using equation (5.20), we get:

$$\tau = \rho \frac{2+P_e}{2} \quad (5.23)$$

Now we proceed to find the probability P_e . Since the probability of the system being empty must be the same for both the FJ-2-M/M/1 and the $M/G_c/1$ systems, namely:

$$P(\bar{n}=0) = P(\bar{n}_1=0, \bar{n}_2=0) = (1-\rho)^{\frac{3}{2}}$$

and since for an M/GI/1 system $P(\bar{n}=0) = 1-\tau$, we get:

$$P_e = \frac{2(1-\rho)}{\rho} [1-\sqrt{1-\rho}] \quad (5.24)$$

Using the above expression in equation (5.23), yields:

$$\tau = 1 - (1-\rho)^{\frac{3}{2}} \quad (5.25)$$

Recall that the stability condition of the FJ-2-M/M/1 system is $\rho \leq 1$. On the other hand, the stability condition for the $M/G_c/1$ system is $\tau < 1$. Since the two conditions must be equivalent, we must then have $\tau < 1$ for all the permissible values of ρ . Indeed, equation (5.25) gives:

$$\lim_{\rho \rightarrow 0} \tau = 0 \quad \lim_{\rho \rightarrow 1} \tau = 1 \quad (5.26)$$

and equation (5.24) gives:

$$\lim_{\tau \rightarrow 0} P_e = 1 \quad \lim_{\tau \rightarrow 1} P_e = 0 \quad (5.27)$$

The above equation stresses the fact that under very light traffic conditions, the service time of a bulk job is distributed as the maximum of two exponentials, whereas under very heavy traffic conditions, this service time is exponentially distributed.

We now proceed to determine the average response time of a bulk job. Let $W(2)$ denote the average waiting time of a bulk job in the $M/G_c/1$ system. From M/GI/1 theory [Klei75], we have $W(2) = \frac{W_0}{1-\tau}$, where $W_0 = \frac{\lambda \bar{X}_c^2}{2}$, therefore we obtain:

$$W(2) = \frac{3 \left[1 - (1-\rho)^{\frac{3}{2}} \right]}{2\mu(1-\rho)^{\frac{3}{2}}}$$

and, since $T(2) = W(2) + \bar{X}_c$, we get:

$$T(2) = \frac{3 \left[1 - (1-\rho)^{\frac{3}{2}} \right]}{2\mu(1-\rho)^{\frac{3}{2}}} + \frac{1 - (1-\rho)^{\frac{3}{2}}}{\lambda} \quad (5.28)$$

Let $N(2)$ denote the average number of bulk jobs in the $M/G_c/1$ system; using Little's result [Litt61], we obtain:

$$N(2) = 1 - (1-\rho)^{\frac{3}{2}} + \frac{3\rho \left[1 - (1-\rho)^{\frac{3}{2}} \right]}{2(1-\rho)^{\frac{3}{2}}}$$

Interpretation of the Results

For the $M/G_c/1$ system, the formulae that involve only the average service time of a bulk job are exact, such as the utilization of the system given by (5.25) and the probability P_e of having $\bar{n}_b=0$ just after a bulk job departure from the system, given by formula (5.24). What make the previous analysis an approximation are the second and higher moments of the bulk job service time distribution that assumes a geometric behavior with parameter P_e , as indicated by (5.17). Unfortunately, the average waiting time of a bulk job, and consequently the bulk job response time and the average number of bulk jobs in the system, depend on the second moment of the bulk job service time distribution. In the real system, as depicted in Figure 5.2:(b), the positive correlation between consecutive bulk job service times tends to decrease the variance of such service times (or equivalently the second moment since the average bulk job service time is the same in both systems). Consequently, we expect the approximation to give a higher bulk job response time. Figure 5.3 depicts the average response time of a bulk job given by the approximation, namely formula (5.28), and that of the exact value given by formula (5.9), as a function of the utilization factor ρ . As we expected, the average response time given by formula (5.28) is higher than the one given by formula (5.9). It is interesting to notice that this discrepancy increases with ρ . Figure 5.4, depicts the discrepancy versus ρ ; namely it plots the ratio of the average response time given by (5.9) to the average response time given by (5.28), as a function of the utilization factor ρ . We observe that for small values of ρ , the two formulae give close values, and as ρ approaches unity, the ratio approaches zero. This figure represents how much correlation (information) we lose by representing the bulk job service times as independent identically distributed random variables as given by equation (5.17).

- [Toba85] F.A. Tobagi and H. Kanakia, "A Comparison of Distributed and Centralized Processing Systems," *Int. Sem. on Comp. Net. and Perf. Evaluation, Held at Tojo Kaikan, Tokyo, September 18-20, 1985.*
- [Tows78] D. Towsley, K.M. Chandy, and J.C. Browne, "Models for Parallel Processing Within Programs: Application to CPU:I/O and I/O:I/O Overlap," *Communications of the ACM*, Vol. 21, No. 10, October 1978, pp. 821-831.
- [Tows83] D. Towsley, "An Approximate Analysis of Multiprocessor System," *ACM Sigmetrics conference on Measurement and Modeling of Computer Systems, Minneapolis, Minnesota, August 29-31, 1983.*
- [Tows80] D.F. Towsley, "Queueing Models with State-Dependent Routing," *JACM*, Vol. 27, No. 2, April 1980, pp. 323-337.
- [Wang85] Y.T. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, Vol. C-34, No. 3, March 1985.
- [Wats72] W.J. Watson, "The TI ASC - A Highly Modular and Flexible Super Computer Architecture," *Proceedings of the AFIPS Fall Joint Computer Conference, AFIPS Press, 1972*, pp. 221-228.
- [Wats74] W.J. Watson and H.M. Carr, "Operational Experiences with the TI Advanced Scientific Computer," *AFIPS NCC Proceedings, 1974*, pp. 389-397.
- [Whit85] P.W. White, *Vectorization of Weather and Climate Models for the CYBER-205 (in Super Computer Applications)*, New York, London: Plenum Press, 1985.
- [Whit84] W. Whitt, "Minimizing Delays in the GI/G/1 Queue," *Operations Research*, Vol. 32, 1984, pp. 41-51.
- [Widd80] L.C. Widdoes, "The S-1 Project: Developing High-Performance Digital Computers," *Digest of Papers: IEEE Compcon 80*, Spring 1980, pp. 282-291.
- [Wolf82] R.W. Wolff, "Poisson Arrivals Sees Time Averages," *Operations Research*, Vol. 30, 1982, pp. 223-231.
- [Yann79] M. Yannakakis, C.H. Papadimitriou, and H.T. Kung, "Locking Policies: Safety and Freedom from Deadlock," *Proceedings of the 20th ACM Proceedings on the Foundations of Computer Science, 1979*, pp. 283-287.

[Yosh77]

Y. Yoshioka, T. Nakamura, and R. Sato, "An Optimum Solution of the Queueing System," *Electronics and Communications in Japan*, Vol. 60 (B8), August 1977, pp. 590-591.