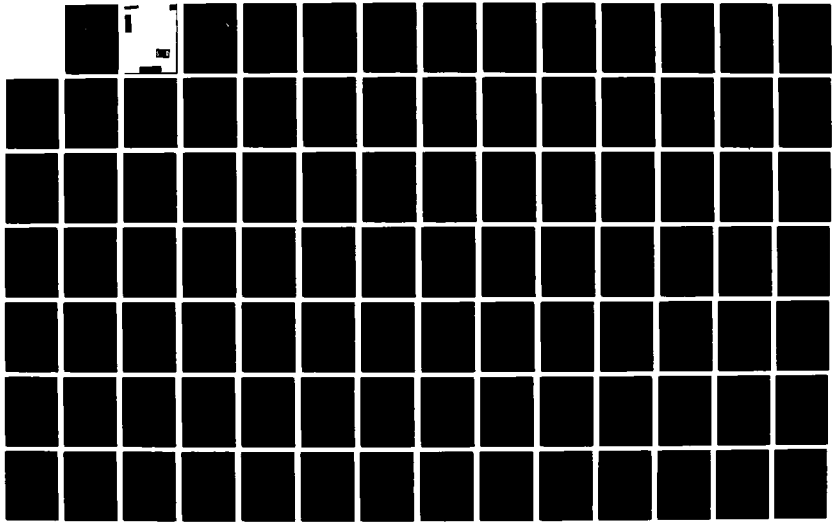
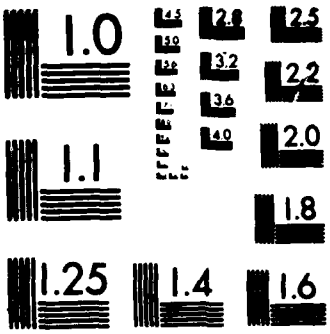
AD-A183 619 MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR (U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE R B OSBORNE MAY 87 MIT/LCS/TR-375 1/4

UNCLASSIFIED N00014-83-K-0125 F/G 12/7 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

MIT/LCS/TR-375

MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR

Randy Brent Osborne

May 1987

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A183619 12

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR-375	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125
6a. NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

DTIC ELECTE JUL 29 1987 S D

11. TITLE (Include Security Classification)
Modeling the Performance of the Concert Multiprocessor

12. PERSONAL AUTHOR(S)
Osborne, Randy Brent

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ____ TO ____	May 1987	307

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
modeling, multiprocessors, Markov chains, queueing theory, Markovian decision theory

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The performance of the Concert Multiprocessor is investigated using probabilistic models. Analysis proceeds by decomposing Concert along its natural hierarchies into a Multibus subsystem and a Ringbus subsystem. Each subsystem is modeled in isolation ignoring the interactions between subsystems.

A series of Multibus models is developed based on a very simple processor model and some simplifying assumptions. These models are analyzed using Markov chains and queueing theory. Ways to relax some of the assumptions and treat more general processor models are discussed.

The Ringbus is a novel and previously unanalyzed interconnection scheme which is of independent interest. Analysis of the Ringbus subsystem concentrates on a general version of the Ringbus which lacks the topological constraints of the Ringbus actually employed in Concert. The determination of the optimum throughput of the Ringbus and associated optimum arbiter algorithm is formulated as a Markovain decision problem. This problem is

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little, Publications Coordinator	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

U.S. Government Printing Office 1985-607-047

Unclassified

87 7 22 054

19. solved for various cases as the available computational resources allowed. Approximations, bounds, and a few general results are derived for more computationally intensive cases. The version of the Ringbus used in Concert is also analyzed for a simple case. Finally, the simplifying assumptions made in analyzing the Ringbus are considered.

The subsystem models can be integrated to produce an overall model by matching the first moments of the interactions between subsystems. This integration was performed for several cases. The results obtained via integration agreed well with those obtained via simulation.

Two sets of simulations are presented. The first set compares the performance of Ringbus architectures and arbiter algorithms in an environment close to that in the actual Concert system and presents results for some cases for which the Markovian decision theory problem is too computationally expensive to solve. The second set presents the expected performance of the actual Concert system for different parameter values.

12

MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR

by

RANDY BRENT OSBORNE

This research was supported by the Defense Advanced Research Projects Agency and was monitored by the Office of Naval Research under contract number N00014-83-K-0125.

Modeling the Performance of the Concert Multiprocessor

by

Randy Brent Osborne

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 1986 in partial fulfillment of the
requirements for the Degree of Master of Science in
Electrical Engineering and Computer Science

ABSTRACT

The performance of the Concert Multiprocessor is investigated using probabilistic models. Analysis proceeds by decomposing Concert along its natural hierarchies into a Multibus subsystem and a Ringbus subsystem. Each subsystem is modeled in isolation ignoring the interactions between subsystems.

A series of Multibus models is developed based on a very simple processor model and some simplifying assumptions. These models are analyzed using Markov chains and queueing theory. Ways to relax some of the assumptions and treat more general processor models are discussed.

The Ringbus is a novel and previously unanalyzed interconnection scheme which is of *independent interest*. Analysis of the Ringbus subsystem concentrates on a general version of the Ringbus which lacks the topological constraints of the Ringbus actually employed in Concert. The determination of the optimum throughput of the Ringbus and associated optimum arbiter algorithm is formulated as a Markovian decision problem. This problem is solved for various cases as the available computational resources allowed. Approximations, bounds, and a few general results are derived for more computationally intensive cases. The version of the Ringbus used in Concert is also analyzed for a simple case. Finally, the simplifying assumptions made in analyzing the Ringbus are considered.

The subsystem models can be integrated to produce an overall model by matching the first moments of the interactions between subsystems. This integration was performed for several cases. The results obtained via integration agreed well with those obtained via simulation.

Two sets of simulations are presented. The first set compares the performance of some Ringbus architectures and arbiter algorithms in an environment close to that in the actual Concert system and presents results for some cases for which the Markovian decision theory problem is too computationally expensive to solve. The second set presents the expected performance of the actual Concert system for different parameter values.

Name and title of thesis supervisor:

Robert H. Halstead, Jr.
Associate Professor of Electrical Engineering and Computer Science

Keywords and phrases:

Modeling, multiprocessors, Markov chains, queueing theory, Markovian decision theory

To my parents
and Kathrin

Acknowledgments

I wish to thank the following people:

Domenic Costanzino, for giving me his application for the EECS graduate program at MIT;

Robert Donaldson, for encouraging me to apply to MIT;

Lakshmi Dasari, for convincing me (somewhat) that Cambridge wasn't the hole that I imagined;

Tom Sterling, for introducing me to the Real Time Systems Group and Bert Halstead; and

Bert Halstead, for allowing me to pursue my interests.

Table of Contents

1. Introduction

1.1 Introduction 17

1.2 The Concert Multiprocessor 20

1.2.1 Multibus 24

1.2.2 RIB 25

1.2.3 Ringbus Arbiter 27

1.3 Modeling Details 30

1.3.1 Processor Model 30

1.3.2 Major Assumptions 34

1.3.3 Factors for Study 34

1.3.4 Overall Performance Metric 35

1.3.5 Decomposition and Integration 35

1.4 Previous Work 40

1.5 Overview of Thesis 41

2. Multibus Models

2.1 Introduction 43

2.2 Deterministic Model 47

2.3 Probabilistic Model - General Behaviour 50

2.4 Exponential Distributed Processing and Service Times - M/M/1//N Model 53

2.5 Exponential Distributed Processing and General Service - M/G/1//N Model 56

2.5.1 Exponential Distributed Processing and Deterministic Service - M/D/1//N Model 61

2.5.2 Comments 65

2.6 General Processing and Exponential Access Time Distributions - G/M/1//N Model 68

2.6.1 Product Form Solutions 68

2.6.2 G/M/1//N Model as a Queueing Network 73

2.7 General Processing and Access Time Distributions - G/G/1//N 75

2.7.1 Mean Waiting Time in PH/PH/1//N Model 75

2.8 Multibus Model with Long Word Accesses 86

2.8.1 Analysis of Model with Long Word Accesses 88

2.8.1.1 Asymptotic Behaviour 88

2.8.1.2 Deterministic Behaviour 90

2.8.1.3 Product Form Solution 94

2.8.1.4 Simulations 98

2.9 Multibus Model with Long Word and Ringbus Accesses 105

2.9.1 Analysis of Multibus Model with Long Word and Ringbus Accesses 107

2.9.1.1 Asymptotic Behaviour 107

2.9.1.2 Deterministic Behaviour 108

2.9.1.3 Product Form Solution 111

2.9.1.4 A Special Case 113

2.9.2 The Single Processor Equivalent of the Multibus 113

2.10 Extensions 117

2.10.1 Non-identical processors 117

2.10.2 More Complex Processor Models 117

2.10.3 Time Dependent Behaviour 123

2.10.4 Dependent Processors 127

2.11 Conclusions 130

2.12 Future Work Required 131

3. The Ringbus Model

3.1 Introduction 133

3.2 Ringbus Model Formulation 139

3.2.1 Markovian Decision Formulation 141

3.3 Optimal Arbiter for Four Slices and Grant Duration of One Round 147

3.4 Optimal Arbiter for Four Slices and Grant Duration Greater than One Round 163

3.4.1 Deterministic Grant Duration of 2, 3, and 4 Rounds 166

3.4.2 Deterministic Grant Duration: The General Case 169

3.4.2.1 General Characteristics of Optimum Throughput 169

3.4.2.2 Bounds on the Optimum Throughput with Deterministic Grant Duration 170

3.4.2.3 Approximating the Optimum Throughput with Deterministic Grant Duration 173

3.4.3 Geometric Grant Duration 175

3.4.4 Other Grant Duration Distributions 177

3.5 Optimal Arbiter for Six Slices and Grant Duration of One Round 178

3.5.1 Bounds on the Optimal Throughput 188

3.5.1.1 Flow Model Bound 188

3.5.1.2 Crossbar Bound 189

3.5.1.3 Number of Segments Bound 192

3.5.1.4 Discussion 194

3.6 Optimal Arbiter for Eight Slices 195

3.6.1 General Characteristics of the Optimum Throughput 198

3.6.1.1 Slope for Very Light Traffic 198

3.6.1.2 Shape Along a Ray with Fixed Ratio of Nonnull Probabilities 199

3.6.1.3 Maximum Points 199

3.6.1.4 Shape Along Cross Sections 199

3.6.2 Number of Segments Bound 199

3.7 The Symmetric Ringbus With More Than Eight Slices 202

3.7.1 Throughput as a Function of the Number of Slices for Some Special Cases 202

3.8 The Performance of the Concert Ringbus 208

3.8.1 The Effect of Asymmetrical Access Paths 209

3.8.2 The Effect of the Rotating Priority Arbitration Algorithm 209

3.8.3 The Effect of Asymmetrical Access Paths and the Rotating Priority Arbitration Algorithm 215

3.9 The Ringbus in the Concert Environment 219

3.9.1 The Equivalent Model of the Ringbus 221

3.10 Conclusions 227

3.11 Suggestions for Future Work 231

4. Integration and Simulation

4.1 Introduction 233

4.2 Integration 233

4.3 Simulation I: The Ringbus in the Concert Environment 247

4.4 Simulation II: The Actual Concert System 264

5. Conclusions 269

Appendix A - Measurement Details 271

Appendix B - Proofs 289

References 305

List of Figures

Figure 1.1: Top level view of Concert 20

Figure 1.2: An example of simultaneous connections on Ringbus 21

Figure 1.3: The Concert Multiprocessor 23

Figure 1.4: Multibus arbitration signals 24

Figure 1.5: RIB access paths 25

Figure 1.6: Access paths to neighbouring RIB 26

Figure 1.7: Logic diagram of arbiter 28

Figure 1.8: Ringbus arbiter timing 29

Figure 1.9: One cycle of a processor 30

Figure 1.10: Processor model 32

Figure 1.11: Alternate processor model 33

Figure 1.12: Subsystem definitions 36

Figure 1.13: Abstract view of Concert 37

Figure 1.14: Decomposition into models 38

Figure 2.1: Finite customer queueing system 45

Figure 2.2: Deterministic model of Multibus 49

Figure 2.3: t_u/t_d vs. N for general probabilistic case 53

Figure 2.4: Markov chain of M/M/1//N model 53

Figure 2.5: $M/M/1/N$ model of Multibus 55

Figure 2.6: Canonic ladder arrangement of stages 57

Figure 2.7: M/D/1//N model of Multibus 63

Figure 2.8: Various r stage Erlangian density functions 67

Figure 2.9: Queueing network for G/M/1//N model 73

Figure 2.10: State transition diagram for $E_3/E_3/1//3$ system 79

Figure 2.11: Generator matrix for $E_3/E_3/1//3$ example 80

Figure 2.12: Generator matrix for $PH_3/PH_3/1//3$ system 80

Figure 2.13: Basic Multibus model 86

Figure 2.14(a): Extended Multibus model 86

Figure 2.14(b): Class transition diagram of extended Multibus model 87

Figure 2.15: Representative cases of $\bar{t}_{w_i}/\bar{t}_m$ vs. N in deterministic case 92

Figure 2.16: Queueing network model of Multibus with long word accesses
Processing time: general, access time: exponential, $\gamma = .62$ 99

Figure 2.17: Queueing network model of Multibus with long word accesses
Processing time: Erlangian (E_3), access time: deterministic, $\gamma = .62$ 101

Figure 2.18: Queueing network model of Multibus with long word accesses
Processing time: exponential, access time: deterministic, $\gamma = .62$ 102

Figure 2.19: Queueing network model of Multibus with long word accesses
Processing time: hyperexponential (H_3), access time: deterministic, $\gamma = .62$ 103

Figure 2.20(a): Multibus model with Ringbus accesses 105

Figure 2.20(b): Class transition diagram 106

Figure 2.21: Representative cases of $\bar{t}_{w_i}/\bar{t}_m$ vs. N in deterministic case 109

Figure 2.22: Queueing network model with processor activities 119

Figure 2.23: A simple Petri net 129

Figure 3.1: Access paths of Concert Ringbus and Symmetric Ringbus 136

Figure 3.2: Logic diagram of Symmetric Ringbus *arbiter* 138

Figure 3.3: Examples of r_i 142

Figure 3.4: Rotational and flip symmetry 147

Figure 3.5: Optimum average number of grants per round for
Symmetric Ringbus with four slices and one round grant duration 149

Figure 3.6: Some possible decisions in states 20, 34, and 40 150

Figure 3.7: Leftover sets associated with request subsets
(a) and (b) for each of the three states in Figure 3.6 151

Figure 3.8: Some possible decisions in state 38 152

Figure 3.9: Optimum decision regions for states 20, 34, 38, and 40 153

Figure 3.10: Leftover sets from request subsets (a) and (b) in states 20, 34, and 40 157

Figure 3.11: Optimum decision regions for states 20, 34, and 40 159

Figure 3.12: Optimum decision region for state 20 160

Figure 3.13: Average number of grants per round with four slices
and one round grant duration 162

Figure 3.14: Average number of grants in progress per round for
deterministic grant durations of 1, 2, 3, and 4 rounds for $S = 4$ 167

Figure 3.15: Exponential approximation and actual values of optimum gain along some rays for $d = 3$ and 4 and $S = 4$ 174

Figure 3.16: Average number of grants in progress per round for various geometrically distributed grant durations 176

Figure 3.17: Optimal throughput (average number of grants per round) for Ringbus with six slices and one round grant durations 179

Figure 3.18: An example of a request set for which the estimated optimal decision is to grant less than the maximum number of requests 180

Figure 3.19: Optimal throughput, subject to the maximum reward constraint, with six slices and one round grant durations 182

Figure 3.20: Optimal throughput, subject to the maximum number of segments constraint, with six slices and one round grant durations 184

Figure 3.21: Some possible decisions 186

Figure 3.22: Optimal throughput for crossbar and Ringbus, each with six slices and one round grant durations 191

Figure 3.23: Optimal throughput of the number of segments model 193

Figure 3.24: Optimal throughput of Ringbus with eight slices and one round grant durations along axes of feasible probability region 196

Figure 3.25: A crossbar cell 205

Figure 3.26: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1} = p_1$ 210

Figure 3.27: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1} = .5p_1$ 211

Figure 3.28: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1} = 0$ 212

Figure 3.29: Throughout with rotating priority algorithm and optimum throughput, both with symmetrical access paths, for four slices and one round grant duration with $p_{-1} = p_1$ 213

Figure 3.30: Example of a disadvantage of the rotating priority algorithm 214

Figure 3.31: Staggered request probabilities 214

Figure 3.32: A comparison of throughputs for four slices and one round grant duration with $p_{-1} = p_1$ 216

Figure 3.33: A comparison of throughputs for four slices and one round grant duration with $p_{-1} = p_1$ 217

Figure 3.34: A comparison of throughputs for four slices and one round grant duration with $p_{-1} = p_1$ 218

Figure 3.35: Point of view of single processor equivalent 221

Figure 3.36: Point of view of Ringbus 222

Figure 3.37: Combined points of view of single processor equivalent and Ringbus 223

Figure 3.38: Signals when $\bar{t}_p^{MBeqv}$ large 225

Figure 3.39: Signals when $\bar{t}_p^{MBeqv} = 0$ 225

Figure 4.1: Two situations leading to a null request 242

Figure A.1: Byte and word access 272

Figure A.2: Long word access 272

Figure A.3: Illustration of definitions 274

Figure A.4: Multibus access time - other port unloaded 277

Figure A.5: Multibus access time - HSB port loaded 278

Figure A.6: Multibus access time of slice global memory - Ringbus port loaded 278

Figure A.7: Typical Ringbus read access 279

Figure A.8: Typical Ringbus read access with $t_p^{MBeqv} = 0$ 281

Figure A.9: Ringbus read access time distribution 284

Figure A.10: Ringbus access time distribution - t_p^{MBeqv} large and other port loaded 286

List of Tables

Table 3.1: States after symmetry extraction 148

Table 3.2: Rewards and transition probabilities for decisions (a) and (b) in states 20, 34, and 40 157

Table 3.3: Results from number of segments model for eight slices 200

Table 3.4: t^{cw-ccw} for various values of S 204

Table 4.1: Integration example 1 240

Table 4.2: Integration example 2 243

Table 4.3: Integration example 3 244

Table 4.4: Integration example 4 245

Table 4.5: Ringbus destination probabilities 248

Table 4.6: Ringbus simulation results 251

Table 4.7: Percent increase in throughput for $\bar{t}_p = 5.0c$ relative to that for rotating priority arbiter algorithm with asymmetrical access paths 260

Table 4.8: Saturation throughput for various algorithms and destination probabilities 261

Table 4.9: Concert simulation results 265

Table 4.10: Necessary condition for Ringbus saturation in the actual Concert system 268

Chapter 1

Introduction

1.1 Introduction

The goal of the research reported in this thesis is to model the performance of the Concert Multiprocessor [A2] in order to answer the following questions:

1. What is the performance of the system as designed and built with respect to some metric?
2. Why is the performance as it is? What factors influence the performance, what is the sensitivity of the performance to these factors, and what are the limitations of the system design?
3. How can the performance be improved and where should the design be modified to achieve this improvement? What are the critical sections and bottlenecks in the design?

An answer to the first question satisfies a natural curiosity; an answer to the second gives users ideas how to structure programs and applications to achieve the best possible performance of the Concert system; and finally, an answer to the third indicates how to achieve better performance in future designs. Another outcome of the work described herein is that it provides a starting point for future modeling efforts. The experience and knowledge gained through this research can be used to guide the development and applications of higher level and/or more complex models.

The performance metric used in this research is the throughput of the system. This metric is simple and yet represents the basic goal of multiprocessor systems. However, throughput is a rather crude metric to use when comparing the performance of different systems because structural and organizational differences often cause the definition of throughput to differ. Fortunately, the main use of throughput in this thesis is to gauge the change in performance due to variations in the parameters of the system or due to small modifications of the system. Throughput is well suited for this kind of study.

The system is modeled at the memory access level and thus throughput in this case is the average number of memory accesses per unit time. Each processor is assumed to spend most of its time accessing its associated local memory. (Organization of the system will be discussed in detail in the next chapter.) The processor model employed is the simplest imaginable in such a case: the processor spends some time performing local processing after which it makes a non-local memory access (for which it may have to wait for bus mastership) and then resumes local processing. The operation of each processor is assumed to be independent of that of all other processors. The reasons for the memory access modeling level, the simple processor model, and the assumption of independent processors are the same: at this point in time not enough is known about the languages, programming models, programs, and applications to obtain more detailed models. Furthermore, the Concert system is designed to be a testbed for the examination of many different multiprocessor ideas. The common denominator of all Concert applications is the system itself and that is where this research is focused. The basic premise of this research is to start with some very simple models, develop them fully, evaluate them, and then determine how the models can be improved. Complexity is always easy to add to models, sometimes to the point that they become unwieldy, but it is more difficult to add complexity in such a way that keeps the models simple but accurate. Thus this thesis should be considered as the first step in an iterative cycle to obtain models incorporating additional features such as processor dependencies, language issues, and programming models.

Because of the size and complexity of the Concert Multiprocessor, direct modeling of the system - even with the simple processor model - would be a formidable task. The approach taken in the sequel is to decompose the system into subsystems along the lines of the system's natural hierarchies. Each subsystem is analyzed in detail and then all the subsystem models are integrated to determine the performance of the total system. Analytical models are used for each subsystem. The functional equations associated with analytical models allow easy prediction and quick evaluation of the effect of various changes in the model parameters. In short, they allow a lot of ground to be covered in a structured manner and this makes them ideally suited to the first step of the iterative cycle described earlier.

Simulation is employed in this thesis in a few instances where the analytical models become intractable or unmanageable. However, the main use of simulation is to determine the accuracy of the integrated models.

The research described herein started when the author joined the Concert Project just after construction had begun. Thus this work in no way affected the design of the system as described in the next chapter and in Anderson [A2]. The optimum time to begin modeling is during the design stage. Unfortunately only the most rudimentary (and flawed†) simulations were performed

† In his simulation of the Ringbus arbiter, Anderson created a queue of requests for each slice (defined in section 1.2) and terminated the simulation when all the queues emptied. However, if a queue emptied and at least

at that time. Although conducted after the design stage, this research is still extremely useful in answering the three basic questions posed earlier.

This thesis is organized into four chapters and each chapter is divided into sections. The next section in this first chapter describes the Concert Multiprocessor. The section after that presents more details on the modeling level and modeling strategy. The factors considered in this study and the assumptions made are discussed in detail. The final two sections in this chapter briefly discuss previous work in this area and preview the following chapters.

one queue was still nonempty the simulation still ran and still collected statistics with null requests (i.e. the absence of a request) generated for each slice with an empty queue. Thus the statistics were biased by the stream of null requests when a queue emptied.

1.2 The Concert Multiprocessor†

Concert is a tightly-coupled, shared memory multiprocessor. It consists of multiple processors, each executing portions of code, communicating through shared memory to cooperate on the solution of a large task (or tasks). It is classified as a multiple instruction stream, multiple data stream (MIMD) computer [F1].

The Concert Multiprocessor consists of a hierarchy of time-shared (i.e. circuit-switched) buses. At the top level, eight *slices* are interconnected by bus segments as shown in Figure 1.1.

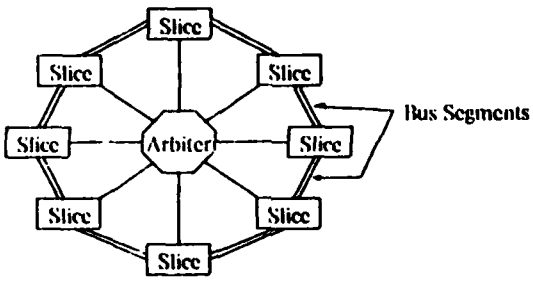


Figure 1.1: Top level view of Concert

Circuitry within each slice connects the two adjacent bus segments either to different internal slice resources or to each other so that all internal slice resources are bypassed. An electrical connection can be established from a resource within one slice - the source - to a resource within a different slice - the destination - by an appropriate connection of the bus segments within the source and destination slices and by joining the bus segments together in all slices between the source and destination. Each bus segment is bidirectional, thus source and destination slices may be connected by a path in either the clockwise or the counterclockwise directions. More than one source-destination connection can be supported simultaneously provided that 1) there is a contiguous connection of segments from each source to its destination, and 2) each bus segment and each slice resource is allocated to at most one source-destination connection. Various simultaneous connections are depicted in Figure 1.2.

† Only the details of the design which are felt to be relevant to the modeling effort in the sequel are discussed here. See Anderson [A2] for more complete information.

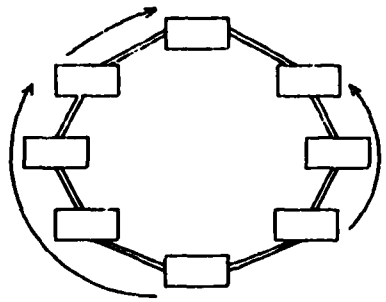


Figure 1.2: An example of simultaneous connections on Ringbus

Note that a maximum of eight simultaneous connections can be supported (e.g. if each slice and its immediate clockwise neighbour comprise a source-destination pair). Once a connection is established from source to destination, that connection is maintained and all the resources involved in that connection remain allocated to only that connection until the source slice no longer requires the connection. A central arbiter, shown in Figure 1.1, controls the allocation and connection of the bus segments. The ring of bus segments shown in Figures 1.1 and 1.2 is called the Ringbus.

Each slice consists of up to eight processor-local memory pairs (one local memory block per processor is the usual, but not necessary, configuration), a global memory block, a time-shared bus called the Multibus, and a Ringbus Interface Board (RIB). Each processor communicates with its local memory over a dedicated bus called the high speed bus (HSB). This bus is private to the processor and independent of the Multibus and other high speed buses. All the processors and memories (both local and global) are also connected to the Multibus. The Multibus, global memory (via a HSB), and the Ringbus segments adjacent to that slice connect to the RIB. Various access paths and circuitry inside the RIB (described in section 1.2.2) allow these items to be interconnected. The resources of a slice that are available for interslice communication can be divided into two mutually exclusive groups: source resources and destination resources. The processors connected to the Multibus are the only source resources. The destination resources consist of the global memory and some global registers (which are inside the RIB).

Only three types of communication, all originated by processors, can occur in the Concert Multiprocessor.† A processor can communicate - i.e. access - its local memory via the HSB, the local memory of other processors on its slice and the global memory of its slice via the Multibus, and the global memory of other slices (and the global registers of its slice) via the Multibus and

† Communication can also be originated by other potential Multibus masters,such as I/O devices. However, these other potential masters essentially behave like processors.

Ringbus. We term these types of accesses HSB, Multibus, and Ringbus accesses respectively. Note that a processor can **not** communicate directly with other processors or the local memory of processors on other slices; such communication must occur through the local or global memory. All bus transactions in Concert are single memory transactions - read, write, or read-modify-write. Successive accesses require establishment of direct bus connections from source processor to destination memory for each access. Thus there is no store and forward mechanism or anything of this kind on the Ringbus or elsewhere.

The structure of a four slice version of the Concert Multiprocessor is illustrated in Figure 1.3. This figure shows all major interconnections within Concert and illustrates some representative accesses from each of the three types of accesses.

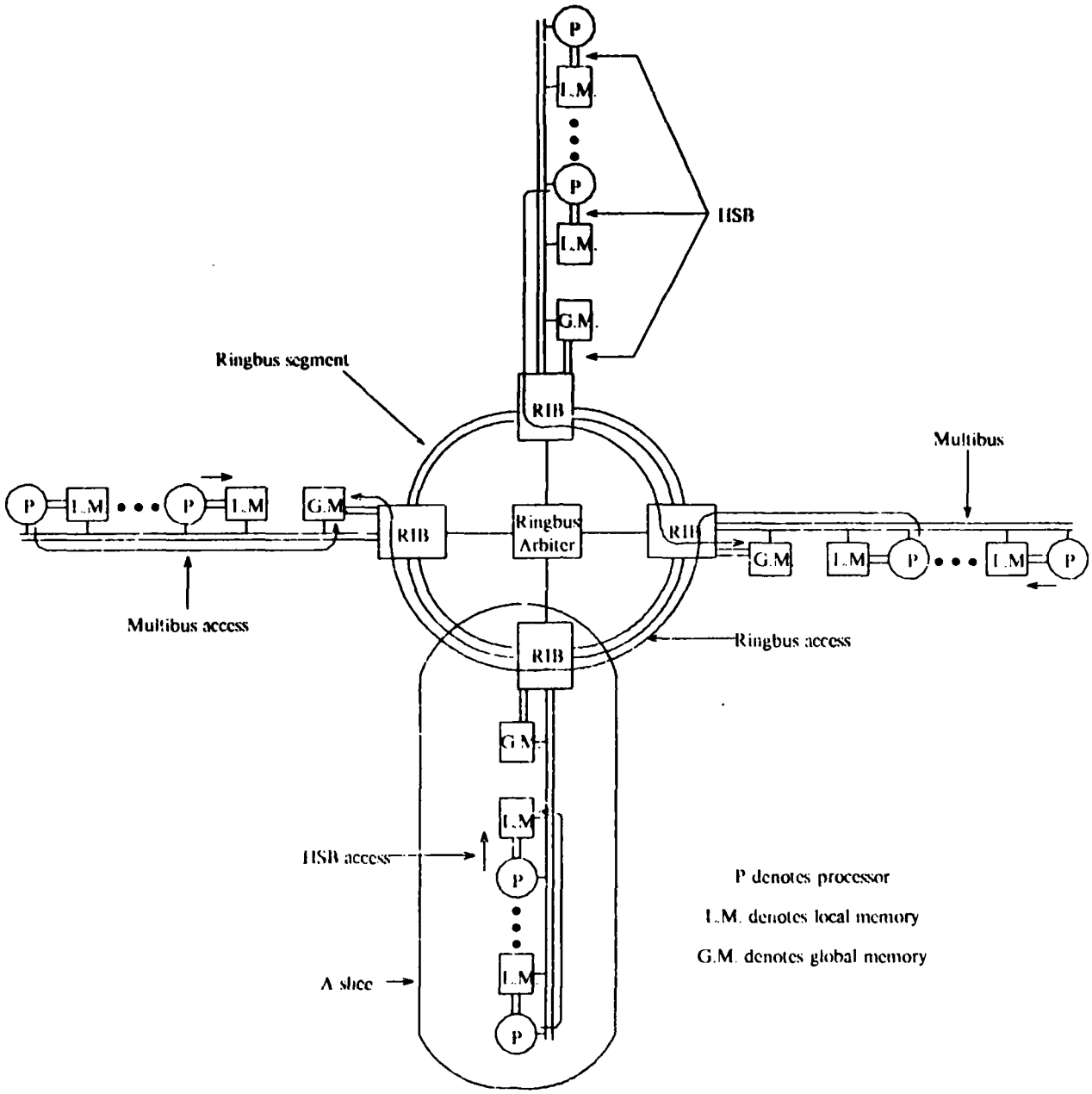


Figure 1.3: The Concert Multiprocessor (only four slices shown)

The Multibus (including high speed buses, processors and memories), RIB, and arbiter are now discussed in more detail.

1.2.1 Multibus

The Multibus is an IEEE 796 standard multi-master bus. An additional bus, which runs parallel to this 796 bus, is physically divided into shorter independent bus segments each of which serves as the high speed bus for a processor. The processors and memories are commercially available dual-ported boards (Microbar Inc. products DBC68K and DBR50 respectively) that each have one Multibus and one HSB port. As described earlier the HSB is private to a processor; thus there is only one processor per HSB. The processors are based on the Motorola MC68000 microprocessor.

When a memory access is initiated, a processor first attempts to access the desired location on the HSB. If this attempt is successful, the memory access proceeds. If it is not successful, the processor accesses the location via the Multibus. Thus a processor accesses its own local memory over its HSB and the local memory of other processors or global memory over the Multibus. Accesses on the HSB take considerably less time then accesses on the Multibus due to the differences between the HSB and Multibus protocols.

Contention for the mastership of the Multibus is resolved by a round-robin arbitration unit. This unit takes a maximum of two Multibus clock cycles (10 MHz clock) as pictured in Figure 1.4.

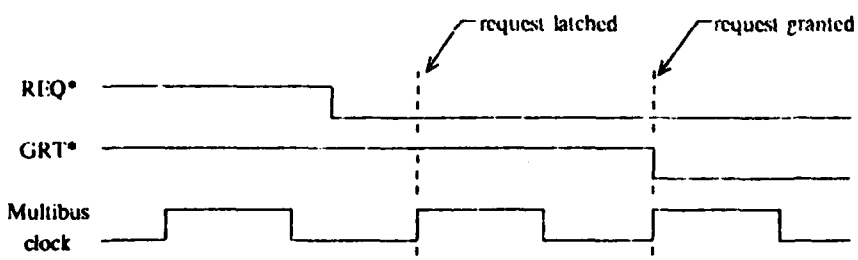


Figure 1.4: Multibus arbitration signals

One cycle is required to latch the request lines and another is required for the arbitration and propagation delay. This arbitration unit grants possession of the bus to a processor for only as long as it takes to complete a single memory access, which cannot exceed 16 bits. The 68000 can perform byte (8 bit), word (16 bit), and long word (32 bit) operations. Long word operations consist of two separate 16 bit accesses; thus a processor must gain control of the bus twice for a long word access. Other processors may seize the bus between these two accesses.

Contention also exists for local memories since a local memory can be addressed simultaneously over a processor's HSB and over the Multibus. This contention is resolved by arbitration circuitry on the dual-ported memory boards.

1.2.2 RIB

When the RIB recognizes a memory access on the Multibus in the Ringbus address space (i.e. a Ringbus access), it decodes the destination slice from the address of the access and sends a request to the Ringbus arbiter for a connection between the Multibus of the source slice and the destination slice. When the Ringbus arbiter grants the request, it directs some number of RIBs to form a path between the source and destination and then it lets the memory access at the source slice proceed.

A diagram of the access paths within the RIB is shown in Figure 1.5. Arrows denote the directionality of the paths and lines perpendicular to a path denote a switch which can be either open or closed.

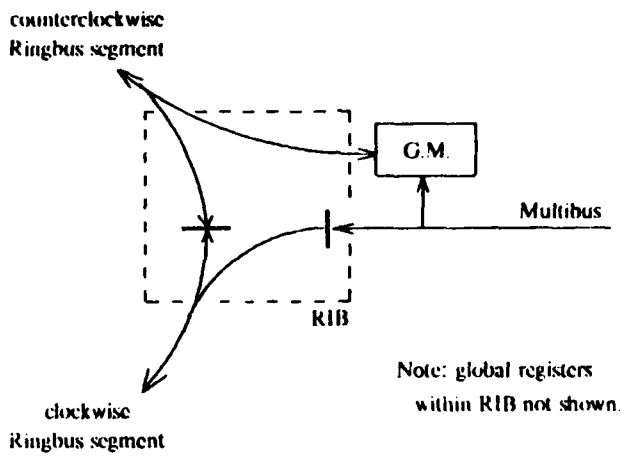


Figure 1.5: RIB access paths

Notice that the Ringbus access paths are asymmetrical. Memory accesses enter the Ringbus on the segment to the clockwise direction of the source RIB and exit via the Ringbus segment to the counterclockwise direction of the destination RIB. This causes the Ringbus to be biased toward memory accesses in the clockwise direction around the Ringbus. As depicted in Figure 1.6, a memory access to a neighbouring RIB in the clockwise direction requires one Ringbus segment compared to three for the neighbouring slice in the the counter clockwise direction. (This last access could also be made in the clockwise direction. For a Ringbus with eight segments, this would require seven segments.)

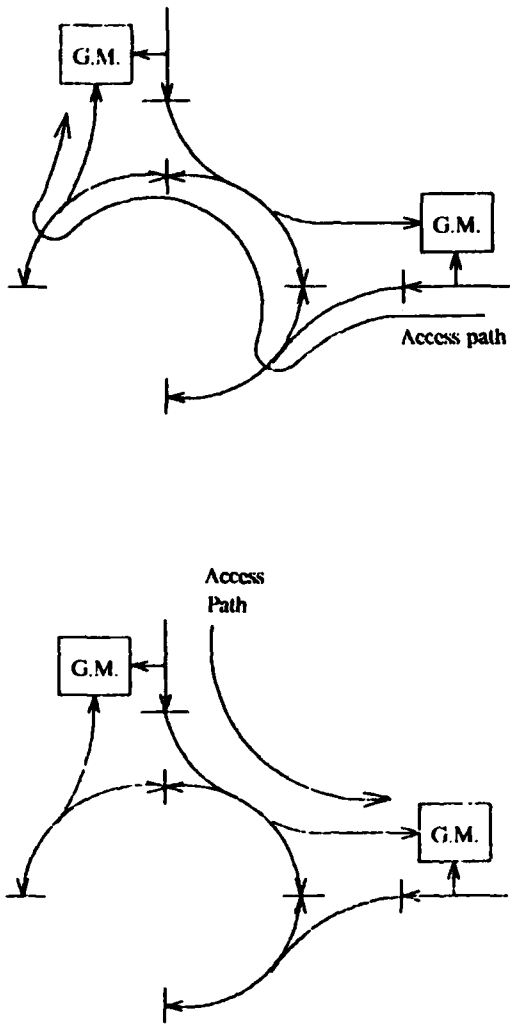


Figure 1.6: Access paths to neighbouring RIB

The asymmetrical access paths clearly reduce the maximum number of accesses that can occur simultaneously on the Ringbus if any of the accesses take place in the counter clockwise direction. The designers of the Concert system felt that the asymmetrical access paths would simplify the Ringbus arbiter (see section 5.2.2 in Anderson [A2]).

The same dual-ported memory boards used for the local memories on the Multibus are used for the global memories. As indicated in Figures 1.3 and 1.5, the Multibus port of the global memory connects directly to the to the Multibus of that slice. The HSB port of the global memory connects to the Ringbus. As before, arbitration circuitry on the global memory board handles simultaneous Multibus and Ringbus accesses to that board. Note that all accesses to global

memory require some portion of the Ringbus, except for accesses to the global memory in the same slice as the processor making the access.

There are also a small number of global registers located in the RIB (they are not shown on any of the Figures) for the purpose of various sundry activities such as resetting the slice, interrupting processors in the slice from a processor external to the slice, enforcing read and/or write protection on the slice's global memory, and some limited performance monitoring. These registers are accessed in the same manner as the global memory except that a slice cannot access its global registers directly from the Multibus. All global register accesses require the Ringbus.

1.2.3 Ringbus Arbiter

The arbiter uses a rotating priority scheme to ensure that all requests eventually get granted. If the slices are numbered consecutively from 0 to $S-1$, where S is the number of slices, then the priority of slice i is $pri(i) = (i - n) \bmod S$ where n is the current top priority slice. A request is held at the top priority until it is granted at which time n is updated to the next slice in the counterclockwise direction that has a pending request. A number of algorithms may be used to grant any combination of lower priority requests that do not conflict with each other or with any grants (i.e. memory accesses) in progress. The particular algorithm used in this case grants a request only if it does not conflict with any requests at higher priority levels or grants in progress. Only the direction requiring the smallest number of Ringbus segments is considered for granting the requests. In the case of a tie in the number of segments required in clockwise and counterclockwise directions, the clockwise direction is chosen.

The arbiter incorporates a clever design. The Ringbus segments required for each request are determined from the destination of the request. Since requests are only granted in one direction as mentioned earlier, there is no ambiguity in determining which segments are required. Each Ringbus segment is provisionally granted to a request. The request to which a particular segment is granted is determined by the priority of the requests. When a request has been granted all the segments that it requires, the request is granted.

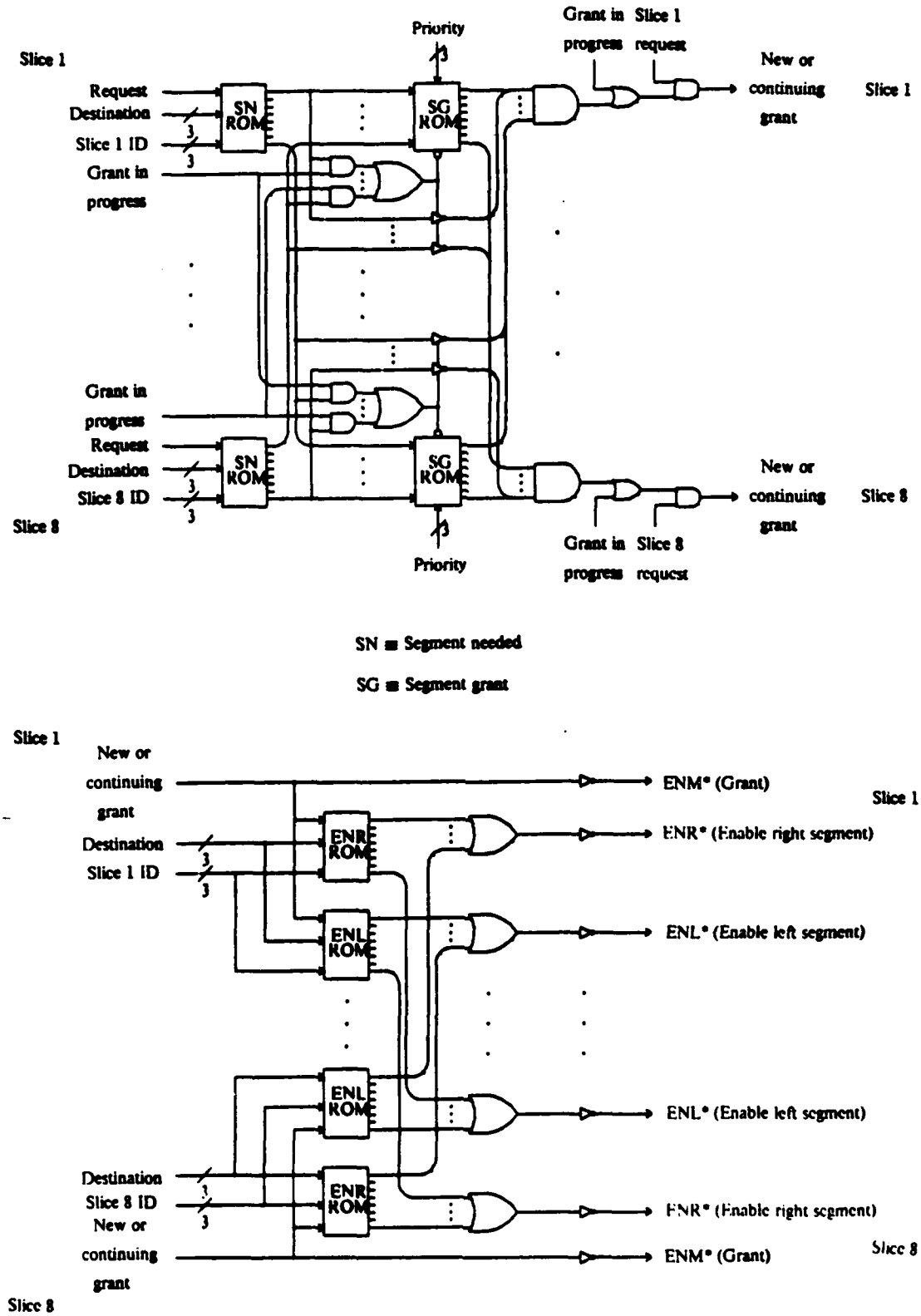


Figure 1.7: Logic diagram of arbiter

A logic diagram of the arbiter is presented in Figure 1.7. The SN Rom determines the segments required for each request. Each of the SG Roms, one for each segment, determines the request to which that segment is granted. The SG Roms automatically grant a segment to all requests that do not require it. Thus the eight segment grant lines just need to be ANDed to determine if the request has all the required segments. To prevent a "request" from being granted when there is in fact no request, the grant line is ANDed with the request line. Some additional logic bypasses the SG Roms to prevent the interconnection of the required segments from being changed while a grant using them is still in progress.

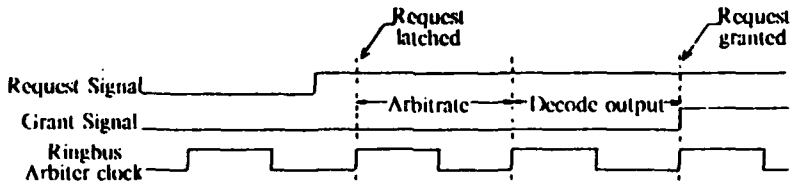


Figure 1.8: Ringbus arbiter timing

The arbitration time for this arbiter is between two and three arbiter clock cycles. As indicated in Figure 1.8, once the requests are latched into the arbiter, one cycle is required for the arbitration and another cycle is required to decode and latch the grant lines.

1.3 Modeling Details

1.3.1 Processor Model

We assume a simple probabilistic model for each processor based on accesses to non-local memory (i.e. those memory locations which a processor can only access via the Multibus). We partition the operation of a processor into three phases: 1) processing, 2) waiting, and 3) accessing. (We add a fourth phase later.) The processing phase corresponds to the interval between the completion of one memory access via the Multibus and the request for the next memory access via the Multibus. (A processor must request the Multibus and be granted its use by the Multibus arbitration circuitry before a memory access may proceed.) Only local (i.e. HSB) memory accesses may occur during this interval. We consider the instructions for each processor to be stored mainly in its local memory. Thus we regard the operation of a processor as consisting of periods of processing (hence the name processing phase), where the processor is accessing instructions and data stored entirely within its local memory, punctuated by accesses to global memory for data and other instructions.

The waiting phase corresponds to the interval between the generation of a Multibus request and the initiation of the access corresponding to that request. A Multibus memory access from one processor may have to wait for the completion of other Multibus accesses before it can begin. The accessing phase corresponds to the interval during which a Multibus access is in progress by that processor: it is the entire duration for which the processor maintains uninterrupted mastership of the Multibus. These three phases correspond to the operation of a processor from the point of view of the Multibus.

The interval for which a processor is in the processing phase we call the processing time, denoted by t_p; the interval for which a processor is in the waiting phase we call the waiting time for a memory request, denoted by t_w; and the interval for which a processor is in the accessing phase we call the access time, denoted by t_a. One cycle of a processor, consisting of these three times, is depicted in Figure 1.9.

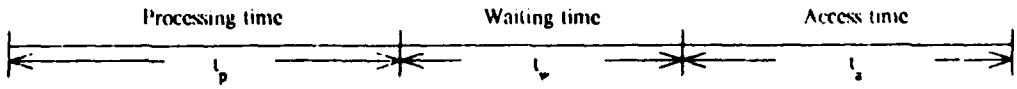


Figure 1.9: One cycle of a processor

More precise definitions of t_p, t_w, t_a in terms of Multibus signals are given in section 2 of Appendix A. The waiting time, t_w, is defined so that it is always zero when there is only one

processor on a Multibus. The delay of the Multibus arbitration circuitry is included in the access time.

We consider t_p, t_w, and t_a to be random variables. t_p and t_a have given probability distributions which serve as inputs to the processor model. The probability distribution of t_w, which is determined by the contention for use of the Multibus, is the output. Given that a processor gains mastership of the Multibus for a memory access, we assume that the access requires use of the Ringbus with probability ψ, in which case we call it a Ringbus access, and that it requires use of only the Multibus with probability $1-\psi$, in which case we call it a Multibus access. Given that a Ringbus access occurs, we assume that its destination is the global memory or a global register connected to Ringbus slice i with probability p_i^{RB}, $i = -(S/2-1), \cdots, -1, 1, 2, \cdots$, or $S/2$. The number of slices is S and i denotes the position of a slice with respect to the one from which the access originates. Negative numbers indicate the counterclockwise direction, positive numbers indicate the clockwise direction around the Ringbus relative to the slice originating the access. Thus $i = -2$ indicates the second slice along the Ringbus in the counterclockwise direction from the slice originating the access and $i = 2$ indicates the second slice in the clockwise direction. We call the set of p_i^{RB} the Ringbus destination probabilities. Since in most applications, accesses to the global registers will be infrequent, we ignore accesses by a processor to the global registers in its own slice. We assume that all Ringbus accesses have the same access time distribution and that all Multibus accesses have the same access time distribution (which in general will differ from that for Ringbus accesses). The Ringbus access time distribution is an equivalent model of the entire Ringbus from the perspective of the Multibus (we talk about this more in section 1.2.5): it includes any waiting time imposed on a Ringbus access by the Ringbus arbiter.

We have just assumed that all Multibus accesses have the same distribution. We now examine this assumption in more detail. In the absence of traffic on the HSB ports of the global memory boards, all Multibus accesses would actually have the same access time distribution. However, since the boards are dual-ported, traffic on one port of a memory board affects traffic on the other port. Thus Multibus accesses may have different access time distributions depending on the memory board accessed and the traffic intensity on the board's HSB port. There are two different cases to consider depending on the destination of a Multibus access.

Case 1: The destination is a local memory, in which case some processor connects to the HSB port of the memory board. In this case the Multibus access time can be greatly affected by the HSB traffic on the local memory board from the processor - compare Figures A.4 and A.5 in Appendix A.

Case 2: The destination is a global memory. In this case the HSB port may either be unconnected or connected to the RIB. A comparison of Figures A.4 and A.6 reveals that the access time is essentially the same for these two choices of HSB connections.

We conclude that if the majority of Multibus accesses are to global memory, then the access time distribution is essentially the same for every access as we assumed earlier. Finally, we note that a comparison of Figures A.9 and A.10 in Appendix A reveals that Ringbus access times are only slightly affected by the traffic intensity on the Multibus port of a global memory board.

We assume that reads and writes have the same access time distribution. This assumption is supported by the results in section 3.3 of Appendix A; for Multibus accesses, the access time distribution for reads and writes differ insignificantly and for Ringbus accesses, the access time distribution for reads and writes differ significantly. We ignore read-modify-write accesses, since they usually occur infrequently compared to reads and writes. (The effect of read-modify-writes can be included by incorporating access times near that of read-modify-writes in the access time distribution for reads and writes.) We assume that byte and word accesses have the same access time distribution. This assumption is again supported by the results in section 3.3 of Appendix A.

Just as the traffic intensity on the HSB port of a memory board affects the Multibus access time of that board, the traffic intensity on the Multibus port of a memory board affects the HSB access time of that board. Since the processing time distribution implicitly includes the HSB access time of its associated local memory, the processing time distribution of a processor depends on the traffic intensity on the Multibus port of its local memory. However, since the processing time distribution is an exogenous input and possibly different for each processor (although we assume it to be the same for each processor in Chapter 2 and 3), we can simply accommodate any such dependencies by using an appropriate processing time distribution. In addition, the argument which we presented above for the access time distribution will work to some extent for the processing time distribution (we can't be sure of the extent since we haven't made any measurements of the effect of Multibus port traffic on the processing time distribution).

The processor model presented so far in illustrated in Figure 1.10.

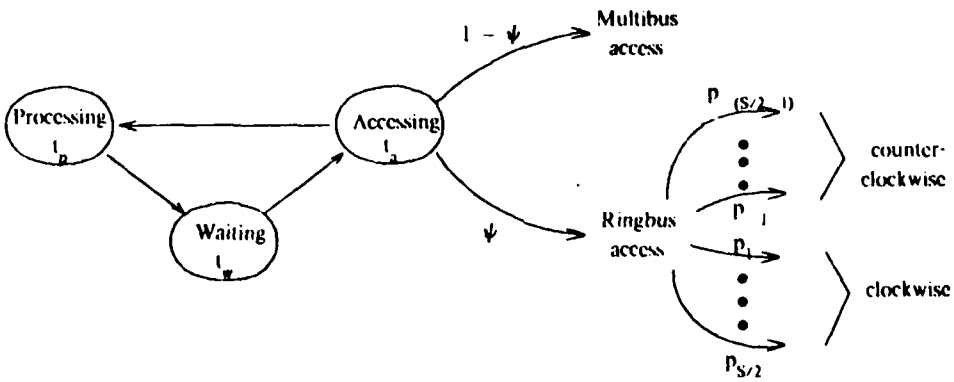


Figure 1.10: Processor model

The one remaining embellishment of the processor model concerns long word accesses. The access of a 32 bit long word involves two consecutive word accesses on the 16 bit wide data paths of Concert. However, the two word accesses on the Multibus are not necessarily consecutive since a processor does not maintain mastership of the Multibus between them. After the first word access of a long word completes, a processor waits some amount of time, which we call the recovery time, before requesting the Multibus for the second word access of the long word. Other processors may seize the Multibus in this time and cause the second word access to wait even if the first word access did not wait. Since a long word access consists of word accesses, we can certainly incorporate long word accesses in the processor model as presented so far. However, this may not be a good model - especially if a processor generates a lot of long word accesses - since the processing times in such a model are not correlated with the first word access of a long words when in reality the processing times are strongly correlated with the first word access of long words.

We add a fourth phase - recovery - to our processor model to create an alternate model for long word accesses. In this model we assume given that a processor gains mastership of the Multibus for a memory access, the access represents the first word of a long word access with probability β and a regular byte or word access with probability $1-\beta$. Given that the access does represent the first word of a long word access, the processor generates a request for the second word of the long word after a recovery time denoted by t_r. This second word access has the same destination - Multibus or Ringbus slice i - as the first word access. Again, we assume that t_r is a random variable with some given probability distribution. A more precise definition of t_r in terms of Multibus signals is given in section 2 of Appendix A. This alternate processor model is illustrated in Figure 1.11.

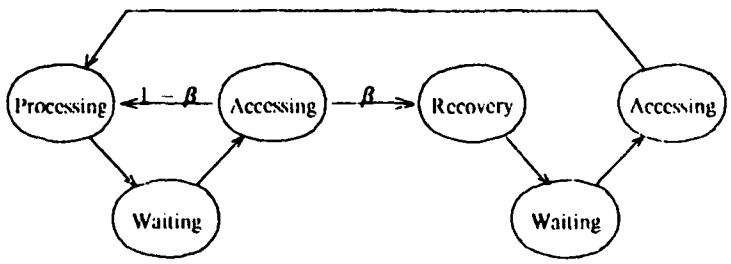


Figure 1.11: Alternate processor model

1.3.2 Major Assumptions

The major assumptions which we make throughout this thesis are:

1. The random variables t_p and t_a for each processor are stationary (i.e. their probability distributions are independent of time). We also assume that the probabilities β, ψ, and p_i^{RB} for each processor are independent of time.
2. Concert is an ergodic system - i.e. long term time averages converge to the values computed for stochastic steady state.
3. Each processor model is entirely independent of all other processor models and everything else. More precisely, all processing and access time random variables, t_p and t_a, are stochastically independent of each other and everything else. Also, all other probabilities β, ψ, and p_i^{RB} are stochastically independent of each other and everything else.
4. The overall model of Concert is in stochastic steady state.

The independence assumptions in 3 simplify the models. Various dependencies of the random variables can be included in the models (as discussed in section 2.10.4) but doing so increases the number of states and complexity of the models. Furthermore it is not clear at the present time what the dependencies are and how significant they are. Certainly factors such as the programs run on the system, the language in which the programs are expressed, and the distribution of the programs about the system influence the number and magnitude of the dependencies, but how does one intelligently express them in a model? Dealing with such questions and the various dependencies is beyond the scope of this thesis. Instead, we adopt a conservative approach: we assume that there are no dependencies and determine the performance as predicted by these simple models. Future research can be devoted to developing more detailed models to incorporate additional factors. The performance predicted by the models with the independence assumptions can be used to bounds the performance predicted by the same models with dependencies. Thus the independence assumptions allow simple models that yield bounds on the performance of more complex models.

Ways to relax the assumptions in 1 and 3 are discussed in section 2.10 in relation to the Multibus model.

1.3.3 Factors for Study

The factors we study in this research are:

1. The processing time distribution.
2. The Multibus access time distribution. (The Ringbus access time distribution is an equivalent model of the entire Ringbus from the perspective of a processor on a Multibus and thus it is dictated by the Ringbus. However, we do consider it as a factor for study in conjunction with

the Multibus model in section 2.9.)

3. The probability of a Ringbus access, ψ, and the Ringbus destination probabilities p_i^{RB}. We also consider the probability of a long word access β, when using our alternate processor model.
4. The number of processors on a Multibus, i.e. in a slice.
5. The number of slices.
6. The Ringbus access paths.
7. The Ringbus arbitration algorithm.

1.3.4 Overall Performance Metric

We use throughput as the performance metric of the overall model. We regard the throughput of a processor as the number of Multibus and Ringbus accesses completed per unit time. Thus the throughput of a processor is equal to $\frac{1}{\bar{t}_{cyc}}$ where $\bar{t}_{cyc}$ is the cycle time given by

$$\bar{t}_{cyc} = \bar{t}_p + \bar{t}_{w_1} + \beta \bar{t}_{w_2} + (1+\beta)((1-\psi)\bar{t}_{amb} + \psi \bar{t}_{arb}).$$

$\bar{t}_{w_1}$ denotes the mean waiting time per Multibus request for a byte, word, or first word af a long word access and $\bar{t}_{w_2}$ denotes the mean waiting time per Multibus request for the second word of a long word access. $\bar{t}_{amb}$ and $\bar{t}_{arb}$ denote the mean access time for Multibus and Ringbus accesses respectively. The total throughput is thus $\sum_{i \in P} \frac{1}{\bar{t}_{cyc_i}}$ where $\bar{t}_{cyc_i}$ is the mean cycle time for processor i and P is the set of all processors.

1.3.5 Decomposition and Integration

We divide the overall Concert system into a number of subsystems: one for each Multibus and one for the Ringbus. Each Multibus subsystem consists of all the processors, local memories, and global memories connected to the Multibus. The Ringbus subsystem consists of the Ringbus arbiter and everything connected to the RIBs except for the Multibus. This definition of the subsystems is illustrated in Figure 1.12.

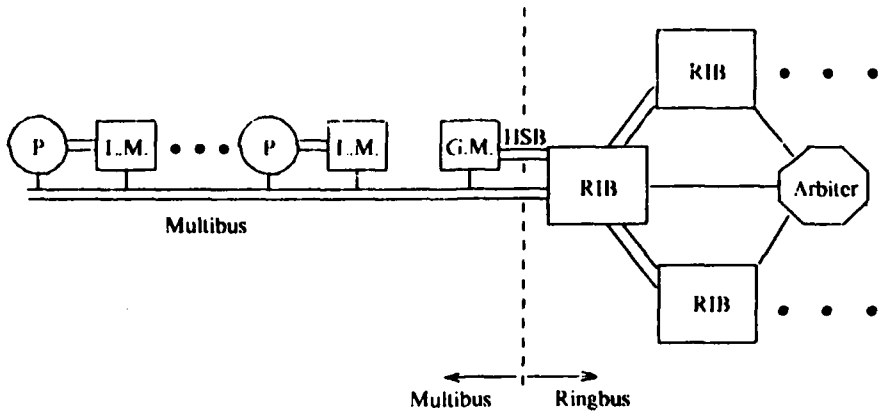


Figure 1.12: Subsystem definitions

Note that the global memory module connected to each RIB is included in the subsystem for the corresponding Multibus and in the subsystem for the Ringbus - we view it as being shared by the two subsystems. Thus there are two points of interaction between each Multibus subsystem and the Ringbus subsystem: the Multibus connection to the RIB and the global memory connected to the RIB. However, the interaction through the global memory connected to the RIB is especially weak. Measurements reported in section 3.3 of Appendix A reveal that the access time distribution for accesses via one port of the global memory connected to the RIB is hardly affected by heavy loading on the other port of the global memory. (Compare Figures A.4 and A.6 and Figures A.9 and A.10.) We ignore the interaction between Multibus and Ringbus subsystems through global memory in the rest of this thesis. The single remaining point of interaction between each Multibus subsystem and the Ringbus subsystem falls on a natural hierarchical boundary and thus represents a natural demarcation point between the subsystems.

Figure 1.13 gives an abstract view of the overall system.

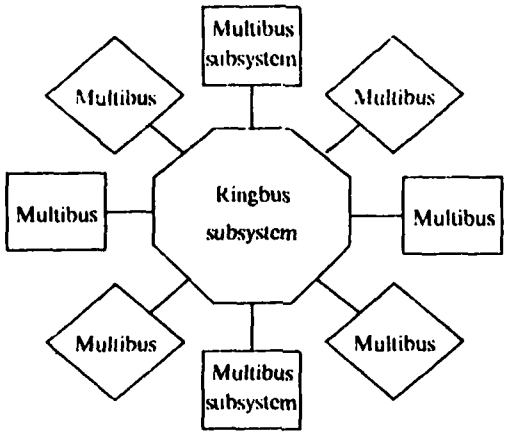


Figure 1.13: Abstract view of Concert

We can regard each subsystem as a black box. Each black box can be represented by an equivalent lumped model, just as a black box in an electrical circuit an be replaced by its Thevenin equivalent circuit. The Thevenin equivalent model of the Multibus subsystem is a single processor model of the sort described in section 1.2.1. This single processor represents the characteristics of the Ringbus accesses from the entire Multibus subsystem. Let the interval between the completion of one access on the Multibus with a Ringbus destination and the start of the next access on the Multibus with a Ringbus destination be called the Ringbus spacing. Then the processing time distribution of the single processor equivalent of the Multibus is equal to the probability distribution of the Ringbus spacing. We make no distinction between word and long word accesses for the Ringbus access spacing; thus we take $\beta = 0$ for the single processor. The probability of choosing Ringbus destination i in the single processor model, which we denote by p_i^{MBeqv}, is equal to the probability that a Ringbus access in the Multibus subsystem is for destination i. Finally, we have $\psi = 1$ for the single processor equivalent. The access time distribution is given by the Ringbus model. The Thevenin equivalent model of the Ringbus subsystem is some access time distribution for each Multibus-RIB connection. This access time distribution for a connection is the distribution of the time from the occurrence of a Ringbus request to completion of that Ringbus access for all Ringbus requests on that connection.

We decompose the overall model of Concert into Multibus and Ringbus models. As shown in Figure 1.14, Thevenin equivalent models are used to represent the other models connected to a particular model.

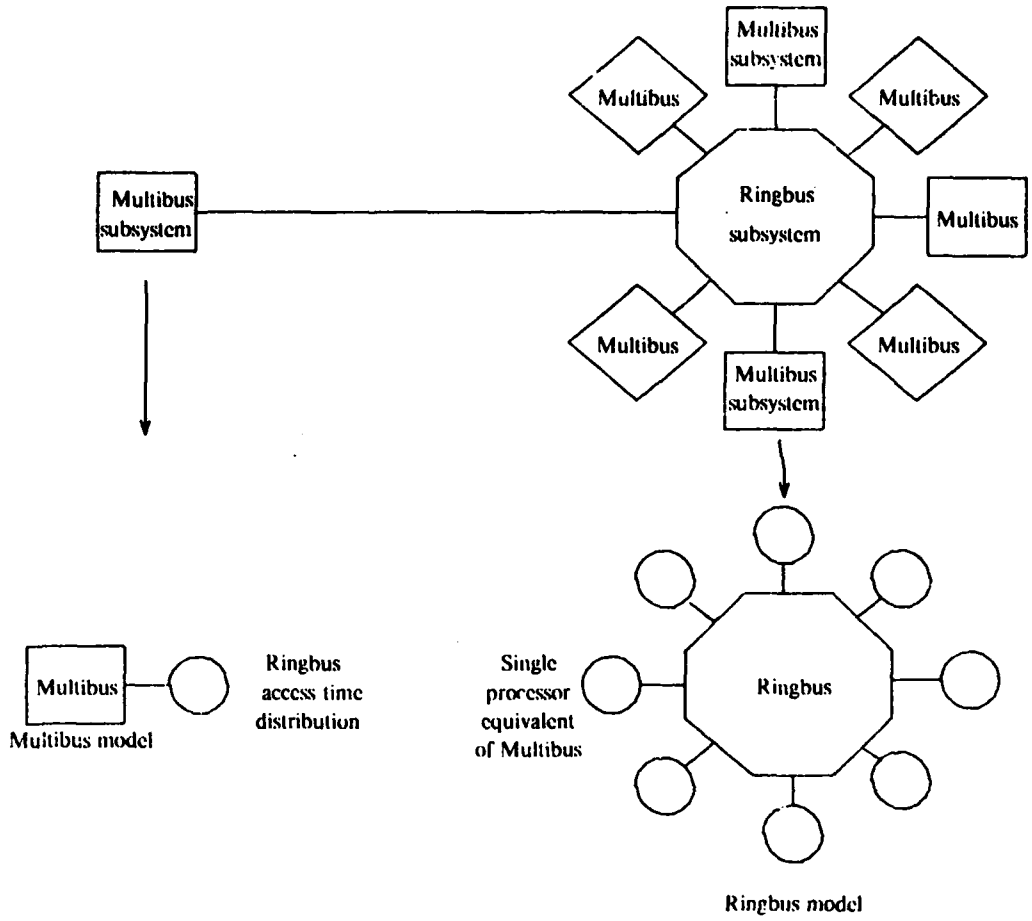


Figure 1.14: Decomposition into models

Given some Ringbus access distribution, the Multibus model can be analyzed. Likewise, given some processing time distribution and Ringbus destination probabilities, the Ringbus model can be analyzed. However, the solutions of these decomposed models do not necessarily correspond to the solutions of the subsystems in the overall system since the models are dependent. The Ringbus access time distribution is given by the Ringbus model, which depends on the single processor model of the Multibus. The single processor model of the Multibus is given by the Multibus model, which depends on the Ringbus access time distribution. Integration is the process of solving the models such that all these dependencies are satisfied. In a sense, integration amounts to matching the boundary conditions - i.e. interactions - between each pair of models to obtain a coherent overall model.

We perform the integration iteratively. First we assume some Multibus single processor

model and some Ringbus access time distribution. Then we solve the Multibus and Ringbus models to obtain a new Multibus single processor model and a new Ringbus access time distribution. We analyze the models again to obtain updated models and repeat until the improvement on successive iterations is sufficiently small. We do not discuss the the existence and uniqueness issues associated with integration. It should be clear later that in our case integration leads to a unique solution.

We make a number of assumptions and approximations to simplify integrating the models:

1. We assume that the Multibus models are identical in every respect: each has the same number of processors and all the processors are identical.
2. We assume that the Ringbus model is symmetrical with respect to each Multibus.

These two assumptions mean that only one Multibus model (and the Ringbus model) needs to be involved in the integration.

3. We approximate the processing time distribution of the single processor model of the Multibus by an exponential distribution.
4. We approximate the Ringbus access time distribution by an exponential distribution.

These two approximations ease the analysis of the models. Since an exponential distribution is completely specified by its first moment, these two approximations also considerably ease the integration of the models, since the integration now effectively reduces to first moment matching (i.e. we just have to determine the mean processing time of the single processor model of a Multibus and the mean Ringbus access time).

Of course, these assumptions and approximations limit the applicability and accuracy of the integration. The accuracy of the performance predictions obtained via integration of the models is assessed by comparison with simulations.

1.4 Previous Work

Single bus multiprocessors like the Multibus subsystem have been studied by many. The basic queueing system formulation of the Multibus model in Chapter 2 has appeared and has been studied in many guises. It appeared as a machine repairman model as early as 1935 [K2]. With the advent of Kleinrock's popular volume [K3], the M/M/1//N model of the basic queueing system has become a classic. Jaiswals' [J1], or alternately Benson and Cox's [B2], solution of the M/D/1//N model is also well known. The theory of product form queueing networks which we apply is well known, although we utilize Kelly's powerful and elegant quasi-reversibility approach [K1] to queueing networks rather than the more well known local balance BCMP approach [B1].

We are not aware of other studies dealing with our particular extensions to the basic queueing system model of the Multibus. However, the extensions are simple and the results we obtain follow from straightforward application of product form queueing network theory, so others may have derived similar results. The specific recursive solution technique we discuss for the PH/PH/1//N model is, to the best of our knowledge, new, although Herzog, Woo, and Chandy [H2] have already outlined the solution of general queueing systems by recursive methods.

The Ringbus subsystem, on the other hand, is a novel interconnection scheme which, to the best of our knowledge, was not studied (or conceived) before Anderson [A2]. Anderson focused on the design of a workable Ringbus: he only performed the most rudimentary simulations (see footnote in section 1.1). We study the optimum performance obtainable with a Ringbus. We formulate the Ringbus arbitration problem as a Markovian decision problem and treat it by the well known techniques of Howard [H4] and Odoni [O2].

The decomposition/integration approach to modeling Concert was inspired by Courtois [C5]. The techniques applied in this approach are standard.

1.5 Overview of Thesis

We study the Multibus model in detail in Chapter 2 and lay the foundation in section 2.9 for later integration with the Ringbus model. In Chapter 3 we study the Ringbus model. We concentrate mainly on the optimum performance of the Ringbus and the arbitration algorithm which achieves this performance. In Chapter 4 we integrate the Multibus and Ringbus models and make a few performance predictions to demonstrate the integration technique. We compare these predictions to simulation results. In the remainder of Chapter 4, we present the results of computer simulations of the overall Concert model.

Chapter 2

Multibus Models

2.1 Introduction

In this chapter we study the Multibus subsystem in detail. We use the processor model described in section 1.3 to construct various increasingly complex models of the Multibus. We assume, as mentioned in section 1.3.2, that all processor models are stationary and independent. To ease analysis, we assume in addition that all processor models are identical in every respect. The extension to non-identical processors, discussed in section 2.10.1, is straightforward but increases the complexity of the analysis without necessarily contributing much insight.

When all processors are identical, the mean cycle time of a processor, $\bar{t}_{cyc}$, is the same for every processor. (This follows from symmetry arguments.) Thus the throughput of the Multibus is given by $\frac{N}{\bar{t}_{cyc}}$ where N is the number of processors and

$$\bar{t}_{cyc} = \bar{t}_p + \bar{t}_{w_1} + \beta \bar{t}_{w_2} + (1+\beta)((1-\psi)\bar{t}_{amb} + \psi \bar{t}_{arb}).$$

$\bar{t}_{w_1}$ denotes the mean waiting time per Multibus request for a byte, word, or first word of a long word access and $\bar{t}_{w_2}$ denotes the mean waiting time per Multibus request for the second word of a long word access. $\bar{t}_{amb}$ and $\bar{t}_{arb}$ denote the mean access time for Multibus and Ringbus accesses respectively.

Since $\bar{t}_{w_1}$ and $\bar{t}_{w_2}$ are the only parameters which determine the throughput of the Multibus which are not exogenous inputs to the Multibus model, the performance metric for the Multibus effectively reduces to the pair $(\bar{t}_{w_1}, \bar{t}_{w_2})$. In this chapter we take the performance metric to be the mean total waiting time per cycle defined by $\bar{t}_{w_T} = \bar{t}_{w_1} + \beta \bar{t}_{w_2}$. This gives a single quantity for the performance, as with throughput, and is more closely related to the Multibus models than

throughput.

All the processors on a Multibus and the Multibus arbitration circuitry are synchronized by a master clock with a 100nsec period (one master clock per Multibus). Thus the Multibus subsystem inherently operates in discrete time. We model this discrete time operation with continuous time models to take advantage of the simple, powerful, and well developed modeling methods available in continuous time, such as product form queueing networks. It is argued in the following paragraphs that there is not much loss of precision in this approach.

We are not interested in modeling the Multibus at the level of the Multibus clock. Such detail is unnecessary for our purposes. Furthermore, any model based on the state of the Multibus at every rising edge of the Multibus clock would be unwieldy due to the large number of such states required. Rather, we are interested in modeling the Multibus at the event level. We define an event to be a request for a Multibus access or the completion of a Multibus access. (We do not consider the initiation of a Multibus access to be an event since either it is equivalent to a request for a Multibus access if there are no other Multibus accesses pending or in progress or it is equivalent to the completion of a Multibus access if a Multibus is in progress. Similarly, we do not consider the initiation or completion of processing to be an event since they are equivalent respectively to the completion of a Multibus access and a request for a Multibus access). Because the Multibus actually operates in discrete time synchronous with the rising edges of the Multibus clock, the time between successive events is the some integer multiple of 100nsec and one or two or more events can occur simultaneously. In modeling the Multibus in continuous time at the event level, we make the following two approximations.

1) We assume that the time between successive events can take on continuous values.

2) We assume that only one event can occur at a time.

The first approximation introduces a maximum error of ±50nsec in interevent times. Since in the actual Multibus the processing time is at least 600nsec and the access time is at least 1000nsec (see Appendix A), the loss of precision introduced by the first approximation is small. For the second approximation, we note that the probability of two or more events occurring in the same Multibus clock period is small. Thus there will probably only be a very small loss of precision due to the second approximation. Therefore there should not be much loss of precision introduced by electing to model the Multibus in continuous time.

The Multibus subsystem can be modeled as a queueing system with a finite number of customers. Consider the case in which $\psi = 0$ and $\beta = 0$ - i.e. only Multibus accesses and no explicit treatment of long word accesses - for each processor model. Denote the number of processors by N. We can represent the operation of each processor by a customer which visits service centers (servers). Once a customer arrives at a server, it remains there for a period of time governed by

the service time probability distribution for that server. Let there be N servers, called processor servers since they represent the N processors, each with an identical service time distribution equal to the processing time distribution. Let there be one server, called a Multibus server since it represents Multibus accesses, with its service time distribution equal to the Multibus access time distribution. (Since all Multibus accesses have the same access time distribution, it is sufficient to have just one server to represent a Multibus access.) Finally, let there be no more than one customer in service at a server at any instant and let there be N customers.

Each of the N customers behaves as follows. A customer visits a processor server and remains there for some processing time after which it joins a queue of other customers waiting to visit the Multibus server. When the customer eventually visits the Multibus server, it remains there for some access time and then it returns to the same processor server.

This processor-queue-Multibus cycle of a customer represents the processing-waiting-accessing cycle of the processor model (with $\psi = 0$ and $\beta = 0$). The finite customer queueing system is pictured in Figure 2.1 below. The circles represent servers.

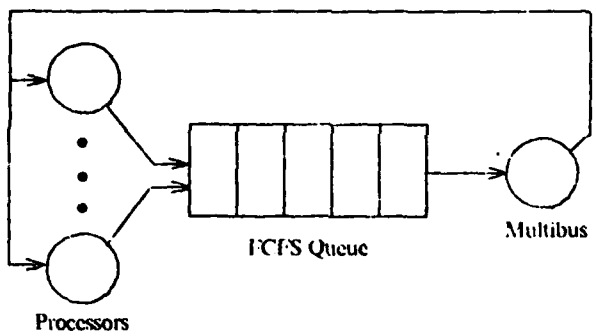


Figure 2.1: Finite customer queueing system

To faithfully model the operation of the Multibus arbitration circuitry, the queueing discipline at the Multibus server should be round-robin. However, to ease analysis, we will assume that this queueing discipline is first-come-first-served (FCFS). Interestingly, there is no loss of precision with this assumption. Since the Multibus server is work-conserving (i.e. the server is always busy while there remains work for it to do) and since all customers are identical (i.e. same processing and access time distribution for each customer), the mean waiting time per access on the Multibus, $\bar{t}_w$,† is the same for both queueing disciplines [M1]. Of course, the waiting time distributions will

† $\bar{t}_w$ is the mean waiting time per access for any access on the Multibus - byte, word, first word of long word, and second word of long word. If $\beta = 0$, $\bar{t}_w = \bar{t}_{w_1}$. In general $\bar{t}_{w_1} \neq \bar{t}_{w_2}$, so $\bar{t}_w \neq \bar{t}_{w_1} \neq \bar{t}_{w_2}$

not necessarily be the same (intuitively, one expects the variance of the waiting time to be greater with the FCFS discipline than with the round-robin discipline), but this doesn't matter since our performance metric just depends on the mean waiting time, $\bar{t}_w$.

We call the finite customer queueing system, depicted in Figure 2.1, with a FCFS queueing discipline, the basic queueing model of the Multibus. In later sections we extend this basic queueing model, known as the machine repairmen model in the queueing theory literature, to accommodate $\psi \neq 0$ and $\beta \neq 0$. A convenient notation to describe the basic queueing model is $S_1/S_2/1//N$. S_1 and S_2 represent symbols denoting, respectively, the processing and access time distributions. The 1 indicates a single server queue and N indicates the total number of customers. Some commonly used symbols are M for memoryless (i.e. exponential), D for deterministic, E_r for r stage Erlangian, and G for general. Thus M/M/1//N denotes a basic queueing model with exponential processing and access times and N processors.

A rather exhaustive analytical treatment of the basic queueing model with different processing and access time distributions is presented in section 2.2 through 2.7. Section 2.2 deals with deterministic processing and access times. Section 2.3 characterizes the general behaviour of $\bar{t}_w$ for probabilistic processing and access times. Sections 2.4, 2.5, and 2.6 develop results for the M/M/1//N, M/G/1//N, and G/M/1//N models respectively. Most of section 2.6 is devoted to describing the known results for a class of queueing networks with convenient product form solutions. These results are heavily utilized in sections 2.8 and 2.9. Section 2.7 presents a recursive technique for handling general processing and access time probability distributions. This is believed to be the first demonstration of a reasonable solution method specifically for the G/G/1//N model.

Generalizations of the basic queueing model to handle $\beta \neq 0$ and $\psi \neq 0$ are covered in sections 2.8 and 2.9. Section 2.8 treats the case with $\beta \neq 0$ and $\psi = 0$ and section 2.9 treats the general case with $\beta \neq 0$ and $\psi \neq 0$. Section 2.9 discusses the decomposition of Concert into Multibus and Ringbus models and develops the hooks for the later integration of these two models. Specifically, the single processor equivalent of the Multibus is developed and relations yielding its parameters are derived.

Lastly, section 2.10 discusses the relaxation of the four major assumptions of 1) identical processors, 2) simple processor model, 3) stationary processor model, and 4) independent processors. The most important sections in Chapter 2 are 2.2, 2.3, 2.4, 2.8, and 2.9. Section 2.6 is also important, but only as a primer on product form solutions of queueing networks for sections 2.8 and 2.9. Sections 2.5 and 2.7 are, in some sense, icing on the cake.

2.2 Deterministic Model

In this first model, both t_p and t_a are deterministic quantities.

Initially the independent processors are unsynchronized. However, due to the determinism of t_p and t_a, every time two or more memory requests occur at the same time that the bus is currently in use, the processors originating those requests are synchronized with each other and with the processor currently using the bus. The synchronization does not occur at the instant of conflict but rather at the instant the access in progress terminates and the request at the head of the FCFS queue waiting for the bus begins its access. At this instant, the two respective processors are synchronized so that the cycle of the one just beginning its access lags the other by exactly t_a. Similarly each processor which has a request in the queue is synchronized so as to lag exactly t_a behind the processor of the previous access. Since t_p is also deterministic and the same for every processor, the synchronized processors will make their next requests at intervals of t_a.

Theorem 2.1

With independent identical processors with deterministic processing time t_p and deterministic access time t_a served by a single bus in FCFS order, the waiting time per request after at most two cycles of every processor is the same for every request. Moreover, after at most two cycles of every processor the FCFS queue is either always empty or always nonempty at the instant a request arrives at the queue.

The proof of this Theorem is given in appendix B.

By construction $t_w = 0$ when N, the number of processors on the Multibus, is one. Let N^* be defined as the saturation point: in the steady state for $N \leq N^*$, $t_w = 0$ (corresponding to the queue always empty when a request arrives), and for $N > N^*$, $t_w > 0$ (corresponding to the queue always nonempty when a request arrives). This saturation point is the maximum number of processors for given t_p and t_a that the bus can support in steady state and maintain $t_w = 0$.

The maximum number of processors that the bus can handle with zero wait time for a request is one (for the bus in use) plus the maximum number of additional processors that can be processing, but not waiting, while the one processor is currently using the bus. This maximum number of processors is given by $\left\lfloor \frac{t_p}{t_a} \right\rfloor$.† Thus $N^* = \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$.

For each processor added above N^*, all processors will share equally (after initial transients

† $\lfloor x \rfloor$ denotes the smallest integer less than x.

die out) the wait incurred by the addition of each processor above the saturation point, if all processors are identical and bus arbitration is FCFS. To find t_w for this case, we may equate the arrival rate of requests to the bus system to the service rate of requests at the bus system. We have then:

$$\frac{N}{(t_p + t_a + t_w)} = \frac{1}{t_a},$$

from which we obtain $t_w = Nt_a - (t_p + t_a)$.

The wait per request normalized by the access time is

$$\frac{t_w}{t_a} = N - (\frac{t_p}{t_a} + 1).$$

At this point (and in the sequel) it is more convenient to consider N^* and N as continuous rather than discrete quantities. The saturation point is thus redefined as $N^* = \frac{t_p}{t_a} + 1$. Although the discussion will consider N and N^* as continuous quantities, these quantities should be understood to be in fact discrete whenever they are given a physical interpretation.

Substituting for N^*, we obtain $\frac{t_w}{t_a} = \begin{cases} 0, & N \leq N^* \\ N - N^*, & N > N^* \end{cases}$, which completely describes the behavior of t_w in the steady state for the deterministic case (see Figure 2.2).

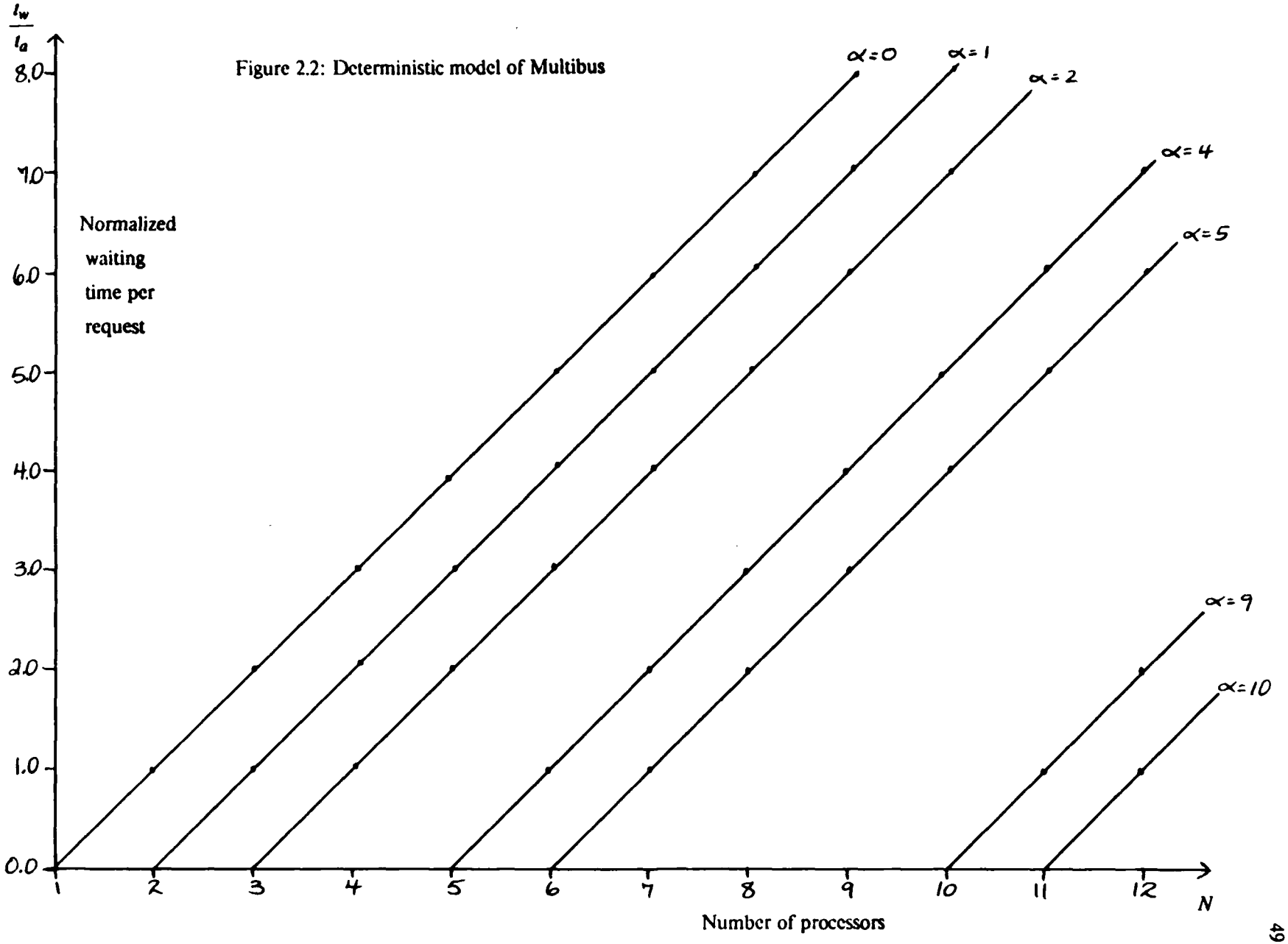


Figure 2.2: Deterministic model of Multibus

2.3 Probabilistic Model - General Behaviour

We now consider t_p and t_a to be stationary random variables with given probability distributions. We assume that the random variable t_p for each processor and the random variables t_a are independent of each other and all other random variables as discussed in section 2.1. We also make the reasonable assumption that the random variables t_p and t_a have finite means i.e. $E[t_p]<\infty$ and $E[t_a]<\infty$.

In addition to these assumptions and those in section 1.3.2 we make the following existence and ergodicity assumptions in this section.

Existence and Ergodicity Assumptions

1. We assume that the mean waiting time per request, $\bar{t}_w$, exists. More precisely, we assume that a stationary probability distribution exists for t_w (since $\bar{t}_w$ is defined in terms of its probability distribution function). Let the waiting time of the nth request to enter the queue be denoted by t_{w_n} so that $\{t_{w_n}\}$, $n\geq 1$, is a sequence of the waiting times of successive requests. The assumption means that $\lim_{n\to\infty} Pr(t_{w_n}\leq y)$ exists and equals some function $W(y)$ where $Pr(t_{w_n}\leq y)$ is the probability distribution of the waiting time of the nth request and $W(y)$ is *the stationary probability distribution for* t_w.
2. We assume that the waiting time process is ergodic so that ensemble averages equal (discrete) time averages i.e. we assume that $\bar{t}_w = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} t_{w_i}$.
3. We assume that the time averages necessary for any application of Little's Law to the queueing system described in section 2.1 exist. Little's Law is the following statement:

   Consider any system at which customers arrive, spend time in the system, and depart. Let $N(t)$ be the number of arrivals at the system in the interval $[0,t]$, $l(t)$ be the number of customers in the system at time t, and w_k be the time spent in the system by the kth customer to arrive. If the following limits exist and are finite

   $$\lambda = \lim_{t\to\infty} \frac{N(t)}{t}, \text{ average arrival rate}$$

   $$L = \lim_{t\to\infty} \frac{1}{t}\int_0^t l(s)ds, \text{ average number in system}$$

   $$W = \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} w_i, \text{ average time in system}$$

   then $L=\lambda W$ [S2].

These assumptions are necessary to ensure that the results developed in this section are strictly correct. All the following sections in this chapter deal with specific probability distributions

and/or specific situations for which these assumptions are valid in all cases; thus it is unnecessary to state them in the sequel. However, this section deals with unspecified general distributions for which it is difficult to show that these assumptions are valid in all cases.

The purpose of the first assumption is straightforward - $\bar{t}_w$ must exist before we can talk about it. The second assumption ensures that the average waiting time derived from an application of Little's Law equals $\bar{t}_w$. The third assumption ensures that it is valid to apply Little's Law. Note that if the time averages in this third assumption exist, then they must be finite since we are dealing with a closed queueing system. If one is willing to deal with a time average for the waiting time per request rather than an ensemble average (i.e. a mean), then only the third assumption is necessary. We present and prove some conditions in Appendix B for which the three assumptions are valid.

We now consider the general behaviour of the mean waiting time per request, $\bar{t}_w$, subject to the preceding assumptions. For a single processor we still have $\bar{t}_w = 0$. We can derive a general formula for $\bar{t}_w$ with N processors using Little's Law.

Let $\bar{n}$ denote the mean number of requests queued for service and currently in service on the bus. Let $\bar{n}_p$ denote the mean number of processors which are processing (i.e. which do not have an outstanding request). Let ρ denote the probability of the server (i.e. the bus) being busy. Let λ^{eff} denote the mean arrival rate of requests to the bus. Since the system is closed with a finite number of requests, λ^{eff} is also the mean service rate of requests.

Then by Little's Law we have: $\bar{t}_w = \frac{\bar{n}}{\lambda^{eff}} - \bar{t}_a$. Applying Little's Law twice more we have $\rho = \lambda^{eff}\bar{t}_a$ and $\bar{n}_p = \lambda^{eff}\bar{t}_p$. Since $\bar{n} + \bar{n}_p = N$ we thus have $\frac{\bar{t}_w}{\bar{t}_a} = \frac{N - \bar{n}_p}{\rho} - 1$ and $\bar{n}_p = \rho\frac{\bar{t}_p}{\bar{t}_a}$, yielding

$$\frac{\bar{t}_w}{\bar{t}_a} = \frac{N}{\rho} - \frac{\bar{t}_p}{\bar{t}_a} - 1 = \frac{N}{\rho} - N^*$$

where we now define $N^* = \frac{\bar{t}_p}{\bar{t}_a} + 1$. This same result can be obtained by considering the throughput balance equation $\frac{N}{\bar{t}_p + \bar{t}_w + \bar{t}_a} = \frac{\rho}{\bar{t}_a}$. It follows from the definition given above that $0 \le \rho \le 1$, and thus $\frac{\bar{t}_w}{\bar{t}_a} \ge N - N^*$. For the deterministic case with $N > N^*$, $\rho = 1$ and thus the lower bound for $\frac{\bar{t}_w}{\bar{t}_a}$ is achieved by the deterministic case. Note that as $N \to \infty$, $\rho \to 1$, thus yielding the same asymptotic behaviour as derived in section 2.2. For $N \le N^*$, we have that $\frac{\bar{t}_w}{\bar{t}_a} \ge 0$

and this lower bound is again achieved by the deterministic case. Therefore $\frac{\bar{t}_w}{\bar{t}_a} \geq max(0, N - N^*)$ where the lower bound is achieved by the deterministic case. We summarize this result as a lemma:

Lemma 2.1

The mean waiting time per request in the previously described queueing system model with stationary processing and access times with means $\bar{t}_p < \infty$ and $\bar{t}_a < \infty$ respectively and subject to the previous assumptions is bounded from below by the mean wait per request in the deterministic model with the same processing and access times $\bar{t}_p$ and $\bar{t}_a$ respectively.

Proof:

Given in the above development.

We can also say that $\bar{w}(N+1) - \bar{w}(N) \geq 0$ (where we use the notation $\bar{w}(N)$ to indicate the mean waiting time $\bar{t}_w$ in an N processor system). This follows since adding another processor cannot cause the mean waiting time to decrease. In addition, it seems intuitive that $\bar{w}(N+1) - \bar{w}(N) \leq \bar{t}_a$: an arriving request in the $N+1$ processor system ought to see at worst one more request in the queue than it would in the corresponding N processor system. The following theorem justifies this intuitive feeling.

Theorem 2.2

Consider the queueing model described previously with stationary processing and access time distributions with means $\bar{t}_p < \infty$ and $\bar{t}_a < \infty$ respectively and subject to the previous assumptions. Then $\bar{w}(N+1) - \bar{w}(N) \leq \bar{t}_a$ where $\bar{w}(N)$ denotes the mean waiting time in a N processor model.

Proof:

Given in Appendix B.

The foregoing allows us to conclude that the mean wait per request for any stationary processing and access time distributions has a curve of the general shape indicated below. The randomness introduced by the probability distributions rounds the "knee" of the curve.

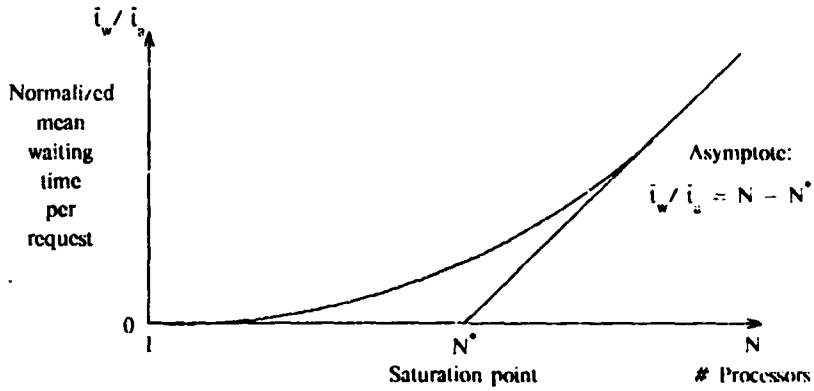


Figure 2.3: $\bar{t}_w/\bar{t}_a$ vs. N for general probabilistic case

The following sections consider the behaviour of $\bar{t}_w$ for specific probability distributions and for modifications to the basic queueing model.

2.4 Exponential Distributed Processing and Service Times - M/M/1//N Model

The analysis of this M/M/1//N model is straightforward. Following Kleinrock [K3], we define state k to represent k requests queued for service or in service ($0 \leq k \leq N$) resulting in the (discrete state continuous time) birth-death Markov chain depicted in the state transition diagram below.

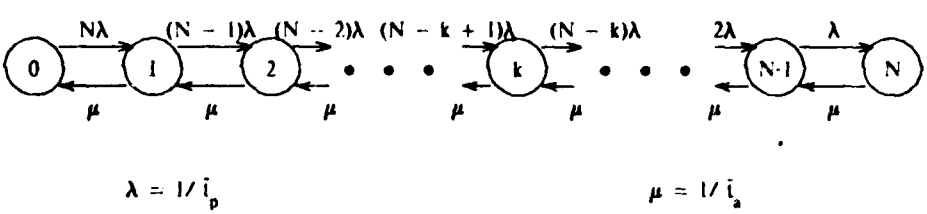


Figure 2.4: Markov chain of M/M/1//N model

The steady state probabilities, π_k, satisfy the local balance equations $\pi_k \cdot p_{k,k+1} = \pi_{k+1} \cdot p_{k+1,k}$, $0 \leq k < N$, where $p_{i,j}$ denotes the probability of going from state i to j in one transition. From the local balance equations we obtain:

$$\pi_k = \pi_0 \frac{N!}{(N-k)!} (\frac{\lambda}{\mu})^k, \; 1 \leq k \leq N \tag{2.1}$$

The mean queue length is $n_q = \sum_{k=2}^{N}(k-1)\pi_k$ and the average arrival rate to the queue is $\lambda_{eff} = \lambda \sum_{k=0}^{N-1}(N-k)\pi_k$. By Little's Law, the mean queueing time (mean time wait in queue before being served) is $\bar{t}_w = \frac{n_q}{\lambda_{eff}}$. The normalized mean wait per request is thus

$$\frac{\bar{t}_w}{\bar{t}_a} = \mu \bar{t}_w = \alpha \cdot \frac{\sum_{k=2}^{N} \frac{N!(k-1)\alpha^{-k}}{(N-k)!}}{\sum_{k=0}^{N-1} \frac{N!\alpha^{-k}}{(N-k-1)!}}, \tag{2.2}$$

where $\alpha = \frac{\mu}{\lambda} = \frac{\bar{t}_p}{\bar{t}_a}$.

Results for the case $\alpha = 1.0$, 2.0, 5.0, and 10.0 are displayed in Figure 2.5.

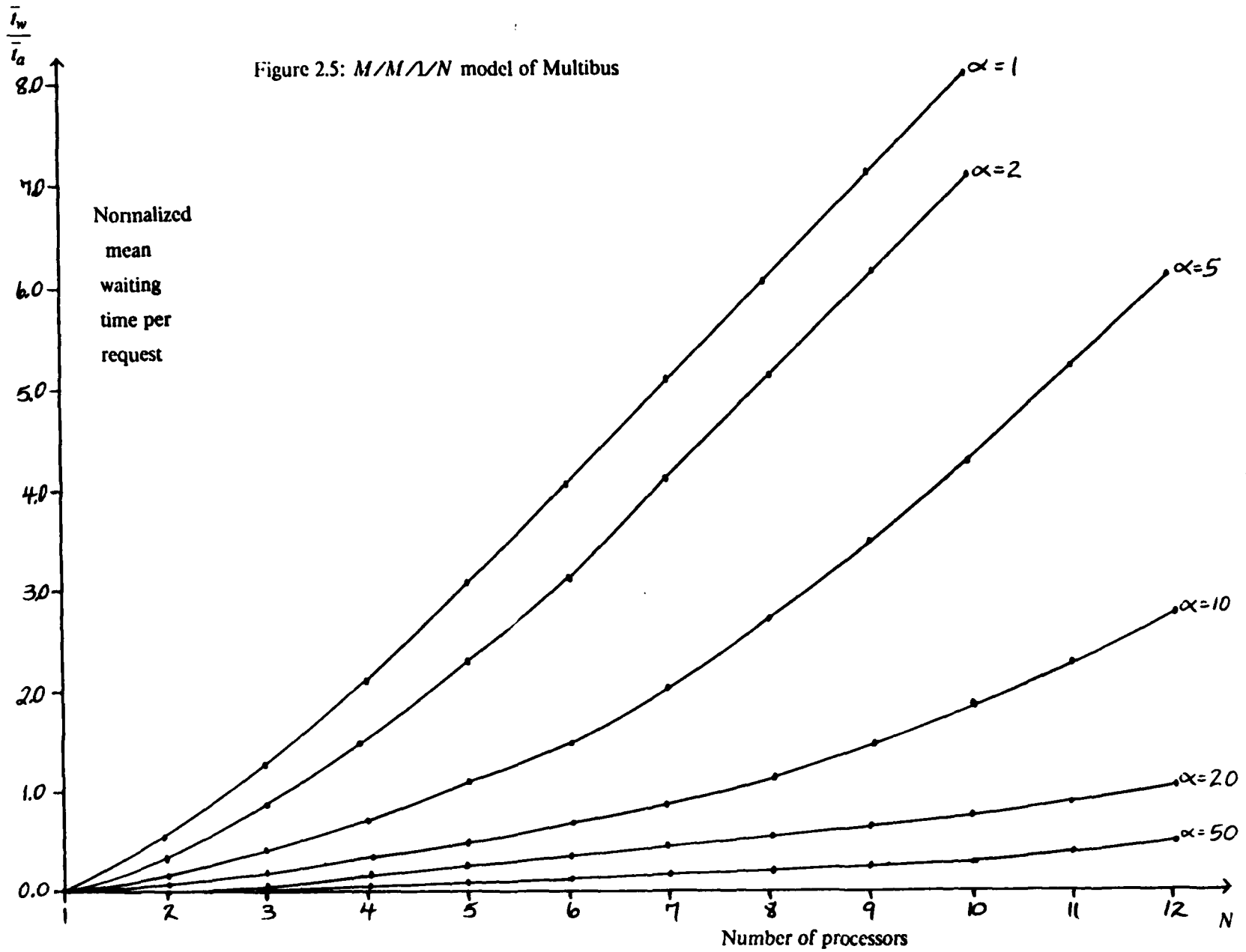


Figure 2.5: $M/M/1/N$ model of Multibus

2.5 Exponential Distributed Processing and General Service - M/G/1//N Model

In this section we generalize the M/M/1//N model of the previous section to include any stationary memory access (or service) time distribution. With a general service distribution, the probability distribution of the remaining service time, given that there is a customer in service, depends on the time that the customer has already been in service. In such a case, the service time distribution is said to have memory. Since a state must include all history or must summarize all the history of the system relevant to predicting the future of the system, the state description of whatever system the server is in must include the expended service time (or alternately, the time remaining in the service of the customer), whenever a customer is in service.

For example, one state description of the M/G/1//N system is to let the states be (k,t) where k requests are queued for service or in service and the request presently in service has been in service for time t, $1 \leq k \leq N$, $t \geq 0$; and (0) when no requests are queued for service or in service. The exponential distribution has the special property that the probability distribution of the time remaining is independent of the time expired so far. This memoryless property is the reason why the service time completed so far is irrelevant for the M/M/1//N model (which is why the state in the previous section was simply (k), $(0 \leq k \leq N)$, and is the reason why the processing time completed so far at each processor is irrelevant for both the M/G/1//N and M/M/1//N models.

The fact that time must be included in the state description complicates the analysis of the M/G/1//N model. We must now deal with an uncountably infinite number of states rather than the finite number of the M/M/1//N model. Three analytical methods are common for finding the steady state distribution of the number of requests queued for service or in service, from which we can then find the mean waiting time per request.

1. Stages

In this method, the server is subdivided into a number of stages where each stage has an exponential service distribution and only a single customer is allowed into the system of stages at a time (just as only a single customer is in the original server at a time). Considering the entry and exit points of the server to be special stages with zero service time, the next stage a customer enters after leaving the present stage is governed by a probability distribution which may depend on the present stage. The mean service time in each stage may also depend on the stage. Cox [C4] has shown that it is possible to synthesize any probability density which has a rational Laplace transform by a system of stages as just described.† Cox has also shown that the system of stages in Figure 2.6 is canonic in that it captures the full generality of densities which can be synthesized by

† If complex values are permitted for the exponential parameters. (Recall that an exponential distribution is fully characterized by a single parameter equal to the reciprocal of the mean.)

the method of stages. In particular, feedback and/or feedforward paths add no further generality. It is sometimes convenient to consider series-parallel or parallel-series arrangements of the stages, rather than the ladder arrangement in Figure 2.6.

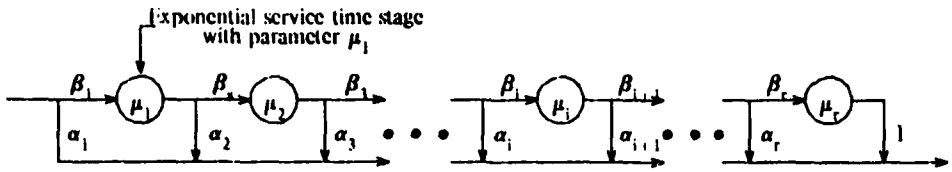


Figure 2.6: Canonic ladder arrangement of stages

The advantage of the method of stages is that the state space is now finite, or at worst countably infinite. This arises because each stage in the server is exponential and thus it suffices for the state to include just the stage in which the customer is, rather than the time completed so far in the service.

The resulting state transition diagram will be similar to that in Figure 2.6 in the previous section except that the the states are more conveniently arranged in a two dimensional manner and transitions are not limited to nearest neighbours. The equilibrium equations relating the steady state probabilities are easily obtained. Since these are linear equations it is in principle straightforward to find the steady state probabilities. Note these are the steady state unconditional probabilities; they must be summed over the appropriate states to obtain the steady state marginal probabilities such as the number of requests queued for service or in service.

The method of stages has three disadvantages. First, closed form results are difficult to obtain except in special cases due to the complexity of solving a large number of simultaneous linear equations. Thus it is difficult to determine how the result varies as a function of the input parameters such as mean arrival and mean service times without recomputing the result for each set of parameters.

Second, the exponential parameter and next stage probability distribution must be found for each stage, preferably so as to minimize the number of stages required to represent a given probability distribution. This can be accomplished by matching either the poles and zeroes of the Laplace transform of the stage system with the poles and zeroes of the Laplace transform of the given probability density or by matching polynomial coefficients of the two Laplace transforms (both amount to the same thing). In either case, the matching involves solving a set of nonlinear equations relating the stage parameters. The number of stages required is equal to the number of poles in the Laplace transform of the given probability density, assuming all pole-zero

cancellations have been removed. As might be imagined, certain interconnections of the stages make the solution of the simultaneous equations easier than others. While straightforward in principle, finding the stage parameters requires a substantial amount of work in the general case.

Third, only probability densities with rational Laplace transforms can be handled exactly in a finite number of stages. However, since any nonrational function can be approximated arbitrarily closely by rational functions, we can in principle use the method of stages for any arbitrary (stationary) probability density. The problem in practice is how to best approximate a given distribution by one that has rational transform.

2. Imbedded Markov Chain

In this method, the two dimensional state description (k, t) of the system is reduced to a one dimensional state description (k) by looking at the system only at select points in time. These points must be such that given the number in the system,* (k), at one such point, and the inputs to the system, then at the next point in time we can calculate the number in the system. Thus these points must implicitly include the time that has been expended on the customer in service.

One set of such points is the service departure times - i.e. the time at which a customer completes service. At a departure instant, the expended service of the next customer is zero (and the residual service of the present customer is zero) and the time to the next departure is given by the unconditional service time distribution as long as at least one customer is left in the system. If the system is empty, the time to the next departure instant is given by the convolution of the arrival time distribution (which is exponential with parameter $N\lambda$ for the M/G/1//N case) with the unconditional service time distribution.†

The behavior of the system at the imbedded points - the departure instants - can be described by a Markov chain. Let the state of the Markov chain be the number of customers in the system immediately after a departure. The transition probabilities can be determined from the arrival and service time distributions. The steady state solution of the Markov chain gives the steady state probability of finding k customers in the original system at the departure instants, but not the correct steady state probability at arbitrary times between departures. (It actually does give the correct results at all times if the customer population N is infinite and the arrival time distribution is exponential.) However, the mean waiting time, as we are concerned with in this chapter, is sufficient to determine the probability that the server is idle and this is easy to

* By system we mean in this case the FCFS queue and its server.

† The probability distribution of the sum of two independent random variables is the convolution of the two respective probability densities.

determine. Thus the steady state solution of the one dimensional imbedded Markov chain at departure instants is sufficient to find the mean waiting time.

Other sets of points exist which may be used to derive an imbedded Markov chain but they are not as convenient since the expended service of the next customer will not be zero (otherwise we have the same set of points as before). This necessitates handling the messy case when a customer does not remain in service long enough to reach the imbedded point.

The advantage of the imbedded Markov chain method is that general service time distributions may be handled explicitly and without solving for a myriad of parameters as in the stage method. The disadvantage is again that it is difficult to obtain closed form results. This is principally due to all the bookkeeping required to keep track of the number of "active" arrival generators in the finite population case. Such bookkeeping is unnecessary in the infinite population case and explicit results for the mean waiting time (depending only on the mean arrival rate and the mean and variance of the service time!) and the waiting time distribution can be obtained.

3. Supplementary Variables

In this method the problem posed by the two dimensional discrete-continuous state space (k,t) (for $k \neq 0$) is attacked directly by solving the related differential difference equations. Closed form results for arbitrary service time densities can be obtained by this method. We give the main results below, from the derivation of Jaiswal [J1]. Let

ρ be the server utilization i.e. the probability that the server is busy

$\bar{b}$ be the mean busy period of the server (the mean time interval between the server being idle)

$\frac{1}{\lambda}$ be the mean of the exponential processing time

t_a be the service time (i.e. access time) with density $f(t_a)$ and mean $\bar{t}_a$

Then

$$\bar{b} = \bar{t}_a \sum_{i=0}^{N-1} \binom{N-1}{i} \frac{1}{\varphi(i)} \tag{2.3}$$

where $\varphi(m) = \begin{cases} \prod_{i=1}^{m} \frac{F^*(i\lambda)}{1-F^*(i\lambda)}, & m \neq 0 \\ 1, & m = 0 \end{cases}$

and $F^*(s) = E[e^{-st_a}] = \int_0^\infty e^{-st_a} f(t_a)\,dt_a$, the Laplace transform of $f(t_a)$.

Finally $\rho = \dfrac{\bar{b}}{\bar{b} + \dfrac{1}{N\lambda}} = \dfrac{1}{1 + \dfrac{1}{N\lambda\bar{b}}}$.

By applying Little's Law twice we can determine the mean waiting time (i.e. queueing time) per request, $\bar{t}_w$. From Little's Law we have $\rho = \lambda_{eff}\bar{t}_a$ where $\lambda_{eff} = \lambda \times$ average number processors running $= \lambda(N - L)$ and L denotes the mean number of request in the FCFS queue or in service. Thus $L = N - \dfrac{\rho}{\bar{t}_a\lambda} = N - \alpha\rho$ where $\alpha = \dfrac{\bar{t}_p}{\bar{t}_a} = \dfrac{1}{\bar{t}_a\lambda}$. Again from Little's Law we have $\bar{t}_w = \dfrac{L}{\lambda_{eff}} - \bar{t}_a$. Therefore

$$\frac{\bar{t}_w}{\bar{t}_a} = \frac{L}{\rho} - 1 = \frac{N}{\rho} - \alpha - 1$$

Substituting for ρ, we obtain

$$\frac{\bar{t}_w}{\bar{t}_a} = N - \alpha - 1 + \frac{1}{\bar{b}\lambda} = N - N^* + \frac{\alpha}{\bar{b}_N} \tag{2.4}$$

where $\bar{b}_N = \dfrac{\bar{b}}{\bar{t}_a}$ (the normalized mean busy period i.e. the average number of consecutive requests served without an intervening idle period).

Equation which should be familiar as just $\dfrac{\bar{t}_w}{\bar{t}_a}$ in the deterministic case for $N \geq N^*$ plus $\dfrac{\alpha}{\bar{b}_N}$.

Equation 2.4 might lead one to conjecture that the maximum difference in mean waiting time per request between the M/G/1//N and deterministic model (section 2.2) occurs at the knee $N = N^*$. The following lemma shows that this conjecture is indeed correct, in even a more general setting, provided N^* is an integer. The treatment must be more careful for non-integer N^* since the queueing system model allows only integer N. The general idea, however, still holds when N^* is non-integer. (The graphs have been drawn as continuous in N to emphasize the trends.)

Lemma:

Let $w(N)$ be the mean waiting time per request in a G/G/1//N queueing system with arbitrary processing and access time distributions with means $\bar{t}_p$ and $\bar{t}_a$ respectively. Let $w_D(N)$ be the mean waiting time per request in a D/D/1//N queueing system with constant processing and access times $\bar{t}_p$ and $\bar{t}_a$ respectively. Then the difference $w(N) - w_D(N)$ is maximum at either $N = \lfloor N^* \rfloor$ or $N = \lceil N^* \rceil$ where $N^* = \alpha + 1$, $\alpha = \dfrac{\bar{t}_p}{\bar{t}_a}$.

Proof:

Consider $1 \leq N \leq N^*$:

From section 2.2 $w_D(N) = 0$ in this range. In addition $w(N+1) - w(N) \geq 0$ for every $N \geq 1$, i.e. $w(N)$ is nondecreasing in N. Thus $w(N) - w_D(N)$ is maximum for $1 \leq N \leq N^*$ when N is the largest integer less than or equal to N^* - i.e. $N = \lfloor N^* \rfloor$.

Consider $N^* \leq N$:

From section 2.2 $w_D(N) = N - N^*$ and $w_D(N+1) - w_D(N) = \bar{t}_a$ in this range. In addition $w(N+1) - w(N) \leq \bar{t}_a$ by Theorem 2.2. Let N^o be the smallest integer greater than or equal to N^* - i.e. $N^o = \lceil N^* \rceil$ - and let $w(N^o) - w_D(N^o) = \delta$.

Then $w(N^o+1) - w_D(N^o+1) \leq w(N^o) - w_D(N^o) = \delta$. By induction on $n = 0,1,2,\cdots$ we have $w(N^o+n) - w_D(N^o+n) \leq \delta$ for all $n \geq 0$. Thus $w(N) - w_D(N)$ is maximum for $N^* < N$ when $N = \lceil N^* \rceil$.

Therefore $w(N) - w_D(N)$ is maximum at either $N = \lfloor N^* \rfloor$ or $N = \lceil N^* \rceil$.

Remark:

If N^* is noninteger these two points are distinct and the one at which the maximum occurs depends on $N^* - \lfloor N^* \rfloor$ and $w(N)$.

2.5.1 Exponential Distributed Processing and Deterministic Service - M/D/1//N Model

We now consider as a special case of the foregoing a model with deterministic (constant) memory access times. This special case is interesting for two reasons. The first reason is that memory accesses on the isolated Multibus directed to the global memory have a relatively constant duration. There is still randomness associated with the access time due to such factors as read-modify-write accesses (which have a significantly longer access time than normal read and write accesses) and variations in the propagation delays of the logic circuitry and signal paths. If we consider read-modify-write accesses to be so infrequent that they can be ignored, we can get some idea of the Multibus access time distribution by referring to section 3 of Appendix A. Roughly 90% or more of the Multibus accesses to the global memory take 1.00 or 1.10 μsec. Thus a constant access time seems like a reasonable approximation in this case. However, memory accesses on the Multibus directed to local memory modules can vary over a much wider range (as indicated in Figure A.5 in Appendix A) due to the HSB traffic on the other port of the accessed memory. Thus a constant access time does not seem like a reasonable approximation in this case.

The second reason for considering the deterministic case is that the mean wait per request in the deterministic case provides a lower bound on the mean wait per request for all M/G/1//N models with the same mean processing and access times. Thus although the exact access time distribution may not be known (or may be too variable to be considered constant), we can still bound the behavior of the mean waiting time.

Theorem 2.3

Given that the mean processing and access times are the same in both the M/G/1//N system and the M/D/1//N system, the mean waiting time (queueing time) in the M/G/1//N system is bounded from below by the mean waiting time in the M/D/1//N system.

Proof:

Following Price [P3], and referring to the M/G/1//N results presented earlier, we have:

$\bar{t}_w$ is strictly increasing in L,

L is strictly decreasing in ρ,

ρ is strictly increasing in $\bar{b}$,

$\bar{b}$ is strictly decreasing in $\varphi(i)$, and

$\varphi(i)$ is strictly increasing in the function $F^*(s)$.

Thus $\bar{t}_w$ is minimized when $F^*(s)$ is minimized. Now from Jensen's Inequality [P1 p.434] $F^*(s) = E[e^{-st_a}] \geq e^{-sE[t_a]} = e^{-st_a}$, which is the transform of a deterministic function. Therefore a constant service time of duration $\bar{t}_a$ gives a lower bound on the mean waiting time among all distributions with the same mean $\bar{t}_a$.

All three methods mentioned earlier for the M/G/1//N model have been applied to the solution of the M/D/1//N model. Benson and Cox [B2] used the method of stages. They obtained a closed form solution for a service distribution consisting of a cascade of r exponential stages (called an r stage Erlangian distribution and denoted by E_r) and then took the limit as $r \rightarrow \infty$. Raskin [R1] employed an imbedded Markov chain. Jaiswal obtained the closed form solutions presented earlier using the technique of supplementary variables. In addition, Ashcroft [A3] has derived a solution for the M/G/1//N model starting with an expression for the mean busy period.

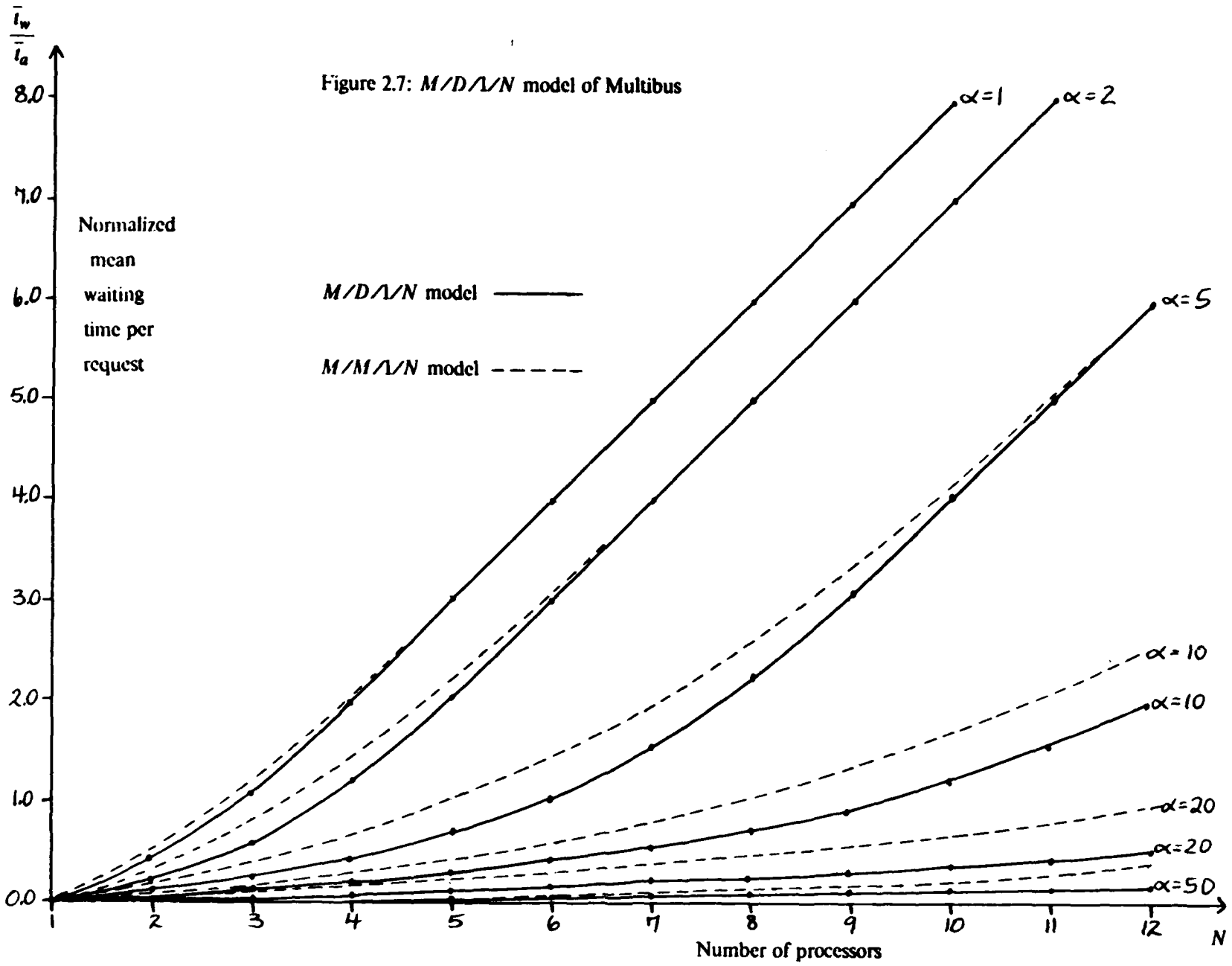


Figure 2.7: $M/D/1/N$ model of Multibus

The actual results for the mean waiting time per request in the M/D/1//N model are plotted in Figure 2.7 for the same cases as in the M/M/1//N model. (The data for this Figure is taken from Ashcroft's paper.) For purposes of comparison, the earlier M/M/1//N results are also plotted. Note that the M/M/1//N and M/D/1//N results are very similar except around the "knee" of the curves.

We also observe the following:

For a given α, the difference in mean waiting time for the M/M/1//N and M/D/1//N models first increases with N, and then decreases with N. Similarly, for a given N, the difference first increases with α and then decreases with α. The maximum difference in the mean waiting times occurs close to the "knee" at $N = N^*$ and increases with N^* (in fact the maximum difference occurred at either $N = \lfloor N^* \rfloor$ or $N = \lfloor N^* \rfloor + 1$ in the cases in which numerical results were computed).

The validity of these observations in the general case may be ascertained by examining the difference $\bar{w}(N)_{M/M/1//N} - \bar{w}(N)_{M/D/1//N} = \alpha\left(\frac{1}{\bar{b}_{N_{M/M/1//N}}} - \frac{1}{\bar{b}_{N_{M/D/1//N}}}\right)$ where $\bar{b}_N = 1 + \sum_{i=1}^{N-1} \binom{N-1}{i} \prod_{m=1}^{i} \left[\frac{1}{F^*(m\lambda)} - 1\right]$. ($\bar{b}_N$ is the normalized mean busy period given by $\bar{b}_N = \frac{\bar{b}}{\bar{t}_a}$ - see equations 2.3 and 2.4.)

For exponential service: $F^*(m\lambda) = \frac{1}{\frac{m}{\alpha} + 1}$, thus $\bar{b}_{N_{M/M/1//N}} = 1 + \sum_{i=1}^{N-1} \binom{N-1}{i} \prod_{m=1}^{i} \frac{m}{\alpha}$.

For deterministic service: $F^*(m\lambda) = e^{-\frac{m}{\alpha}}$, thus $\frac{1}{F^*(m\lambda)} - 1 = e^{\frac{m}{\alpha}} - 1 = \frac{m}{\alpha} + g(\frac{m}{\alpha})$, using the series expansion of $e^{\frac{m}{\alpha}}$ (i.e. $g(\frac{m}{\alpha}) = \sum_{n=2}^{\infty} \frac{1}{n!} \left(\frac{m}{\alpha}\right)^n$), and therefore $\bar{b}_{N_{M/D/1//N}} = 1 + \sum_{i=1}^{N-1} \binom{N-1}{i} \prod_{m=1}^{i} \left[\frac{m}{\alpha} + g(\frac{m}{\alpha})\right]$.

Rather than examining the difference $\bar{w}(N)_{M/M/1//N} - \bar{w}(N)_{M/D/1//N} \equiv \Delta w(N)$ directly, it is easier to rewrite $\Delta\bar{w}(N)$ as $\left[\frac{\bar{w}(N)_{M/M/1//N}}{w(N)_{M/D/1//N}} - 1\right] \bar{w}(N)_{M/D/1//N}$ and examine the ratio

$$\frac{\bar{w}(N)_{M/M/1//N}}{\bar{w}(N)_{M/D/1//N}} = \frac{N - N^* + \frac{\alpha}{\bar{b}_{N_{M/M/1//N}}}}{N - N^* + \frac{\alpha}{\bar{b}_{N_{M/D/1//N}}}} \equiv r_w.$$

For $N=1$, $\bar{b}_{N_{M/M/1//N}}=\bar{b}_{N_{M/D/1//N}}=1$ and hence $r_w=1$. As $N\to\infty$, $\bar{b}_{N_{M/M/1//N}}$ and $\bar{b}_{N_{M/D/1//N}}\to\infty$ (at different rates) and $N-N^*\to\infty$; hence $r_w\to 1$. For $N=N^*$, $r_w=\dfrac{\bar{b}_{N_{M/D/1//N}}}{\bar{b}_{N_{M/M/1//N}}}$ which is clearly greater than 1 for $N>1$. Thus $\Delta\bar{w}(N)$ must increase and then later decrease with N.

For small values of α, $\bar{b}_{N_{M/M/1//N}}$ and $\bar{b}_{N_{M/D/1//N}}$ are large and hence $r_w\approx 1$. For large values of α, $\bar{b}_{N_{M/M/1//N}}\approx\bar{b}_{N_{M/D/1//N}}$ and again $r_w\approx 1$. For all values of α, $\bar{b}_{N_{M/D/1//N}}\geq\bar{b}_{N_{M/M/1//N}}$. In particular, $\bar{b}_{N_{M/D/1//N}}>\bar{b}_{N_{M/M/1//N}}$ and thus $r_w>1$ for medium values of α. Since r_w is continuous in α, this is enough to conclude that $\Delta\bar{w}(N)$ increases with α and later decreases with α (although not necessarily monotonically).

2.5.2 Comments

It is difficult to say much more of interest about the M/G/1//N model without some knowledge of the access time distribution; indeed, the mean waiting time per request is completely specified by the closed form expression given earlier once the distribution is known.

From section 3.3 of Appendix A we see that all access times must be in the range 1.02 μsec to 1.82 μsec (allowing for best and worst case propagation delays and traffic on the other memory port and assuming no read-modify-writes). One might conjecture that because this access time distribution is more "deterministic" than an exponential one with the same mean (and certainly does not have the long tails of the exponential), the mean waiting time ought to be bounded from above by that for an exponential distribution with the same mean. This is indeed the case as the following argument shows.

Recall from equation 2.4 that the mean waiting time per request is given by $\dfrac{\bar{t}_w}{\bar{t}_a}=N-N^*+\dfrac{\alpha}{\bar{b}_N}$,

where $\bar{b}_N=1+\sum_{i=1}^{N-1}\left|\begin{matrix}N-1\\ i\end{matrix}\right|\prod_{m=1}^{i}\left|\dfrac{1}{F^*(s)}-1\right|$.

As discussed in the proof of Theorem 2.3, $\dfrac{\bar{t}_w}{\bar{t}_a}$ is strictly increasing in $F^*(s)$. Thus to show $\bar{t}_{w\,M/M/1//N}\geq\bar{t}_w$ it suffices to show that $F^*(i\lambda)_{M/M/1//N}\geq F^*(i\lambda)$ for all i and $\lambda\geq 0$.

Theorem 2.4

Let $F^*_{ab}(s)$ denote the Laplace transform of the probability density function $f_{ab}(x)$ with mean $\bar{x}$ where $f_{ab}(x) = 0$ for $x \notin [a,b]$; $0<a$ and $b<2a$. Let $F^*(s)_{M/M/1//N}$ denote the Laplace transform of the exponential density function with the same mean $\bar{x}$. Then $F^*(s)_{M/M/1//N} \geq F^*_{ab}(s)$ for s real and $s \geq 0$.

The proof of this theorem is given in Appendix B. For the case at hand $a = 1.02$ and $b = 1.82 < 2a$, thus $F^*(i\lambda)_{M/M/1//N} \geq F^*_{ab}(i\lambda))$ for every i and $\lambda \geq 0$.

Theorems 2.3 and 2.4 imply that the mean waiting time for the $M/G/1//N$ model as presented in section 2.5 is bounded above and below by the $M/M/1//N$ and $M/D/1//N$ models respectively, with the same mean processing and access times. Therefore a quick characterization of the mean waiting time of the $M/G/1//N$ model with any access time distribution (obeying the restrictions in Theorem 2.4) can be obtained from the $M/M/1//N$ and $M/D/1//N$ models. Furthermore, by analogy with the Pollaczek-Khinchin formula for the mean waiting time in the $M/G/1$ queue ‡, one would expect the mean waiting time to vary approximately linearly with the square of the coefficient of variation of the access time distribution given by $C_x^2 = \frac{\sigma_x^2}{\bar{x}^2}$. However, as Price [P3] points out by means of example, this can be misleading since the variance can be dominated by a few long access times which have little effect on the mean waiting time.

A reasonable model for the access time distribution is an r stage Erlangian distribution. Figure 2.8 shows how the Erlangian density function varies with r.

‡ The $M/G/1$ queue is an *open* queueing model (as opposed to the closed models considered in this chapter) with a Poisson arrival process and a general service process independent of the arrival process. The mean waiting time in the queue is $\bar{t}_w = \frac{\rho\bar{x}(1+C_x^2)}{2(1-\rho)}$ where arrivals occur at rate λ, service has mean $\bar{x}$ and variance σ_x^2, and $\rho = \lambda\bar{x}$, $C_x^2 = \frac{\sigma_x^2}{\bar{x}^2}$.

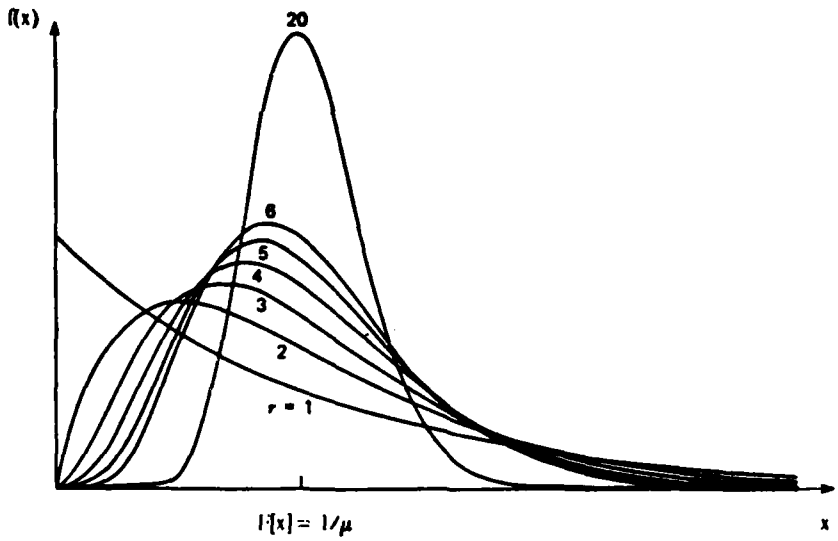


Figure 2.8: Various r stage Erlangian density functions

For the M/E_r/1//N model it is easy to show that the mean waiting time per request is upper bounded by that for the M/M/1//N, if the mean processing and access times are the same in each case. As above, to show $\bar{t}_{w_{M/M/1//N}} \geq \bar{t}_{w_{M/E_r/1//N}}$, it suffices to show that $F^*(i\lambda)_{M/M/1//N} \geq F^*(i\lambda)_{M/E_r/1//N}$ for all i and $\lambda \geq 0$.

$$F^*(i\lambda)_{M/M/1//N} = \frac{\alpha}{i+\alpha} \text{ and } F^*(i\lambda)_{M/E_r/1//N} = \left[\frac{r\alpha}{i+r\alpha}\right]^r.$$

Since $\frac{\alpha}{i+\alpha} = \left[\frac{r\alpha}{i+r\alpha}\right]^r$ for $r=1$ and $\left[\frac{r\alpha}{i+r\alpha}\right]^r$ is strictly decreasing in r, the exercise is completed. Note that in the limit as $r \to \infty$ the Erlangian distribution approaches a deterministic distribution. Thus $\bar{t}_{w_{M/M/1//N}} \geq \bar{t}_{w_{M/E_r/1//N}} \geq \bar{t}_{w_{M/D/1//N}}$.

2.6 General Processing and Exponential Access Time Distributions -- G/M/1//N Model

We now consider the effect of the processing time distribution on the mean waiting time per request. For this section we keep the service time distribution exponential to facilitate comparison with the earlier M/M/1//N model and to determine the relative effect of changes in processing and service time distributions with respect to the M/M/1//N model.

The G/M/1//N model could be solved using any of the three methods described in section 3. However they all become cumbersome because whatever method is chosen must essentially be applied N times since there are N general distributions. The state description must, explicitly or implicitly, contain the processing time completed so far at each processor that is busy and the number of requests waiting for or in service. Thus there are anywhere from 0 to N continuous variables in the state description. This leaves the imbedded Markov chain and supplementary variable methods hopelessly complicated for reasonable values of N. Direct application of the method of stages is also very complicated. However, in the special case of the G/M/1//N model - due to the exponential access time distribution - the solution of the equilibrium equations has a very simple form.

2.6.1 Product Form Solutions

In certain cases the steady-state probabilities for a system of two or more interconnected queues have the following form:

Let the vector x_i denote the state of queue i, and let π_{x_i} denote the steady state probability of that state when queue i is in isolation. Then the overall, or global, state of the system is given by $X = (x_1, x_2, \ldots, x_n)$. Denote the steady-state probability of global state X by π_X. Then $\pi_X = C\prod_{i=1}^{n}\pi_{x_i}$, where C is a normalizing constant.

Any system in which the steady-state probabilities can be expressed in such a form is said to have a *product form* solution. Product form solutions are extremely convenient in that one can dispense with solving the global equilibrium equations; it is sufficient to solve for the steady-state probabilities for each queue in isolation. In the following we summarize the main results known pertaining to product form solutions in queueing networks as described by Kelly [K1].

The principal result is the following:

Suppose there are n queues (the queue is thought of as a black box here and includes the server for that queue) and a total of k classes of customers in the overall system. For each queue i assume that no more than one customer enters or leaves the queue at any point in time. Let each customer in queue i belong to some class k in the total set of classes $K(i)$ visiting that queue and assume that customers cannot change class as they pass through the queue. Let the state of

queue i at time t be denoted by $x_i(t)$ and assume that the state information allows the number of customers of each class at the queue to be determined. If each queue i is quasi-reversible in isolation, then the equilibrium probability distribution has the product form given above.

A queue i is quasi-reversible if:

1) its state $x_i(t)$ is a stationary Markov process,
2) the arrival times of class k customers, $k \in K(i)$ after time t are independent of $x_i(t)$ before or at t,
3) the departure times of class k customers, $k \in K(i)$ after time t are independent of $x_i(t)$ at or after t, and
4) the mean rate of class k arrivals and departures is equal for every $k \in K(i)$.

If a queue is quasi-reversible, then points 2 and 3 imply that the arrival and departure processes of class k customers are independent Poisson processes.

Two types of queues are known to be quasi-reversible. In both types, the arrival process of class k customers is Poisson with rate $\lambda(k)$, giving a total arrival rate of $\lambda = \sum_k \lambda(k)$. The first type is distinguished by exponentially distributed service times with the same mean service for all classes of customers (although the mean may vary with the number of customers in the queue). Kelly [K1] describes this type of queue as follows:

Assume we are dealing with queue i and let n_i be the total number of customers in the queue.

(i) Each customer requires an amount of service which is a random variable exponentially distributed with mean μ.

(ii) A total service effort is supplied at the rate $\varphi_i(n_i)$, where $\varphi_i(n_i) > 0$ if $n_i > 0$.

(iii) A proportion $\gamma_i(l, n_i)$ of this effort is directed to the customer in position l, $(1 \le l \le n_i)$. When this customer completes service and leaves the queue, the customers in positions $l+1, l+2, \ldots, n_i$ *move to positions* $l, l+1, \ldots, n_i - 1$ *respectively.*

(iv) A customer arriving at queue i moves into position l $(1 \le l \le n_i + 1)$ with probability $\delta_i(l, n_i + 1)$. Customers previously in positions $l, l+1, \ldots, n_i$ move to positions $l+1, l+2, \ldots, n_i + 1$ respectively.

The amount of service a customer requires at queue i is assumed to be independent of the amount of service the same customer requires in other queues and independent of the amount of service all other customers in queue i require. For example, a FCFS queue with K classes of customers, each class with Poisson arrivals of rate $\lambda(k)$, $k \in K$ and the same exponentially distributed service for all customers can be described by:

$$\gamma(l, n) = \begin{cases} 1, & l = 1 \\ 0, & l = 2, \ldots, n \end{cases}$$

$$\delta(l,n)=\begin{cases}1, & l=n+1\\ 0, & l=1,\ldots,n\end{cases}$$

$$\varphi(n)=1$$

Quasi-reversible queues of the first type (also called generalized M/M/· queues), can be described by the state $x(t)=(n,c(1),\ldots,c(n))$ where n is the number of customers in the queue and $c(l), 1\le l\le n$, is the class of the customer in the lth position of the queue. The state $x(t)$ is a stationary Markov process with steady-state probability

$$\pi_x=\kappa\prod_{j=1}^{n}\frac{\lambda(c(j))}{\mu\varphi(j)},$$

where κ is a normalizing constant [K1].

The steady-state probability of the non-Markovian state $x^I(t)=(n(1),n(2),\ldots,n(K))$, where $n(k)$ is the number of customers of class k in the queue, can be found by considering all possible ways of arranging n customers in k classes.

$$\pi_{x^I}=\kappa\left[\prod_{j=1}^{n}\frac{1}{\varphi(j)}\right]\frac{n!}{n(1)!\,n(2)!\cdots n(K)!}\rho_1^{n(1)}\rho_2^{n(2)}\cdots\rho_k^{n(K)} \tag{2.5}$$

$$\rho_k=\frac{\lambda(k)}{\mu}$$

Finally, the steady-state probability of the non-Markovian state $x^{II}(t)=(n)$, can be found by summing π_{x^I} over all possible ways to arrange n customers.

$$\pi_{x^{II}}=\kappa\left[\prod_{j=1}^{n}\frac{1}{\varphi(j)}\right]\sum_{n(1)+n(2)+\cdots+n(K)=n}\frac{n!}{n(1)!\,n(2)!\cdots n(K)!}\rho_1^{n(1)}\rho_2^{n(2)}\cdots\rho_k^{n(K)},$$

(where

$$\sum_{n(1)+n(2)+\cdots+n(K)=n}$$

means

$$\sum_{n(1)=0}^{n}\sum_{n(2)=0}^{n}\cdots\sum_{n(K)=0}^{n}$$

such that $n(1)+n(2)+\cdots+n(K)=n$ at all times) yielding

$$\pi_{x^{II}}=\kappa\rho^n\left[\prod_{j=1}^{n}\frac{1}{\varphi(j)}\right],\quad \rho=\sum_{k=1}^{K}\frac{\lambda(k)}{\mu}.$$

The second type of quasi-reversible queues is described by Kelly [K1] in a similar manner. The description is the same as above except for:

(i) The service required by a customer is a random variable from an arbitrary distribution which may depend on the class of the customer.

(iv)Same as above except the symmetry condition $\delta(l,n_i+1)=\gamma(l,n_i+1)$ is imposed for every $l=1,\ldots,n_i+1$.

Queues of this second type are called symmetric queues. For example, a server- sharing queue (essentially a round-robin queue with infinitesimal quantum size so all customers are effectively simultaneously in service) can be described by $\gamma(l,n)=\frac{1}{n}$, $l=1,2,\ldots,n$; $n\geq 0$, and $\varphi(n)=1$. A last come first served (LCFS) queue with preemption can be described by $\gamma(l,n)=1$, $l=n$, $n=1,2,\ldots$, and $\varphi(n)=1$ for $n\geq 0$. Finally, an infinite server queue can be described by $\gamma(l,n)=n$, $n\geq 1$, and $\varphi(n)=\frac{1}{n}$, $l=1,2,\ldots,n$; $n\geq 1$.

Note that a FCFS queue is *not* a symmetric queue. Therefore a FCFS queue with anything other than the same exponentially distributed service for all customers (as described in the first type of quasi-reversible queues) does not fit into the two types of quasi-reversible queues just described. Indeed, such FCFS queues are not quasi-reversible since the departure process at time t is not independent of the state $x(t)$ after t (i.e. given the state describing the customers in the queue and the service time expended on the customer presently in service, some information about the next departure time(s) can be ascertained). As a result, no product form solutions are known for such FCFS queues.

As for the generalized M/M/· queues mentioned earlier, we can describe a symmetric queue by Markov process, find the resulting steady-state probabilities, and then sum over various states to find the steady-state marginal probabilities. Skipping the intermediate steps (which follow directly from the steady-state probability distribution given in Kelly [K1]), we have for the non-Markovian state $X(t)=(n,c(1),\ldots,c(n))$ (n and $c(l)$ are as defined before) the steady state probability distribution

$$\pi_X=\prod_{j=1}^{n}\frac{\lambda(c(j))}{\varphi(j)}E[z(c(j))],$$

where $E[z(c(j))]$ is the mean service requirement of a class $c(j)$ customer.

For the non-Markovian states $x^I(t)=(n(1),\ldots,n(K))$ and $x^{II}(t)$ (n), we get the same results as before with ρ_k and ρ now as follows:

$$\rho_k=\lambda(k)E[z(k)] \qquad \rho=\sum_{k=1}^{K}\rho_k$$

The only feature of networks of quasi-reversible queues that has not yet been discussed is the routing of customers within the network. The routing is formulated as follows: upon departing from a queue a customer of class k joins class l with probability r_{kl}.† By adding a sufficient number of classes, routing can include dependency on previously visited queues and classes as well as on the initial class. For example, a deterministic route can correspond to each input class. In addition, routing can depend on quite detailed previous history (such as actual service times) provided that the next class depends only on the present class and that the queues remain quasi-reversible with respect to the classes.

The effective arrival rate of customers of class k to the queueing network is $\lambda^{eff}(k)=\lambda(k)+\sum_l \lambda^{eff}(l)r_{lk}$, where $\lambda(k)$ is the arrival rate of class k customers from a source external to the network (external arrivals are assumed to belong to a Poisson process). The steady-state probability distribution of each quasi-reversible queue in isolation is computed assuming the the arrival process of each customer class is Poisson with rate given by the effective arrival rate of that class in the network. The overall steady-state probability distribution of the network is the product of the steady-state probability distribution of each queue in isolation.

In the steady state the various classes of customers in the network can either:

1. form closed loops with no arrivals or departures, or
2. form no loops.

(Closed loops with arrivals and no departures and closed loops with departures and no arrivals obviously cannot exist in steady-state.)

If all classes of customers form no loops, then the effective arrival rates are uniquely defined by $\lambda^{eff}(k)=\lambda(k)+\sum_l \lambda^{eff}(l)r_{lk}$. In this case the network is said to be open and the normalizing constant in the product form equation is $C=1$. If all classes form closed loops with no arrivals or departures then the effective arrival rates are given up to an multiplicative constant by $\lambda^{eff}(k)=\sum_l \lambda^{eff}(l)r_{lk}$. In this case the network is said to be closed and the normalizing constant is such that the sum of all probabilities is 1. Otherwise the network is said to be mixed. In this case $\lambda^{eff}(k)$ is uniquely determined for those classes that form no loops and determined up to a constant for those classes that form closed loops.

We conclude this section on product form solutions by noting that the same results have

† Departures from the network can be handled by defining a certain class for departed customers. However, it is traditional to avoid defining a explicit class for departures, resulting in $\sum_k r_{kl}<1$ if class k customers can leave the network.

been reached by others, notably Baskett et al [B1], by close examination of the global balance equations in the method of stages. For certain cases these global balance equations reduce to local balance equations for which it is easy to determine the equilibrium probability distribution. Kelly's treatment via the quasi-reversibility of the queues generalizes earlier work (distributions with non-rational Laplace transforms and any queue fitting the description given earlier for generalized M/M/· queues or symmetric queues can be treated) and unifies it through the concept of quasi-reversibility.

2.6.2 G/M/1//N Model as a Queueing Network

The G/M/1//N model can be considered as a closed queueing network with a FCFS queue with an exponential service time distribution - same mean for all customers - and an infinite server as depicted below:

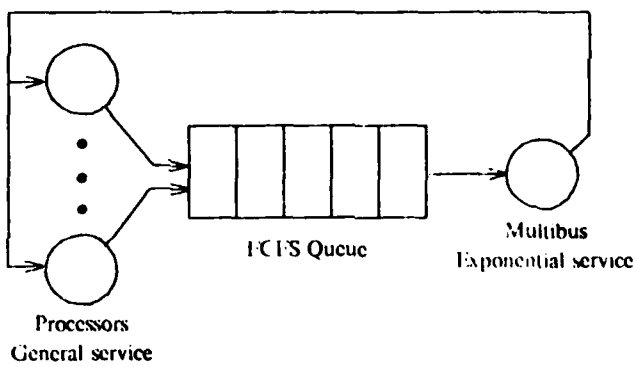


Figure 2.9: Queueing network for G/M/1//N model

All customers are identical. Let all customers in the infinite server queue be class 1 with mean service time $\bar{t}_p$. Let all customers in the FCFS queue be class 2 with mean service time $\bar{t}_a$. Thus $r_{12} = r_{21} = 1$ and $\lambda^{eff}(1) = \lambda^{eff}(2)$. Each queue is quasi-reversible in isolation. Therefore from section 2.6.1 we have for the infinite server queue with a state of $x_1 = (n_1)$:

$$\pi_{x_1} = \kappa_1 \frac{\rho_1^{n_1}}{n_1!}, \quad \rho_1 = \lambda^{eff}(1)\bar{t}_p.$$

For the FCFS queue we have for a state of $x_2 = (n_2)$:

$$\pi_{x_2} = \kappa_2 \rho_2^{n_2}, \quad \rho_2 = \lambda^{eff}(2)\bar{t}_p.$$

Thus for the overall state $X = (x_1, x_2) = (n_1, n_2)$: we have

$$\pi_X = \kappa_1 \kappa_2 \frac{\left[\lambda^{eff}(1)\bar{t}_p\right]^{n_1}}{n_1!} \left[\lambda^{eff}(2)\bar{t}_a\right]^{n_2}$$

Since $n_1 + n_2 = N$, the state reduces to $X = (n_2)$ and the steady-state probability distribution of n_2 customers in the FCFS queue is:

$$\pi_X = \kappa \frac{N!}{(N - n_2)!} \left[\frac{\bar{t}_a}{\bar{t}_p}\right]^{n_2}, \quad 0 \le n_2 \le N ,$$

and κ is a normalizing constant ($\kappa = \dfrac{\kappa_1 \kappa_2 \left[\lambda^{eff} \bar{t}_p\right]^N}{N!}$).

Aside from the change in notation, this equation is exactly the same as equation 2.1 in section 2.4 for the steady-state probability of n_2 customers in the M/M/1//N system. Therefore both the M/M/1//N and G/M/1//N models have exactly the same mean waiting times per request if the mean processing and access times are the same respectively for each model. (The reader is thus referred to the graph for the M/M/1//N case in lieu of a graph here.) This is a surprising result considering that the processing time distribution is arbitrary. As we shall see in the next section, the key to this behavior is the exponential distribution of the service time at the FCFS queue.

2.7 General Processing and Access Time Distributions - G/G/1//N

In this section we consider the full generality of the basic model studied so far. Unfortunately, the G/G/1//N model is difficult to solve exactly. We no longer have the convenience of memoryless (i.e. exponential) processing times as in the M/G/1//N case or the luck to have a product form solution due to the exponential service time as in the G/M/1//N case. Imbedded Markov chains and supplementary variable methods are hopelessly complex. This leaves the method of stages, as complicated as it may be. Of course, as mentioned in section 2.5, explicit closed form solutions cannot generally be obtained with the method of stages. Simulation is also a possible alternative. However, simulation is not very useful to systematically determine the effect of various parameter changes, so we leave it as a last resort. Approximation, which does not suffer from this weakness, is perhaps the most attractive alternative in this case. Rather than pursue a lengthy investigation of approximation techniques for the G/G/1//N system, we refer the reader to Halachmi and Franta [H1] and Whitt [W2].

One simple way to approximate the solution of the G/G/1//N model is to replace the FCFS queue by either a server-sharing queue or a LCFS queue. Both of these queues are symmetric and the processors can be represented by an infinite server queue as in section 2.6.2. Therefore both queues are quasi-reversible and a product form solution exists. In fact the analysis and solution is exactly the same as that in section 2.6.2! Thus this approximation gives no more information than that in section 2.4. (Actually it does: it demonstrates that under different service disciplines the G/G/1//N model has very simple solutions.)

2.7.1 Mean Waiting Time in PH/PH/1//N Model

In this section we derive, using the method of stages, a solution for the mean waiting time per request in the G/G/1//N model. Our approach is to relate the solution of the G/G/1//N model to the solution of the G/G/1//(N-1) model (i.e. the same model - same processing and access time distributions - just one less processor) and then find the solution by solving a smaller problem based on the solution of the G/G/1//(N-1) model. This recursive approach was motivated by the proof of Theorem 2.2 in Appendix B. Herzog, Woo, and Chandy [H2] have outlined in general terms the solution of queueing problems by a recursive technique so the concept we apply is not new. However, we have not found any references in the literature concerning recursive techniques specifically applied to the G/G/1//N system. General motivation for much of the content in this section, such as the block partitioning of the generator matrix and the PH distribution, is due to the work of Neuts [N1].

Neuts has studied continuous time Markov processes with a countably infinite number of states where the generator matrix † has the following (canonical) block matrix form:

† The nondiagonal elements of a generator matrix Q, i.e. q_{ij} for $i \neq j$, indicate the transition rate from state i to state j in the associated continuous time Markov process. The diagonal elements are given by

$$Q = \begin{bmatrix} B_0 & A_0 & 0 & 0 & . & . \\ B_1 & A_1 & A_0 & 0 & . & . \\ B_2 & A_2 & A_1 & A_0 & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{bmatrix}$$

where all matrices are $m \times m$. Neuts [N1] has shown the following result concerning such processes:

> If the matrix Q is irreducible and positive recurrent (explained below), then the stationary probability vector $\underline{\pi}$ of Q when partitioned to agree with the partitioning of Q has the matrix-geometric form:
>
> $$\underline{\pi}_i = \underline{\pi}_0 R^i \quad i > 0$$
>
> where the matrix R is the minimal nonnegative* solution of $\sum_{k=0}^{\infty} R^k A_k = 0$.

The matrix Q is irreducible if the system has no independent subsystems; that is, if all subsystems interact and are dependent. This ensures that the steady state solution (if it exists) is independent of the initial state. It is usually evident by inspection or construction that Q is irreducible. Requiring that Q be positive recurrent is essentially just requiring that the process is stable (i.e. the queue size does not grow indefinitely) so that a steady state exists. We will not be concerned about positive recurrence here since our closed system G/G/1//N model will have only a finite number of states and we will assume it to be irreducible; thus the corresponding matrix Q will necessarily be positive recurrent.

We will hypothesize that the steady state probability vector of the G/G/1//N model (when represented by the method of stages) has a similar matrix-geometric form. Our G/G/1//N model will have only a finite number of states; thus our approach will be similar to but different than that outlined above for infinite dimensional systems. The key aspect of Neuts' result is the matrix-geometric form of the steady state probability vector.

In the following, we will use the phase distribution (denoted by PH) originated by Neuts [N1]. The PH distribution is really just a convenient matrix formulation of the method of stages. (Indeed, some authors use "phase" instead of "stage".) This formulation provides a much needed

$q_{ii} = -\sum_{j \neq i} q_{ij}$. A generator matrix Q has the property that $\underline{\pi} Q = 0$ in the steady state where $\underline{\pi}$ is the vector of steady state probabilities.

* Minimal in the sense that $R \leq X$ (element-wise) for any other solution $X \neq R$ of $\sum_{k=0}^{\infty} X^k A_k = 0$.

structure for the method of stages and unifies many widely disparate formulations of Erlangian, series/parallel, and stage type distributions. PH distributions are, however, a subset of those obtained by Cox [C4] in that all the poles of the Laplace transform of a PH distribution are real (as opposed to the complex poles allowed in Cox's formulation). This restriction to real poles allows PH distributions to be directly related to finite state Markov processes and allows them to be realizable using only real arithmetic.

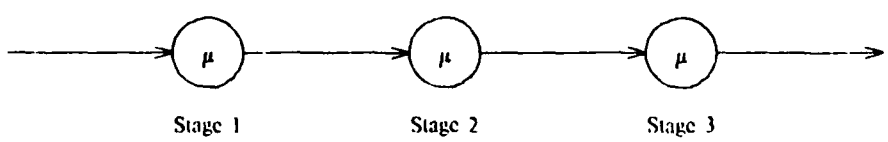
A continuous parameter PH distribution $F(\underline{x})$ on $[0,\infty)$ has the following formulation:

$$\hat{Q} = \begin{vmatrix} T & \underline{T}^0 \\ 0 & 0 \end{vmatrix}$$

where T is a $m \times m$ nonsingular (i.e. invertible) matrix, $\underline{T}^0$ is a $m \times 1$ column vector, and $T\underline{e} + \underline{T}^0 = 0$ where $\underline{e}$ is an $m \times 1$ column vector of 1's. The matrix $\hat{Q}$ represents the generator of a $m+1$ state Markov process. The transition between any state $i \in 1, 2, \cdots, m$ and state $j \in 1, 2, \cdots, m$, $j \neq i$, is governed by an exponential distribution with rate T_{ij}. Similarly, the transition between any state $i \in 1, 2, \cdots, m$ and state $m+1$ is governed by an exponential distribution with rate T_i^0 $(T_{ii} = -(T_i^0 + \sum_{j \neq i} T_{ij}))$. The states $1, 2, \cdots, m$ are transient and state $m+1$ is absorbing. The initial probability vector is $(\underline{\alpha}, \alpha_{m+1})$ where $\underline{\alpha}$ is a $1 \times m$ row vector and α_i is the probability of starting in phase i. ($\underline{\alpha}\underline{e} + \alpha_{m+1} = 1$.) The random variable x is defined as the time until absorption in the above Markov process. The distribution of x is $F(x) = 1 - \underline{\alpha} e^{Tx} \underline{e}$, $x > 0$. The pair $(\underline{\alpha}, T)$ is called the representation of $F(x)$ and the dimension of the square matrix T is called the order of $F(x)$.

As an example, a third order Erlangian distribution (E_3) can be formulated as a PH distribution as follows:

Erlangian:



Each stage has an exponential distribution with rate μ

PH distribution:

$$S = \begin{bmatrix} -\mu & \mu & 0 \\ 0 & -\mu & \mu \\ 0 & 0 & -\mu \end{bmatrix} \qquad \underline{S}^0 = \begin{bmatrix} 0 \\ 0 \\ \mu \end{bmatrix} \qquad \underline{\alpha} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

We now consider the G/G/1//N model where the processing time distribution is PH with representation $(\underline{\alpha},T)$, order m, and $\alpha_{m+1}=0$ and the access time distribution is PH with representation $(\underline{\beta},S)$, order v, and $\beta_{v+1}=0$. The states in the resulting PH/PH/1//N model can be described by:

$$(n,s,t_1,\ldots,t_N)$$

where n is the number of requests queued for or in service, $0 \le n \le N$; s is the current phase of the service (i.e. access time distribution), $1 \le s \le v$; t_i is the current phase of the processing at processor i, $1 \le t_i \le m$; and s and t are simply omitted (or taken to be zero) when there is no request in service or when processor i is idle, respectively.

This gives a total of $m^N + \sum_{j=0}^{N} vm^j$ states. Since all the processors are assumed to be identical, we can reduce the number of states by considering the state description:

$$(n,s,p_1,p_2,\ldots,p_m)$$

where p_i denotes the number of processors in which the processing is in phase $i, 1 \le i \le m$, $0 \le p_i \le N-n$, $\sum_{i=1}^{m} p_i = N-n$, and n and s are as before. This gives a total of $\binom{N+m-1}{m-1} + \sum_{j=0}^{N-1} v\binom{N-j+m-1}{m-1} + v$ states.

As an example, consider the $E_2/E_2/1//N$ system with $N=3$. The state transition diagram for the system is given in Figure 2.10. The corresponding generator matrix, if the states are labeled in lexicographical order (i.e. in order (0,0,0,3),(0,0,1,2),(0,0,2,1),(0,0,3,0),(1,1,0,2), (1,1,1,1),(1,1,2,0),(1,2,0,2),(1,2,1,1),(1,2,2,0),(2,1,0,1), (2,1,1,0),(2,2,0,1),(2,2,1,0),(3,1,0,0),(3,2,0,0)), is given in Figure 2.11. Notice the block tridiagonal form of Q. A process having a matrix Q of this form is called a quasi-birth death (QBD) process. Figure 2.12 shows the generator matrix for the general case of a PH/PH/1//N system with the processing time distribution of order 2, the access time distribution of order 2, and $N=3$. Again, note the block tridiagonal form of Q.

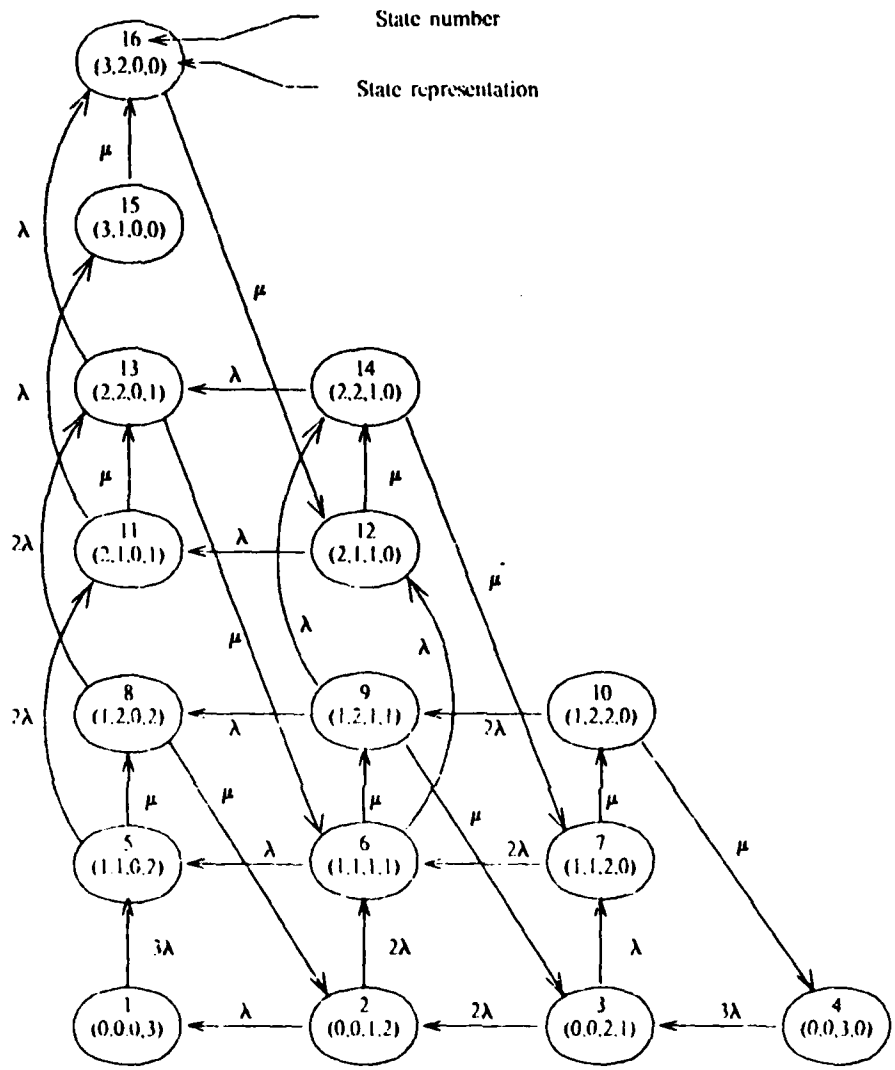


Figure 2.10: State transition diagram for $E_2/E_2/1//3$ system

$$T = \begin{bmatrix} -\lambda & \lambda \\ 0 & -\lambda \end{bmatrix} \qquad \underline{T}^0 = \begin{bmatrix} 0 \\ \lambda \end{bmatrix} \qquad \underline{\alpha} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$S = \begin{bmatrix} -\mu & \mu \\ 0 & -\mu \end{bmatrix} \qquad \underline{S}^0 = \begin{bmatrix} 0 \\ \mu \end{bmatrix} \qquad \underline{\beta} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix}
-3\lambda & 0 & 0 & 0 & 3\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\lambda & -3\lambda & 0 & 0 & 0 & 2\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2\lambda & -3\lambda & 0 & 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3\lambda & -3\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2\lambda-\mu & 0 & 0 & \mu & 0 & 0 & 2\lambda & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \lambda & -2\lambda-\mu & 0 & 0 & \mu & 0 & 0 & \lambda & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2\lambda & -2\lambda-\mu & 0 & 0 & \mu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mu & 0 & 0 & 0 & 0 & 0 & -2\lambda-\mu & 0 & 0 & 0 & 0 & 2\lambda & 0 & 0 & 0 \\
0 & 0 & \mu & 0 & 0 & 0 & 0 & \lambda & -2\lambda-\mu & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\
0 & 0 & 0 & \mu & 0 & 0 & 0 & 0 & 2\lambda & -2\lambda-\mu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda-\mu & 0 & \mu & 0 & \lambda & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & -\lambda-\mu & 0 & \mu & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mu & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda-\mu & 0 & 0 & \lambda \\
0 & 0 & 0 & 0 & 0 & 0 & \mu & 0 & 0 & 0 & 0 & 0 & \lambda & -\lambda-\mu & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & \mu \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu & 0 & 0 & 0 & -\mu
\end{bmatrix}$$

Figure 2.11: Generator matrix for $E_2/E_2/1//3$ example

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \qquad \underline{T}^0 = \begin{bmatrix} T_1^0 \\ T_2^0 \end{bmatrix} \qquad \underline{\alpha} = \begin{bmatrix} \alpha_1 & \alpha_2 \end{bmatrix}$$

$$S = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \qquad \underline{S}^0 = \begin{bmatrix} S_1^0 \\ S_2^0 \end{bmatrix} \qquad \underline{\beta} = \begin{bmatrix} \beta_1 & \beta_2 \end{bmatrix}$$

$$Q = \begin{bmatrix}
\Sigma & 3T_{12} & 0 & 0 & 3\beta_1 T_2^0 & 0 & 0 & 3\beta_2 T_2^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T_{12} & \Sigma & 2T_{21} & 0 & \beta_1 T_1^0 & 2\beta_1 T_2^0 & 0 & \beta_2 T_1^0 & 2\beta_2 T_2^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2T_{12} & \Sigma & T_{21} & 0 & 2\beta_1 T_1^0 & \beta_1 T_2^0 & 0 & 2\beta_2 T_1^0 & \beta_2 T_2^0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3T_{12} & \Sigma & 0 & 0 & 3\beta_1 T_1^0 & 0 & 0 & 3\beta_2 T_1^0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\alpha_2 S_1^0 & \alpha_1 S_1^0 & 0 & 0 & \Sigma & 2T_{21} & 0 & S_{12} & 0 & 0 & 2T_2^0 & 0 & 0 & 0 & 0 & 0 \\
0 & \alpha_2 S_1^0 & \alpha_1 S_1^0 & 0 & T_{12} & \Sigma & T_{21} & 0 & S_{12} & 0 & T_1^0 & T_2^0 & 0 & 0 & 0 & 0 \\
0 & 0 & \alpha_2 S_1^0 & \alpha_1 S_1^0 & 0 & 2T_{12} & \Sigma & 0 & 0 & S_{12} & 0 & 2T_1^0 & 0 & 0 & 0 & 0 \\
\alpha_2 S_2^0 & \alpha_1 S_2^0 & 0 & 0 & S_{21} & 0 & 0 & \Sigma & 2T_{21} & 0 & 0 & 0 & 2T_2^0 & 0 & 0 & 0 \\
0 & \alpha_2 S_2^0 & \alpha_1 S_2^0 & 0 & 0 & S_{21} & 0 & T_{12} & \Sigma & T_{21} & 0 & 0 & T_1^0 & T_2^0 & 0 & 0 \\
0 & 0 & \alpha_2 S_2^0 & \alpha_1 S_2^0 & 0 & 0 & S_{21} & 0 & 2T_{12} & \Sigma & 0 & 2T_1^0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \alpha_2\beta_1 S_1^0 & \alpha_1\beta_1 S_1^0 & 0 & \alpha_2\beta_2 S_1^0 & \alpha_1\beta_2 S_1^0 & 0 & \Sigma & T_{21} & S_{12} & 0 & T_2^0 & 0 \\
0 & 0 & 0 & 0 & \alpha_2\beta_1 S_1^0 & \alpha_1\beta_1 S_1^0 & 0 & \alpha_2\beta_2 S_1^0 & \alpha_1\beta_2 S_1^0 & 0 & T_{12} & \Sigma & 0 & S_{12} & T_1^0 & 0 \\
0 & 0 & 0 & 0 & \alpha_2\beta_1 S_2^0 & \alpha_1\beta_1 S_2^0 & 0 & \alpha_2\beta_2 S_2^0 & \alpha_1\beta_2 S_2^0 & 0 & S_{21} & 0 & \Sigma & T_{21} & 0 & T_2^0 \\
0 & 0 & 0 & 0 & \alpha_2\beta_1 S_2^0 & \alpha_1\beta_1 S_2^0 & 0 & \alpha_2\beta_2 S_2^0 & \alpha_1\beta_2 S_2^0 & 0 & 0 & S_{21} & T_{12} & \Sigma & 0 & T_1^0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_2\beta_1 S_1^0 & \alpha_2\beta_2 S_1^0 & \alpha_2\beta_2 S_1^0 & \alpha_1\beta_2 S_1^0 & \Sigma & S_{12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_2\beta_1 S_2^0 & \alpha_2\beta_2 S_2^0 & \alpha_2\beta_2 S_2^0 & \alpha_1\beta_2 S_2^0 & S_{21} & \Sigma
\end{bmatrix}$$

Σ denotes $Q_{ii} = -\sum_{j \neq i} Q_{ij}$.

Figure 2.12: Generator matrix for $PH_2/PH_2/1//3$ system

If we label the states in the same lexicographical order in the general case, then we obtain the generator:

$$Q = \begin{vmatrix} B_0 & C_0 & 0 & 0 & . & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & . & 0 & 0 \\ 0 & A_2 & B_2 & C_2 & . & 0 & 0 \\ 0 & 0 & A_3 & B_3 & . & 0 & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & B_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & . & A_N & B_N \end{vmatrix}$$

where:

B_i is a square matrix of dimension $v\begin{pmatrix} N-i+m-1 \\ m-1 \end{pmatrix}$ denoting the transition rates between states with i requests in the queue;

A_i is a $v\begin{pmatrix} N-i+m-1 \\ m-1 \end{pmatrix} \times v(i)\begin{pmatrix} N-i+m \\ m-1 \end{pmatrix}$ matrix denoting the transition rates from states with i requests in the queue to states with $i-1$ requests in the queue ($v(i)=1$ if $i=1$ and v otherwise);

and, C_i is a $v\begin{pmatrix} N-i+m-1 \\ m-1 \end{pmatrix} \times v\begin{pmatrix} N-i+m-2 \\ m-1 \end{pmatrix}$ matrix denoting the transition rates from states with i requests in the queue to states with $i+1$ requests in the queue.

More details about these matrices will be given later as necessary. We partition the steady state probability vector $\underline{\pi}$ (given by $\underline{\pi}Q=0$ and $\sum_i \pi_i = 1$) into the vectors $\underline{\pi}_0, \underline{\pi}_1, \ldots, \underline{\pi}_N$ matching the partitioning of Q. The steady state equations are now:

$$\underline{\pi}_0 B_0 + \underline{\pi}_1 A_1 = 0 \tag{2.6}$$

$$\underline{\pi}_{i-1} C_{i-1} + \underline{\pi}_i B_i + \underline{\pi}_{i+1} A_{i+1} = 0\,, \quad 0 < i < N \tag{2.7}$$

$$\underline{\pi}_{N-1} C_{N-1} + \underline{\pi}_N B_N = 0 \tag{2.8}$$

One way to solve these equations is to adapt Neuts' matrix-geometric approach. Since the matrices are now functions of i, consider a rate matrix that is a function of i i.e. $R(i)$, and guess that $\underline{\pi}_i$ has the form $\underline{\pi}_i = \underline{\pi}_0 R(1)R(2)\cdots R(i-1)R(i)$. Substituting this expression for $\underline{\pi}_i$ into the steady state equations we obtain:

$$\underline{\pi}_{N-1}(C_{N-1} + R(N)B_N) = 0$$

$$\underline{\pi}_{i-1}(C_{i-1} + R(i)B_i + R(i)R(i+1)A_{i+1}) = 0\,, \quad 0 < i < N$$

$$\underline{\pi}_0(B_0 + R(1)A_1) = 0$$

If the appropriate inverses exist we have:

$$R(N) = -C_{N-1}B_N^{-1}$$

$$R(i) = -C_i(B_i + R(i+1)A_{i+1})^{-1}, \quad 0<i<N$$

$$R(1) = -B_0A_1^{-1}$$

A solution technique by iterative substitution is now apparent. This particular matrix-product approach is again - to the best of our knowledge - new. However, it is a rather infeasible approach. The main difficulty is posed by finding the inverses of the various matrices. For large N and even just small values of m and v, the dimension of B_i for small i is very large, implying that large dimensional matrices must be inverted. Finding the inverses of large matrices is computationally very inefficient. Furthermore, the inverse of a sparse matrix is usually quite dense. Therefore it is difficult to use any sparsity present in the A_i, B_i, and C_i matrices to reduce the computational requirements in any of the other matrix operations. The non-sparsity also implies large storage requirements. Another difficulty is posed by the varying dimensions of all the matrices involved: even $R(i)$ has a size that is a function of i. This makes any practical implementation difficult and complex since the solution of each $R(i)$ is essentially a special case. Finally, a great deal of work is required for the solution with N processors ($N+1$ matrix inverses and many matrix multiplies and adds) and it must all be repeated if we also want the solution for $N+1$ processors.

The key idea in this section is the following simple observation.

Some of the steady state probabilities of the G/G/1//N system are related by a multiplicative constant to the steady state probabilities of the same system with one less processor (i.e. G/G/1//(N-1)). Specifically, for our PH/PH/1//N system with state $(n,s,p_1,p_2,\ldots,p_m)$ and steady state probabilities for N processors denoted by $\pi(N)(n,s,p_1,\ldots,p_m)$ we have:

$$\pi(N)(n,s,p_1,\ldots,p_m) = C\,\pi(N-1)(n-1,s,p_1,\ldots,p_m), \quad 2\le n\le N$$

where C is a constant. The proof of this relation for the general case (i.e. for the G/G/1//N system) is given during the proof of Theorem 2.2 in Appendix B.

Therefore, if we denote our earlier partitioned steady state vector as $\pi^N = (\pi_0^N, \ldots, \pi_N^N)$ for the PH/PH/1//N system and $\pi^{N-1} = (\pi_0^{N-1}, \ldots, \pi_{N-1}^{N-1})$ for the identical system with one less processor, we have

$$\pi^N = C\,(\pi_0^{N-1}, \ldots \tag{2.9}$$

We can determine π^N and π^{N-1} [illegible] π^N [illegible]

$$\underline{\pi}_0^N B_0 + \underline{\pi}_1^N A_1 = 0 \tag{2.10(a)}$$

$$\underline{\pi}_0^N C_0 + \underline{\pi}_1^N B_1 = -\underline{\pi}_2^N A_2 = -C\underline{\pi}_1^{N-1} A_2 \tag{2.11(b)}$$

Assuming that Q is irreducible (which we will assume in the rest of this section), equations 2.10(a) and 2.10(b) represent $\binom{N+m-1}{m-1} + v\binom{N+m-2}{m-1}$ linearly independent equations in the same number of unknowns. If Q is irreducible, all the rows of Q are linearly independent, and thus B_0^{-1} exists, yielding

$$\underline{\pi}_0^N = -\underline{\pi}_1^N A_1 B_0^{-1} \quad \text{and} \quad \underline{\pi}_1^N (B_1 - A_1 B_0^{-1} C_0) = -C\underline{\pi}_1^{N-1} A_2.$$

Finally $(B_1 - A_1 B_0^{-1} C_0)^{-1}$ also exists if Q is irreducible, yielding

$$\underline{\pi}_1^N = -C\underline{\pi}_1^{N-1} A_2 (B_1 - A_1 B_0^{-1} C_0)^{-1}$$

Let π_i^N denote the steady state probability of i requests in the queue. That is, $\pi_i^N = \underline{\pi}_i^N \cdot \underline{e}$ where $\underline{e}$ denotes a column vector of 1's of appropriate dimension. Then the constant C can be determined by the requirement that $\sum_{i=0}^{N} \pi_i^N = 1$. Therefore we now have a recursive formulation for determining $\underline{\pi}_i^N$, $0 \le i \le N$, for any $N \ge 1$. The solution for $N = 1$ can be found by solving $\underline{\pi}_0^1 B_0 + \underline{\pi}_1^1 A_1 = 0$ and $\underline{\pi}_0 C_0 + \underline{\pi}_1^1 B_1 = 0$ where B_0 is $m \times m$, A_1 is $m \times v$, C_0 is $v \times m$, and B_1 is $v \times v$. (Note that the dimensions of the matrices are functions of N.) If Q is irreducible we have $\underline{\pi}_1^1 = -\underline{\pi}_0^1 (B_1 - A_1 B_0^{-1} C_0)^{-1}$ where the inverses exist. In addition we have $\underline{\pi}_1^1 \cdot \underline{e} + \underline{\pi}_0^1 \cdot \underline{e} = 1$, yielding $\underline{\pi}_0^1 (I - (B_1 - A_1 B_0^{-1} C_0)^{-1}) \cdot \underline{e} = 1$. This equation is easy to solve for reasonable values of m and v.

The mean waiting time for any N can be determined by applying Little's Law twice, as in section 2.3, to yield

$$\frac{C\sum_{i=1}^{N-1}(i+1)\pi_i^{N-1} + \pi_1^N}{C\sum_{i=1}^{N-1}\pi_i^{N-1} + \pi_1^N} - 1 \tag{2.11}$$

Since the normalization factor for the π_i^N cancels out of equation 2.11, it is not necessary to determine the constant C in equation 2.9 if the π_i^N are just being used to compute $\frac{\bar{t}_w}{t_a}$.

To avoid the computational inefficiencies associated with the matrix inversions and to retain the advantages afforded by sparse matrices, it is best to solve equations 2.10 and 2.11 using Gaussian elimination or Gauss Siedel iteration. The B_0 matrix is very sparse. In the following, the state corresponding to row i is denoted by $(n^i, v^i, p_1^i, \ldots, p_m^i)$ and the state corresponding to column j

is denoted by $(n^j, s^j, p_1^j, \ldots, p_{in}^j)$. Element (i,j) of B_0 is given by:

$$i)\ i \neq j:\ (B_0)_{ij} = \begin{cases} p_l^j T_{lk} & \text{if } n^i = n^j = 0,\ s^i = s^j, \text{ and } l \text{ and } k \text{ are the unique values (if any) such that} \\ & \quad p_q^i = p_q^j, \text{ for every } q \neq l,k, \text{ and } p_l^j = p_l^i - 1 \geq 0,\ p_k^j = p_k^i + 1 \leq N \\ 0 & \textit{otherwise} \end{cases}$$

$$ii)\ i = j:\ (B_0)_{ii} = -\sum_{i \neq j} (B_0)_{ij} - \sum_{i \neq j} (C_0)_{ij}$$

The C_0 matrix is in general not as sparse. Element (i,j) of C_0 is given by:

$$(C_0)_{ij} = \begin{cases} p_l^i \underline{T}_l^0 \beta_k & \text{if } n^i = 0,\ n^j = 1,\ s^i = 0,\ s^j = k, \text{ and } l \text{ is the unique value (if any) such that} \\ & \quad p_q^i = p_q^j, \text{ for every } q \neq l \text{ and } p_l^j = p_l^i - 1 \geq 0 \\ 0 & \textit{otherwise} \end{cases}$$

The B_1 matrix is again very sparse. Element (i,j) of B_1 is given by:

$$i)\ i \neq j:\ (B_1)_{ij} = \begin{cases} p_l^j T_{lk} & \text{if } n^i = n^j = 1,\ s^i = s^j, \text{ and } l \text{ and } k \text{ are the unique values (if any) such that} \\ & \quad p_q^i = p_q^j, \text{ for every } q \neq l,k, \text{ and } p_l^j = p_l^i - 1 \geq 0,\ p_k^j = p_k^i + 1 \leq N - 1 \\ S_{tu} & \text{if } n^i = n^j = 1,\ s^i = t,\ s^j = u,\ p_q^i = p_q^j \\ 0 & \textit{otherwise} \end{cases}$$

$$ii)\ i = j:\ (B_1)_{ii} = -\sum_{i \neq j} (B_1)_{ij} - \sum_{i \neq j} (A_1)_{ij} - \sum_{i \neq j} (C_1)_{ij}$$

where

$$(C_1)_{ij} = \begin{cases} p_l^i \underline{T}_l^0 & \text{if } n^i = 1,\ n^j = 2,\ s^i = s^j, \text{ and } l \text{ is the unique value (if any) such that} \\ & \quad p_q^i = p_q^j, \text{ for every } q \neq l \text{ and } p_l^j = p_l^i - 1 \geq 0 \\ 0 & \textit{otherwise} \end{cases}$$

The A_1 matrix is given by:

$$(A_1)_{ij} = \begin{cases} \underline{S}_l^0 \underline{\alpha} & \text{ifn } i = 1,\ n^j = 0,\ s^i = l, s^j = 0, \text{ and } l \text{ is the unique value (if any) such that} \\ & \quad p_q^i = p_q^j, \text{ for every } q \neq k \text{ and } p_k^j = p_k^i + 1 \leq N \\ 0 & \textit{otherwise} \end{cases}$$

The sparsity of all these matrices depends on the exact form of the phase distributions for the processing and access times. In the special case of Erlangian service, the matrix A is very sparse. The matrices still have large dimensions for large N, but now we can efficiently employ the sparsity of the matrices to reduce both the computational and storage requirements.

There are three drawbacks to the recursive approach described to determine the mean waiting time. First, as just mentioned, the matrices are still large for large N. Furthermore, the size of the matrices is still a function of N. Second, one cannot obtain the solution for N processors without investing the work to determine the solution for 1, 2, 3, . . . , and $N-1$ processors. Sometimes this is a convenient built-in advantage. For instance, in this thesis we have continually been interested in the solution for 1, 2, 3, . . . , N processors so a recursive solution based on the solution for $N-1$ processors is not a hindrance. In fact, the recursive solution is very efficient in a case like this since no extra work is performed. Third, as with all recursive computational procedures, small numerical errors propagate very well throughout the chain of calculations.

As a final remark, the recursive method really amounts to solving equations 2.6, 2.7, and 2.8. It just happens that the intermediate results solve the same problem for smaller N.

2.8 Multibus Model with Long Word Accesses

We now extend the model of the isolated Multibus considered so far to include long word accesses, as discussed when the processor model was introduced. Long word accesses are modeled as follows: at the end of the processing time interval, the processor decides with a probability β that its memory access will be a long word access and with a probability $1 - \beta$ that its memory access will be either a word or byte access. The probability β is assumed identical for all processors and independent of the state of all other processors and memory access. A long word access actually requires two successive word accesses on the Multibus. With the Multibus system employed in Concert, there is an interval of 600 to 700 nanoseconds between these two accesses during which the processor releases control of the bus to any pending requests. Because of the round-robin arbitration on the Multibus, all the pending requests are served before the second access of the long word access. Therefore a long word access is essentially two independent accesses: 600 to 700 nsec after the first word access is completed, the request for the second word is generated and joins the end of the queue for Multibus service.

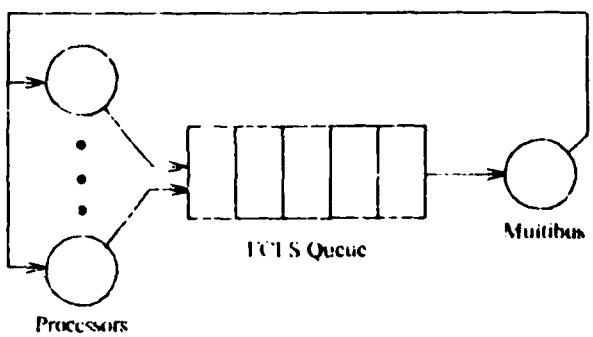


Figure 2.13: Basic Multibus model

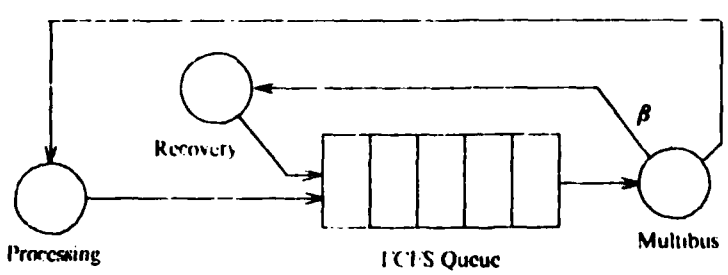


Figure 2.14(a): Extended Multibus model

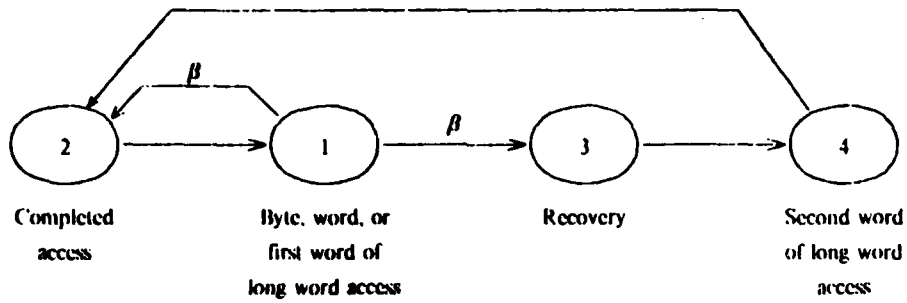


Figure 2.14(b): Class transition diagram of extended Multibus model

The basic Multibus model, depicted in Figure 2.13 above, can be extended to include long word accesses. This extended Multibus model is depicted in Figure 2.14(a). Note that the circle labeled "processing" denotes all the processors which are processing and the circle labeled "recovery" denotes all the processors which are recovering: these circles do not denote individual processors. Figure 2.14(b) shows a class transition diagram of the model. The details of the model are as follows.

Let the request for a byte, word, or the first word of a long word access from any processor i ($1 \leq i \leq N$) be represented by a customer of class 1. Upon completion of this access, the class 1 customer becomes either a class 2 customer with probability $1-\beta$ or a class 3 customer with probability β. Class 2 customers represent fully completed memory accesses - byte, word, and long word (both word accesses) - and class 3 customers represent half completed long word accesses - only the first word access completed. Upon receiving a class 2 customer, processor i begins processing and after a time period t_p, governed by the processing time distribution, processor i generates another request, represented as a class 1 customer. Upon receiving a class 3 customer, processor i waits a recovery time t_r (a random variable given by a recovery time distribution) before generating a class 4 customer, representing the request for the second word of a long word access. Upon completion of this second word access (all word accesses are governed by the same access time distribution), the class 4 customer becomes a class 2 customer and returns to processor i. Exactly N customers are always somewhere in the closed loop of classes 1,2,3, and 4.

Conceptually there is no difference between:

Method 1: the processor deciding when it generates a request that the request corresponds to a long word access, and

Method 2: the server deciding when it completes a word access that the access corresponds to a long word access (and hence requires a second word access). (This method is depicted

in Figure 2.14.)

In method 2 there is no need to distinguish between byte or word accesses and the first word of a long word access. Method 2 therefore requires one less class per processor than method 1.

The processing time random variable, t_p, at each processor is assumed to be identically distributed for all processors and independent of all other random variables. The recovery time random variable, t_r, at each processor is also assumed to be identically distributed for all processors and independent of all other random variables. Finally, the access time random variable, t_a, for each byte or word access is assumed to be identically distributed for all such accesses, irrespective of class, and independent of all other random variables.

2.8.1 Analysis of Model with Long Word Accesses

2.8.1.1 Asymptotic Behaviour

For sufficiently large N the bus will constantly be in use, yielding a bus throughput of $\frac{1}{\bar{t}_a}$ word accesses per unit time. Since each processor cycle (processing time plus word or long word memory access) requires an average of $1+\beta$ word accesses, we obtain the throughput balance equation:

$$\frac{(1+\beta)N}{\bar{t}_{cyc}} = \frac{1}{\bar{t}_a} \tag{2.12}$$

where $\bar{t}_{cyc}$ is the average cycle time given by:

$$\bar{t}_{cyc} = \bar{t}_p + \bar{t}_{w_1} + \bar{t}_a + \beta(\bar{t}_r + \bar{t}_{w_2} + \bar{t}_a), \tag{2.13}$$

$\bar{t}_{w_1}$ is the average waiting time for a byte or word access or the first word access of a long word

and $\bar{t}_{w_2}$ is the average waiting time for the second word access of a long word.

In general $\bar{t}_{w_1} \neq \bar{t}_{w_2}$ since the waiting time of the second word of a long word access is correlated with the waiting time of the first word. For any particular long word access we have

$t_{w_2} = max(0, \sum_{i=1}^{n_{t_{w_1}} + n_{(t_a + t_r)}} t_{a_i} - t_r)$ where

$n_{t_{w_1}}$ is the number of requests joining the queue after a request (for the first word of a long word) during the waiting time t_{w_1} of that request

$n_{(t_a + t_r)}$ is the number of requests joining the queue during the actual access time and

recovery time ($t_a + t_r$) of the request (for the first word)

and t_{a_i} denotes a particular sample of the access time distribution.

$n_{t_{w_1}} + n_{(t_a + t_r)}$ is the total number of requests which arrive after the request for the first word but before the request for the second word of a long word access. The quantity $n_{t_{w_1}}$ is related to t_{w_1} and thus t_{w_1} and t_{w_2} are correlated. In particular, $t_{w_2}=0$ only if all requests that arrived in $t_{w_1} + t_a + t_r$ are completely served in time t_r. Certainly, $t_{w_2}=0$ is in general more difficult to attain the larger t_{w_1} is - i.e. $t_{w_2}=0$ is in general a stricter requirement than $t_{w_1}=0$. Thus we expect t_{w_1} and t_{w_2} to have different probability distributions.

The mean total waiting time (or wasted time) per processor cycle is $\bar{t}_{w_T} = \bar{t}_{w_1} + \beta \bar{t}_{w_2}$. Manipulating the equations 2.12 and 2.13 we have:

$$\bar{t}_{w_T} = (1+\beta)N\bar{t}_a - \bar{t}_p - (1+\beta)\bar{t}_a - \beta\bar{t}_r$$

If we normalize $\bar{t}_{w_T}$ by the mean word access time $\bar{t}_a$, we have

$$\frac{\bar{t}_{w_T}}{\bar{t}_a} = (1+\beta)N - \alpha - (1+\beta) - \beta\gamma \tag{2.14}$$

where $\alpha = \frac{\bar{t}_p}{\bar{t}_a}$, as before, and $\gamma = \frac{\bar{t}_r}{\bar{t}_a}$. Equation 2.14 describes a function of N with an asymptotic slope of $1+\beta$ and a knee at $1 + \frac{\alpha + \beta\gamma}{1+\beta}$. The effect of the long word accesses, through the parameter β, is to increase the asymptotic slope compared with the case with only word accesses. The knee increases with β if $\gamma > \alpha$ and decreases with β if $\gamma < \alpha$.

Normalizing instead by the mean memory access time $\bar{t}_m = \bar{t}_a + \beta(\bar{t}_r + \bar{t}_a)$ yields:

$$\frac{\bar{t}_{w_T}}{\bar{t}_m} = \frac{N}{1 + \frac{\beta\gamma}{1+\beta}} - \frac{\alpha}{(1+\beta)+\beta\gamma} - 1 \tag{2.15}$$

As a function of N, $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ has an asymptotic slope of $\frac{1}{1 + \frac{\beta\gamma}{1+\beta}}$, which is always less than or equal to 1, and a knee again at $1 + \frac{\alpha + \beta\gamma}{1+\beta}$.

2.8.1.2 Deterministic Behaviour

Consider now the case when t_p, t_a, and t_r are deterministic quantities. The maximum memory access time is $2t_a + t_r$. Regarding this as the access time and proceeding as in section 2.2 we obtain $\bar{t}_{w_T}=0$ for $N \leq \left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1$. In the actual Multibus $0<\bar{t}_r<\bar{t}_a$ (see Appendix A). Taking $0<t_r<t_a$ here, we find that queueing must occur (i.e. $\bar{t}_{w_T}>0$) for $N > \left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1$ when $\beta>0$. The reason that $\bar{t}_{w_T}>0$ under these conditions is that no request can be completely served in the recovery time (since $t_r<t_a$), thus in order to maintain $\bar{t}_{w_T}=0$ only one request can be served in the entire $2t_a + t_r$ interval. However, this is impossible for $N > \left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1$, hence some requests must occasionally wait. The case with $\beta = 0$ reduces to that discussed in section 2.2, for which no queueing occurs until $N > \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$.

In the actual Multibus $0<\bar{t}_r \leq \bar{t}_p$ (see Appendix A), thus $0<t_r \leq t_p$. We can view the recovery time t_r as a shortened processing time. Thus the processing time is t_p with probability $1-\beta$ and t_r with probability β (with the restriction that one processing time of t_p follows every processing time of t_r). When $N > \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$ and $\beta = 0$, we know from section 2.2 that the bus is always busy. The following theorem shows that the bus is in fact always busy when $N > \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$ regardless of the value of β.

Theorem 2.5

Consider the Multibus model with long word accesses described in the beginning of section 2.8. If

1) t_p and t_r are deterministic variables such that $0<t_r \leq t_p$,
2) t_a is a random variable with minimum value $t_{a_{min}} \geq t_r$,
3) $N > \left\lfloor \frac{t_p}{t_{a_{min}}} \right\rfloor + 1$, and
4) each of the N processors has completed at least two memory accesses - byte, word, or first or second word access of a long word

then the fraction of time that the bus is busy, denoted by ρ, is 1.

Proof:

Suppose to the contrary that $\rho<1$. Then there must be at least one memory request such that the bus is idle immediately prior to that request. Choose one such memory request. Denote the time at which that request occurs by τ and the processor from which it originated by k. There are two cases to consider.

Case 1: At time τ processor k just completed a processing time interval (of duration t_p).

Immediately prior to time $\tau - t_p$, each of the $N-1$ processors other than processor k either must have a memory request pending (and waiting) or must be in the midst of a processing or a recovery period (since all processors have completed at least two memory accesses). Since $t_r \leq t_p$, all of these processors (if any) in the midst of a processing or recovery period must generate at least one memory request before time τ. Therefore there must be at least $N-1$ memory requests pending or generated in the interval $(\tau - t_p, \tau]$. In order that the bus be idle immediately prior to time τ, all of these memory requests must be completely served before time τ. Since there are at least $N-1$ of these memory requests, we must at least have $(N-1)t_{a_{\min}} < t_p$. Or, since N is an integer, we must have $N \leq \left\lfloor \frac{t_p}{t_{a_{\min}}} \right\rfloor + 1$.

Case 2: At time τ processor k just completed a recovery time interval (of duration t_r)

Since the bus is idle immediately prior to time τ and $t_r \leq t_{a_{\min}}$, there can be no memory requests pending or generated in the interval $[\tau - t_r, \tau)$. Furthermore, no memory requests can be pending or generated in the interval $(\tau - t_r - t_{a_{\min}}, \tau)$, otherwise the bus would not be idle immediately prior to time τ. In order that there be no memory requests in the interval $(\tau - t_r - t_{a_{\min}}, \tau)$, all the other $N-1$ processors must be processing during the interval $(\tau - t_r - t_{a_{\min}}, \tau)$. Thus each of these $N-1$ processors must begin processing in the interval $[\tau - t_p, \tau - t_r - t_{a_{\min}}]$, implying that at least $N-2$ memory accesses occur in the interval $[\tau - t_p, \tau - t_r - t_{a_{\min}}]$. Therefore at least $N-1$ memory accesses occur in the interval $[\tau - t_p, \tau)$, i.e. $(N-1)t_{a_{\min}} < t_p$. Or since N is an integer, $N \leq \left\lfloor \frac{t_p}{t_{a_{\min}}} \right\rfloor + 1$.

From Case 1 and 2 we conclude that $N \leq \left\lfloor \frac{t_p}{t_{a_{\min}}} \right\rfloor + 1$ is a necessary condition in order that $\rho<1$. Since by hypothesis $N > \left\lfloor \frac{t_p}{t_{a_{\min}}} \right\rfloor + 1$, we must have $\rho \geq 1$.

Our throughput balance equation (equation 2.12) can be written for general ρ as follows:

$$\frac{(1+\beta)N}{\bar{t}_{cyc}} = \frac{\rho}{\bar{t}_a}.$$

We conclude from this that $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ equals its asymptotic value for $N \geq \left| \frac{t_p}{t_a} \right| + 1$ since $\rho = 1$ for N in this range.

Figure 2.15 illustrates representative cases of $\bar{t}_{w_T}/\bar{t}_m$ vs. N in the deterministic case.

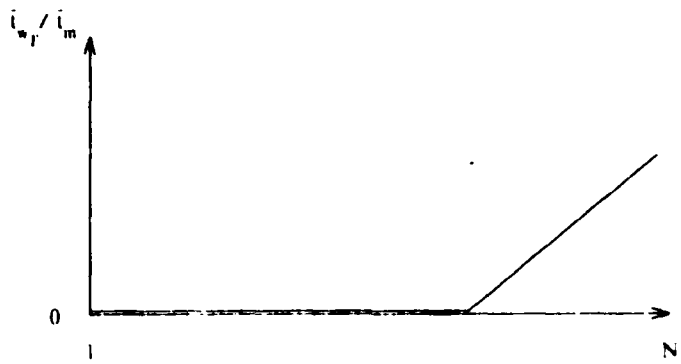


Figure 2.15(a): $\beta = 0$

Knee: $\alpha + 1$ Asymptotic slope: 1

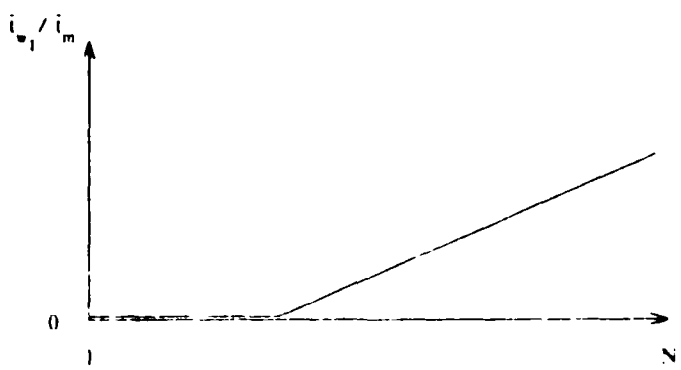


Figure 2.15(b): $\beta = 1$

Knee: $\frac{\alpha}{(2+\gamma)} + 1$ Asymptotic slope: $\frac{1}{1+\frac{\gamma}{2}}$

Figures 2.15(a) and 2.15(b) depict $\frac{t_{w_l}}{t_m}$ for $\beta = 0$ and $\beta = 1$ respectively. (N is treated as a continuous parameter in Figure 2.15, thus the floor functions are neglected. Note also that $\alpha = \frac{t_p}{t_a}$ and $\gamma = \frac{t_r}{t_a}$.) For $\beta > 0$ we have three cases:

1. for $N \leq \left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1$, $t_{w_l} = 0$,
2. for $\left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1 < N < \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$, $\frac{t_{w_l}}{t_m}$ is strictly positive and greater than or equal to its asymptotic value (assuming $t_a > 0$), and
3. for $N \geq \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$, $\frac{t_{w_l}}{t_m}$ equals its asymptotic value.

These three cases are illustrated in Figure 2.15(c). $N_l^* \equiv \left\lfloor \frac{t_p}{2t_a + t_r} \right\rfloor + 1$ and $N_u^* \equiv \left\lfloor \frac{t_p}{t_a} \right\rfloor + 1$.

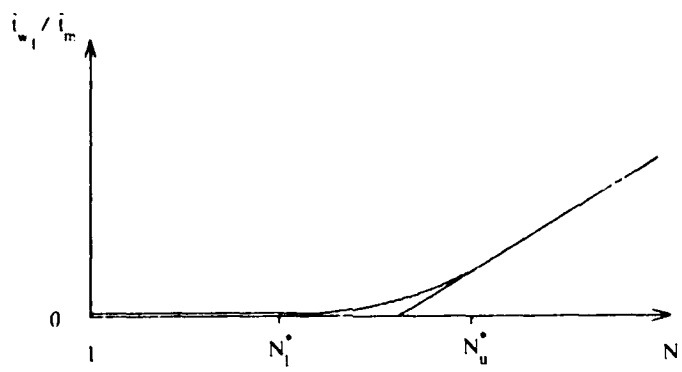


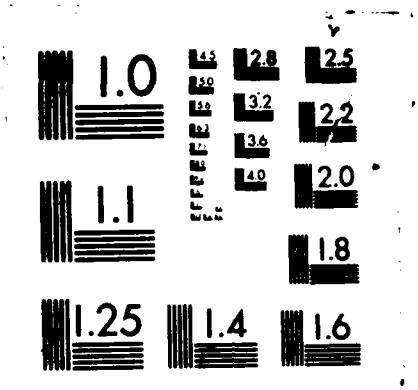
Figure 2.15(c): $0 < \beta < 1$

Knee: $\frac{\alpha + \beta\gamma}{1 + \beta} + 1$ Asymptotic slope: $\frac{1}{1 + \frac{\beta\gamma}{1 + \beta}}$

The curves in Figure 2.15(c) are rounded in the knee area due to the randomness introduced by the probabilistic choice of word vs. long word access. Because of this rounding, the knee cannot always be interpreted as the maximum value of N for which $t_{w_l} = 0$ can be maintained.

Finally, note that deterministic t_p, t_r, and t_a yield a lower bound on t_{w_l} over all possible sta-

AD-A183 619 MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR 2/4
(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
COMPUTER SCIENCE R B OSBORNE MAY 87 MIT/LCS/TR-375
UNCLASSIFIED N00014-83-K-0125 F/G 12/7 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

tionary processing, recovery, and access time probability distributions for all N if $\beta = 0$ or 1 and at least for $N \leq \frac{\alpha}{2+\gamma} + 1$ and $N \geq \alpha + 1$ if $0 < \beta < 1$.

2.8.1.3 Product Form Solution

The Multibus model with long word accesses that was presented earlier has a product form solution if the access time is exponentially distributed. The processing and recovery time distributions may be completely arbitrary.

Let the global state be $\underline{X} = (\underline{x}_P, \underline{y})$ where $\underline{x}_P$ represents the state of the processors (where class 1 customers originate) and $\underline{y}$ represents the state of the FCFS queue for Multibus service. The processors can be considered as comprising an infinite server since there is always a free processor available for an arriving customer. Therefore the processors form a quasi-reversible queue (with respect to a Markovian state description). The exponentially distributed access time, independent of class, renders the FCFS queue quasi-reversible (again with respect to a Markovian state description). The quasi-reversibility of all the queues in isolation yields the product form:

$$\pi_X = \pi_{x_P} \cdot \pi_y$$

Let $\underline{x}_P = (n_P, n_R)$ where n_P is the number of customers in class 2 (i.e. processing) and n_R is the number of customers in class 3 (i.e. recovering).

Let $\underline{y} = (n_{A_1}, n_{A_2})$ where n_{A_1} is the number of customers in class 1 (i.e. byte or word or first access of long word) and n_{A_2} is the number of customers in class 4 (i.e. second access of long word).

Let λ_j^{eff} represent the effective arrival rate of class j customers; $j = 1,\ldots,4$. Then from the results in section 2.6.1 we have:

$$\pi_{x_P} = \frac{(\lambda_2^{eff} \bar{t}_p)^{n_P}}{n_P!} \cdot \frac{(\lambda_3^{eff} \bar{t}_r)^{n_R}}{n_R!}$$

$$\pi_y = \frac{(n_{A_1} + n_{A_2})!}{n_{A_1}! n_{A_2}!} \bar{t}_a^{(n_{A_1} + n_{A_2})} \left[\lambda_1^{eff}\right]^{n_{A_1}} \left[\lambda_4^{eff}\right]^{n_{A_2}}$$

Now $\lambda_4^{eff} = \lambda_1^{eff} - \beta\lambda_1^{eff}$ and $\lambda_2^{eff} = (1-\beta)\lambda_1^{eff} + \lambda_4^{eff} = \lambda_1^{eff}$. Thus the steady state probability of the global state $\underline{X} = (n_P, n_R, n_{A_1}, n_{A_2})$ is

$$\pi_X = \frac{\left[\lambda_1^{eff} \bar{t}_a\right]^N \alpha^{n_P} (\beta\gamma)^{n_R} (n_{A_1} + n_{A_2})! \beta^{n_{A_2}}}{n_P! n_R! n_{A_1}! n_{A_2}!} \tag{2.16}$$

Since $n_P + n_R + n_{A_1} + n_{A_2} = N$, we can rewrite this as:

$$\pi_{\underline{X}} = C\left|\begin{matrix} N \\ n_P\ n_R\ n_{A_1}\ n_{A_2}\end{matrix}\right| \alpha^{n_P}(\beta\gamma)^{n_R}(n_{A_1}+n_{A_2})!\beta^{n_{A_2}},$$

for some normalizing constant C ($C = \dfrac{\left[\lambda_1^{eff}\bar{t}_u\right]^N}{N!}$).

The mean number of requests for a byte, word, or the first word of a long word access in the FCFS queue is

$$\overline{n_{A_1}} = \sum_{n_{A_1}=0}^{N} n_{A_1} \sum_{n_P=0}^{N} \sum_{n_R=0}^{N} \sum_{n_{A_2}=0}^{N} \pi_{\underline{X}} \qquad n_P + n_R + n_{A_2} = N - n_{A_1}$$

Similarly, the mean number of requests for the second word of a long word in the FCFS queue is:

$$\overline{n_{A_2}} = \sum_{n_{A_2}=0}^{N} n_{A_2} \sum_{n_P=0}^{N} \sum_{n_R=0}^{N} \sum_{n_{A_1}=0}^{N} \pi_{\underline{X}} \qquad n_P + n_R + n_{A_1} = N - n_{A_2}$$

Clearly $\overline{n_{A_1}} = \overline{n_{A_2}}$ when $\beta = 1$ and $\overline{n_{A_2}} = 0$ when $\beta = 0$.

If we let the global state be $\underline{X}' = (n_{A_1}, n_{A_2})$ then

$$\pi_{\underline{X}'} = C'\left[\sum_{n_P=0}^{N} \sum_{n_R=0}^{N} \frac{(N-n_{A_1}-n_{A_2})!}{n_P!n_R!}\alpha^{n_P}(\beta\gamma)^{n_R}\right]\frac{(n_{A_1}+n_{A_2})!}{n_{A_1}!n_{A_2}!}\frac{\beta^{n_{A_2}}N!}{(N-n_{A_1}-n_{A_2})!}$$

$$= C'(\alpha+\beta\gamma)^{N-n_{A_1}-n_{A_2}}\frac{N!}{(N-n_{A_1}-n_{A_2})!}\frac{(n_{A_1}+n_{A_2})!}{n_{A_1}!n_{A_2}!}\beta^{n_{A_2}}$$

$$= C''\left|\begin{matrix} N \\ n_{A_1}\ n_{A_2}\ N-n_{A_1}-n_{A_2}\end{matrix}\right|\left[\frac{1}{\alpha+\beta\gamma}\right]^{n_{A_1}}\left[\frac{\beta}{\alpha+\beta\gamma}\right]^{n_{A_2}}(n_{A_1}+n_{A_2})!$$

Thus

$$\overline{n_{A_1}} = C''N!\sum_{n_{A_1}=0}^{N}\frac{n_{A_1}}{n_{A_1}!}\left[\frac{1}{\alpha+\beta\gamma}\right]^{n_{A_1}}\sum_{n_{A_2}=0}^{N-n_{A_1}}\frac{(n_{A_1}+n_{A_2})!}{n_{A_2}!(N-n_{A_1}-n_{A_2})!}\left[\frac{\beta}{\alpha+\beta\gamma}\right]^{n_{A_2}}$$

$$\overline{n_{A_2}} = C''N!\sum_{n_{A_2}=0}^{N}\frac{n_{A_2}}{n_{A_2}!}\left[\frac{\beta}{\alpha+\beta\gamma}\right]^{n_{A_2}}\sum_{n_{A_1}=0}^{N-n_{A_2}}\frac{(n_{A_1}+n_{A_2})!}{n_{A_1}!(N-n_{A_1}-n_{A_2})!}\left[\frac{1}{\alpha+\beta\gamma}\right]^{n_{A_2}}$$

Interchanging the order of summation for $\overline{n_{A_1}}$ we have

$$\overline{n_{A_1}} = C''N!\sum_{n_{A_2}=0}^{N}\left[\frac{\beta}{\alpha+\beta\gamma}\right]^{n_{A_2}}\sum_{n_{A_1}=0}^{N-n_{A_2}}\frac{n_{A_1}(n_{A_1}+n_{A_2})!}{n_{A_1}!n_{A_2}!(N-n_{A_1}-n_{A_2})!}\left[\frac{1}{\alpha+\beta\gamma}\right]^{n_{A_1}}$$

$$= C'' N! \sum_{n_{A_2}=0}^{N-1} \left[\frac{\beta}{\alpha+\beta\gamma} \right]^{n_{A_2}} \sum_{n_{A_1}=1}^{N-n_{A_2}} \frac{(n_{A_1}+n_{A_2})!}{(n_{A_1}-1)!\,n_{A_2}!\,(N-n_{A_1}-n_{A_2})!} \left[\frac{1}{\alpha+\beta\gamma} \right]^{n_{A_1}}$$

and

$$\overline{n_{A_2}} = C'' N! \sum_{n_{A_2}=1}^{N} \left[\frac{\beta}{\alpha+\beta\gamma} \right]^{n_{A_2}} \sum_{n_{A_1}=0}^{N-n_{A_2}} \frac{(n_{A_1}+n_{A_2})!}{n_{A_1}!\,(n_{A_2}-1)!\,(N-n_{A_1}-n_{A_2})!} \left[\frac{1}{\alpha+\beta\gamma} \right]^{n_{A_1}}$$

$$= C'' N! \beta \sum_{n_{A_2}=1}^{N} \left[\frac{\beta}{\alpha+\beta\gamma} \right]^{n_{A_2}-1} \sum_{n_{A_1}=0}^{N-n_{A_2}} \frac{((n_{A_1}+1)+(n_{A_2}-1))!}{n_{A_1}!\,(n_{A_2}-1)!\,(N-(n_{A_1}+1)-(n_{A_2}-1))!} \left[\frac{1}{\alpha+\beta\gamma} \right]^{n_{A_1}+1}$$

By renaming n_{A_1} and n_{A_2} in the above expression for $\overline{n_{A_2}}$ we see that $\overline{n_{A_1}} = \beta \overline{n_{A_2}}$, as one might have expected (naively) from the outset.

If we let $\underline{X}''' = (n_P, n_R, n_S)$ where n_S is the total number of requests in the FCFS queue, then:

$$\pi_{X'''} = C''' \frac{\alpha^{n_P}}{n_P!} \frac{(\beta\gamma)^{n_R}}{n_R!} (1+\beta)^{n_S} = C^{IV} (1+\beta)^{n_S} \frac{(N-n_S)!}{n_P!\,n_R!} \alpha^{n_P} (\beta\gamma)^{n_R} \frac{N!}{(N-n_S)!}$$

Finally, if we let $\underline{X}^{V} = (n_S)$, then

$$\pi_{X^{V}} = C^{V} (\alpha+\beta\gamma)^{N} \frac{N!}{(N-n_S)!} \left[\frac{1+\beta}{\alpha+\beta\gamma} \right]^{n_S}$$

$$= C^{VI} \frac{N!}{(N-n_S)!} \left[\frac{1+\beta}{\alpha+\beta\gamma} \right]^{n_S}$$

where

$$C^{VI} = \left[\sum_{n_S=0}^{N} \frac{N!}{(N-n_S)!} \left[\frac{1+\beta}{\alpha+\beta\gamma} \right]^{n_S} \right]^{-1}$$

Note that this is exactly the same result we obtain in section 2.4 for the M/M/1//N model if we replace $\frac{\lambda}{\mu}$ by $\frac{1+\beta}{\alpha+\beta\gamma}$ = ratio of mean service requirement per cycle to mean processor time (processing plus recovery) per cycle.

The average number of requests in the queue is

$$\overline{n_S} = C^{VI} \sum_{n_S=0}^{N} n_S \left[\frac{1+\beta}{\alpha+\beta\gamma} \right]^{n_S} \frac{N!}{(N-n_S)!}$$

and $\overline{n_S} = \overline{n_{A_1}} + \overline{n_{A_2}} = (1+\beta)\overline{n_{A_1}}$.

By Little's Law: $\bar{t}_{w_1} = \frac{\overline{n_{A_1}}}{\lambda_1^{eff}} - \bar{t}_a$, $\bar{t}_{w_2} = \frac{\overline{n_{A_2}}}{\lambda_4^{eff}} - \bar{t}_a$, and the mean waiting time for any access is $\bar{t}_w = \frac{\overline{n_s}}{\lambda_1^{eff} + \lambda_4^{eff}} - \bar{t}_a$. Since $\lambda_4^{eff} = \beta\lambda_1^{eff}$ and $\overline{n_{A_2}} = \beta\overline{n_{A_1}}$, we have $\bar{t}_{w_1} = \bar{t}_{w_2} = \bar{t}_w = \frac{\overline{n_s}}{\lambda_f^{eff}} - \bar{t}_a$ where $\lambda_f^{eff} = \lambda_1^{eff} + \lambda_4^{eff}$. (In general $\bar{t}_w = \frac{\bar{t}_{w_1}}{(1+\beta)} + \frac{\beta\bar{t}_{w_2}}{(1+\beta)}$.)

It is possible to arrive at $\bar{t}_{w_1} = \bar{t}_{w_2}$ (and hence $\overline{n_{A_1}} = \beta\overline{n_{A_2}}$) via a simpler route. A closed network of quasi-reversible queues has the property that at the instant a customer arrives at a queue the probability distribution of all other customers is the same as the equilibrium distribution obtained if they were the only customers in the network (Kelly [K1]). An arriving customer essentially "sees" the network as it would behave in equilibrium without itself. Therefore class 1 and class 4 customers arriving at the FCFS queue each "see" the queue as it would behave in equilibrium with $N-1$ customers - each see the same distribution of customers. Both classes of customers thus have the same waiting time distribution.

Denoting the probability that the FCFS queue server (i.e. the Multibus) is busy by ρ, we have, again by Little's Law, $\rho = \lambda_f^{eff}\,\bar{t}_a$. The bus utilization ρ is given by $\rho = 1 - C^{VI}$. Therefore

$$\frac{\bar{t}_{w_1} = \bar{t}_{w_2} = \bar{t}_w}{\bar{t}_a} = \frac{n_s}{\rho} - 1 = \frac{C^{VI}}{(1-C^{VI})}\left[\sum_{n_s=1}^{N} n_s \frac{N!}{(N-n_s)!}\left[\frac{1+\beta}{\alpha+\beta\gamma}\right]^{n_s}\right] - 1$$

This is the same result as obtained with the M/M/1//N model when, as just noted above, we replace $\alpha = \frac{\mu}{\lambda}$ in the M/M/1//N model by $\frac{\alpha+\beta\gamma}{1+\beta}$. Therefore $\frac{\bar{t}_w}{\bar{t}_a}$ is asymptotic to $N - \frac{\alpha+\beta\gamma}{1+\beta} - 1$ for large N (at least in the case when all processors are identical and all the queues are quasi-reversible in isolation). Since in this case we know that $\bar{t}_w = \bar{t}_{w_1} = \bar{t}_{w_2}$, it is easy to confirm this asymptotic behaviour. Equating throughputs for large N we have:

$$\frac{(1+\beta)N}{\bar{t}_p + \bar{t}_{w_1} + \bar{t}_a + \beta(\bar{t}_r + \bar{t}_{w_2} + \bar{t}_a)} = \frac{1}{\bar{t}_a}$$

and thus $\frac{\bar{t}_w}{\bar{t}_a} = N - \frac{\alpha+\beta\gamma}{1+\beta} - 1$, for large N as deduced by comparison with the result for the M/M/1//N model.

Since $\bar{t}_{w_1} = \bar{t}_{w_2} = \bar{t}_w$ the mean total waiting time per processor cycle is simply $\bar{t}_{w_f} = (1+\beta)\bar{t}_w$. Although $\bar{t}_{w_f}$ is more meaningful than $\bar{t}_w$ as an indication of throughput degradation, we choose

to give the results in terms of $\bar{t}_w$ for three reasons. First, as just mentioned the two are trivially related by a multiplicative constant. Second, $\bar{t}_w$ or more specifically, $\frac{\bar{t}_w}{\bar{t}_a}$ unifies the results of the current model with the results of the earlier models and facilitates direct comparisons. Third, the asymptotic slope of $\bar{t}_w$ is independent of all parameters except N, unlike the case with $\bar{t}_{w_r}$. Thus graphical results for $\bar{t}_w$ can be presented without the possible clutter created by asymptotes intersecting.

Actual measurements (see Appendix A) indicate that $\bar{t}_a = 1.04$ μsec for reads and 1.06 μsec for writes and that $\bar{t}_r = .65$ μsec. Taking $\bar{t}_a = 1.05$ μsec and $\bar{t}_r = .65$ μsec yields $\gamma = .62$. The minimum possible value for t_p is .60 or .70 μsec with almost equal probability; thus $\alpha \geq .62$. Figure 2.16 shows $\bar{t}_w/\bar{t}_a$ vs. N for various combinations of $\alpha \geq .62$ and $0 \leq \beta \leq 1$ with $\gamma = .62$. Note that with $\beta = 0$ the model reduces to the G/M/1//N model.

The mean waiting time per request is very sensitive to the value of β. Indeed, since $\frac{\partial}{\partial \beta}\left[\frac{\alpha + \beta\gamma}{1+\beta}\right] = \frac{\gamma - \alpha}{(1+\beta)^2} < 0$ (since $\alpha \geq \gamma$), the knee of $\frac{\bar{t}_w}{\bar{t}_a}$ varies from $\frac{\alpha + .61}{2} + 1$ to $\alpha + 1$, which represents close to a 100% change in $\bar{t}_w$ (with respect to $\bar{t}_w$ for $\beta = 1$) for large α.

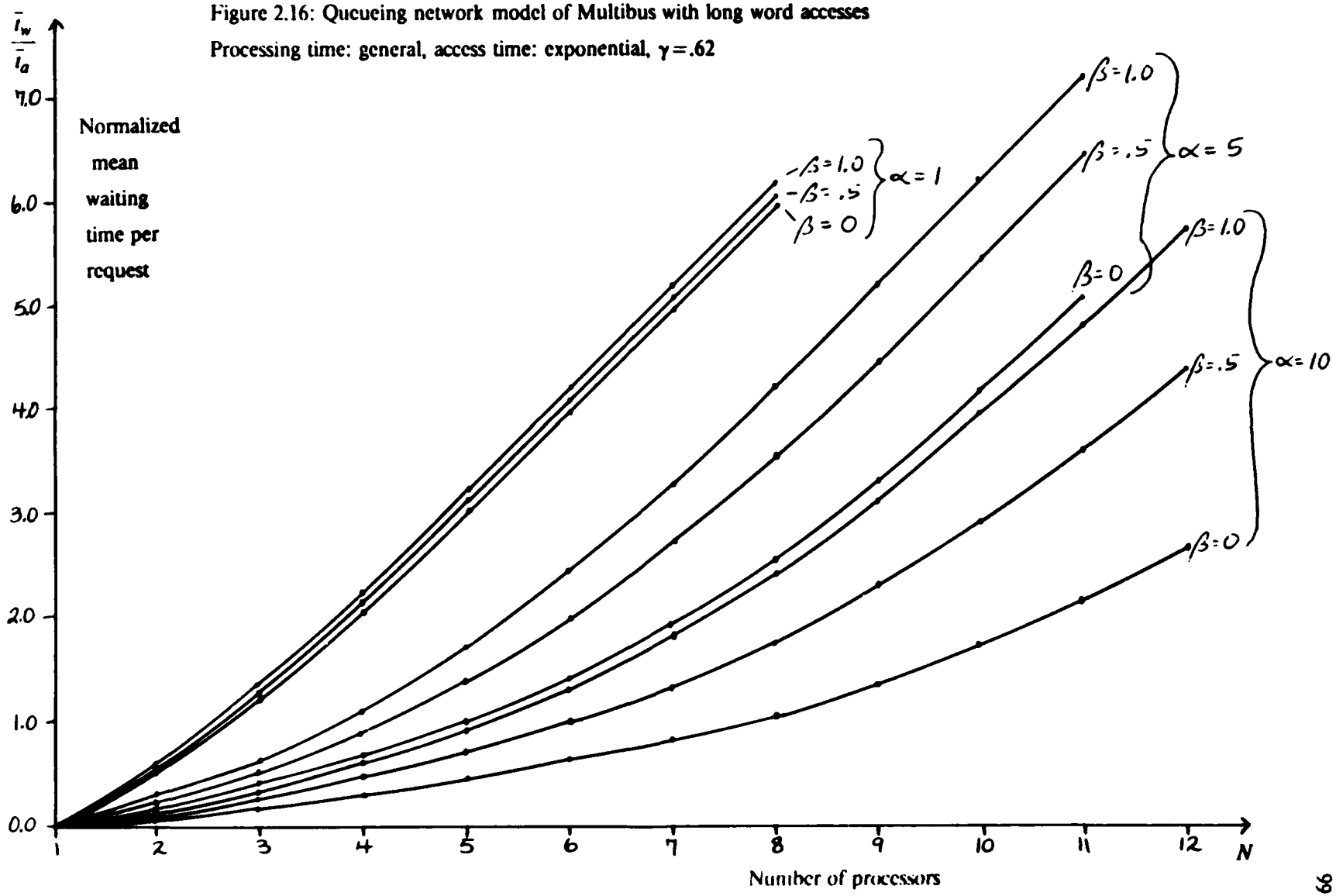
2.8.1.4 Simulations

In this section we explore the case when the access time is not exponentially distributed and thus the solution does not (in general) have the convenient product form as in the previous section. As demonstrated in section 2.7.2, exact results could be obtained by the method of stages. However, this method requires substantial work and does not yield great insight. Approximate results could be obtained by a diffusion model as in Halachmi and Franta [H1] or by the methods discussed and referenced by Whitt [W2]. While such approximate results can yield a great deal of insight, they are more difficult to obtain in this case - due to the complexities added by long word accesses - than in section 2.7 and they are, of course, just approximate.

In order to obtain a qualitative understanding of the effect of different processing time distributions on the mean waiting time per request, we simulated the system with different α and β parameters for different processing time distributions. The access time distribution was kept deterministic throughout to approximate the actual Multibus access time distribution. The error in this approximation is presumably quite small since the variance of the actual access time is small (see section 3.3 in Appendix A). The results from all the previous models lead us to conjecture that the mean waiting time for a given processing time distribution and a given mean access time is minimized by a deterministic access time. Thus the mean waiting time with the actual access time distribution will likely only be greater. The recovery time distribution was also kept deterministic throughout.

Figure 2.16: Queueing network model of Multibus with long word accesses

Processing time: general, access time: exponential, $\gamma = .62$



Three different processing time distributions were considered: third order Erlangian (i.e. E_3), exponential, and third order hyperexponential (with parameters $\alpha_1 = .6$ $\alpha_2 = .3$ $\alpha_3 = .1$ and $\lambda_1 = \lambda$, $\lambda_2 = .1\lambda$, $\lambda_3 = .001\lambda$ where $\bar{t}_p = \frac{\alpha_1}{\lambda_1} + \frac{\alpha_2}{\lambda_2} + \frac{\alpha_3}{\lambda_3}$). The chief difference between these distributions is in their coefficient of variation defined as $C_{t_p} = \frac{\sigma_{t_p}}{\bar{t}_p}$ The coefficient of variation, C_{t_p}, is a measurement of the amount of variation or randomness about the mean normalized by the mean. The following table gives C_{t_p} for the three distributions considered.

Processing time distribution	Coefficient of variation C_{t_p}
Erlangian (E_3)	$\frac{1}{\sqrt{3}} \approx .5773$
Exponential	1
Hyperexponential (H_3, parameters as above)	$\sqrt{10.1} \approx 3.178$

The simulation results for $\alpha = 1.0, 5.0, 10.0$ and $\beta = 0, .5, 1.0$ are presented in Figures 2.17, 2.18 and 2.19. Note that the vertical axis is the mean waiting time per access for *any* access - i.e. the first or second word of a long word - denoted by $\bar{t}_w$. We found in general that $\bar{t}_{w_1} \leq \bar{t}_w \leq \bar{t}_{w_2}$ where $\bar{t}_{w_1}$ and $\bar{t}_{w_2}$ are the mean waiting times for the first and second word respectively. The difference $\bar{t}_{w_2} - \bar{t}_{w_1}$ increased with N and approached a constant as the mean waiting time approached its asymptotic value (interestingly, $\bar{t}_w \approx \frac{\bar{t}_{w_1} + \bar{t}_{w_2}}{2}$). These findings are consistent with the discussion in section 2.8.1.1: the waiting time for the second word of a long word access is correlated with the waiting time of the first word of the same long word access.

Figure 2.17: Queueing network model of Multibus with long word accesses
Processing time: Erlangian (E_3), access time: deterministic, $\gamma = .62$

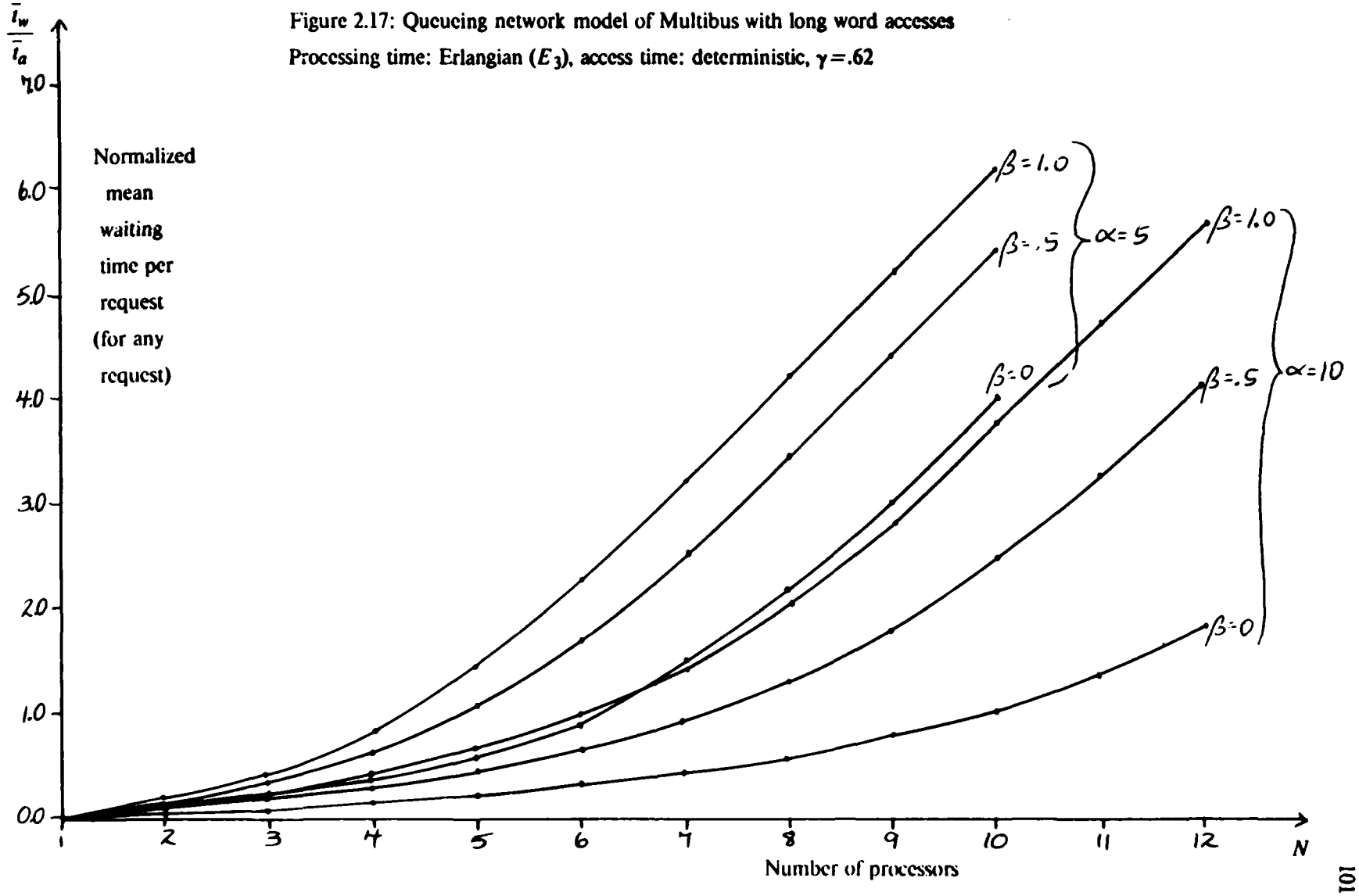
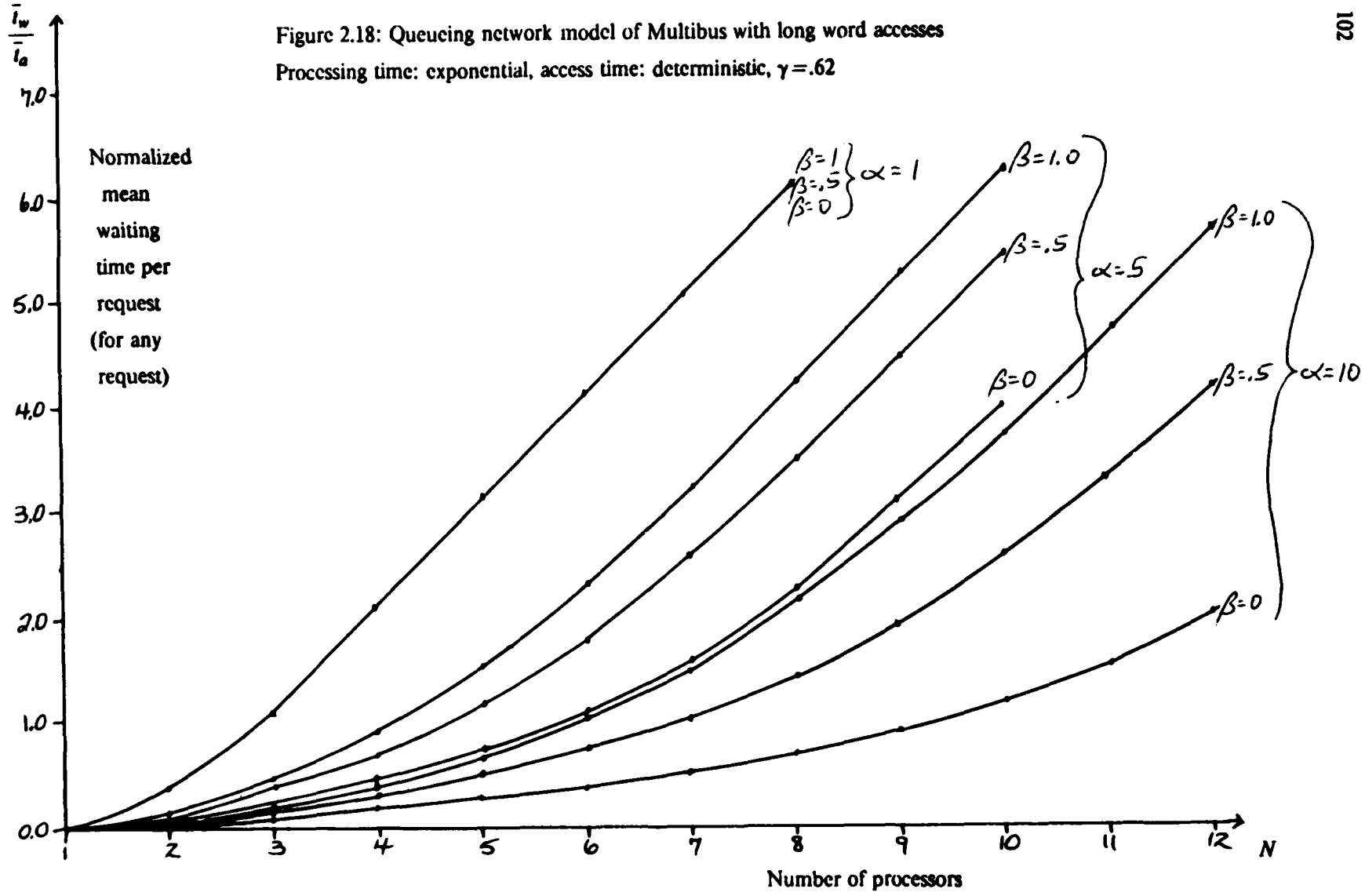


Figure 2.18: Queueing network model of Multibus with long word accesses

Processing time: exponential, access time: deterministic, $\gamma = .62$



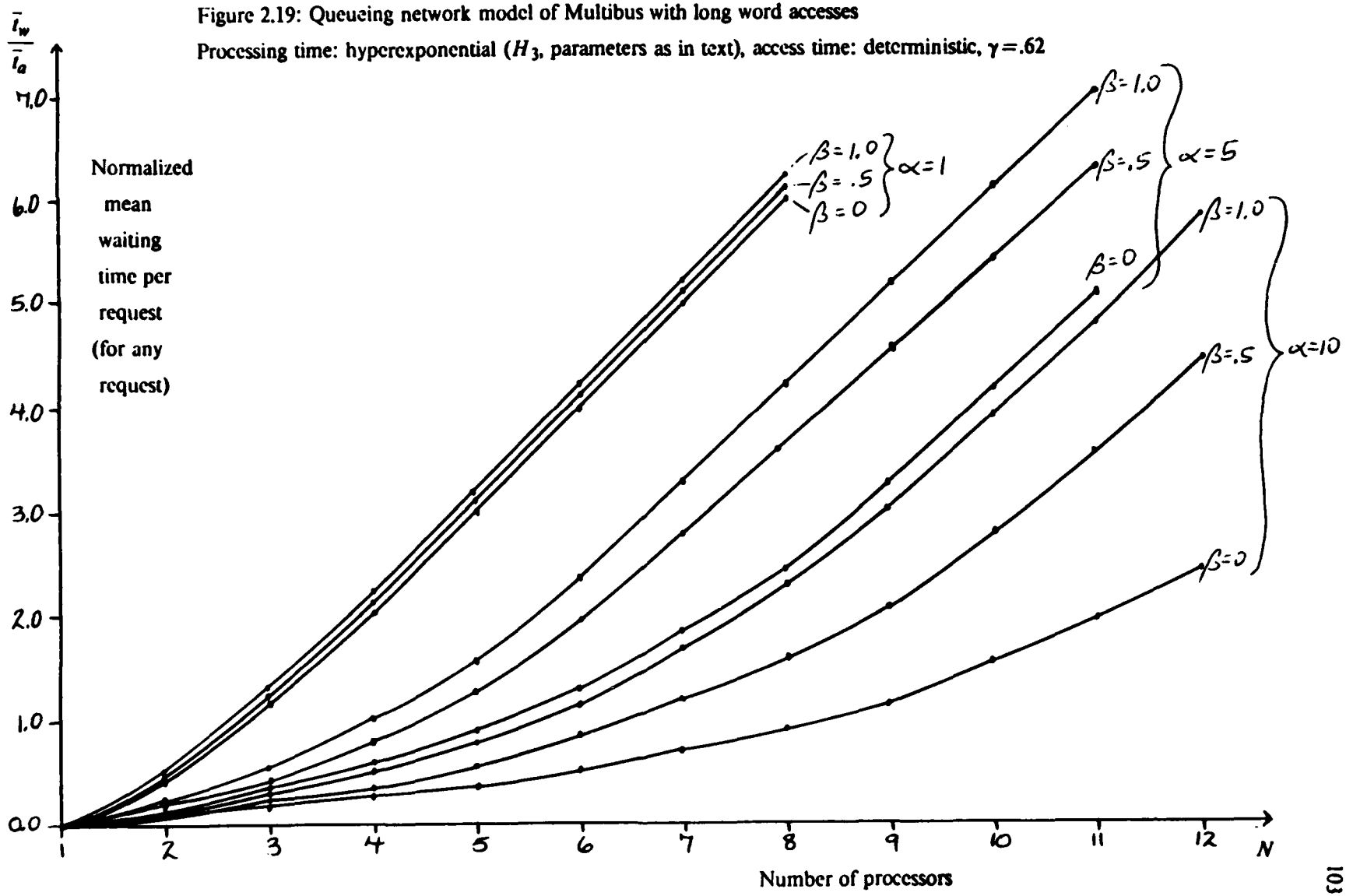


Figure 2.19: Queueing network model of Multibus with long word accesses

Processing time: hyperexponential (H_3, parameters as in text), access time: deterministic, $\gamma = .62$

A careful examination of Figures 2.17, 2.18, and 2.19 reveals that for any given α and β the curves differ only in the knee area. In each case, the mean waiting time in the knee area is least for the Erlangian distribution and greatest for the hyperexponential distribution. This finding is consistent with our findings with the previous models: the mean waiting time in the knee area generally increases as the "randomness" of the (processing and access) distributions increases. In each case however, the change in the mean waiting time due to the different processing time distributions is much less than the change due to different values of the parameter β. For example, for $\alpha = 10.0$, the Erlangian curve is at most about .2 below the same curve for the exponential, and the hyperexponential curve is at most about .5 above the same curve for the exponential.

The difference in mean waiting times effected by exponential versus deterministic distributions for the access time can be ascertained by comparing Figures 2.5 and 2.18. The difference in mean waiting times is greatest in the knee area of the curves and increases with N, as observed with the earlier models. For $\alpha = 10.0$, the difference is at most about .70. Changing β from .5 to 1.0 results in a change of at most about 1.5 in the mean waiting time. Therefore, for the distributions considered, the mean waiting time is more sensitive to the value of β than the form of the distribution. Indeed, the value of β determines the asymptotic value of the mean waiting time and the location of the knee in the mean waiting time curve. The processing and access time distributions just determine the "sharpness" of the knee.

The above discussion suggests that it is best to study the factors influencing the parameter β, while perhaps assuming analytically tractable exponential distributions for the processing and access times, before studying in detail the effect of different distributions.

2.9 Multibus Model with Long Word and Ringbus Accesses

In this section the Multibus model discussed so far is interfaced with the Ringbus. As described in section 1.2.5 we have decomposed the overall Concert model into two models - the Multibus and the Ringbus - to make analysis tractable. When analyzing one model, the operation of the other is replaced by an equivalent lumped model. In this section we replace the Ringbus by its equivalent access time distribution. In the sequel we will be interested in approximating the Ringbus access time distribution by one with a small number of parameters (in particular a single parameter) so that we can easily solve for the interaction between the Multibus and Ringbus models. For now we consider the Ringbus access time distribution to be general and unspecified.

We can extend the Multibus model with long word accesses that was developed in section 2.8 to include Ringbus accesses. We regard a Ringbus access as occurring with probability ψ and a Multibus access as occurring with probability $1 - \psi$; otherwise the model remains as in section 2.8. Actually, any Ringbus access begins as a Multibus access. The Ringbus interface board (RIB) determines which Multibus accesses are permitted to use the Ringbus based on the address at which the read and/or write is to be performed. Recall from section 1.3 that we term a memory operation - read and/or write - that occurs in the Ringbus address space (i.e. requires the Ringbus) a Ringbus access. Similarly, we call a memory operation that occurs in the Multibus address space (i.e. does not require any portion of the Ringbus) a Multibus access. Thus a Ringbus access requires mastership of the Multibus, but the actual access occurs in the Ringbus address space.

The new model can be described more precisely by introducing classes of customers as in section 2.8. We now require a total of seven classes; the classes 1 through 4 are the same as in section 2.8.

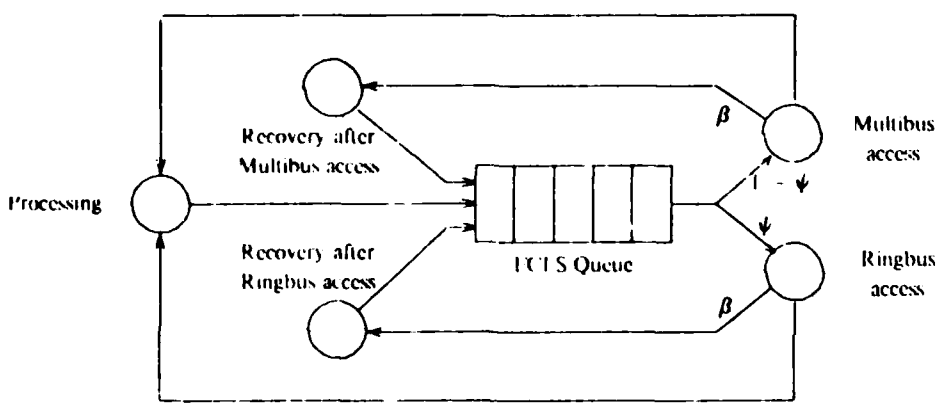


Figure 2.20(a): Multibus model with Ringbus accesses

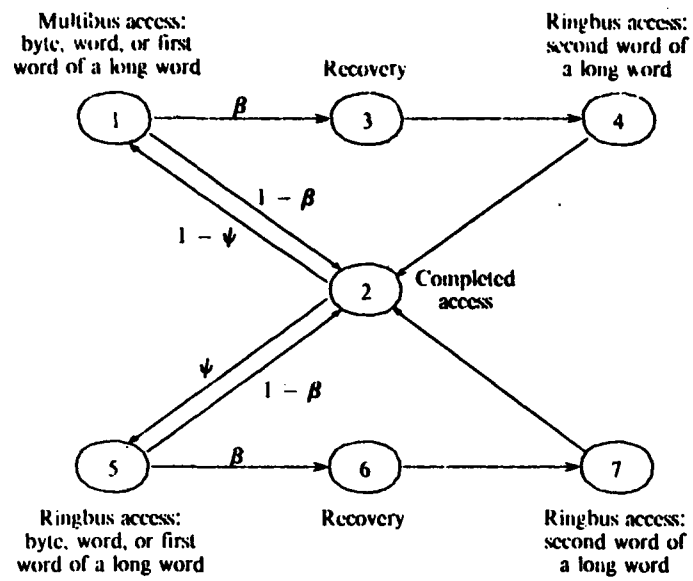


Figure 2.20(b): Class transition diagram

Figure 2.20(a) depicts the new model and Figure 2.20(b) shows a class transition diagram. As in Figure 2.14 in section 2.8, the circle in Figure 2.20(a) labeled "processing" denotes all processors which are processing and the circles labeled "recovery" denote the processors which are recovering. The details of the model are as follows:

Let the request for a byte, word, or the first word of a long word access from any processor be represented by a customer of class 1 for a Multibus access or by a customer of class 5 for a Ringbus access. After a class 1 customer completes its access, it becomes either a class 2 customer with probability $1-\beta$ or a class 3 customer with probability β, and returns to any free processor (all processors are considered identical). Class 2 customers represent fully completed memory accesses - byte, word, and long word (both accesses) - and class 3 customers represent half completed long word Multibus accesses - only the first word completed. Upon receiving a class 3 customer, a processor waits a time t_r given by the recovery time distribution before generating a class 4 customer, representing the request for the second word of a long word Multibus access. Upon completion of this second word access, the class 4 customer becomes a class 2 customer and returns to any free processor.

With probability $1-\psi$ this request is for a Multibus access and is represented by a customer of class 1; with probability ψ this request is for Ringbus access and is represented by a customer of class 5.

After a class 5 customer completes its access, it becomes either a class 2 customer with probability $1-\beta$ or a class 6 customer with probability β, and returns to any free processor. Class 6 customers represent half completed long word Ringbus access - only the first word completed. Upon receiving a class 6 customer, a processor waits a time t_r given by the same recovery time distribution as before and then generates a class 7 customer, representing the request for the second word of a long word Ringbus access. Finally, upon completion of this second word access, the class 7 customer becomes a class 2 customer and returns to any free processor.

Customer classes 5, 6, and 7 are completely analogous to classes 1, 3, and 4 respectively, except that the former refer to Ringbus access and the latter to Multibus accesses. Exactly N customers are always somewhere in the closed loop of classes 1 through 7.

As in our previous model, the processing time distribution is identical for all processors, and the recovery time distribution is the same for all processors. There are two separate access time distributions: one for Multibus accesses and one for Ringbus accesses. The Multibus access time distribution is the same for all byte and word (first or second word of long word) Multibus accesses and the Ringbus access time distribution is the same for all byte and word (first or second word of long word) Ringbus accesses. We denote the access time of a Multibus access by the random variable t_{amb}, and the access time of a Ringbus access by the random variable t_{arb}. The random variables t_p, t_r, t_{amb}, t_{arb} are each assumed to be independent of other random variables and independent of all classes.

2.9.1 Analysis of Multibus Model with Long Word and Ringbus Accesses

2.9.1.1 Asymptotic Behaviour

The Multibus throughput is now $\dfrac{\rho}{(1-\psi)\bar{t}_{aMB}+\psi\bar{t}_{aRB}}$ where ρ is the fraction of time (i.e. probability in steady state) that the Multibus is busy. The throughput balance equation is thus:

$$\frac{(1+\beta)N}{\bar{t}_{cyc}} = \frac{\rho}{\bar{t}_a} \tag{2.17}$$

where $\bar{t}_a$ is the average access time given by $\bar{t}_a = (1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}$ and $\bar{t}_{cyc}$ is the average cycle time given by $\bar{t}_{cyc} = \bar{t}_p + \bar{t}_{w_1} + \bar{t}_a + \beta(\bar{t}_r + \bar{t}_{w_2} + \bar{t}_a)$

As in section 2.8, $\bar{t}_{w_1}$ is the average waiting time for a byte or word access or the first word access of a long word and $\bar{t}_{w_2}$ is the average waiting time for the second word access of a long word. Now however, $\bar{t}_{w_1}$ and $\bar{t}_{w_2}$ refer to the average waiting time of both Multibus and Ringbus accesses. It is certainly possible to partition $\bar{t}_{w_1}$ and $\bar{t}_{w_2}$ each into one component for Multibus

accesses and another for Ringbus accesses, but we choose to continue looking at the overall waiting time per request. Note that in general, $\bar{t}_{w_1} \neq \bar{t}_{w_2}$, as discussed for the case in section 2.8.

The mean total waiting time (or wasted time) per processor cycle is $\bar{t}_{w_T} = \bar{t}_{w_1} + \beta \bar{t}_{w_2}$. Combining this equation with the equation for $\bar{t}_{cyc}$ and equation 2.17 yields:

$$\bar{t}_{w_T} = (1+\beta)\frac{N}{\rho}\bar{t}_a - \bar{t}_p - (1+\beta)\bar{t}_a - \beta \bar{t}_r$$

As discussed in section 2.8, we choose to normalize $\bar{t}_{w_T}$ by the mean memory access time $\bar{t}_m = \bar{t}_a + \beta(\bar{t}_r + \bar{t}_a)$ in order to retain our earlier interpretation of the knee. Thus

$$\frac{\bar{t}_{w_T}}{\bar{t}_m} = \frac{\frac{N}{\rho}}{1 + \frac{\beta\gamma}{(1+\beta)(1+\psi(\zeta-1))}} - \frac{\alpha}{\beta\gamma + (1+\beta)(1+\psi(\zeta-1))} - 1 \tag{2.18}$$

where $\alpha = \frac{\bar{t}_p}{\bar{t}_{aMB}}$, $\gamma = \frac{\bar{t}_r}{\bar{t}_{aMB}}$, and $\zeta = \frac{\bar{t}_{aRB}}{\bar{t}_{aMB}}$

As a function of N, $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ has a knee at $\frac{\alpha + \gamma\beta}{(1+\beta)(1+\psi(\zeta-1))} + 1$ and an asymptotic slope of $\frac{1}{1 + \frac{\beta\gamma}{(1+\beta)(1+\psi(\zeta-1))}}$, which is always less than or equal to 1. As $N \to \infty$, $\rho \to 1$, so $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ is asymptotic to equation 2.18 with $\rho = 1$.

2.9.1.2 Deterministic Behaviour

Consider now the case when t_p, t_r, t_{amb}, and t_{arb} are deterministic quantities. The maximum memory access time is $2t_{arb} + t_r$ (assuming that $t_{arb} > t_{amb}$). Thus $\bar{t}_{w_T} = 0$ for $N \leq \left\lfloor \frac{t_p}{2t_{arb} + t_r} \right\rfloor + 1 = \left\lfloor \frac{\alpha}{2\zeta + \gamma} \right\rfloor + 1 \equiv N_l^*$, where N_l^* corresponds to the knee when $\beta = 1$ and $\psi = 1$. The following theorem shows that the bus is busy when $N \geq \left\lfloor \frac{t_p}{t_{aMB}} \right\rfloor + 1 = \lfloor \alpha \rfloor + 1 \equiv N_u^*$ regardless of the value of β and ψ. N_u^* corresponds to the knee when $\beta = 0$ and $\psi = 0$.

Theorem 2.6

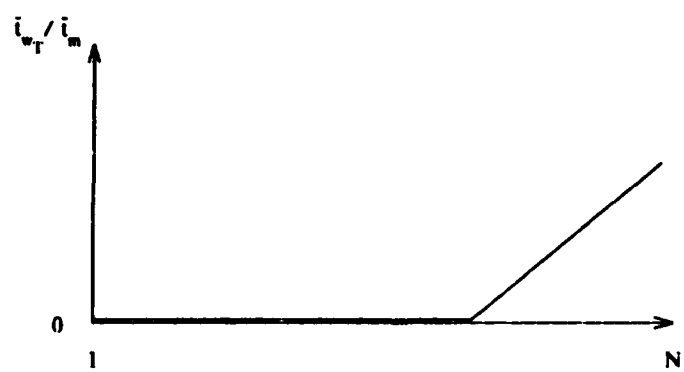
Consider the Multibus model with long word and Ringbus accesses described in the beginning of section 2.9. If t_p, t_{aMB}, t_{aRB}, and t_r are deterministic variables such that $t_r \leq t_p$, $t_r \leq t_{aMB} \leq t_{aRB}$, $t_r > 0$, and $N > \left| \frac{t_p}{t_{aMB}} \right| + 1$ and if each of the N processors has completed at least two memory accesses - byte, word, or first or second word access of a long word - then the fraction of time that the bus is busy, denoted by ρ, is 1.

Proof:

Given by Theorem 2.5 with $t_{a_{min}} = t_{aMB}$.

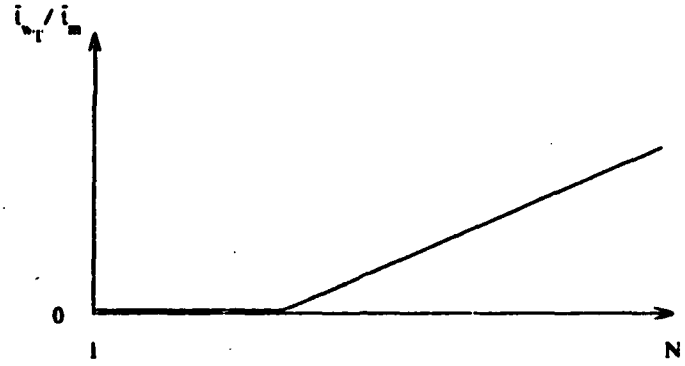
From Theorem 2.6 we conclude that $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ equals its asymptotic value for $N \geq N_u^*$. For $N_l^* < N < N_u^*$ and $0 < \beta < 1$ and/or $0 < \psi < 1$, $\frac{\bar{t}_{w_T}}{\bar{t}_m}$ is strictly positive, again by an argument similar to that in section 2.8.1.2.

The three possible cases are depicted in Figure 2.21. As discussed in section 2.8.1.2, the curve in Figure 2.21(c) is rounded in the knee area due to the randomness introduced by the probabilistic choice of Multibus versus Ringbus access and word versus long word access.



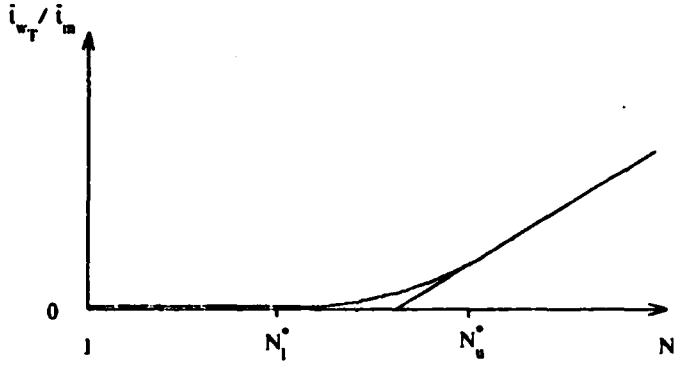
(a): $\beta = 0$ and $\psi = 0$

Knee: $\alpha + 1$ Asymptotic slope: 1



(b): $\beta = 1$ and $\psi = 1$

Knee: $\dfrac{\alpha+\gamma}{2\zeta} + 1$ Asymptotic slope: $\dfrac{1}{1+\dfrac{\gamma}{2\zeta}}$



(c): $0<\beta<1$ and/or $0<\psi<1$

Knee: $\dfrac{\alpha+\beta\gamma}{(1+\beta)(1+\psi(\zeta-1))} + 1$ Asymptotic slope: $\dfrac{1}{1+\dfrac{\beta\gamma}{(1+\beta)(1+\psi(\zeta-1))}}$

Figure 2.21: Representative cases of $\bar{i}_{w_T}/\bar{i}_m$ vs. N in deterministic case

2.9.1.3 Product Form Solution

The FCFS queue for the Multibus is no longer quasi-reversible in general, since the service time depends on the class of the customer; Ringbus accesses may have a different service time distribution than Multibus accesses. Certainly, the FCFS queue remains quasi-reversible if $\psi=0$ and the Multibus access time distribution is exponential or if $\psi=1$ and the Ringbus access time distribution is exponential. The analysis in either of these two cases is the same as in section 2.8. However, we are interested in the general case when $0<\psi<1$. Since the FCFS queue is not quasi-reversible for $0<\psi<1$ (unless the Multibus and Ringbus access time distributions are identical), we cannot use the product form results in section 2.6.1 to give an exact result (no product form solutions are known for non-symmetric FCFS queues). We can however, find an exact product form solution for a slightly different model than the one in which we are interested.

Consider the model presented at the beginning of section 2.9 with general processing, recovery and access time distributions. Obtain a new model by replacing the FCFS queue for the Multibus by a server-sharing queue. (A server-sharing queue is essentially a round-robin queue with infinitesimal quantum size so all queued customers are in service simultaneously.) Since the server-sharing queue is quasi-reversible, this new model has an exact product form solution. We will now derive the exact solution for this new model and use it to approximate the solution of our original model with a FCFS queue.

Let the global state be $\underline{X}=(\underline{x}_P,\underline{y})$ where $\underline{x}_P$ represents the state of the processors and $\underline{y}$ represents the state of the server-sharing queue for use of the Multibus. As in section 2.8.1.2, the processors can be considered as comprising an infinite server and thus they form a quasi-reversible queue (with respect to a Markovian state description). As mentioned earlier, the server-sharing queue is also quasi-reversible (with respect to a Markovian state description). The quasi-reversibility of all the queues in isolation yields the product form:

$$\pi_X=\pi_{x_P}\pi_y$$

Let $\underline{x}_P=(n_2;n_3,n_6)$ and $\underline{y}=(n_1,n_5;n_4,n_7)$ where n_i is the number of customers in class i. Let λ_i^{eff} represent the effective arrival rate of class i customers. Then from the results in section 2.6.1 we have:

$$\pi_{x_P}=\frac{(\lambda_2^{eff}\bar{t}_p)^{n_2}}{n_2!}\cdot\frac{(\lambda_3^{eff}\bar{t}_r)^{n_3}}{n_3!}\cdot\frac{(\lambda_6^{eff}\bar{t}_r)^{n_6}}{n_6!}$$

$$\pi_y=\frac{(n_1+n_5+n_4+n_7)!}{n_1!n_5!n_4!n_7!}(\lambda_1^{eff}\bar{t}_{aMB})^{n_1}(\lambda_5^{eff}\bar{t}_{aRB})^{n_5}(\lambda_4^{eff}\bar{t}_{aMB})^{n_4}(\lambda_7^{eff}\bar{t}_{aRB})^{n_7}$$

Now $\lambda_4^{eff}=\lambda_3^{eff}=\beta\lambda_1^{eff}$, $\lambda_7^{eff}=\lambda_6^{eff}=\beta\lambda_5^{eff}$, $\lambda_1^{eff}=(1-\psi)\lambda_2^{eff}$, and $\lambda_5^{eff}=\psi\lambda_2^{eff}$.

Simplifying, we finally obtain:

$$\pi_X = \frac{C^{v}\alpha^{n_P}(\beta\gamma)^{n_R}(n_{A_1}+n_{A_2})!(1-\psi+\psi\zeta)^{(n_{A_1}+n_{A_2})}\beta^{n_{A_2}}}{n_P!n_R!n_{A_1}!n_{A_2}!} \tag{2.19}$$

where $n_p = n_2$, $n_R = n_3 + n_6$, $n_{A_1} = n_1 + n_5$, and $n_{A_2} = n_4 + n_7$. As before, $\alpha = \frac{\bar{t}_p}{\bar{t}_{aMB}}$, $\gamma = \frac{\bar{t}_r}{\bar{t}_{aMB}}$, and $\zeta = \frac{\bar{t}_{aRB}}{\bar{t}_{aMB}}$. Note that equation 2.19 is exactly the same as equation 2.16 in section 2.8.1.3 except for the $(1-\psi+\psi\zeta)^{(n_{A_1}+n_{A_2})}$ term. We can imagine a similar term in equation 2.16, i.e. $1^{(n_{A_1}+n_{A_2})}$, and thus both have exactly the same form.

Using the results of section 2.8.1.3 we immediately have:

1) the steady-state probability of a total of n_s, $0<n_s<N$ requests in the queue is

$$Prob(n_s \text{ in queue}) = C^{v}\frac{N!}{(N-n_s)!}\left[\frac{(1+\beta)(1-\psi+\psi\zeta)}{\alpha+\beta\gamma}\right]^{n_s}$$

where C^{v} is a normalizing constant

2) $\bar{t}_{w_1} = \bar{t}_{w_2} = \bar{t}_w$

3) $$\frac{\bar{t}_w}{\bar{t}_a} = \frac{C^{v}}{1-C^{v}}\left[\sum_{n_s=1}^{N} n_s \frac{N!}{(N-n_s)!}\left[\frac{(1+\beta)(1-\psi+\psi\zeta)}{\alpha+\beta\gamma}\right]^{n_s}\right] - 1$$ where

$\bar{t}_a = (1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}$.

Points (1) and (3) are the same results as obtained with the M/M/1//N model in section 2.4 (equations 2.1 and 2.2) when $\alpha = \frac{\mu}{\lambda}$ in the M/M/1//N model is replaced by $\frac{\alpha+\beta\gamma}{(1+\beta)(1-\psi+\psi\zeta)}$. Therefore with a server-sharing queue, $\frac{\bar{t}_w}{\bar{t}_a}$ is asymptotic to $N - \frac{\alpha+\beta\gamma}{(1+\beta)(1-\psi+\psi\zeta)}$ for large N. Point (2) implies that $\bar{t}_{w_f} = (1+\beta)\bar{t}_w$. Thus $\frac{\bar{t}_{w_f}}{\bar{t}_m} = \frac{(1+\beta)}{1+\beta+\frac{\beta\gamma}{1+\psi(\zeta-1)}} \times \frac{\bar{t}_w}{\bar{t}_a}$.

To gauge the accuracy of the result for the server-sharing model as an approximation for the original model, consider the Multibus model with long word accesses in section 2.8. The product form solution of this model is exactly the same for FCFS and server-sharing disciplines at the Multibus queue. However, the product form solution with the server-sharing discipline is more comprehensive: it is exact for general distributions for the processing, recovery, and access times (i.e. it is not limited to an exponential access time distribution as with the FCFS discipline). Since

the product form solution is the same for FCFS and server-sharing disciplines, the simulations reported in section 2.8.1.4 may be used to determine the accuracy of the solution for the server-sharing discipline in approximating the solution for the FCFS discipline. We reach the same conclusion as in section 2.8.1.4: the approximation is excellent except in the knee area and in general, $\bar{t}_{w_1} \neq \bar{t}_{w_2}$. In the knee area, $\bar{t}_w$ with the server-sharing discipline is too large for some processing time distributions (those with $C_{t_p} < 1$ it seems) and too small for other processing time distributions (those with $C_{t_p} > 1$ it seems). Extrapolating, we expect roughly the same the accuracy of our server-sharing model in section 2.9.1.3 as an approximation for the original FCFS model.

It is important to temper the previous sentence with the observation that we are basing our extrapolation to the case with general Ringbus access time distribution (of possibly large variance) on the simulations performed for deterministic access times. However, the accuracy of the server-sharing model will likely remain very good except around the knee area where we expect the greatest inaccuracies to accrue. We have chosen not to perform any simulations to determine further the accuracy of our server-sharing model. The reason is that, as in section 2.8.1.3, we expect the mean waiting time to be more sensitive to the values of the parameters, such as β and ψ, than the exact form of the probability distributions. Therefore it seems best to study the factors influencing the parameters before studying the effect of the probability distributions.

2.9.1.4 A Special Case

In the special case when the processing time is exponentially distributed and there are no long word accesses (i.e. $\beta = 0$) an exact result for the average waiting time per request can be obtained from the M/G/1//N results in section 2.5. Since there are no long word accesses, we can combine the Multibus access time and Ringbus access time distributions into one access distribution. Specifically, if the Multibus access time distribution is $\text{Prob}(t_{amb} \leq t) = F_{amb}(t)$ and the Ringbus access time distribution is $\text{Prob}(t_{arb} \leq t) = F_{arb}(t)$, then the overall access time distribution is $F_a(t) = (1-\psi)F_{amb}(t) + \psi F_{arb}(t)$. The average waiting time $\bar{t}_w$ can be determined by applying the formulae in section 2.5 with $F^*(s) = \int_0^\infty e^{-sx}\, dF_a(x)$.

2.9.2 The Single Processor Equivalent of the Multibus

As discussed in section 1.2.4, we have decomposed the overall model of Concert into Multibus and Ringbus models and when dealing with one of these models, we replace the other models by equivalent models. Up to this point we have examined the Multibus model: we have assumed some Ringbus access time distribution and determined the performance of the Multibus model with that distribution. Now we examine the single processor equivalent model of the

Multibus.

The single processor equivalent of the Multibus is characterized by a processing time distribution, Ringbus destination probabilities, and $\beta = 0$, $\psi = 1$ (as discussed in section 1.2.4). The processing time distribution presents the most difficulty - we must find the probability distribution of the Ringbus spacing.† The Ringbus destination probabilities are trivial to determine. Since we have assumed that all processors in the Multibus model are identical, the Ringbus destination probabilities for the entire Multibus are the same as that for one processor. Thus the Ringbus destination probabilities for the single processor equivalent, denoted by p_i^{MBeqv}, are given by $p_i^{MBeqv} = p_i$ for all i, where the Ringbus destination probabilities for each processor in the Multibus model are denoted by p_i.

The Ringbus spacing probability distribution is very difficult to find in closed form. Instead, we choose to approximate the Ringbus spacing distribution by another distribution with the same first moment. We could also use higher moments in the approximation of the Ringbus spacing distribution, thereby achieving greater accuracy. However, higher moments are progressively more difficult to obtain from the Multibus model. We therefore choose to stick with our simple first moment approximation and evaluate the results before considering more complex and accurate approximations. Indeed, the results so obtained may be sufficiently accurate that more accurate approximations are unnecessary. To ease analysis, we choose an exponential distribution, which is completely parameterized by its first moment, to approximate the Ringbus spacing distribution. Recall from section 1.2.4 that the processing time probability distribution of the single processor equivalent is equal to the Ringbus spacing probability distribution. Thus we have just approximated the processing time distribution of the single processor equivalent by an exponential distribution. Let the mean of this distribution be denoted by $\bar{t}_p^{MBeqv}$.

The Ringbus access time distribution is also very difficult to find in closed form (as we shall see in Chapter 3). For the same reasons as above, we choose to also approximate the Ringbus access time distribution by an exponential distribution. Since both the processing time distribution of the single processor equivalent and the Ringbus access time distribution are thus completely specified by their respective first moments, integrating the Multibus and Ringbus models reduces to first moment matching, rather than the (considerably) more difficult task of matching continuous distributions.

We now determine the mean processing time of the single processor equivalent of the Multibus in terms of Multibus parameters. The mean time between initiation of Ringbus accesses is $\bar{t}_p^{MBeqv} + \bar{t}_{aRB}$. This mean time is also given by $\dfrac{\bar{t}_{cyc}}{(1+\beta)N\psi}$ where

† In section 1.2.4 we defined the Ringbus spacing to be the interval between the completion of one access on

$\bar{t}_{cyc} = \bar{t}_p + \bar{t}_{w_1} + \bar{t}_a + \beta(\bar{t}_r + \bar{t}_{w_2} + \bar{t}_a)$ and $\bar{t}_a = (1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}$. Thus the mean processing time of the single processor equivalent is given by

$$\bar{t}_p^{\,MBeqv} = \frac{\bar{t}_{cyc}}{(1+\beta)N\psi} - t_{aRB} \tag{2.20}$$

To proceed further we require a relationship between $\bar{t}_{w_T} = \bar{t}_{w_1} + \beta\bar{t}_{w_2}$ and the Multibus parameters. We choose to use the exact results for the server-sharing queue model developed in section 2.9.1.3 to approximate the general case. There is, of course, some error involved with this approximation, but at least we have a convenient result expressing the relationship between $\bar{t}_{w_T}$ and the Multibus parameters. As discussed in section 2.9.1.3, the server-sharing queue model should give fairly accurate results for $\bar{t}_{w_T}$ except around the knee area. Substituting the equation for $\bar{t}_{cyc}$ into equation 2.20 we have:

$$\bar{t}_p^{\,MBeqv} = \frac{\bar{t}_p + \beta\bar{t}_r}{(1+\beta)N\psi} + \frac{\bar{t}_a\left(\frac{\bar{t}_w}{\bar{t}_a}+1\right)}{N\psi} - \bar{t}_{aRB} = \bar{t}_{aMB}\left\{\frac{(\alpha+\beta\gamma)}{(1+\beta)N\psi} + (1-\psi+\zeta\psi)\frac{\left(\frac{\bar{t}_w}{\bar{t}_a}+1\right)}{N\psi} - \zeta\right\}$$

where $\frac{\bar{t}_w}{\bar{t}_a}$ is the mean waiting time per request for the M/M/1//N model of section 2.4 (equations 2.1 and 2.2) with $\frac{\mu}{\lambda} = \frac{\alpha+\beta\gamma}{(1+\beta)(1-\psi+\psi\zeta)}$.

For small N, $\frac{\bar{t}_w}{\bar{t}_a} \approx 0$, and thus $\bar{t}_p^{\,MBeqv} \approx \bar{t}_{aMB}\left\{\frac{(\alpha+\beta\gamma)}{(1+\beta)N\psi} + \frac{1-\psi}{N\psi} - \zeta(1-\frac{1}{N})\right\}$. Therefore $\bar{t}_p^{\,MBeqv}$ is approximately linear in ζ for small N. For large N, $\frac{\bar{t}_w}{\bar{t}_a} \approx N - \frac{(\alpha+\beta\gamma)}{(1+\beta)(1-\psi+\psi\zeta)} - 1$ and thus $\bar{t}_p^{\,MBeqv} \approx \bar{t}_{aMB}\frac{(1-\psi)}{\psi}$, a constant.

As we shall see in section 3.9.1, we need one more quantity from the single processor equivalent of the Multibus when we integrate the Multibus and Ringbus models. This quantity, which we denote by P_{RB} is the probability that at the termination of a Ringbus access, the Multibus queue is nonempty and the request at the head of the queue is a Ringbus request. In other words,

$P_{RB} = Prob$(a customer departing from the Multibus queue leaves a Ringbus request

at the head of the queue | the customer departing is a Ringbus request)

the Multibus with a Ringbus destination and the start of the next such access on the Multibus.

A closed network of quasi-reversible queues has the property that when a customer of a given class arrives at or departs from a queue, the other customers in the system are distributed according to the steady-state probability distribution obtained if they were the only customers in the system [Theorem 3.12 of Ref. K1]. Thus P_{RB} in the server-sharing queue approximation of the general case is given by the steady-state probability of the customer in service - i.e. at the head of the queue - representing a Ringbus access in a $N-1$ processor system. (N is the number of processors in the original system.) We denote this probability by ρ_{RB}^{N-1}. Let ρ^{N-1} denote the steady-state probability of there being any customer in service in a $N-1$ processor system.

Using Little's Law we have $\rho_{RB}^{N-1} = \lambda_{RB}^{eff(N-1)}\bar{t}_{aRB}$ and $\rho^{N-1} = \lambda_{TOT}^{eff(N-1)}\bar{t}_a$. From section 2.9.1.3 we have

$$\lambda_{RB}^{eff(N-1)} = \lambda_5^{eff(N-1)} + \lambda_7^{eff(N-1)}$$

$$= \psi(1+\beta)\lambda_2^{eff(N-1)}$$

$$\lambda_{TOT}^{eff(N-1)} = \lambda_1^{eff(N-1)} + \lambda_3^{eff(N-1)} + \lambda_5^{eff(N-1} + \lambda_7^{eff(N-1)}$$

$$= (1+\beta)\lambda_2^{eff(N-1)}$$

and

$$\bar{t}_a = ((1-\psi)+\zeta\psi)\bar{t}_{aMB}$$

Thus

$$\rho_{RB}^{N-1} = \frac{\psi\bar{t}_{aRB}}{\bar{t}_a}\rho^{N-1} = \frac{\psi\zeta}{1-\psi+\zeta\psi}\rho^{N-1}$$

We have already noted that the server-sharing queue model in section 2.9.1.3 has the same solution for $\bar{t}_w/\bar{t}_a$ as a M/M/1//N queue model with $\frac{\mu}{\lambda} = \frac{\alpha+\beta\gamma}{(1+\beta)(1-\psi+\psi\zeta)}$. The same holds for the probability that the server is busy. That is, ρ^{N-1} is the probability that the server is busy in a M/M/1//N-1 system with $\frac{\mu}{\lambda} = \frac{\alpha+\beta\gamma}{(1+\beta)(1-\psi+\psi\zeta)}$. Finally, $P_{RB} = \frac{\psi\zeta}{1-\psi+\psi\zeta}\rho^{N-1}$.

2.10 Extensions

The Multibus models considered so far have four main weaknesses:

1. All processors are identical.
2. The processor model is very simple, perhaps too simple.
3. The processor model is stationary, i.e. time independent.
4. All processors are independent.

We assumed points 1 through 4 in the previous sections to obtain simple and analytically tractable models. In this section we consider extensions to relax each of these assumptions.

2.10.1 Non-identical processors

This case is straightforward to handle by simply adding more states to the Multibus model to represent the different combinations of non-identical processors. For example, we can change the state description of the M/M/1//N model in section 2.3 from (n), where n represents the number of requests waiting for or in service to $(n, c_1, c_2, \cdots, c_n)$, where n is the same as before and c_i represents the processor from which the ith request in the queue originated. In a sense, we now have N classes of customers (for N processors) where there is one class per processor. Similarly, we can add classes to the Multibus model with Ringbus accesses in section 2.9 to distinguish the respective processors at which requests originate. For example, we could choose the classes $7(i-1)+1, 7(i-1)+2, \cdots, 7(i-1)+7, 1 \leq i \leq N$, where i denotes the originating processor and $7(i-1)+1, 7(i-1)+2, \cdots, 7(i-1)+7$ represent the 7 classes (as in section 2.9) associated with the originating processor i. A product form solution, similar to that developed in section 2.9.1.2, can be developed with respect to these classes.

Since the processors are now non-identical, the mean waiting time per request, $\bar{t}_w$, is not necessarily the same for the requests of all processors. This complicates the calculation of the throughput. It is probably best to consider the mean waiting time per request from processor i, for all i, rather than the mean waiting time for *any* request given by $\bar{t}_w$.

Note that while the case with non-identical processors is straightforward to handle, the state space required and the complexity of the analysis increases without necessarily contributing much insight.

2.10.2 More Complex Processor Models

This case can again be handled by increasing the number of states representing the Multibus model. We assume in this subsection that the processors are identical, independent, and stationary. These assumptions can be relaxed by the methods discussed in the preceding and succeeding subsections.

Within the assumptions stated above, we can make the processor model arbitrarily complex and, provided that we can find a Markovian state description of the processor, we can augment the state of the Multibus model with the state of each processor model and in principle solve for the steady-state probability distribution. Once we know the steady-state probability distribution we can in principle determine any related performance measurement of interest. The difficulty, of course, is with the "in principle" part.

One quite general way to proceed is to approximate the entire Multibus model (including the processor models) by a queueing network model with a product form solution. One advantage of this approach is that we can deal with the model at a more abstract level. The states need not be Markovian: it suffices that each queue is quasi-reversible in isolation with respect to some Markovian state description but we need not find or deal with such a description. A second advantage is that we can obtain analytical expressions for the steady-state distributions and hence for the performance measures of interest. A disadvantage is that inevitably some simplifying assumptions are involved. In some cases the necessary simplifying assumptions may obscure or eliminate the features of interest. In such cases one must resort to other methods such as simulation. (There is a paucity of methods for dealing with large non-product form systems.)

A way to extend the processor model using a queueing network model is to consider the processor operation as consisting of a set of activities, say $A_1, A_2, \cdots, A_m$. One activity might correspond to program execution in the processor's local memory, another might correspond to reading or writing global data, and yet another might correspond to busy waiting on some global memory location, and so on. (Of course, with our independence assumption, the period of time spent busy waiting must be independent of the operation of the other processors.) Associated with each activity is some interarrival time of requests for the Multibus, some interarrival time of requests for the Ringbus, a probability distribution for the time spent in that activity, and a probability distribution for the next activity (which may depend on the previous activities and the time in each). We can describe the overall Multibus model by a queueing network by regarding the activities as queues (several queues may be necessary to describe each activity) and the operation of the processors as customers which move from queue to queue. The customers can belong to classes which represent the previous queue(s) visited, the service time at a queue, and so on (provided each queue remains quasi-reversible with respect to the classes). Finally, the transition from one class to another can be governed by a probability distribution depending only on the present class.

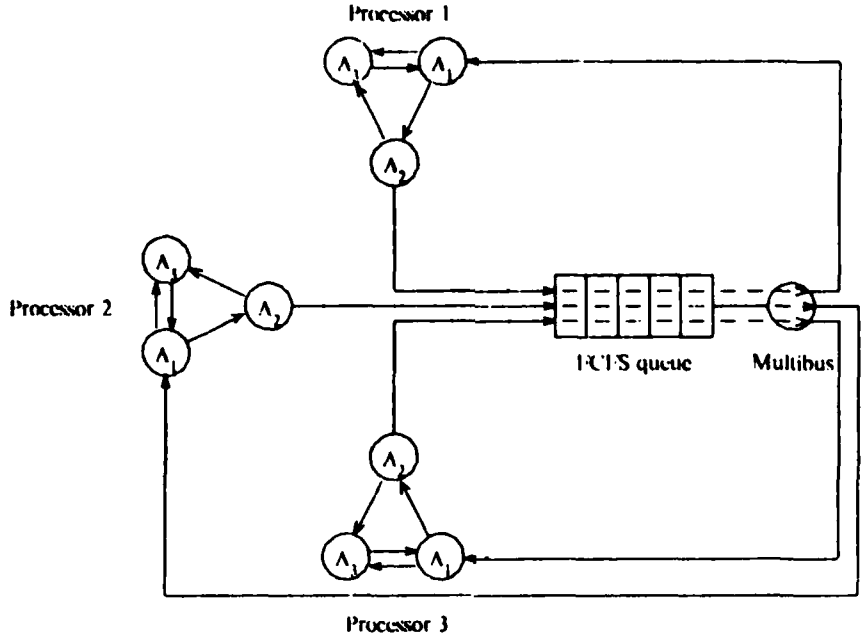


Figure 2.22: Queueing network model with processor activities

A queueing network model for a three processor system with three activities each is depicted in Figure 2.22.

If each queue is quasi-reversible in isolation, then the global state probability has a product form solution. Since there is at most one customer per class (we assume that there is a total of one customer in all the classes associated with a single processor - more than one would correspond to a multi-tasking processor), the service time distribution at each queue except the FCFS Multibus queue may be completely general. As discussed in sections 2.8 and 2.9, the service time distribution at the Multibus queue must either be exponential with the same mean for all customers or the queue discipline must be server-sharing.

To illustrate the activity-based queueing network model more concretely, we consider the following general case.

Let there be N, not necessarily identical processors. Let the model for processor i consist of $Q(i)$ queues $Q^i{}_1, Q^i{}_2, \cdots, Q^i{}_{P(i)}$ and the Multibus queue (which is common to all N processors models). Let there be a finite set of customer classes $C(i)$ associated with each processor i. Each customer class visits at least one queue. Upon departing from a queue, a customer of class k joins class l with probability $r^i{}_{kl}$ (for processor i). The classes associated with processor i form a single closed loop including all $Q(i)$ queues and the Multibus queue, thus $\sum_{k \in C(i)} \sum_{l \in C(i)} r^i{}_{kl} = 1$.

Let there be exactly one customer in the closed loop of classes corresponding to each processor. Thus the service time distribution at each queue except the Multibus queue may be completely general. (In addition, any customer at a queue other than the Multibus queue must be receiving service.) We assume that the service time distribution at each queue is independent of customer class. (For non-Multibus queues we can simply add more queues and classes to circumvent this restriction.) We also assume either that the Multibus queue discipline is FCFS and the service time distribution is exponential or that the Multibus queue discipline is server-sharing and the service time distribution is general.

By adding a sufficient number of queues and classes, the general case just described can handle or approximate a wide range of activities and processor models. As stated earlier in this section, the classes can represent quite detailed history, such as previous queues visited and the service times at those queues. Therefore one can even have an approximate distribution for the time spent in an activity by defining classes to represent the time elapsed in a certain activity. (This technique will be discussed in more detail in section 2.10.3.) By construction, each queue in the general case just described is quasi-reversible in isolation and thus the global state probability has a product form solution. We now investigate this solution.

Let the global state be $\underline{X}=(\underline{x}_{P_1},\cdots,\underline{x}_{P_N},\underline{y})$ where $\underline{x}_{P_i}$ represents the state of processor i and y represents the state of the Multibus queue. Then we have the product form solution:

$$\pi_X=\prod_{i=1}^{N}\pi_{x_{P_i}}\cdot\pi_y.$$

Let $\underline{x}_{P_i}=(\underline{q}^i{}_1,\cdots,\underline{q}^i{}_N)$ where $\underline{q}^i{}_j$ denotes the state of queue j for processor i and let $\underline{q}^i{}_j=(n^i{}_j(l),n^i{}_j(k),\cdots)$ for each class $l,k,\cdots\in C(i)$ where $n^i{}_j(k)$ denotes the number of customers in class k at queue j. Let $\lambda^i{}_j$ denote the effective arrival rate of class j customers for processor i. Conservation of flow yields

$$\lambda^i{}_j=\sum_{k\in C(i)}\lambda^i{}_k r^i{}_{kj}\,,\quad j\in C(i) \tag{2.21}$$

Then the steady-state probability of state $\underline{q}^i{}_j$ is

$$\pi_{\underline{q}^i{}_j}=Cn^i{}_j!\prod_{k\in C(i)}\frac{\left[\rho^i{}_{jk}\right]^{n^i{}_j(k)}}{n^i{}_j(k)!}$$

where $n^i{}_j=\sum_{k\in C(i)}n^i{}_j(k)$, $\rho^i{}_{jk}=\bar{s}^i{}_j\lambda^i{}_k$, and $\bar{s}^i{}_j$ is the mean service time at queue j for processor i.

If we let $\underline{q}^i{}_j=(n^i{}_j)$ we have

$$\pi_{q^i_j} = \frac{C^i \left[\rho^i_j\right]^{n^i_j}}{n^i_j!}$$

where $\rho^i_j = \sum_{k \in C(i,j)} \rho^i_{jk}$ and $C(i,j)$ is the set of classes arriving at queue j for processor i. Thus the steady-state probability of state $\underline{x}_{P_i}$ is

$$\pi_{x_{P_i}} = C^i \prod_{j=1}^{Q(i)} \frac{\left[\rho^i_j\right]^{n^i_j}}{n^i_j!}$$

Let $n^i = \sum_{j=1}^{Q(i)} n^i_j$. Then if we let $\underline{x}_{P_i} = (n^i)$ we have (since $n^i = 0$ or 1)

$$\pi_{x_{P_i}} = C^i \left[\rho^i\right]^{n^i}$$

where $\rho^i = \sum_{j=1}^{Q(i)} \rho^i_j = \sum_{j=1}^{Q(i)} \left[\bar{s}^i_j \sum_{k \in C(i,j)} \lambda^i_k\right]$.

Let the state of the Multibus queue be represented by $\underline{y} = (m_1, \cdots, m_N)$ where m_i denotes the number of requests in the queue from processor i. Note that $m_i + n^i = 1$. Denote the effective arrival rate of customers at the Multibus queue from processor i by λ^i_{MB} i.e. $\lambda^i_{MB} = \sum_{k \in C(i,MB)} \lambda^i_k$ where $C(i,MB)$ is the set of all classes arriving at the Multibus from processor i. Then

$$\pi_y = m!\left[\bar{t}_a\right]^m \prod_{i=1}^{N} \frac{\left[\lambda^i_{MB}\right]^{m_i}}{m_i!}, \qquad m = \sum_{i=1}^{N} m_i$$

Therefore if the global state is $\underline{X} = (n^1, \cdots, n^N, m_1, \cdots, m_N)$ we have

$$\pi_X = C^{II} m!\left[\bar{t}_a\right]^m \prod_{i=1}^{N} \frac{\left[\rho^i\right]^{1-m_i}\left[\lambda^i_{MB}\right]^{m_i}}{m_i!}$$

$$= C^{III}\left[m_1\, m_2^{m} \cdots m_N\right] \prod_{i=1}^{N} \left[\sum_{j=1}^{Q(i)} \tau^i_j \frac{\sum_{k \in C(i,j)} \lambda^i_k}{\lambda^i_{MB}}\right]^{-m_i} \tag{2.22}$$

where $\tau^i_j = \frac{\bar{s}^i_j}{\bar{t}_a}$ (and $n^i = 1 - m_i$, $m_i = 0$ or 1). The quantity $\frac{\sum_{k \in C(i,j)} \lambda^i_k}{\lambda^i_{MB}}$ is the ratio for processor i of the effective arrival rate at queue j to the effective arrival rate at the Multibus queue.

The mean waiting time per request from processor i, $\bar{t}_{w_i}$, and the mean waiting time per request for *any* request, $\bar{t}_w$, can be derived from equation 2.22.

We note the following two points about equation 2.22:

1. Equation 2.22 is dependent on the details of the model for processor i only through the quantities $s^i{}_j$ and $\frac{\sum_{k \in C(i,j)} \lambda^i{}_k}{\lambda^i{}_{MB}}$ (for $(j = 1, \cdots, Q(i))$. The former quantity is given and the latter can be computed on solving the conservation flow equations 2.21 (within some arbitrary constant). Thus the overall solution for $\bar{t}_{w_i}$ or $\bar{t}_w$ effectively reduces to solving a set of linear equations (equation 2.21) for each of the N processors. Since solving large sets of such equations is relatively easy, the main difficulty with applying queueing networks to model complex processor behaviour is specifying the desired behaviour in terms of queues, service time distributions, and routing probabilities.

2. Consider the model in section 2.4 with exponential processing and access time distributions with non-identical processors. If we let the global state be $\underline{X}_{exp} = (m_1, \cdots, m_N)$ where m_i is the number (0 or 1) of requests from processor i in the Multibus queue, then the steady-state probability of $\underline{X}_{exp}$ is

$$\pi_{\underline{X}_{exp}} = C^{-1} \binom{m}{m_1\ m_2 \cdots m_N} \prod_{i=1}^{N} \left(\frac{\bar{t}_{p_i}}{\bar{t}_a} \right)^{-m_i} \tag{2.23}$$

where $\bar{t}_{p_i}$ is the mean processing time of processor i. (Equation 2.23 follows from equation 2.5 in section 2.6.) Equations 2.22 and 2.23 are identical if $\bar{t}_{p_i}$ is replaced by $\sum_{j=1}^{Q(i)} s^i{}_j \frac{\sum_{k \in C(i,j)} \lambda^i{}_k}{\lambda^i{}_{MB}} \equiv \bar{t}_{p_i}{}^{eff}$. Therefore the most complicated stationary model, when expressed as a queueing network model as described in the general case presented earlier, has the same solution for $\bar{t}_{w_i}$ and $\bar{t}_w$ as the simple exponential processing and access time model (with appropriate $\bar{t}_{p_i}{}^{eff}$)! It is fascinating that the single parameter $\bar{t}_{p_i}{}^{eff}$ suffices in the solution of an (almost) arbitrarily complex model. Of course, the underlying reason for this result is the exponential access time or server-sharing discipline of the Multibus queue.

A possibility to circumvent the difficulty mentioned in point 1 is now apparent. Simulate or actually run a single processor with the desired complex behaviour on a system with exponentially distributed access times (perhaps simulation is best to achieve such access times). Measure the steady-state probability distribution and solve for the $\bar{t}_{p_i}{}^{eff}$ which yields this same probability

distribution with exponentially distributed processing time. Once the value of $\bar{t}_{p_i}^{eff}$ has been determined in such a manner for each different processor, $\bar{t}_{w_i}$ and $\bar{t}_w$ can be computed from equation 2.23. This indirect approach for determining $\bar{t}_{w_i}$ and $\bar{t}_w$ may be cheaper for a large number of processors N than the obvious alternative of simulating the entire system since N simulation runs of a single processor may be cheaper than one simulation run of N processors (for the same degree of accuracy).

Equation 2.22 is a fine result if the performance measures of interest involve just the status of the Multibus queue and do not involve the status of any processors. If the measures of interest involve both the Multibus and the processors, then we cannot simplify the solution of the queueing network model to such a degree. This unfortunately means that the state space may remain large. Finding the solution of queueing networks with a large number of states is computationally expensive. Efficient techniques for handling such cases have been developed by Buzen [B3], Chandy, Herzog, and Woo [C2], Reiser and Kobayashi [R2], Reiser and Sauer [R3], Chandy and Sauer [C3], Lam [L1], and Lam and Lien [L2]. However, even these techniques require a lot of work when the state space is as enormous as it might easily get with complex models.

Another approach when the queueing network remains large after simplification or when product form queueing network models are not applicable, is to decompose the overall model into more manageable submodels, each of which can be studied and solved independently, and integrate the submodel results to obtain an overall solution. Except in special circumstances, such a procedure yields only approximate results and thus several iterations of decomposition and integration may be required to obtain results of sufficient accuracy.

2.10.3 Time Dependent Behaviour

This subsection is directed chiefly towards time dependent behaviour of the processing time distribution. We regard the access time distribution as mainly fixed by the hardware and thus time invariant. However, the probabilities of the different type of accesses - word vs. long word and Multibus vs. Ringbus - may well be time dependent. If these probabilities are time dependent they can be treated in the same manner as the processing time distribution.

We limit our discussion to processor behaviours that can be reasonably well approximated as time independent - i.e. stationary - on a finite number of nonzero time intervals. The idea is to represent each stationary interval of this piece-wise stationary approximation of the processor behaviour by a stationary submodel. The overall processor model then consists of a finite set of such submodels, with exactly one such submodel in effect at each point in time; the duration each submodel remains in effect; and some strategy to choose the next submodel when the time allotted to the present submodel is expended. Each stationary submodel can be arbitrarily complex -

such as those models discussed in sections 2.10.1 and 2.10.2 - as long as it is stationary and independent of all other submodels.

We distinguish two cases based on the time required for a submodel to approach steady-state (i.e. for the transients to die out) relative to the duration of the submodel.

Case 1: For every submodel, the time required to approach steady-state is small relative to the duration of the submodel. (We will not discuss what is "short" enough.) In this case it may be reasonable to approximate the behaviour of each submodel over its entire duration by its steady-state behaviour. The behaviour of the overall model can then be approximated as a piece-wise function of the steady-state behaviour on each submodel interval. In this case it is probably best to represent any performance measure of interest for the overall model by a vector of such performance measures with each element of the vector corresponding to a different submodel. Knowledge of the duration of each submodel and the strategy for choosing submodels allows the average of any performance measure to be determined from its performance measure "vector" on all the submodels. Note, however, that such an average performance may not be too meaningful; at the least, it must be carefully interpreted. Note also that the steady-state behaviour of the other submodels can be determined simply by assuming it is the only submodel. Thus this case has the important attribute that the overall model can be decomposed into a number of smaller and independent submodels.

Case 2: For at least one submodel, the time required to approach steady-state is not small relative to the duration of the submodel. This case is more difficult since the dynamics of the overall model preclude its treatment as independent submodels. (There are certainly situations in which some but not all of the submodels can be treated as independent and approximated by their steady-state behaviour over their entire duration. Perhaps such hybrid situations should be called Case 3.) To handle Case 2 we need to incorporate in the state description somehow the expended time (or remaining time) in the duration of the submodel in effect and the submodel (and perhaps some past history of submodel choices too). Of course we can specify a Markov process which incorporates this additional information* but we again return to the more abstract queueing network models. In fact we return to the activity based queueing network model discussed in the previous subsection.

We consider each submodel to be an activity with some probability distribution, $F_d(t)$, for the time in that activity and some probability distribution for the next activity to enter given the

* It will almost certainly turn out that it is too difficult to treat such Markov processes analytically except in trivial cases.

current activity. (This latter probability can be generalized to depend on past activities.) We represent the set of activities as a queueing network model as follows.

Let there be one queue per activity. The service time at this queue has the same (stationary) distribution as the processing time of that activity. (We consider a situation in which the processing time distribution of the overall model is not stationary. We do not consider any other complications here on the basic queueing model discussed in section 2.1.) Let the classes associated with the queue represent the total amount of processing time elapsed so far while in that activity. Specifically, let there be classes

1. $c^1(i,n\Delta t)$ representing a request for the Multibus from activity i where the cumulative processing time while in activity i is $t \in [n\Delta t,(n+1)\Delta t)$, and
2. $c^2(i,n\Delta t)$ representing a request returning from Multibus service to activity i with cumulative processing time while in activity i of $t \in [n\Delta t,(n+1)\Delta t)$. (We quantize time so we can deal with discrete probabilities for the time being. We have chosen quanta of uniform size for simplicity in the presentation.)

The routing probabilities at queue i (i.e. the queue associated with activity i) are:

$$p(c^2(i,n\Delta t);c^1(j,m\Delta t)) = \begin{cases} \int_{(m-n-1)\Delta t}^{(m-n)\Delta t} f_{p_i}(s)ds, & \text{if } m > n \text{ and } j = i \\ 0, & otherwise \end{cases}$$

where $f_{p_i}(t)$ is the probability density function (pdf) of the processing time at queue i. The routing probabilities at the Multibus queue are:

$$p(c^1(i,n\Delta t);c^2(j,t)) = \begin{cases} e^i(n\Delta t)\cdot p_{ij} & \text{if } t = 0 \text{ and } j \neq i \\ 1 - e^i(n\Delta t) & \text{if } t = n\Delta t \text{ and } j = i \\ 0 & otherwise \end{cases}$$

p_{ij} is the probability that the next activity is j given that the current activity is i ($\sum_{j \neq i} p_{ij} = 1$). $e^i(n\Delta t)$ is the probability that activity i ends in $[n\Delta t,(n+1)\Delta t)$ given that the sum of the processing times incurred while in this activity is $\geq n\Delta t$, i.e.

$$e^i(n\Delta t)=\begin{cases} Prob(n\Delta t \le d_i \le (n+1)\Delta t \mid d_i \ge n\Delta t)=\dfrac{\int_{n\Delta t}^{(n+1)\Delta t} f_{d_i}(s)ds}{\int_{n\Delta t}^{\infty} f_{d_i}(s)ds}, & Prob(d_i < n\Delta t)<1 \\ 1, & Prob(d_i < n\Delta t)=1 \end{cases}$$

where $f_{d_i}(t)$ is the pdf of the duration in activity i.

We now have a queueing network model of the form discussed in the previous subsection. From equation 2.23 we know that the steady-state probability distribution of customers in the Multibus queue (from which we can determine $\bar{t}_{w_i}$ and $\bar{t}_w$) depends on the mean processing time while in each activity and the ratio of the effective arrival rate at each queue to the effective arrival rate at the Multibus.

Denote the effective arrival rate of class $c^l(i,n\Delta t)$ customers by $\lambda(c^l(i,n\Delta t))$. Then the ratio $\dfrac{\sum_{k\in C(i,j)} \lambda^i{}_k}{\lambda^i{}_{MB}}$ of equation 2.22 is given by

$$\frac{\sum_{n=0}^{\infty} \lambda(c^2(i,n\Delta t))}{\sum_{i}\sum_{n=0}^{\infty} \lambda(c^1(i,n\Delta t))} \tag{2.24}$$

The conservation of flow equations are

$$\lambda(c^2(i,n\Delta t))=(1-e^i(n\Delta t))\lambda(c^1(i,n\Delta t)) + \sum_{j\ne i} p_{ji}e^j(n\Delta t)\lambda(c^1(j,n\Delta t))$$

and

$$\lambda(c^1(i,m\Delta t))=\sum_{n=0}^{m-1}\int_{(m-n-1)\Delta t}^{(m-n)\Delta t} f_{p_i}(s)ds\,\lambda(c^2(i,n\Delta t)).$$

Manipulating these equations we have

$$\sum_{m=0}^{\infty} \lambda(c^1(i,m\Delta t))=\sum_{m=0}^{\infty} \lambda(c^2(i,m\Delta t)) \tag{2.25}$$

and

$$\lambda(c^1(i,m\Delta t)) = \sum_{n=0}^{m-1} \int_{(m-n-1)\Delta t}^{(m-n)\Delta t} f_{p_i}(s)ds(1-c^i(m\Delta t))\lambda(c^1(j,m\Delta t)) + \sum_{j\neq i} p_{ji}c^i(n\Delta t)\lambda(c^1(j,n\Delta t)) \tag{2.26}^{\dagger}$$

The ratio in equation 2.24 is determined by the solution of equation 2.26 and the identity 2.25. The system of linear equations 2.26 can be solved for $\lambda(c^1(i,n\Delta t))$ within an arbitrary constant. Therefore, as in the previous subsection, the overall solution for $\bar{t}_{w_i}$ and $\bar{t}_w$ effectively reduces to solving a set of linear equations for each of the N processors.

It is highly desirable to keep the number of time quanta fairly small so that the number of equations to solve in 2.26 is not enormous. The degree of inaccuracy introduced in the solution by the quantization can be estimated by comparing the solution with that obtained with a larger number of quanta.

Finally, this treatment of nonstationary processor behaviour can be extended, along the lines of the previous subsection, to deal with more complex processor behaviour.

2.10.4 Dependent Processors

By dependent processors we mean that for at least one processor i there exists some time t such that the operation of the processor is not statistically independent of the operation of processor j for all $j \neq i$ and for all time $s < t$. To model dependent processors, the state of a processor must be allowed to depend on the state of other processors. This dependency unfortunately precludes the use of queueing network models with product form solutions as we have pursued to this point in this thesis. The reasoning is as follows.

In a queueing network model, the state of a processor is given by the concatenation of the states of all queues representing that processor. Alternatively, we can view the state of a processor as given by the class in which the one customer is in. (There can only be one customer per processor since we are modeling the Multibus at the memory access level and processors are single tasking - i.e. a processor is idle while it has a Multibus memory access pending or in progress.) Thus if the states of two processors are dependent, then some of the respective classes of the processors are dependent - i.e. the present class of the customer for one processor may determine the present or future class of the customer for another processor. But a product form solution is not guaranteed if the class of one customer depends on the class of another since the routing of customers is now effectively dependent on the state of the queueing network. (Walrand's proof [W1]

† Upon taking the limit $\Delta t \to 0$, equation 2.26 becomes a set of Volterra integral equations of the second kind [113].

of the product form for networks of quasi-reversible queues requires that the routing be independent of everything else.) For two processors we can attempt to circumvent the difficulty imposed by dependent classes by introducing "superclasses" to represent all possible pairs (C_i, C_j) where C_i denotes a customer class for processor i. (This can be generalized for more than two processors.) A change in class at processor i then forces a change in the superclass which also forces a change in class at process j. A product form solution can be developed with respect to the superclasses. However, a customer in a superclass can only have one service time distribution at each queue. Yet a customer in a superclass represents two customers of classes C_i and C_j respectively from different processors with possibly vastly different service times at queues. Therefore a queueing network model with superclasses is not representative of the original queueing network model unless the classes C_i and C_j corresponding to each superclass have the same service time distribution at each queue for the two different processors. And if this is the case, the processors are not dependent. Thus we cannot guarantee that a queueing network model for dependent processors possesses a product form solution and represents the operation of the processors.

The above reasoning implies that we cannot model synchronization and mutual exclusion, two principal forms of dependency between processors, with queueing networks and expect product form solutions. In addition, it is well known [S1] that product form solutions cannot be expected for queueing network models involving multiple resource possession. Multiple resource possession occurs when a customer at one queue requires simultaneous service at several queues, thus "possessing" the service resources of those queues. An example of multiple resource possession in Concert is a Ringbus memory access. Such an access requires the simultaneous possession of the Multibus and Ringbus. Thus a product form solution cannot be expected if we model Concert as a queueing network model with a queue for the Multibus and a queue for the Ringbus. This is one reason why we have chosen to decompose Concert into separate Multibus and Ringbus models and regarded Ringbus memory accesses as just requiring a different service time at the Multibus queue.

All the dependencies mentioned above can be handled with sufficiently detailed Markov chain models. However such models suffer from a relatively low level of abstraction: the structure of the model is often obscured and one's energy misdirected by the details of Markov state definitions and transitions. Stochastic Petri Nets (SPNs) [M2] allow modeling at a higher level of abstraction than with Markov chains and can easily handle the sort of dependencies mentioned above. A SPN model is less complex, easier to construct, and has a greater likelihood of being correct than an equivalent Markov chain model.

A Petri Net is a set P of places, a set T of transitions, a set α of directed arcs from places to transitions, a set β of directed arcs from transitions to places, and some initial placement of tokens in places (called a marking). Arcs incident on a given transition are called input arcs and the places

from which these arcs emanate are called input places for that transition. Arcs emanating from a given transition are called output arcs and the places at which these arcs terminate are called output places for that transition. A transition is enabled when there is at least one token in each of its input places. After a transition is enabled, it fires immediately, removing one token from each of its input places and adding one token to each of its output places. (There can be more than one token at a place.) A simple Petri Net is illustrated in Figure 2.23. The circles represent places, the bars represent transitions, and the dots represent tokens. See Peterson [P2] for an extensive discussion of Petri Nets and their properties.

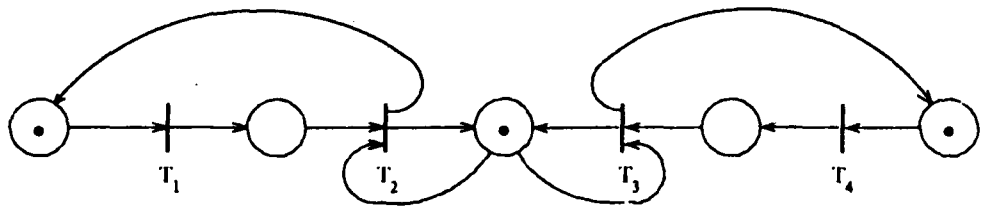


Figure 2.23: A simple Petri net

A Stochastic Petri Net (SPN) is a Petri Net with the following modification. Associated with each transition is a random variable which specifies the interval, called the firing time, between the enabling of that transition and its firing (given that the transition is still enabled at that time). At the instant at which a transition fires - and not before - one token is removed from each of its input places and one token is added to each of its output places. Thus the firing of one transition may cause the disabling of another transition. The probability distribution of the firing time is given and possibly different for each transition. (Petri Nets can also be made stochastic by incorporating probabilistic service times at each place.) With appropriate probability distributions for the transitions T_1, T_2, T_3, and T_4, Figure 2.23 represents a SPN model of a two processor Multibus system. (T_1 represents the processing time of processor 1, T_2 represents the access time of processor 2, T_4 represents the processing time of processor 2, and T_3 represents the access time of processor 2.) More complex SPN models of processors can be developed easily. Performance measures, similar to those derived with our other modeling techniques, can be derived from a SPN. Molloy [M2] has shown that SPNs with exponential firing time distributions are isomorphic to one dimensional Markov chains and thus the performance measures of interest for such SPNs can be determined by their equivalent Markov chains. However, with Molloy's technique relatively small SPNs result in large Markov chains. Such state space explosion makes Molloy's technique unattractive for determining the performance measures of larger SPNs. Wiley [W3] has developed techniques that are more efficient and more general.

2.11 Conclusions

1. The general behaviour of the mean waiting time per request is similar to that depicted in Figure 2.3: the position of the knee and the asymptotic slope depend on $\bar{t}_p$, the mean processing time; $\bar{t}_{aMB}$, the mean Multibus access time; $\bar{t}_r$, the mean recovery time; $\bar{t}_{aRB}$, the mean Ringbus access time; β, the probability of a long word access; and ψ, the probability of a Ringbus access. The exact shape of the waiting time per request versus number of processors curve depends on the probability distributions for the processing, recovery, and access times. Generally, the more "deterministic" these distributions are - i.e. the smaller the variance of the associated random variables - the shallower the knee is. In fact, the mean waiting time per request with deterministic processing, recovery, and access times provides a lower bound on the mean waiting time per request.

2. The mean waiting time per request can be more sensitive to the parameters β and ψ than to the probability distributions for the processing, recovery, and access times. In the cases that we simulated (in section 2.8.1.4), we found that the mean waiting time with various probability distributions was fairly close to that obtained with exponential probability distributions. This suggests that future effort be spent determining appropriate values or ranges of values for the parameters β and ψ and assessing the adequacy of our simple processor model.

3. The assumptions of identical processors and a simple processor model can be removed, as discussed in section 2.10, by expanding our basic queueing network approach. The assumption of time independent behaviour can also be removed, provided the time dependent behaviour can be reasonably approximated by time piece-wise independent behaviour, by expanding the queueing network approach. This approach is trivial in the special case when the overall model can be decomposed into independent submodels for each time scale. Otherwise, this approach is very complicated and probably unreasonably difficult for all but simple models. The assumption of independent processors is the most difficult to remove. In fact, it cannot be removed by any expansion of our queueing network approach (unless one is willing to sacrifice tractability and consider networks without a product form solution). As discussed in section 2.10.4, the behaviour of the Multibus with dependent processors can be modeled with low level Markov chain models, or more preferably, by higher level models such as Stochastic Petri Nets.

4. The performance of the Multibus can be improved by the following:

i) reduce the frequency of long word and Ringbus accesses. Ringbus accesses are especially detrimental to performance because of their extremely long duration, during which all Multibus traffic is blocked. In the actual Concert system, the minimum duration of a Ringbus access is 2.00μsec and the maximum duration is 7×(10×0.200μsec) (the maximum duration for which the required segments can be allocated to other requests) + 2.70μsec = 16.70μsec (assuming no test and set instructions). Most Ringbus accesses will have a

duration somewhere between these two extremes, depending on the processing time, β, and ψ.

Two ways to avoid blocking Multibus traffic when a Ringbus access occurs are to:

a) replace the Multibus by two or more parallel buses, or perhaps just add a private bus for Ringbus accesses, and

b) divide the memory transaction protocol into a memory operation component and an acknowledgment component that occur at separate times between which control of the Multibus may be relinquished to other memory transactions.

Both of these options are costly, although (a) is probably less costly.

ii) decrease the overhead time on non-local memory accesses. Each non-local memory access experiences 100 to 200nsec of delay due to the Multibus arbiter and substantial delays in asserting the BREQ* (Multibus request) signal upon detecting a non-local memory access and in asserting the address and control signals once the BPRN* (Multibus grant) signal is asserted.

iii) reduce the Ringbus access time.

2.12 Future Work Required

1. Evaluate the single processor equivalent model and the Multibus models. Derive appropriate values for the processor model parameters from real programs and compare the performance predicted by the Multibus models with the actual performance.

Ali [A1] has performed some work in this direction. He found excellent agreement between predicted and actual performance of the simple Multibus (no long word or Ringbus accesses) for some artificial programs emulating the simple processor model. For the "real" programs which Ali considered, he found time dependent behaviour to be very important, suggesting that stationary models are inadequate.

2. Improve the processor and Multibus models and develop new ones.

The existing models can be improved to some degree as discussed in section 2.10. However, a better direction in which to proceed is to develop higher level models, such as Stochastic Petri net models. Time and processor dependencies are easier to model at higher levels.

Chapter 3

The Ringbus Model

3.1 Introduction

In this chapter we study the Ringbus subsystem. As discussed in section 1.3.5, we replace each Multibus by a single processor equivalent model. We assume that each Multibus, and thus each single processor equivalent model, is identical in all respects. We also assume that the Ringbus is symmetrical with respect to each Multibus. We make these assumptions so that we can use the abundant symmetry that they imply to simplify considerably the analysis of the Ringbus and the integration of the Multibus and Ringbus models. The treatment in this chapter can be extended easily formally (although not so easily practically) to deal with situations in which these assumptions are not valid. We assume an exponential distribution for the processing time distribution of each single processor equivalent. The reason for this is again to ease analysis. We make no assumptions at this point about the access time distribution; indeed, this distribution is one of the factors for study in this chapter.

The focus of this chapter is the optimum performance of the Ringbus. There are three reasons for this emphasis on the optimum performance. First, the Ringbus is a novel interconnection scheme which has not been studied previously (as far as we know). Thus, knowing the optimum performance of the Ringbus satisfies a natural curiosity. Second, the theoretical maximum improvement in performance of any particular Ringbus design (including the design utilized in Concert) can be determined from the optimum performance of the Ringbus. This theoretical performance improvement is useful in evaluating Ringbus designs. Third, knowledge of the optimum performance of the Ringbus allows the Ringbus to be compared with other interconnection schemes in terms of the optimum performance. Since the Ringbus is a novel interconnection, the optimum performance of the Ringbus is important in establishing the merit of Ringbus-like schemes with other interconnection schemes.

To avoid getting overwhelmed by details or trapped by the small and relatively unimportant differences between various Ringbus designs, we take an abstract view of the Ringbus. This abstract view is as follows. The Ringbus and Ringbus arbiter operate synchronously with an arbiter clock of period c. Requests for the interconnection of source and destination slices arrive from the Multibuses (or in this case the single equivalent processor models of the Multibus) asynchronously with respect to the arbiter clock. On each rising clock edge, the arbiter examines all pending requests and then instantaneously decides which requests should be granted and how the requests should be granted. This decision is implemented immediately so that there is zero delay from the rising edge of the arbiter clock to the time that a segment allocated to a request is used. Once granted, a request lasts exactly some integral number of arbiter clock periods. We assume, without loss of generality, that the duration of a grant (which is what we call a granted request) is encoded in its request rather than determined by the number of clock periods before the request is removed (as it is in the Concert system).† Requests remain pending until they are eventually granted. The Ringbus itself we consider to be just a ring of bus segments under the control of a central arbiter.

The abstract view of the Ringbus given above is really a set of simplifying assumptions. We list the most important of these assumptions below.

1) We ignore the delays of the *RIB circuitry, including the delay to mitigate metastability when* latching the asynchronous request signals from the Multibus.

2) We assume zero arbitration time and zero delay in connecting the bus segments of the Ringbus.

3) We assume grant durations of an integral number of arbiter clock periods.

4) We assume that the minimum time between the termination of a grant of some slice and the next nonnull request from that slice is zero.

In addition, we assume there are no global register accesses.

We term the abstract view of the Ringbus summarized by the above assumptions the isolated Ringbus model. In section 3.9 we discuss the differences between the environment of the isolated Ringbus model (created by these assumptions) and the environment of the Ringbus in the actual Concert system. We also consider the effects these differences have on the performance of the Ringbus. The Multibus-Ringbus interaction, which is simplified by assumptions 1 and 4 above, is

† Since there may be zero time between the termination of a grant and the next nonnull request from a slice in our abstract Ringbus, the arbiter cannot unambiguously differentiate between a continuing grant and a new nonnull request of the same type if the duration of a request is determined by the interval until the request is removed. In the Concert system there is at least one clock period of dead time between successive nonnull requests from the same slice to prevent this ambiguity.

discussed in detail in section 3.9.1 and in section 3.3.2 of Appendix A for the actual Concert system. The Multibus-Ringbus interaction is complicated, detailed, and very dependent on the implementation. This is the reason that we simplified the interaction in our abstract view.

We interpret the Ringbus in a broad sense. We define the Ringbus to be a ring of independent bus segments in which adjacent bus segments may be connected to form longer buses. Associated with each bus segment interconnection point is a slice which is connected to the segments via an access path. All Ringbus accesses originate and terminate at slices. The interconnection of the bus segments occurs in real time under the control of a central arbiter in response to requests originating from slices for paths to other slices. We assume that the arbiter operates in discrete time (although it need not in all cases).

Different Ringbus designs are distinguished by 1) the number of bus segments (which is equal to the number of slices), 2) the access paths between the slices and bus segments, and 3) the arbitration algorithm. In this chapter we only consider Ringbus designs with an even number of slices. In addition, we only consider two different types of access paths: asymmetrical and symmetrical. The Ringbus design utilized in Concert has asymmetrical access paths (as discussed in section 1.2.2.) [See also Figure 3.1.] Hereafter we call this particular Ringbus design - minus the arbitration algorithm - the Asymmetric Ringbus. These asymmetrical access paths impose unnecessary performance limitation. As discussed in section 1.2.2, counterclockwise accesses on the Asymmetric Ringbus require two segments in addition to the segments between the source and destination slices. Symmetrical access paths remove this performance limitation. A Symmetric Ringbus is a Asymmetric Ringbus with symmetrical access paths instead of asymmetrical access paths. Figure 3.1 illustrates the access paths of the Asymmetric Ringbus and the Symmetric Ringbus. We define the Concert Ringbus to be the Ringbus and arbitration algorithm actually used in the Concert system. That is, the Concert Ringbus is a Asymmetric Ringbus with a rotating priority arbitration algorithm† (as discussed in section 1.2.3).

† Anderson [A2] actually calls this arbitration algorithm a rotating priority, full arbitration arbitration algorithm to distinguish it from others he considered during the design of Concert. We will call it simply a rotating priority arbitration algorithm.

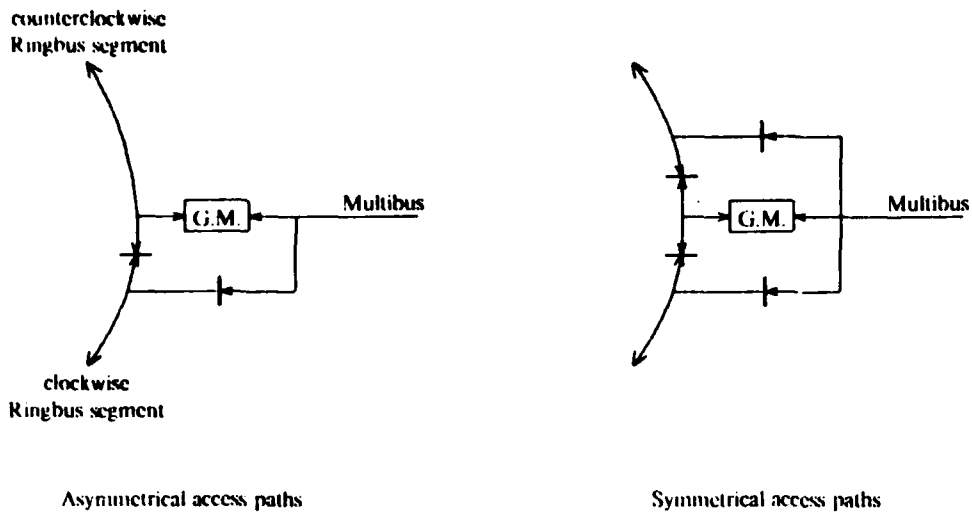


Figure 3.1: Access paths of Asymmetric Ringbus and Symmetric Ringbus

As stated earlier, our chief interest is the optimum performance of the Ringbus. Since the Symmetric Ringbus is a *superset of the Asymmetric Ringbus, the optimum performance of the* Symmetric Ringbus is greater than or equal to that of the Asymmetric Ringbus. For this reason, we concentrate on the optimum performance of the Symmetric Ringbus in this chapter. The Symmetric Ringbus is also easier to analyze since it has more symmetry. In the course of determining the optimum performance we also determine the optimal arbitration algorithm, which is of interest in designing good sub-optimal algorithms.

We briefly consider the optimum performance of the Asymmetric Ringbus for a small number of slices. In addition, we determine the performance of the Concert Ringbus and the performance of the Symmetric Ringbus with the rotating priority arbitration algorithm. A trivial modification to the arbiter in the actual Concert system (which we call the Concert Ringbus arbiter) allows this algorithm to operate with symmetrical access paths. (The additional complexity and circuitry required in the RIB might not be judged as trivial.) The problem from the point of view of the arbiter with symmetrical access paths is that conflicts may now occur at the request destination as well as at the Ringbus segments. Thus the arbiter must arbitrate the destinations as well as the Ringbus segments.

To include this feature, the arbiter just needs to arbitrate for each Ringbus resource - segment or destination - in the same manner in which the arbitration proceeded for the segments in the Concert Ringbus arbiter (see section 1.2.3). The first step is to determine all the Ringbus resources required for each request. As in the Concert Ringbus arbiter, requests would be granted

only in the direction requiring the smallest number of segments, with ties being broken in preference of the clockwise direction. Finally, a request would be granted when it has been granted all the resources that it requires.

A logic diagram for this new arbiter is shown in Figure 3.2. The part count has doubled because we now have double the number of Ringbus resources to arbitrate. However, the size of the parts required is the same. The number of parts is proportional to the number of resources and the size of the parts is exponential to the number of sources.

This new arbiter design, which evidently was overlooked during the design of the Concert system, would result in superior or equivalent performance in all cases. (It certainly cannot result in inferior performance since its functionality is a superset of the other's.)

In section 3.2 we formulate the Ringbus as a discrete time probabilistic model. Time is quantized into discrete intervals, called rounds, which are equal to and synchronous with the arbiter clock period. The performance metric of the Ringbus model is the throughput in terms of the average number of grants completed per round. The optimum performance of the Ringbus model is formulated as a Markovian decision problem.

In sections 3.3 and 3.4 we investigate the optimal arbiter for a Ringbus of four slices. Section 3.3 covers grant durations of one round and section 3.4 covers grant durations greater than one round for two special cases. These special cases are deterministic grant durations and geometrically distributed grant durations.

In section 3.5 we investigate the optimal arbiter for a Ringbus of six slices and develop a number of bounds on the optimum throughput.

Sections 3.6 and 3.7 consider the Ringbus with eight and more slices. Since the computational requirements for these cases exceeds the available resources, we just discuss the general characteristics of the optimum throughput in section 3.6 and the optimum throughput for some special cases in section 3.7.

In section 3.8, we compare the performance of the optimum arbiter algorithms and the rotating priority arbiter algorithm for the Concert and Symmetric Ringbuses.

Finally, in section 3.9 we discuss some of the differences between our abstract Ringbus model and the Ringbus utilized in Concert. We consider the effect that these differences have on performance. The last part of this section develops the hooks for the integration of the isolated Ringbus model with the Multibus model.

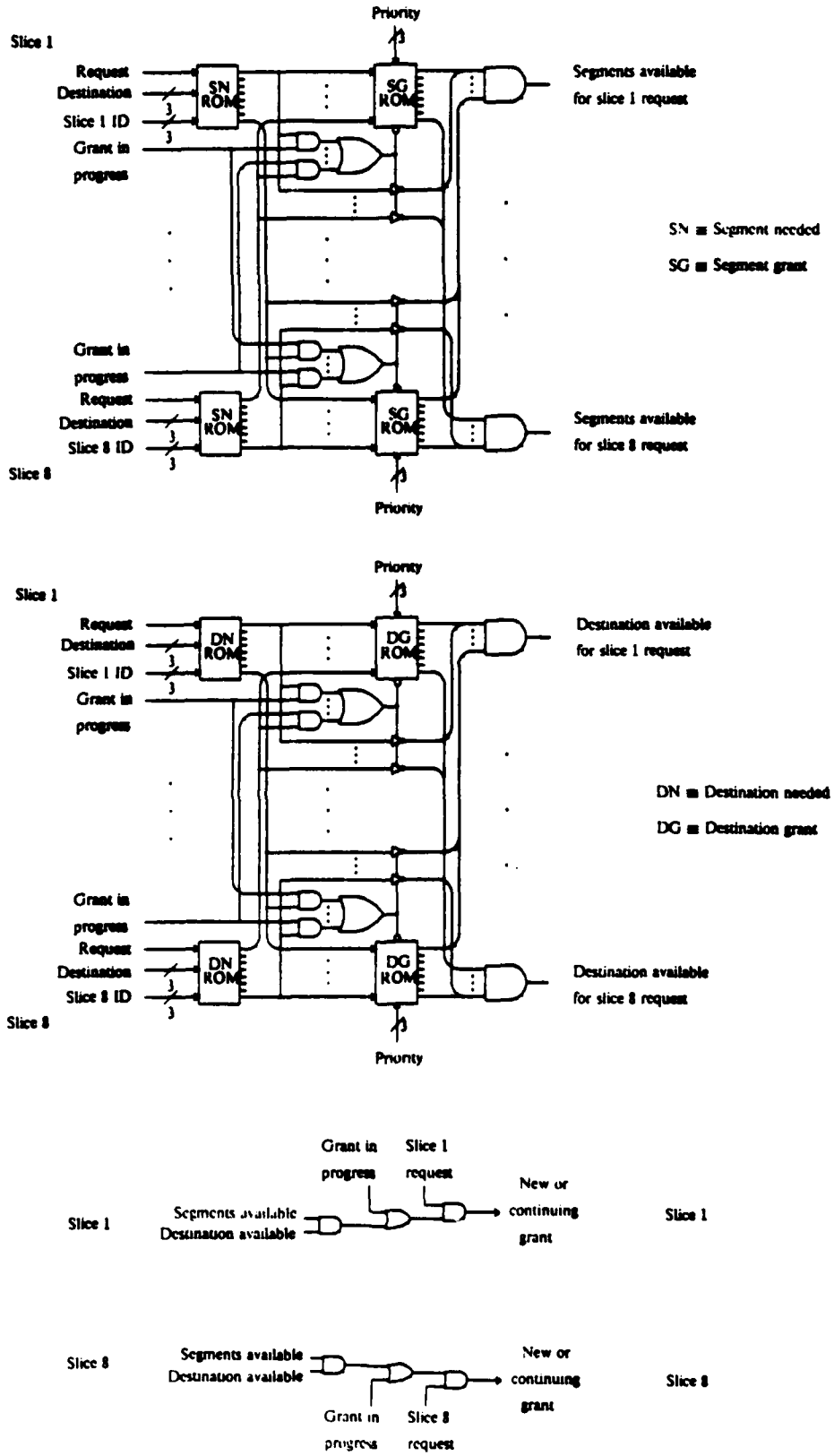


Figure 3.2: Logic diagram of Symmetric Ringbus arbiter

3.2 Ringbus Model Formulation

From the single processor equivalent model of the Multibus, we know that if a Ringbus access occurs in some round then with probability p_i^{MBeqv} its destination is i slices around the Ringbus from the source slice ($i = -(S/2-1), \cdots, -1, 1, 2, \cdots, S/2$). Negative values of i indicate the counterclockwise direction around the Ringbus and positive values indicate the clockwise direction. Note that this probability distribution of requests is independent of the source slice. This is a consequence of our assumption that all Multibus models, and hence all the single processor equivalent model of the Multibus, are identical. Since we assumed an exponential distribution for the processing time distribution of the single processor equivalent model of the Multibus, the probability that the next request at a slice arrives in the ith round after the end of the previous grant at that slice is a constant independent of i. In other words, the number of rounds between the end of a grant and the next request at that same slice (i.e. the discretized processing time of the single equivalent processor model) is a geometric random variable. Because of the memoryless property of a geometric random variable, we can exclude from the state description any information on the number of rounds waited so far for a request to arrive at a slice. Thus the assumption of an exponential distribution for the processing time of the single processor equivalent model simplifies not only the integration of the Multibus and Ringbus models but also the analysis of the Ringbus model.

In each round the arbiter must decide which subset of the current requests to grant based on past and present information only. The arbiter is thus a causal, discrete time decision maker. Decisions are subject to the following constraints:

1. All segments required by a request must be connected as required before or at the same time that the request is granted.
2. Each segment is used for no more than one grant in a round.
3. All segments required by a grant remain connected and allocated for the exclusive use of that grant for the entire duration of the grant.
4. Every pending request eventually gets granted i.e. each request has a bounded waiting time. (This requires a bounded Ringbus access time. In the Concert system each Ringbus access represents a single memory transaction - read, write, or read-modify-write - and the duration of each such transaction is bounded by the Ringbus timeout period.†)

Without loss of generality, we consider the segments referred to in Constraint 1 to be connected at the time that a request is granted. This is in fact how the Ringbus operates in Concert.

† If the addressed memory location at the destination RIB does not respond with an acknowledgment within a given time interval, the destination RIB sends a signal to the source RIB which aborts the Ringbus access

We make the following three simplifications in our formulation of the Ringbus model:

1. We exclude from the state description any information on the duration that a request waits before being granted. This waiting time information is irrelevant when

   i) in modeling the performance of the only non-optimal arbiter algorithm considered in this chapter - the rotating priority arbiter algorithm, and

   ii) determining the optimum performance of the Ringbus without Constraint 4.

   The waiting time information is irrelevant in case i) because the rotating priority algorithm does not utilize this information. We have not presented sufficient machinery at this point to show that the request waiting time information is irrelevant in case ii). In fact, we we have not even completed the formulation of the Ringbus model. Therefore we relegate a precise statement and proof of the irrelevance of this history information, which we call Theorem 3.1, to Appendix B and encourage the reader to examine this theorem after completing subsection 3.2.1.

2. We ignore Constraint 4 when pursuing the optimum performance of the Ringbus. Our reasons are as follows. First, by ignoring Constraint 4, request waiting time information may be excluded from the state description (as justified by Theorem 3.1), thus permitting the analysis to be greatly simplified. Second, ignoring Constraint 4 removes the effect of the maximum permissible waiting time on the the optimum performance so that the optimum performance obtained is the inherent optimum performance of the Ringbus architecture. If the maximum permissible waiting time is sufficiently large, Constraint 4 has negligible effect. If it is sufficiently small (such as equal to its minimum value of $(S-1)D$ where S is the number of slices and D is the maximum duration of an access), Constraint 4 has an enormous effect on the performance. In fact, with a maximum permissible waiting time of $(S-1)D$, the arbiter algorithm must impose some sort of strict priority ordering on requests. Third, any arbiter algorithm can easily be modified to ensure bounded waiting times. Such a modification may, of course, result in a degradation of performance dependent on the maximum permissible waiting time.

   Note that assuming a large enough maximum permissible waiting time is essentially equivalent to ignoring Constraint 4. We prefer to think of ignoring Constraint 4 as assuming such a large enough maximum waiting time.

3. We limit the duration of a grant to have one of the following two simple forms:

   i) a constant duration of d rounds where $d = 1,2,3,$ or 4.

   ii) a geometric probability distribution i.e. the duration is d rounds where d is a random variable with a (memoryless) geometric distribution.

3.2.1 Markovian Decision Formulation

Let the state of the Ringbus (ignoring request waiting times as discussed previously) at the beginning of each round be described by

$$(r_1,d_1;r_2,d_2;\cdots;r_i,d_i;\cdots;r_S,d_S)$$

where r_i denotes the destination of the request at slice i and d_i indicates the duration for which the request has been granted so far.

We express the destination of a request as the number of slices the destination slice is around the Ringbus relative to the source slice. We use positive numbers to indicate the clockwise direction from the source and negative numbers to indicate the counterclockwise direction from the source. Thus $r_i = 2$ indicates a request to the slice two slices along the Ringbus in the clockwise direction from the source slice, and $r_i = -2$ indicates a request to the slice two slices along the Ringbus in the counterclockwise direction from the source slice.

We do not use $r_i = 0$ to indicate a request from slice i to slice i. We assumed earlier that there are no global register accesses, hence such requests do not occur. Instead, we use $r_i = 0$ to indicate that slice i is not requesting a Ringbus destination. We call this absence of a request a null request. The arbiter treats a null request just like a genuine request except that 1) a null request is always granted immediately when it occurs (since there are no resources to be granted for a null request) and 2) a null request always has a duration of only one round. Any two consecutive genuine requests at a slice are separated by some number (possibly zero) of null requests proportional to the processing time between those genuine requests.

A request from slice i is pending (i.e. not yet granted) if and only if $d_i = 0$. The duration d_i is increased by one for each round that the request remains granted. We express the destination of any pending request in terms of the smallest number of slices - either clockwise or counterclockwise direction - the destination slice is relative to the source slice. A tie in the number of slices in each direction is broken in favour of the clockwise direction. Thus for any pending request if the source slice is i and the destination slice is $j \neq i$, then

$$r_i = \begin{cases} x, & x \leq S/2 \\ x - S, & S/2 < x \end{cases}$$

where $x = (j - i) \bmod S$.

A request may be granted in either clockwise or counterclockwise direction. We express the destination of a request once the request is granted in terms of the direction in which the request was granted. Thus if a request is granted from slice i to slice $j \neq i$,

$$r_i = \begin{cases} x, & \text{if granted in clockwise direction} \\ x - S, & \text{if granted in counterclockwise direction} \end{cases}$$

where $x = (j - i) \bmod S$.

Therefore once a request is granted, we use r_i to indicate which segments have been allocated to that request. For the Symmetrical Ringbus, the mapping from r_i to the segments is especially easy: $|r_i|$ indicates the number of segments allocated beginning from slice i and the sign of r_i indicates the direction around the Ringbus in which these segments are allocated. For the Asymmetric Ringbus, the mapping is the same except that two additional segments are required for requests granted in the counterclockwise direction: the segment most immediately clockwise of the source slice and the segment most immediately counterclockwise of the destination slice i. (Thus there is only one direction to grant requests from a slice to its immediate clockwise neighbour i.e. from slice i to slice $(i \bmod S) + 1$.)

An example of the definition of r_i is illustrated in Figure 3.3.

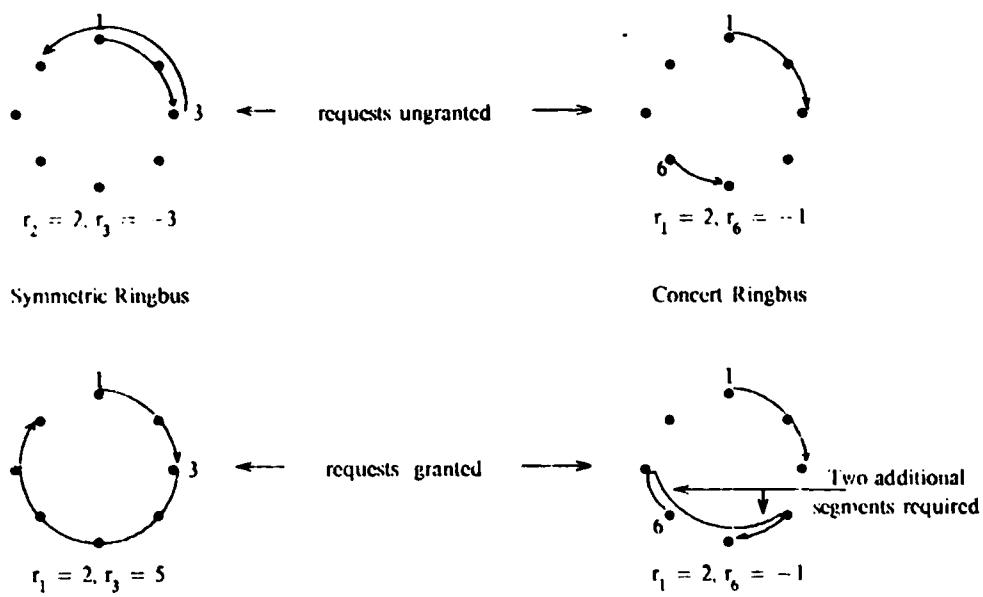


Figure 3.3: Examples of r_i

In some cases the state description simplifies. If all grants have a constant duration of one round, then all the d_i can be eliminated from the state description since a new request - either genuine or null - always replaces a request once it has been granted. If all grants of genuine requests have a duration with a geometric distribution, then we only need a binary variable for d_i. As before, d_i indicates a pending request. $d_i = 1$ indicates that a request has been granted for one or more rounds. The exact duration of the grant in this case is irrelevant since the geometric distribution of the duration is memoryless (i.e. independent of how long the request has been granted

so far).

When a grant at a slice terminates (after one round for a null request and one or more rounds for a nonnull request), a new request arrives at that slice at the beginning of the following round. We denote by p_i the probability that a new request is for a destination i slices along the Ringbus from the source slice, $i=-(S/2-1), \cdots, -1, 1, 2, \cdots, S/2$. As before, negative values of i indicate the counterclockwise direction around the Ringbus from the source and positive values of i indicate the clockwise direction. We denote by p_0 the probability that a new request is a null request. Thus $p_i = \dfrac{p_i^{MBeqv}}{1-p_0}$ for $i \neq 0$.

Given some current state, the next state of the Ringbus depends on the present state, the decisions made in the present state, and the new requests that arrive in the present round. The states of the Ringbus thus comprise a discrete time Markov chain. The state transition probabilities depend on the state and the decision made in that state. Note that going from the present state to the next state has two parts - a deterministic part and a random part. The deterministic part is determined by the decision in the present state: any requests ungranted in the present state or corresponding to grants still in progress in the present state must appear in the next state. The random part is determined by the new requests which arrive to replace the grants which terminated in the present state.

For convenience, we number the states with consecutive integers starting from 1 and we number the possible decisions in each state with consecutive integers starting from 1. We denote the one-step probability from state i to state j by p_{ij}^d where d indicates the decision made in state i. Denote the decision made in state i by $d(i)$ and let $\mathbf{D}=[d(1),d(2),d(3), \cdots]$. We call the decision vector $\mathbf{D}$ a policy: it specifies the decision made in each state, and thus completely specifies the operation of the arbiter. We consider only stationary policies, i.e. policies which are independent of time. In addition, we consider only policies in which there is at least one new grant or grant in progress in each state except for the state with $r_i=0$ for all i. (A new grant is a grant which has a duration of zero so far; it has been granted for the first time in that state. A grant in progress is a grant which has a duration so far of one or more rounds; it has been granted for the first time in some previous state.) We assume that p_i is nonzero for all $i=-(S/2-1), \cdots -1,0,1, \cdots, S/2$. The above restriction on admissible policies and this assumption of nonzero probabilities ensures the following:

1) All states in the Markov chain communicate - i.e. the n step transition probability from state i to state j is nonzero for all i and j and some $n \geq 1$. The Markov chain thus forms a single closed class.

2) The Markov chain is periodic.

These two conditions ensure that the finite Markov chain has a stationary steady-state (Theorem 2 p.29 of Kleinrock [K3]). Denote the steady-state probability of being in state i under policy $\mathbf{D}$ by $\pi_i^{\mathbf{D}}$. The $\pi_i^{\mathbf{D}}$ are given by

$$\pi_i^{\mathbf{D}} = \sum_j \pi_j^{\mathbf{D}} p_{ji}^{d(i)} \quad \text{and} \quad \sum_i \pi_i^{\mathbf{D}} = 1 \tag{3.1}$$

We call the number of new grants in state i under decision $d(i)$, the reward, which we denote by $q_i^{d(i)}$. The average number of new grants per round under policy $\mathbf{D}$ is

$$g^{\mathbf{D}} = \sum_i q_i^{d(i)} \pi_i^{\mathbf{D}} \tag{3.2}$$

where $\mathbf{D} = [d(1), d(2), \cdots]$. The average number of new grants per round is the throughput of the Ringbus. Our objective is to find the maximum throughput, g^{opt}, and the corresponding policy $\mathbf{D}$, subject to given constraints on the decisions and for given probabilities.

The constraints on the decisions fall into three classes which we term logical, topological, and theoretical. The logical constraints, which we discussed at the beginning of section 3.2, impose certain basic conditions on the Ringbus segments independent of the arbiter algorithm and Ringbus design. The topological constraints impose the mapping from a request to the segments required for that request. Different Ringbus topologies, and in particular different access paths, can be expressed in terms of different request to segment mappings. The Asymmetric Ringbus and the Symmetric Ringbus differ only in their mapping of counterclockwise requests to segments: the Asymmetric Ringbus requires two more segments than the Symmetric Ringbus. The theoretical constraints ensure smooth application of the Markovian decision formulation. The limitation to stationary policies is of no concern since any real arbiter implementation would likely operate independent of time anyway. Likewise, the limitation to policies with at least one grant in every state (except for the state with $r_i = 0$ for all i) is of no concern since any optimal arbiter would obviously have at least one grant per round wherever possible. Without this limitation, the Markovian decision problem might have multiple chains and transient states, which complicate the analysis.

The optimal throughput and corresponding policy of the Markovian decision model of the Ringbus can be solved using Howard's policy-iteration method [H4]. We develop some preliminary results following Howard [H4], for future use and then we present Howard's algorithm.

Suppose we ran our Markov chain model of the Ringbus with rewards for n rounds under some policy $\mathbf{D}$. Let $V_i^{\mathbf{D}}(n)$ denote the total expected reward (i.e. total number of new grants) accumulated over the n rounds that we start in state i. $V_i^{\mathbf{D}}(n)$ obeys the recurrence relation:

$$V_i^{\mathbf{D}}(n) = q_i^{d(i)} + \sum_j p_{ij}^{d(i)} V_j^{\mathbf{D}}(n-1), \quad i \in I, \quad n \geq 1 \tag{3.3}$$

where I is the set of all states. Howard has shown that $V_i^{\mathbf{D}}(n)$ has the asymptotic form

$$V_i^{\mathbf{D}}(n) = ng^{\mathbf{D}} + v_i^{\mathbf{D}}, \quad \text{as} \quad n \to \infty \tag{3.4}$$

$v_i^{\mathbf{D}}$ represents the value of starting in state i: $v_i^{\mathbf{D}} - v_j^{\mathbf{D}}$, $i \neq j$, is the difference in the long run expected reward due to starting in state i rather than state j. Substituting equation 3.4 into equation 3.3, we obtain

$$g^{\mathbf{D}} + v_i^{\mathbf{D}} = q_i^{d(i)} + \sum_j p_{ij}^{d(i)} v_j^{\mathbf{D}}. \tag{3.5}$$

If there are N states, equation 3.5 represents N simultaneous equations in $N+1$ unknowns. We rectify this situation by subtracting $v_1^{\mathbf{D}}$ from both sides of equation 3.5 and regarding $g^{\mathbf{D}}$ and the $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ as the unknowns:

$$g^{\mathbf{D}} + (v_i^{\mathbf{D}} - v_1^{\mathbf{D}}) = q_i^{d(i)} + \sum_j p_{ij}^{d(i)} (v_j^{\mathbf{D}} - v_1^{\mathbf{D}}). \tag{3.6}$$

We call these $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ the relative values. We can solve equation 3.6 for $g^{\mathbf{D}}$ and the relative values. Note that we now have an equivalent form for $g^{\mathbf{D}}$:

$$g^{\mathbf{D}} = q_1^{d(1)} + \sum_j p_{1j}^{d(1)} (v_j^{\mathbf{D}} - v_1^{\mathbf{D}}) \tag{3.7}$$

Howard's policy iteration algorithm is the following:

1) Start with some policy $\mathbf{D}$.

2) **Value Determination:** Use the $p_{ij}^{d(i)}$ and $q_i^{d(i)}$ for a given policy $\mathbf{D}$ to solve

$$g^{\mathbf{D}} + (v_i^{\mathbf{D}} - v_1^{\mathbf{D}}) = q_i^{d(i)} + \sum_j p_{ij}^{d(i)} (v_j^{\mathbf{D}} - v_1^{\mathbf{D}}) \tag{3.8}$$

for $g^{\mathbf{D}}$ and the relative values $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$.

3) **Policy Improvement:** For each state i, use the relative values $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ from the previous policy and determine the value or values of k which satisfy:

$$\max_k \left(q_i^k + \sum_j p_{ij}^k (v_j^{\mathbf{D}} - v_1^{\mathbf{D}}) \right) \tag{3.9}$$

If a unique value of k satisfies equation 3.9 then set $d^*(i) = k$. If two or more values of k satisfy equation 3.9 then either one such value of k is $d(i)$ or no such value of k is $d(i)$. In the former case, set $d^*(i) = d(i)$ and in the latter case set $d^*(i)$ equal to an arbitrarily chosen value of k satisfying equation 3.9. The new decision in state i is $d^*(i)$.

4) If policy $\mathbf{D}^*$ is the same as policy $\mathbf{D}$ (i.e. if $d^*(i) = d(i)$ for all i), then stop: $\mathbf{D}^*$ is the optimal policy and $g^{\mathbf{D}^*}$ is the optimal average reward per round. If policy $\mathbf{D}^*$ is not the same as policy

D, then set $D = D^*$ and go to 2.

With precise arithmetic, g^D increases monotonically on each iteration and Howard's algorithm terminates in a finite number of iterations [H4]. However, truncation errors can cause indefinite cycling of the algorithm in a machine implementation.

3.3 Optimal Arbiter for Four Slices and Grant Duration of One Round

In this section we investigate the optimal arbiter for the Symmetrical Ringbus with four slices and a grant duration of one round. In this case the state description is

$$(r_1, r_2, r_3, r_4)$$

where $r_i = -1, 0, 1,$ or 2; $i = 1, 2, 3, 4$. We assume that the request probabilities are symmetrical with respect to the direction around the Ringbus, i.e. $p_1 = p_{-1}$. There are 256 states in this state description. However, this number can be reduced by taking advantage of the abundant symmetry present. There are two types of symmetry present, which we term rotational and flip. These symmetry types are most conveniently viewed geometrically. Imagine the Ringbus represented by four nodes (each representing a slice) connected by arcs (each representing a bus segment) to form a planar diamond shape which has three axes of symmetry: one perpendicular to the plane and two in the plane of the diamond. Rotational symmetry refers to the symmetry about the axis perpendicular to the plane of the Ringbus. Flip symmetry refers to the symmetry about one of the axes in the plane of the Ringbus. Because of the rotational symmetry it does not matter which axis in the plane is chosen for the flip symmetry axis. An example of each symmetry type is illustrated in Figure 3.4.

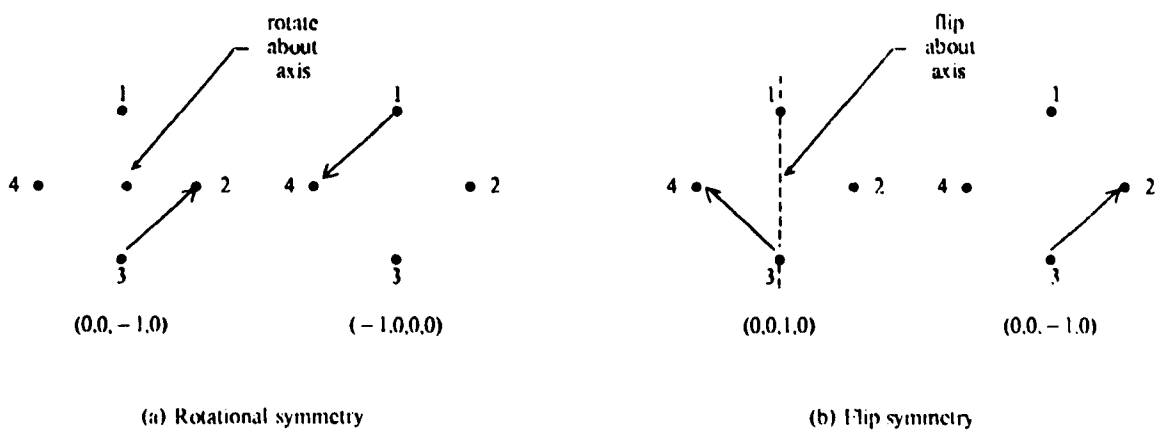


Figure 3.4: Rotational and flip symmetry

Since the request probabilities are identical for each slice and symmetrical with respect to the direction around the Ringbus, by employing both rotational and flip symmetry all eight states $(\pm 1,0,0,0)$, $(0,\pm 1,0,0)$, $(0,0,\pm 1,0)$, $(0,0,0,\pm 1)$ can be seen to be equivalent to $(-1,0,0,0)$. Thus we can replace these eight states by a single equivalent state $(-1,0,0,0)$. By extracting all available symmetry, we eventually end up with a total of 43 states. These states are listed in Table 3.1 along

with the number of original states which reduced to each equivalent state.

State Number	Equivalent State	Reduction Factor
1	0 0 0 0	1
2	-1 0 0 0	8
3	2 0 0 0	4
4	-1 -1 0 0	8
5	-1 0 -1 0	4
6	-1 0 1 0	4
7	-1 0 2 0	8
8	-1 1 0 0	4
9	-1 2 0 0	8
10	1 -1 0 0	4
11	1 2 0 0	8
12	2 2 0 0	4
13	2 0 2 0	2
14	-1 -1 -1 0	8
15	-1 -1 1 0	8
16	-1 -1 2 0	8
17	-1 1 -1 0	8
18	-1 2 -1 0	8
19	-1 1 2 0	8
20	-1 2 1 0	4
21	-1 2 2 0	8
22	1 -1 -1 0	8
23	1 1 2 0	8
24	1 2 -1 0	4
25	1 -1 2 0	8
26	1 2 2 0	8
27	2 -1 2 0	8
28	2 2 2 0	4
29	-1 -1 -1 -1	2
30	-1 -1 -1 1	8
31	-1 -1 -1 2	8
32	-1 -1 1 1	4
33	-1 -1 1 2	8
34	-1 -1 2 1	8
35	-1 -1 2 2	8
36	-1 1 -1 1	2
37	-1 1 -1 2	8
38	-1 1 2 2	4
39	-1 2 -1 2	4
40	-1 2 1 2	4
41	-1 2 2 1	4
42	-1 2 2 2	8
43	2 2 2 2	1

Table 3.1: States After Symmetry Extraction

The optimal arbiter problem can be expressed as a Markovian decision problem based on these 43 states. We number the states as indicated in Table 3.1 and solve this problem using Howard's algorithm [114]. Figure 3.5 shows the gain (i.e. the mean number of grants per round) for various values of p_1 and p_2 ($p_0 = 1 - 2p_1 - p_2$).

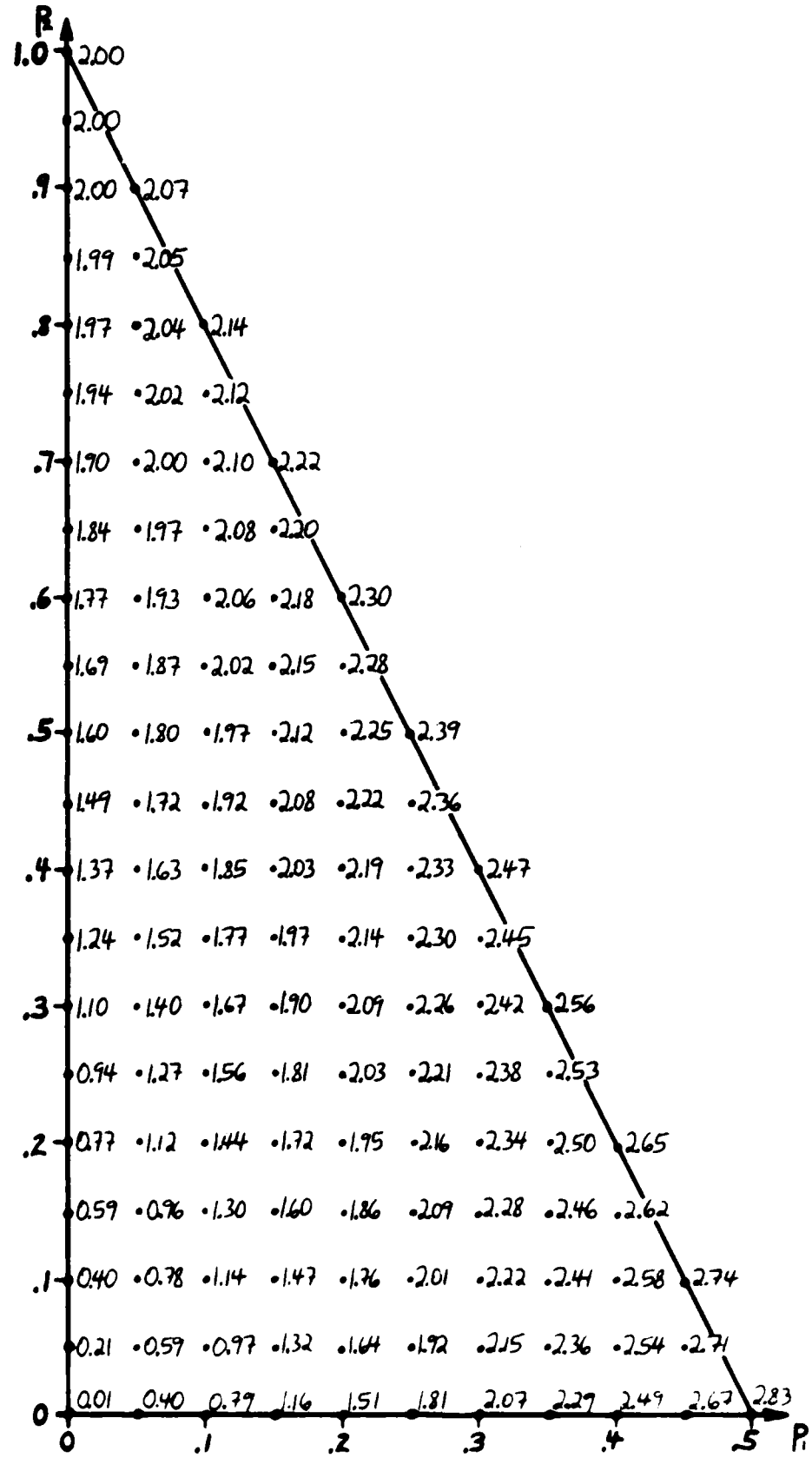


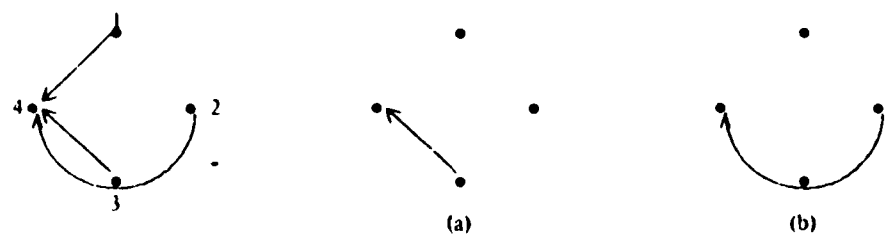
Figure 3.5: Optimum average number of grants per round for Symmetric Ringbus with four slices and one round grant duration

Regardless of the probabilities p_1 and p_2, the optimum decision rule in all states consists of the following two steps:

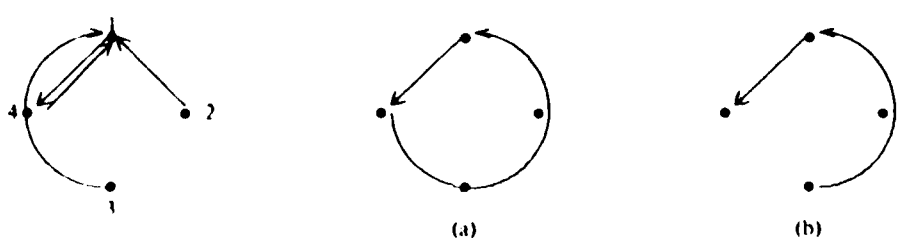
1. Consider only the request subsets for each state that have the greatest number of requests. This amounts to maximizing the immediate reward in each state.
2. Decide which of the request subsets with maximum immediate reward to grant. (This is trivial if there is only one such subset.)

For all states except 20, 34, 38, and 40, and regardless of the probabilities p_1 and p_2, the request subset chosen in step 2 of the decision rule is the one that has the most requests of the longest length - i.e. of length 2 (where we define length to be the number of segments required).

For states 20, 34, 38, and 40, the request subset chosen in step 2 of the decision rule depends on the probabilities p_1 and p_2. States 20, 34, and 40 each have two request subsets with maximum immediate reward as shown in the diagrams in Figure 3.6.



State 20 Maximum reward request subsets



State 34

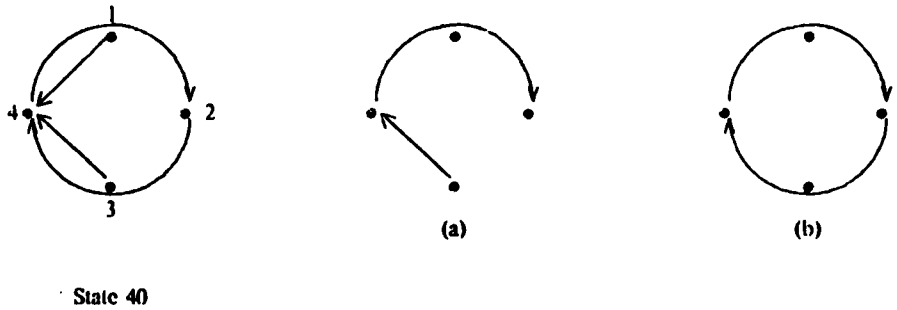


Figure 3.6: Some possible decisions in states 20, 34, and 40

Two sets are associated with each possible decision in a state: a grant set and a leftover set. For a particular state and a particular decision, the grant set consists of all the requests that are granted and null request for each of the ungranted requests. The leftover set consists of all the requests not granted and null requests for each of the granted requests. For example, if request subset (a) is granted in state 20 (see Figure 3.6) then the grant set is (0,0,1,0) and the leftover set is (0,2,0, 1): if request subset (b) is granted, the grant set is (0,2,0,0) and the leftover set is (0,0,1, 1). We can write $R = G_d + L_d$ where R, G_d, and L_d denote the request, grant, and leftover sets respectively, $+$ denotes element-wise addition, and the subscript d indicates that this decomposition of R depends on the decision.

The leftover sets associated with request subsets (a) and (b) in Figure 3.6 are the same for each of the states 20, 34, and 40 (using rotational symmetry for state 34). Thus the decisions in these three states amount to the same decision: should the leftover set be (a) or (b)? (See Figure 3.7.)

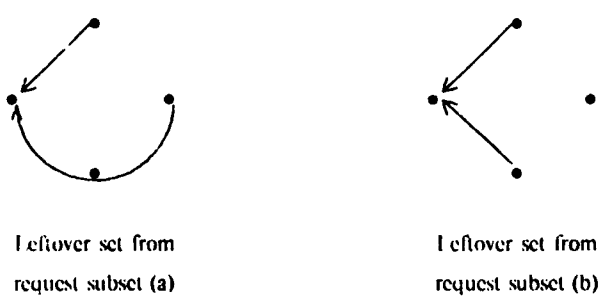


Figure 3.7: Leftover sets associated with request subsets (a) and (b) for each of the three states in Figure 3.6

Of course the decisions in many other groups of states other than 20, 34, and 40 are related through their leftover states. The Markovian decision problem can in fact be formulated in terms of leftover sets rather request sets. Assuming that at least one request is granted in every request set, the number of states required can be reduced by this alternate formulation. However, the transition probabilities are more difficult to determine and the problem structure is less intuitive in this alternate formulation.

State 38 also has two request subsets with maximum immediate reward. These two request subsets and their associated leftover sets are shown in Figure 3.8.

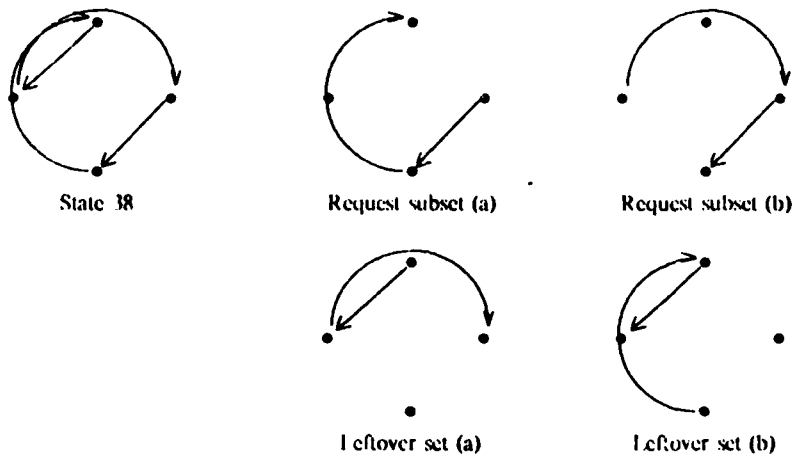


Figure 3.8: Some possible decisions in state 38

Notice the subtle difference between leftover states (a) and (b) in Figure 3.8.

The regions over which request subsets (a) and (b) of Figures 3.6 and 3.8 comprise optimal decisions are shown in Figure 3.9.

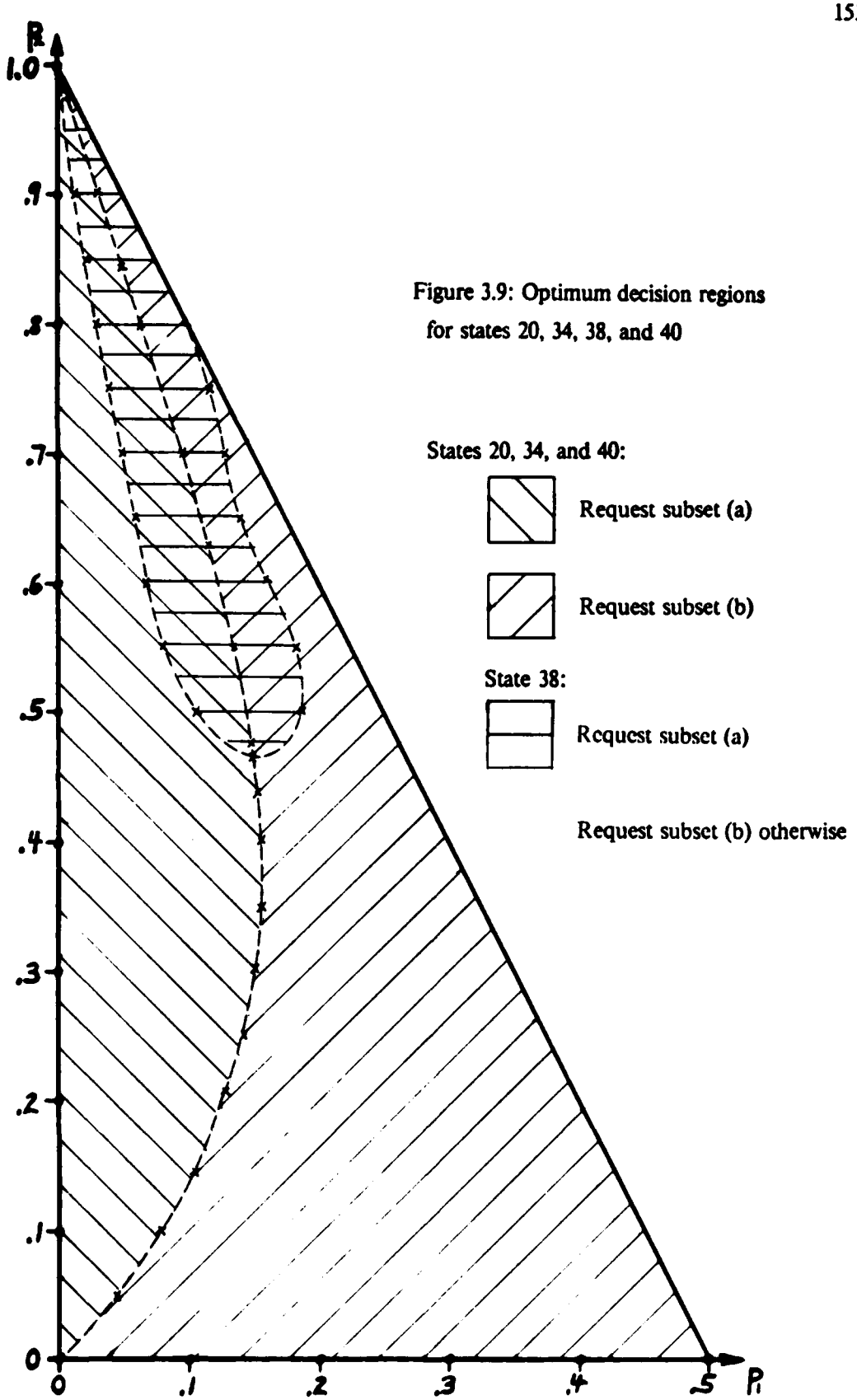


Figure 3.9: Optimum decision regions for states 20, 34, 38, and 40

To the right of the line delineating request subset (a) and (b) for states 20, 34, and 40, step 2 of the decision rule is the same as that mentioned earlier for all the other states: grant the request subset that has the most requests of length 2. In other words, leftover set (b) (of Figure 3.7) is a better choice than leftover set (a) for p_1 and p_2 to the right of the line in Figure 3.9.

We now investigate the regions over which request subsets (a) and (b) for states 20, 34, and 40 are optimal (assuming optimal decisions in all other states). Of course the exact regions over which each of these request subsets is optimal can be computed by applying Howard's policy iteration algorithm. However, the policy iteration yields the optimal decision for only a single point and thus the extent of the regions must be determined by the behaviour at many sample points. This is, in fact, the manner in which the regions shown in Figure 3.9 were established. An analytical form for the boundaries of the regions would be much more useful, but such a form seems intractable. Instead, we consider an approximation.

The basic idea is to approximate the relative value (i.e. $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$) of a state i by the immediate reward, $q_i^{d(i)}$, in that state. First we number the states as listed in Table 3.1. Since there are no genuine requests in state 1, the only possible decision is to grant all the null requests. The immediate reward, q_1, is thus zero. The transition probability, p_{ij}, is simply the probability of the requests arriving that constitute state j. For example, if the transition probability from state 1 to symmetry state 19 (− 1,1,2,0) is $p_{1,19} = 8p_0p_1^2p_2$. (There are 8 ways to go from state 1 to the symmetry state (− 1,1,2,0) - this is the reduction number listed in Table 3.1).

Equation 3.7 thus reduces to

$$g^{\mathbf{D}} = \sum_j p_{1j}(v_j^{\mathbf{D}} - v_1^{\mathbf{D}}). \tag{3.10}$$

(We drop the superscript $d(1)$ on p_{1j} since there is only one possible decision in state 1.) Substituting equation 3.10 into equation 3.6 yields:

$$v_i^{\mathbf{D}} - v_1^{\mathbf{D}} = q_i^{d(i)} + \sum_j (p_{ij}^{d(i)} - p_{1j})(v_j^{\mathbf{D}} - v_1^{\mathbf{D}}). \tag{3.11}$$

Now consider $V_i^{D}(n)$ and the recurrence relation expressed by equation 3.3. Let $V_i^{\mathbf{D}}(0) = 0$ for all i. The difference $V_i^{\mathbf{D}}(n) - V_1^{\mathbf{D}}(n)$ is

$$\begin{aligned} V_i^{\mathbf{D}}(n) - V_1^{\mathbf{D}}(n) &= q_i^{d(i)} + \sum_j (p_{ij}^{d(i)} - p_{1j})(q_j^{d(j)} + \sum_k p_{jk}^{d(j)}(q_k^{d(k)} + \sum_l p_{kl}^{d(k)}(q_l^{d(l)} + \cdots + \sum_s p_{rs}^{d(r)} q_s^{d(s)}))) \\ &= q_i^{d(i)} + \sum_j (p_{ij}^{d(i)} - p_{1j}) q_j^{d(j)} + \sum_j \sum_k (p_{ij}^{d(i)} - p_{1j}) p_{jk}^{d(j)} q_k^{d(k)} + \cdots \\ &+ \sum_j \sum_k \cdots \sum_s (p_{ij}^{d(i)} - p_{1j}) p_{jk}^{d(j)} \cdots p_{rs}^{d(r)} q_s^{d(s)} \end{aligned} \tag{3.12}$$

$$= q_i^{d(i)} + \sum_{m=0}^{n-2} \sum_j \sum_k (p_{ij}^{d(i)} - p_{1j}) \varphi_{jk}(m)^{\mathbf{D}} q_k^{d(k)}$$

where $\varphi_{jk}(m)^{\mathbf{D}}$ is the m step transition probability from state j to state k: $\varphi_{jk}^{\mathbf{D}}(0) = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$, $\varphi_{jk}^{\mathbf{D}}(m+1) = \sum_l p_{jl}^{d(j)} \varphi_{lk}^{\mathbf{D}}(m)$. As $n \to \infty$, $V_i^{\mathbf{D}}(n) - V_1^{\mathbf{D}}(n) \to v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ by equation 3.4 and thus

$$v_i^{\mathbf{D}} - v_1^{\mathbf{D}} = q_i^{d(i)} + \sum_{m=0}^{\infty} \sum_j (p_{ij}^{d(i)} - p_{1j}) \sum_k \varphi_{jk}(m)^{\mathbf{D}} q_k^{d(k)} \tag{3.13}$$

We now have two alternate formulations for the relative values $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$: equation (3.11) and equation (3.13). Equation 3.11 provides a way to calculate the relative values and equation 3.13 allows an interpretation of the relative values. We see from equation 3.13 that $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ is the infinite sum of probabilistically weighted rewards. Rewritten as

$$v_i^{\mathbf{D}} - v_1^{\mathbf{D}} = q_i^{d(i)} - 0 + \sum_{m=0}^{\infty} \sum_j \sum_k p_{ij}^{d(i)} \varphi_{jk}(m)^{\mathbf{D}} q_k^{d(k)} - \sum_{m=0}^{\infty} \sum_j \sum_k p_{1j} \varphi_{jk}(m)^{\mathbf{D}} q_k^{d(k)}, \tag{3.14}$$

our earlier interpretation of $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ as the difference in the average total reward starting in state i relative to stating in state 1 is obvious. (Equation 3.14 can be generalized for $v_i^{\mathbf{D}} - v_j^{\mathbf{D}}$.)

Equation 3.13 suggests that $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ can be computed to arbitrary accuracy simply by summing enough of the terms on the right hand side. One way to approximate $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$, which we now pursue, is by the first term of its infinite series expansion, i.e. $v_i^{\mathbf{D}} - v_1^{\mathbf{D}} \approx q_i^{d(i)}$. This approximation has the merit of avoiding any computation with the transition probabilities. Of course some accuracy is lost in this simple approximation. However this merit is very important when the number of states is so large that it is a great deal of work to compute all the transition probabilities. (Such is the case for six and eight slices as discussed in the sequel.)

In some cases the approximation $v_i^{\mathbf{D}} - v_1^{\mathbf{D}} \approx q_i^{d(i)}$ is exact. Consider those states i in which all the requests can be granted simultaneously without conflict. We call the request sets of such states immediately grantable and we denote the set of such states by IG. If the decision, $d(i)$, in some state $i \in IG$ is such that all the requests are granted, then the leftover set for state i is the same as the leftover set for state 1. Now if two states k and l have the same leftover set, then $p_{kj}^{d(k)} = p_{lj}^{d(l)}$ for all j since the next state is entirely determined by the leftover set and the probability distribution of new request arrivals which is the same for both states. Thus if $d(i)$ is such that all requests are granted, then $p_{ij}^{d(i)} = p_{1j}$ for all j. Equation 3.11 then implies that $v_i^{\mathbf{D}} - v_1^{\mathbf{D}} = q_i^{d(i)}$.

This previous result can be generalized. Consider any two states i and j with decisions $d(i)$ and $d(j)$ such that both states have the same leftover set. Then $p_{ik}^{d(i)} = p_{jk}^{d(j)}$ for all k and

$v_i^D - v_j^D = q_i^{d(i)} - q_j^{d(j)}$. This result follows from the obvious generalization of equation 3.11 to

$$v_i^D - v_j^D = q_i^{d(i)} - q_j^{d(j)} + \sum_k (p_{ik}^{d(i)} - p_{jk}^{d(j)})(v_k^D - v_j^D).$$

The determination of all the relative values $v_i^D - v_1^D$, and hence solving for g^D, thus amounts to determining the difference in relative values of states with different leftover sets. This is consistent with our earlier observation that the Markovian decision problem can be expressed in terms of leftover sets rather than request sets.

Since the relative value in state i, $v_i^D - v_1^D$, represents the difference in the average total reward starting in state i relative to that starting in state 1, (which has only null requests), it seems intuitive that $v_i^D - v_1^D$ should never exceed the number of genuine, i.e. non-null, requests in that state which we denote by n_i. We found that indeed $v_i^D - v_1^D \leq n_i$ for all states i for every case we investigated for four (and six) slices. We were unable to establish if this inequality is true in general.

We now return to our approximation $v_i^D - v_1^D \approx q_i^{d(i)}$ and the determination of an approximate analytical expression for the regions corresponding to request subsets (a) and (b) in states 20, 34, and 40 in the four slice, single round grant duration Symmetric Ringbus. Request subsets (a) and (b) each grant the maximum number of requests possible in each of the states 20, 34, and 40. Thus the choice of request subset (a) or (b) in these three states does not depend on the immediate reward: it depends only on the leftover sets. For a given policy D, request subset (a) results in an improvement in the throughput if

$$\sum_j p_{ij}^{(a)}(v_j^D - v_1^D) > \sum_j p_{ij}^{(b)}(v_j^D - v_1^D)$$

and request subset (b) results in an improvement if

$$\sum_j p_{ij}^{(a)}(v_j^D - v_1^D) < \sum_j p_{ij}^{(b)}(v_j^D - v_1^D)$$

where $i = 20$, 34, or 40 and we have cancelled the immediate rewards from both sides of the inequalities. Approximating $v_j^D - v_1^D$ by $q_j^{d(j)}$, we have:

$$\Delta \equiv \sum_j (p_{ij}^{(a)} - p_{ij}^{(b)}) q_j^{d(j)}$$

If $\Delta > 0$ then request subset (a) is best and if $\Delta < 0$ then request subset (b) is best. Since we already know that the optimal policy consists of granting the maximum number of requests in each state, $q_j^{d(j)}$ is equal to the maximum number of simultaneously grantable requests in state j. The leftover sets from request subsets (a) and (b) are shown below.

Leftover set from request subset (a)

Leftover set from request subset (b)

Figure 3.10: Leftover sets from request subsets (a) and (b) in states 20, 34, and 40

Table 3.2 lists the possible next states (without symmetry removed), the immediate reward in each state, and the transition probability.

Leftover Set (a)			Leftover Set (b)	
Next State	Immediate Reward	Transition Probability	Next State	Immediate Reward
-1, 2,-1,-1	3	p_1^2	-1,-1, 1,-1	3
-1, 2,-1, 0	2	p_0p_1	-1,-1, 1, 0	2
-1, 2,-1, 1	2	p_1^2	-1,-1, 1, 1	2
-1, 2,-1, 2	2	p_1p_2	-1,-1, 1, 2	3
-1, 2, 0,-1	2	p_0p_1	-1, 0, 1,-1	2
-1, 2, 0, 0	1	p_0^2	-1, 0, 1, 0	1
-1, 2, 0, 1	2	p_0p_1	-1, 0, 1, 1	2
-1, 2, 0, 2	2	p_0p_2	-1, 0, 1, 2	2
-1, 2, 1,-1	2	p_1^2	-1, 1, 1,-1	2
-1, 2, 1, 0	1	p_0p_1	-1, 1, 1, 0	2
-1, 2, 1, 1	2	p_1^2	-1, 1, 1, 1	3
-1, 2, 1, 2	2	p_1p_2	-1, 1, 1, 2	3
-1, 2, 2,-1	3	p_1p_2	-1, 2, 1,-1	2
-1, 2, 2, 0	2	p_0p_2	-1, 2, 1, 0	1
-1, 2, 2, 1	2	p_1p_2	-1, 2, 1, 1	2
-1, 2, 2, 2	2	p_2^2	-1, 2, 1, 2	2

Table 3.2: Rewards and Transition Probabilities for Decisions (a) and (b) in States 20, 34, and 40

After some algebra we obtain

$$\Delta = p_0 p_2 - p_0 p_1 - p_1 p_2 - p_1^2$$

which can be further simplified to

$$\Delta = p_1^2 - p_1 - 2p_1 p_2 + p_2 - p_2^2.$$

The boundary between the regions for request subsets (a) and (b) is given approximately by $\Delta = 0$. This approximate boundary is surprisingly close to the exact boundary between the regions as shown in Figure 3.11.

We are not so fortunate with the boundary between the regions in which request subsets (a) and (b) in state 38 (see Figure 3.8) are optimal respectively. An analysis similar to that just completed for states 20, 34, and 40 and again with $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$ approximated by $q_i^{d(i)}$ for all i yields $\Delta = p_1(p_1 - p_2)$. Thus the boundary between the regions for request subsets (a) and (b) in state 38 is approximated by $p_1 = p_2$. This approximate boundary and the exact boundary are shown in Figure 3.12. The large discrepancy in these boundaries indicates that $v_i^{\mathbf{D}} - v_1^{\mathbf{D}} \approx q_i^{d(i)}$ is not a very good approximation in this case. This is to be expected since the difference between leftover sets (a) and (b) is very subtle (see Figure 3.8). We expect the average reward per round to be almost the same for request subsets (a) and (b) over much of the $p_1 = p_2$ probability space. Of course, greater accuracy in estimating the boundary can be achieved by using more terms of equation 3.13 in the approximations of $v_i^{\mathbf{D}} - v_1^{\mathbf{D}}$.

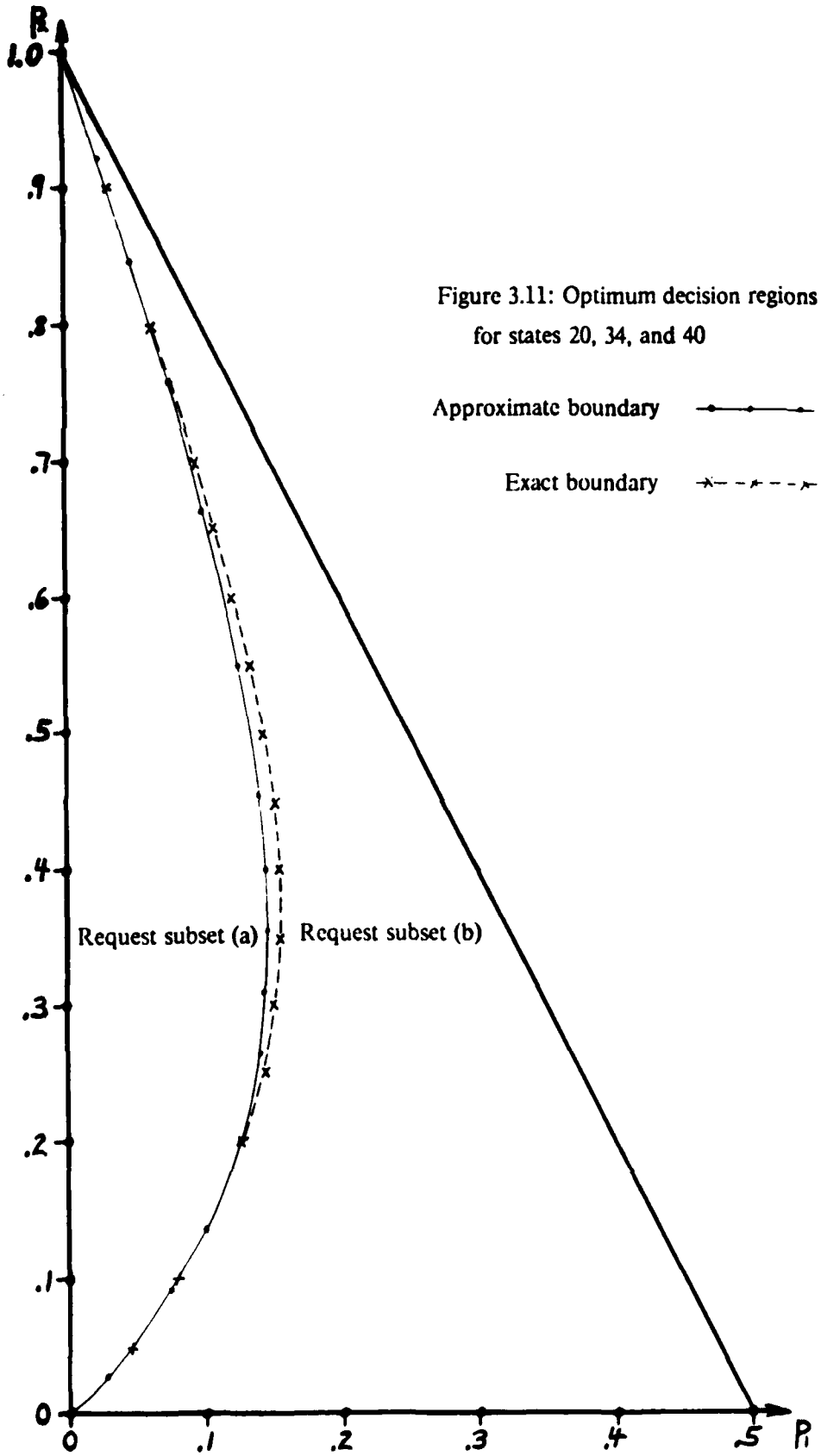


Figure 3.11: Optimum decision regions for states 20, 34, and 40

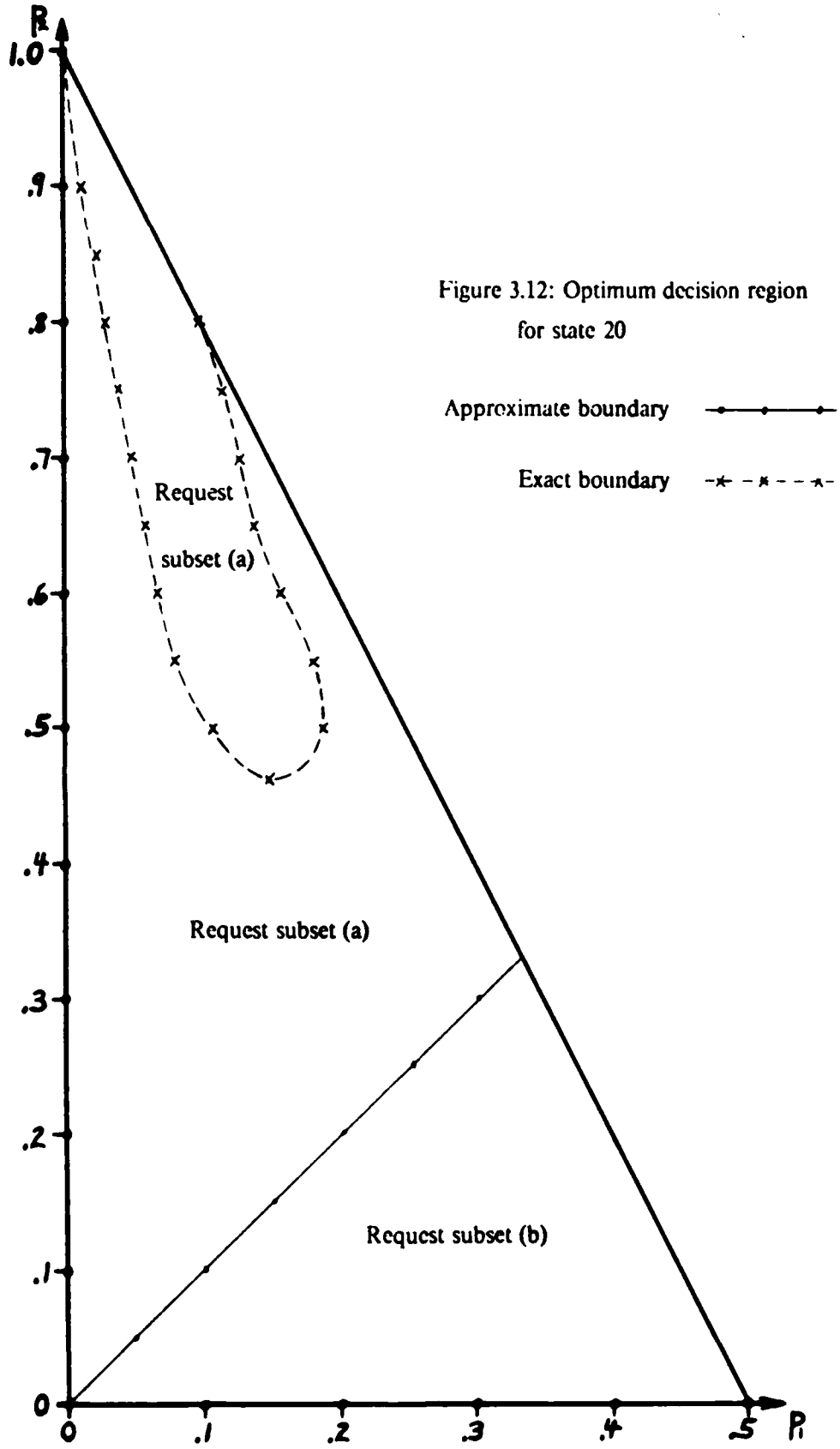


Figure 3.12: Optimum decision region for state 20

We investigated the approximation $v_i^D - v_1^D \approx q_i^{d(i)}$ in two instances. In the first instance we approximated the test quantity (equation 3.9) in step 3 of Howard's policy iteration algorithm by

$$\max_k \left(q_i^k + p_{ii}^{d(i)} q_i^k + \sum_{j \neq i} p_{ij}^{d(i)} q_j^{\max} \right) \tag{3.15}$$

where $q_j^{\max}$ is the maximum number of grants possible in state j. We found that the decision k yielded by this approximate test quantity reliably predicts the optimum decision in state i in most cases. (The main exception was in state 38.) In the second instance we approximated g^{opt} by $g^{est} = \sum_j p_{1j} q_j^{\max}$. This approximation corresponds to granting the maximum number of requests in every state and ignoring the leftover requests. The comparison of the calculated values of g^{opt} and g^{est} shown in Figure 3.13 for various probabilities reveals that g^{est} is a good approximation to g^{opt}. In every case investigated we found $0 \leq g^{opt} - g^{est} \leq 0.22$. Figure 3.13 also shows the optimum average number of grants per round for a crossbar interconnection of four slices. This crossbar interconnection is similar to the Ringbus interconnection except for fewer constraints on which request subsets may be granted. In fact, the only constraints on the request subsets are destination constraints: no two requests that have the same destination can be granted simultaneously. Since a crossbar has fewer constraints than a Ringbus, its performance will always be superior to that of a Ringbus (provided everything except for the interconnections is the same).

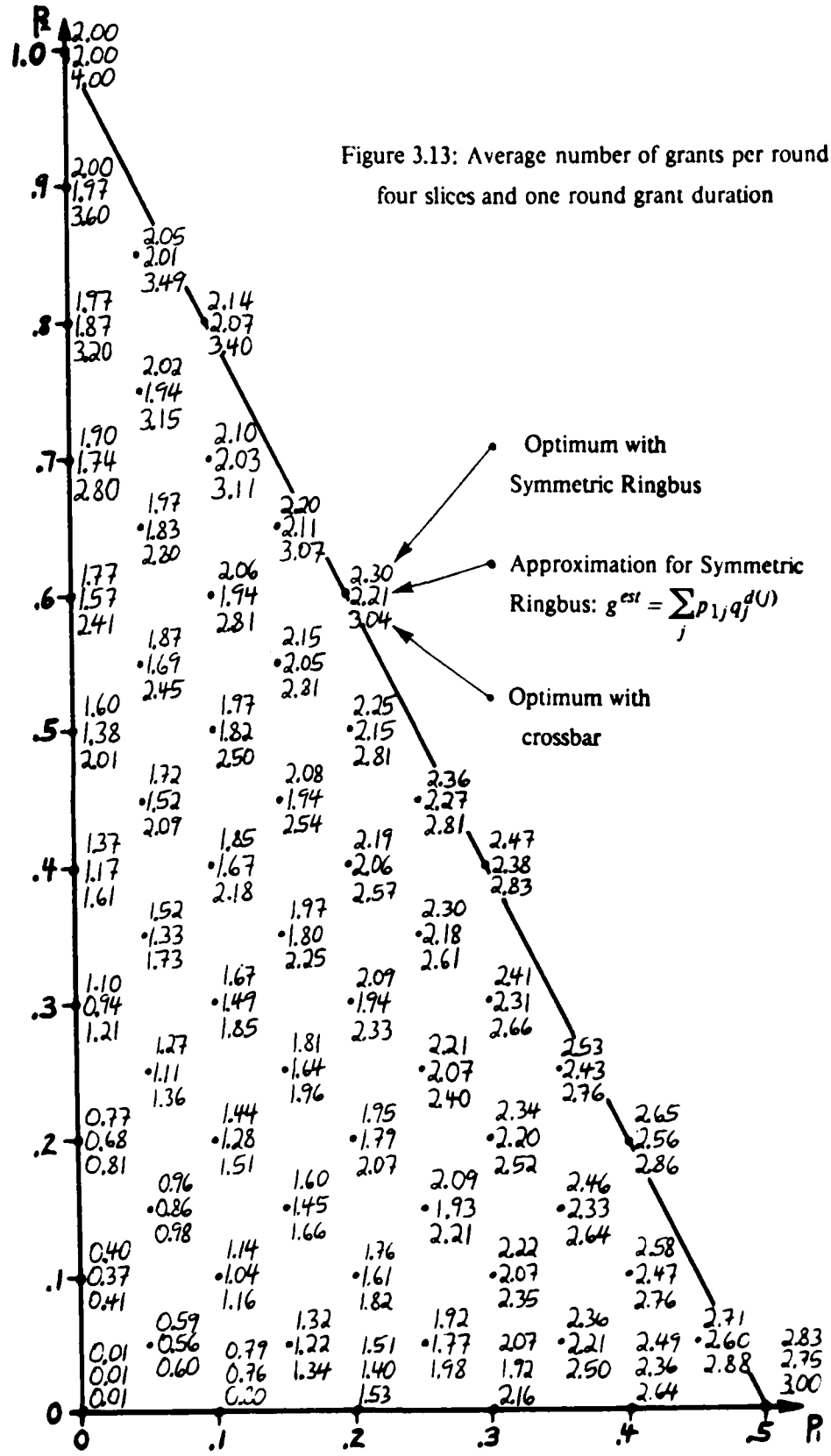


Figure 3.13: Average number of grants per round with four slices and one round grant duration

3.4 Optimal Arbiter for Four Slices and Grant Duration Greater than One Round

In this section we investigate the optimal arbiter for the Symmetrical Ringbus with four slices and deterministic grant durations of 2, 3, and 4 rounds and geometrically distributed grant durations. The basic state description is

$$(r_1, d_1; r_2, d_2; r_3, d_3; r_4, d_4)$$

as described in section 3.2 and we assume symmetric Ringbus probabilities, i.e. p [illegible] p. As in the previous section we apply rotational and flip symmetry to significantly reduce the number of states required. In fact, we start with the same 43 states as in the previous section and add the d to these states to obtain a complete state description. However, one complication arises: a request may be granted in either of two ways on the Ringbus: in the shortest direction for requests of length 1 and the clockwise direction for requests of length 2 (we call this the primary direction) or the longest direction for requests of length 1 and the counterclockwise direction for requests of length 2 (we call this the secondary direction). For a grant in progress [illegible] the state description must include the direction in which the request has been granted [illegible] slots allocated to that grant are known. One way to include this information [illegible] but [illegible] with each grant in progress indicating its direction on the Ringbus.

A more efficient method to include the direction of grant [illegible] is based on the following two observations:

1. If a request of length 1 is granted in the longest [illegible] other request can be granted in the longest direction [illegible] be granted.
2. If a request of length 2 is granted in the counterclockwise direction [illegible] 1 can be granted in the longest direction and [illegible] counterclockwise direction can be granted.

These observations [illegible]

[illegible]

The complete state description is

$$(\ldots, \ldots, \ldots, \ldots, d_1, d_2, \ldots, d_4, \ldots, \mathit{common})$$

The left part $(\ldots, \ldots, \ldots, \ldots)$ refers to one of the 43 states of Table 3.1 obtained after all the rotational and flip symmetry has been removed. The middle part $(d_1, d_2, \ldots, d_4)$ indicates the uninterrupted duration of time for which each of the corresponding requests has been granted. All requests which are not granted have a duration of zero. Only a request corresponding to grants in progress have non-zero values. All such requests have a duration of zero. The right part *common* is the [illegible] grants in progress as a [illegible] *common*. A [illegible]

[illegible] all states for which [illegible] in progress [illegible]

[illegible]

[illegible]

2. Determine $V_i(n+1)$ for all i from:

$$V_i(n+1) = \max_k \left(q_i^k + \sum_j p_{ij}^k V_j(n) \right) \tag{3.16}$$

The policy in round $n+1$ is $\mathbf{D}(n+1) = [d(1), d(2), \cdots]$ where $d(i)$ is equal to the decision k which maximizes the right hand side of equation 3.16 for state i.

3. Increment n, go to step 2.

The real power of value iteration is due to the so-called Odoni bounds [O?] which give upper and lower bounds on the optimal average reward per round. These limits improve on each iteration and eventually converge to the optimal average reward per round. Define $\delta_i(n) = V_i(n) - V_i(n-1)$, $L(n) = \min_i \delta_i(n)$, and $U(n) = \max_i \delta_i(n)$. After the nth iteration of step 2 we have

$$L(n+1) \leq g^{opt} \leq U(n+1).$$

Furthermore, $L(n) \geq L(m)$ and $U(n) \leq U(m)$ for $m \leq n$. Therefore after the nth iteration of step 2, g^{opt} may be estimated by $g^{opt} \approx g^{est}(n+1) = \frac{U(n+1)+L(n+1)}{2}$. As $n \to \infty$, $\frac{U(n)+L(n)}{2} \to g^{opt}$, $V_i(n) - V_j(n) \to v_i - v_j$, and $\mathbf{D}(n) \to \mathbf{D}^{opt}$.

For our purposes, value iteration also has implementation advantages over Howard's policy iteration. With value iteration we need only store the $V_i(n)$ for all states and the request probabilities p_i. The possible decisions k and associated rewards, q_i^k, and transition probabilities, p_{ij}^k, can be computed on the fly. With policy iteration, it is difficult to solve for $g^{\mathbf{D}}$ and the relative values without first having calculated and stored all the q_i and p_{ij} for a particular policy, which requires a lot of storage if the number of states is large.

Of course, with value iteration neither the estimate of g^{opt} nor the estimate of the optimal policy is necessarily exact. However $|g^{est}(n) - g^{opt}|$ can be as small as desired simply by iterating to sufficiently large n. The difference between the upper and lower bounds on g^{opt} indicates the significance of any differences between $\mathbf{D}(n)$ and $\mathbf{D}^{opt}$.

For the large number of states that we are considering, any other known method would also give approximate results. With Howard's policy iteration we would have to use iterative techniques to obtain an approximate solution to the large set of simultaneous equations represented by equation 3.8.

Before considering deterministic and geometrically distributed grant distributions, we introduce [illegible]. We continue to use g to denote the average reward per round. However, we [illegible] round is the number of new and continuing grants in that round

(rather than just the number of new grants in that round).† Thus g is better thought of as the average number of grants in progress per round. Of course, if all grants have the same duration of one round, then g is also the throughput. We define the throughput as the average number of new grants per round and denote it by t. For a general grant duration distribution with mean $\bar{d}$ rounds, the throughput and average number of grants in progress per round are related by $t = g/\bar{d}$. The throughput of the Ringbus is also given by the number of slices divided by the mean cycle time per slice. This yields the throughput balance equation

$$\frac{S}{\frac{p_0}{1-p_0} + w_{RB} + \bar{d}} = t = g/\bar{d} \tag{3.17}$$

where $\frac{p_0}{1-p_0}$ is the mean processing time per slice (p_0 is the probability of a null request), w_{RB} is the mean waiting time per request, $\bar{d}$ is the mean grant duration, and S is the number of slices.

3.4.1 Deterministic Grant Duration of 2, 3, and 4 Rounds

Using value iteration and Odoni's bounds, as described earlier, we obtained estimates of the optimal average number of grants in progress (trivially related to the throughput) and estimates of the optimal policy for the *Symmetric Ringbus with deterministic grant durations* of 2, 3, and 4 rounds. Figure 3.14 shows the optimal average number of grants in progress for these three cases and for grant durations of one round, as investigated earlier, for selected probabilities. All these estimates, except those marked with an asterisk, are within ±.005 of optimal. The asterisks indicate estimates for which a tolerance of ±.005 was not achieved after 100 iterations. The maximum error in these estimates, as determined by the bounds $L(100)$ and $U(100)$, is ±.0175.

† There is no theoretical reason to prefer one of these definitions of the reward over the other. With our definition, the solution of the Markovian decision problem yields the average number of grants in progress per round. With the other definition, the solution yields the throughput. The average number of grants in progress per round and the throughput are trivially related as shown in equation 3.17. There is however an empirical reason to prefer our definition of the reward over the other definition: we found that the value iteration method converged faster with our definition.

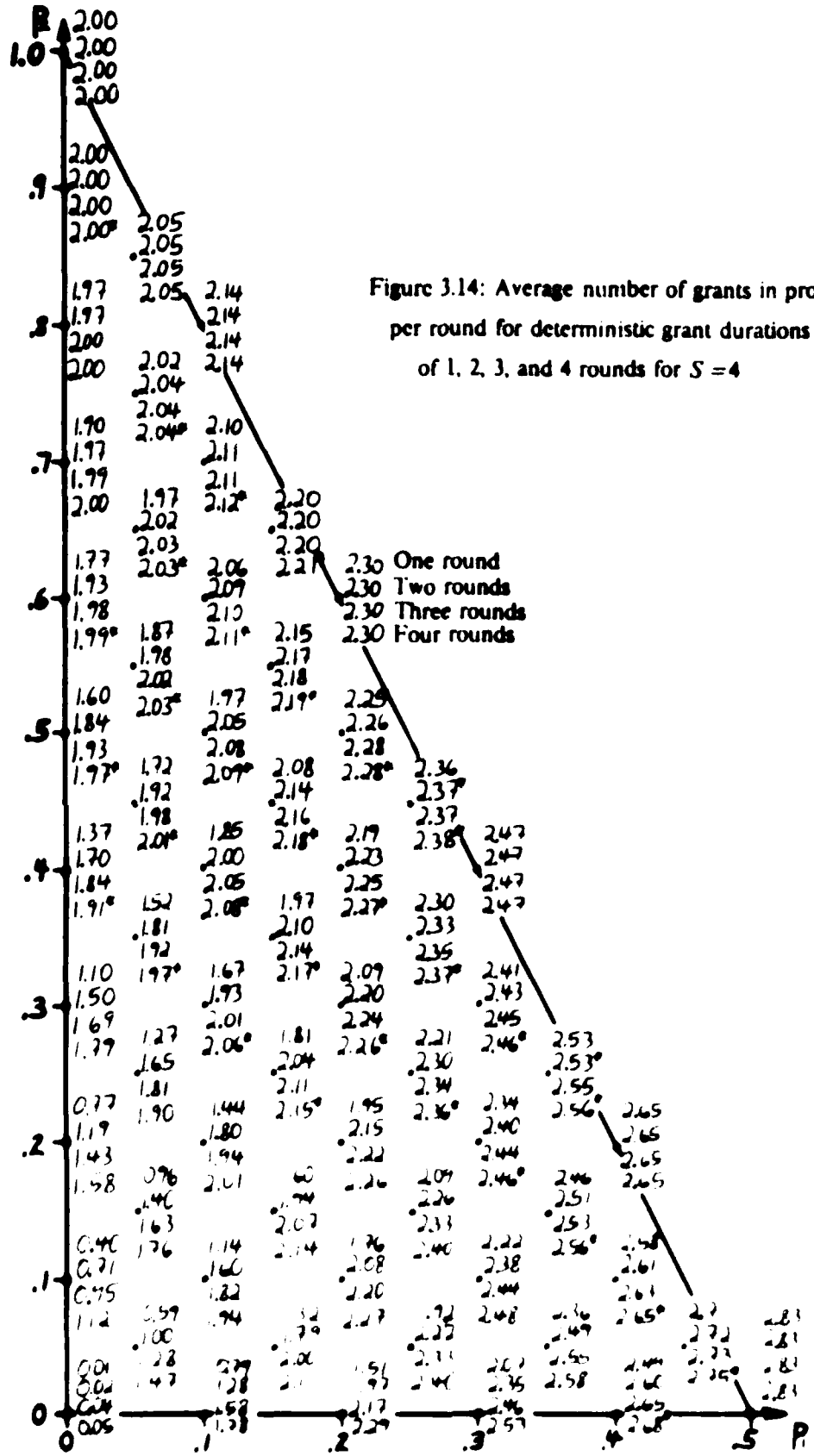


Figure 3.14: Average number of grants in progress per round for deterministic grant durations of 1, 2, 3, and 4 rounds for $S = 4$

Note that the optimal average number of grants in progress (which we will call simply g in the rest of section 3.4) is a strong function of the grant duration, d, when p_0 is large - i.e. light loading. As p_0 decreases, the grant duration has less effect on g. In fact, for $p_0 = 0$, g appears to be independent of the grant duration (for deterministic grant durations). These observations make intuitive sense. When p_0 is large, nonnull requests are rare and occur with almost equal likelihood regardless of d. The difference is that the grants last longer for larger d and thus contribute more to g. When p_0 is small, the Ringbus is nearly saturated with nonnull requests every round and thus d has little effect on g. Section 3.4.2 examines these observations with more rigour.

Because of the large number of states, especially for $d = 4$, it is impossible to discuss here in detail the estimated optimal decisions in each state for $d = 2$, 3, and 4. Instead, we will just discuss one main trend observed in the estimated optimal decisions. It was exhausting enough to examine all the states for various probabilities to determine this trend.

The main trend is the following interesting observation: sometimes the estimated optimal decision in some states grants less than the maximum reward in those states. The states in which this phenomenon was observed fall into two main classes: 1) states with a small (one or two) number of nonnull requests all ungranted so far and 2) states with a large number of requests with grants in progress and a small number of ungranted nonnull requests. An example of a state in the first class is the state

(1 0 0 0 0 0 0, 0, 5)

Although the request in this state is immediately grantable, the optimum estimated decision sometimes is not to grant the request. Another such example is the state

(1 2 0 0, 0, 0, 0 0 5)

Again the request in this state are immediately grantable and again the optimum estimated decision sometimes is to grant neither request. An example of a state in the second class is the state

(1 1 1 [illegible] 0 5) [illegible] d [illegible]

[illegible] the grant duration [illegible] Although the [illegible] request [illegible] duration [illegible] is [illegible] the optimum [illegible] estimated [illegible] sometimes [illegible] grant the request. The [illegible] depend on [illegible] factors

1 [illegible]

2 [illegible]

3 [illegible]

The less than the maximum reward tendency is most pronounced for heavy loading - i.e. small p_0 - and large d. For light loading - i.e. large p_0 - the estimated optimal decision in every state grants the maximum reward.

An explanation for the observed tendencies is that there is a tradeoff between the immediate contribution to throughput obtained by granting a request and the possible future degradation to throughput caused by the constraints imposed on future grants by granting a request. For a grant duration of d rounds, any request granted imposes constraints on which requests may be granted in the $d-1$ rounds after it is first granted. For light loading it is unlikely that a nonnull request will arrive within d rounds of granting a request and thus the future degradation caused by granting requests is negligible. Therefore the tradeoff is in favour of granting requests immediately and hence the tendency towards granting the maximum reward for light loading. For heavy loading, it is very likely that a nonnull request will arrive within d rounds of granting a request and thus the future degradation caused by granting requests can be very significant. This degradation can be reduced by avoiding large differences in the duration that different grants have been in progress. States in the first class (of the two mentioned earlier) achieve this by not granting any requests.

In heavy loading additional nonnull requests will likely arrive very soon, so it makes better sense to wait until these requests arrive and grant all the requests at once rather than grant the initial request and delay the granting of any subsequent nonnull requests until the grant of the initial request terminate. States in the second class avoid differences in the duration of grants in progress by delaying the granting of new nonnull requests until the grants in progress terminates. The result in heavy loading is that all grants in a state tend to have the same duration in progress.

Thus in heavy loading there is a tendency towards granting requests at intervals of d rounds. We term such an algorithm an interval algorithm. Note that an interval algorithm completely eliminates the throughput degradation caused by the constraints a grant imposes on future grants because grants in one interval do not impose constraints on the grants in succeeding intervals.

3.4.2 Deterministic Grant Duration - The General Case

In this subsection we present computational results for deterministic grant durations. We assume a deterministic grant duration of d rounds.

3.4.2.1 General Characteristics of Optimum Throughput

[illegible]

as $\frac{\partial t^{opt}}{\partial(1-p_0)}$ and $\frac{\partial^2 t^{opt}}{\partial(1-p_0)^2}$, need not be.

For any particular policy (i.e. set of decisions in each state), all the derivatives of t with respect to the probabilities will be continuous. However, the optimum policy can vary with the probabilities. Thus the optimum throughput over any portion of the feasible probability region is, in general, a piecewise combination of the throughput of the optimum policy in each subregion. The derivatives of t^{opt} with respect to the probabilities will not, in general, be continuous at the boundaries of the subregions. Fortunately, the number of discontinuities along any ray is finite since there is only a finite number of different policies. Strictly speaking, $\frac{\partial t^{opt}}{\partial(1-p_0)}$ is not defined at such a discontinuity, but it may be defined to have the value of one of the policies at the point of discontinuity. In this sense, $\frac{\partial t^{opt}}{\partial(1-p_0)}$ is strictly positive for all p_0 along a ray (except possibly at the end points).

It is also obvious that $\frac{\partial^2 t^{opt}}{\partial(1-p_0)^2} < 0$ for all p_0 along a ray, except perhaps at discontinuities at the boundaries of the subregions corresponding to different policies. (Recall that the optimum policy can vary with the probabilities. Note also that $\frac{\partial^2 t^{opt}}{\partial(1-p_0)^2}$ is not defined at such discontinuities). Within any particular subregion along a ray, the rate of increase of t^{opt} with $1-p_0$ - i.e. $\frac{\partial t^{opt}}{\partial(1-p_0)}$ decreases as $1-p_0$ increases since there are fewer null requests to replace by nonnull requests (and thus increase t^{opt}) as $1-p_0$ increases. Hence, $\frac{\partial^2 t^{opt}}{\partial(1-p_0)^2} < 0$ within the subregion and thus t^{opt} is convex down in $1-p_0$ within each subregion along any ray. Note that it does not follow from this that t^{opt} is convex down everywhere along a given ray, even if $\frac{\partial^2 t^{opt}}{\partial(1-p_0)^2}$ is redefined at points of discontinuity. In summary, t^{opt} is monotonic in $1-p_0$ everywhere along a ray and convex down in $1-p_0$ within any subregion along a ray.

3.4.2.2 Bounds on the Optimum Throughput with Deterministic Grant Durations

Let $t_p^{d>1}$ and $g_p^{d>1}$ denote the optimal throughput and the optimal average number of grants per round respectively for grants with a deterministic duration of $d>1$ round and for some set of probabilities $p_0, p_1, \ldots, p_S$ - denoted by p^* (S is the number of slices). Similarly let $t_p^{d=1}$ and $g_p^{d=1}$ denote the optimal throughput and optimal average number of grants per round respectively for grants with a duration of one round and for some set of probabilities $p_0, p_1, \ldots, p_S$ - denoted by $p^{*'}$. Now for the same set of probabilities in each case (i.e. $p^* = p^{*'}$) the

optimal throughput with $d>1$ can be no more than the optimal throughput with $d=1$. This follows since for a fixed set of probabilities the optimal throughput cannot increase as d increases. Thus

$$t_{\vec{p}}^{d>1} \leq t_{\vec{p}}^{d=1} \quad \text{or} \quad g_{\vec{p}}^{d>1} \leq d g_{\vec{p}}^{d=1}.$$

For $d>1$ it is possible to grant at least the same average number of grants per round as for the same set of probabilities for $d=1$. The argument is as follows. Restrict the instants at which all new requests - even null requests - for $d>1$ can be granted to the beginning of every d^{th} round in synchrony with some clock of period d rounds. (Note that null requests have a grant duration of one round and nonnull requests have a grant duration of d rounds. Restricting the granting of null requests to every d rounds synchronous with the clock of period d artificially lengthens the grant duration of a null request to d rounds.) At the "arbitration instant" at the beginning of each successive interval of d rounds (synchronous with the clock of period d), grant the request subset corresponding to the optimal decision for that request set with $d=1$ and the same set of probabilities. The result is an arbitration algorithm for $d>1$ which is exactly the same as the optimal arbitration algorithm with $d=1$, the same set of probabilities, and a arbiter clock period of d. That is, by

1) restricting the instants at which new requests - even null requests - can be granted to every d rounds synchronous with some clock of period d, and

2) using this clock of period d as the arbiter clock,

the arbitration problem reduces to that for $d=1$. Thus

$$g_{\vec{p}}^{d=1} \leq g_{\vec{p}}^{d>1} \quad \text{or} \quad \frac{1}{d} t_{\vec{p}}^{d=1} \leq t_{\vec{p}}^{d>1}.$$

We call an arbiter algorithm that operates in accordance with point 1 above an interval algorithm. We call the optimum algorithm subject to this restriction the optimum interval algorithm. As just discussed above, the optimal interval algorithm is exactly the same as and achieves the same throughput as the optimal algorithm for $d=1$.

These lower bounds on $g_{\vec{p}}^{d>1}$ and $t_{\vec{p}}^{d>1}$ can be tightened by removing the restriction that null requests can only be granted at the beginning of every d rounds synchronous with the clock interval of period d. Instead, let null requests be granted immediately whenever they occur as was the case in our original formulation of the arbitration problem. However, this affects the probability of requests as seen by the (restricted) arbiter every arbitration instant. The (restricted) arbiter sees a null request at an arbitration instant if there have been exactly d null requests since the last arbitration instant, otherwise it sees a nonnull request. Thus the (restricted) arbiter sees a null request with probability $(p_0)^d$ and a nonnull request of length i with probability $\frac{p_i(1-(p_0)^d)}{(1-p_0)}$.

Therefore

$$g_{p'}^{d=1} \le g_{p}^{d>1} \quad \text{or} \quad \frac{1}{d} t_{p'}^{d=1} \le t_{p}^{d>1}$$

where $p_0' = (p_0)^d$ and $p_i' = \dfrac{p_i(1-(p_0)^d)}{(1-p_0)}$, $1 \le i \le S/2$.

This lower bound is easily seen to be tighter than the previous one since $\dfrac{\partial g^{d=1}}{\partial(1-p_0)} \ge 0$ for every p_0 along any ray in the feasible probability region.

The complete bounds on the optimal throughput for a deterministic grant duration $d>1$ rounds are:

$$g_{p'}^{d=1} \le g_{p'}^{d=1} \le g_{p}^{d>1} \le d g_{p}^{d=1}$$

$$\text{or} \quad \frac{1}{d} t_{p'}^{d=1} \le \frac{1}{d} t_{p'}^{d=1} \le t_{p}^{d>1} \le t_{p}^{d=1}$$

where $p_0' = (p_0)^d$ and $p_i' = \dfrac{p_i(1-(p_0)^d)}{(1-p_0)}$, $1 \le i \le S/2$. Note that these bounds are expressed completely in terms of the optimal throughput for $d=1$ which is a much simpler problem than for $d>1$.

$g_{p}^{d>1}$ and $t_{p}^{d>1}$ approach their respective upper bounds as $p_0 \to 1$. This can be shown as follows. We have

$$\frac{t_{p}^{d>1}}{t_{p}^{d=1}} = \frac{\dfrac{S}{\dfrac{p_0}{1-p_0} + w_{RB}^{d>1} + d}}{\dfrac{S}{\dfrac{p_0}{1-p_0} + w_{RB}^{d=1} + 1}} = \frac{p_0 + (1-p_0)(w_{RB}^{d=1} + 1)}{p_0 + (1-p_0)(w_{RB}^{d>1} + d)}.$$

Taking the limit as $p_0 \to 1$, for which $w_{RB}^{d=1} \to 0$ and $w_{RB}^{d>1} \to 0$, we have $\lim_{p_0 \to 1} \dfrac{t_{p}^{d>1}}{t_{p}^{d=1}} = 1$ and $\lim_{p_0 \to 1} \dfrac{g_{p}^{d>1}}{g_{p}^{d=1}} = d$. Thus $t_{p}^{d>1} \approx t_{p}^{d=1}$ and $g_{p}^{d>1} \approx d g_{p}^{d=1}$ for $p_0 \approx 1$. The estimated optimal average number of grants per round for light loading shown in Figure 3.14 correlates well with this latter result. This result justifies the intuitive reason given in section 3.4.1 for the strong relationship of $g_{p}^{d>1}$ with d for light loading.

Similarly, $g_{p}^{d>1}$ and $t_{p}^{d>1}$ seem to approach their respective lower bounds as $p_0 \to 0$, as suggested by the results for $p_0 \approx 0$ in Figure 3.14 for $S=4$ and $d=2$, 3 and 4 rounds. We were unable

to prove this conjecture.

3.4.2.3 Approximating the Optimal Throughput with Deterministic Grant Duration

If d, the grant duration, is large, the number of states required to calculate $g_{\vec{p}}^{d>1}$ is very large, making its calculation difficult. A more attractive approach for large d is to approximate $g_{\vec{p}}^{d>1}$. In this subsection we present a simple approximation to $g_{\vec{p}}^{d>1}$ using $g_{\vec{p}}^{d=1}$ and the value of d. Since $t_{\vec{p}}^{d>1}$ is trivially related to $g_{\vec{p}}^{d>1}$, this approximation also applies (although indirectly) to $t_{\vec{p}}^{d>1}$.

The ratio $\dfrac{g_{\vec{p}}^{d>1}}{g_{\vec{p}}^{d=1}}$ has the following properties along a ray:

1) It is a continuous function of $1 - p_0$.

2) $\lim\limits_{p_0 \to 1} \dfrac{g_{\vec{p}}^{d>1}}{g_{\vec{p}}^{d=1}} = d$

3) $\lim\limits_{p_0 \to 1} \dfrac{\partial \left[g_{\vec{p}}^{d>1} / g_{\vec{p}}^{d=1} \right]}{\partial (1 - p_0)} = d(d-1)$

In addition we conjecture that $\lim\limits_{p_0 \to 0} \dfrac{g_{\vec{p}}^{d>1}}{g_{\vec{p}}^{d=1}} = 1$.

We choose to approximate $\dfrac{g_{\vec{p}}^{d>1}}{g_{\vec{p}}^{d=1}}$ by an exponential function with the same value (d) and slope ($-d(d-1)$) at $p_0 = 1$, and the same asymptotic value as conjectured at $p_0 = 0$. Thus our approximation along any ray is

$$\frac{g_{\vec{p}}^{d>1}}{g_{\vec{p}}^{d=1}} \approx (d-1)e^{-d(1-p_0)} + 1 \qquad (3.8)$$

This approximation is clearly suspect in at least two regards. [illegible] every ray. In addition, it is not equal to 1 at $p_0 = 1$ [illegible] ment is its simplicity. Nevertheless, the approximation [illegible] compared to results determined [illegible] approximation (equation 3.8) for $N = 4$, $p = 3/4$, [illegible] function of $1 - p_0$ for $N = 4$ (computed [illegible] $p = p_0$ and $p = \frac{1}{4}$ [illegible]

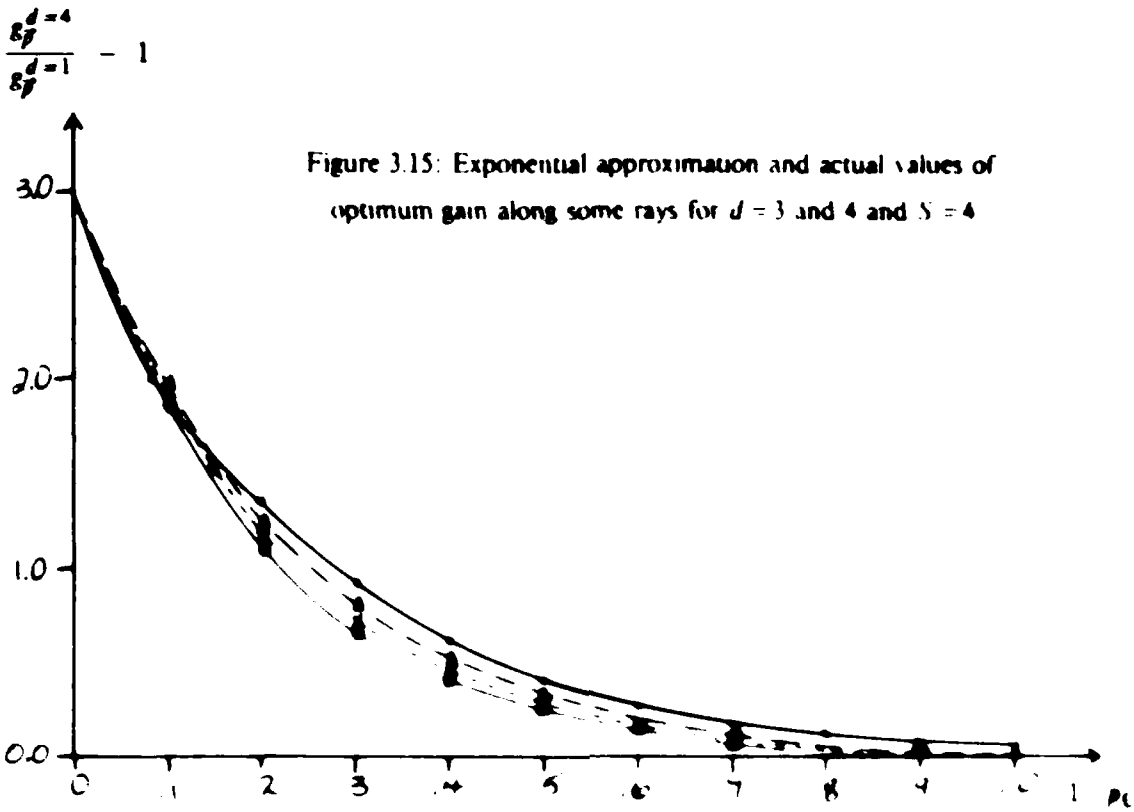
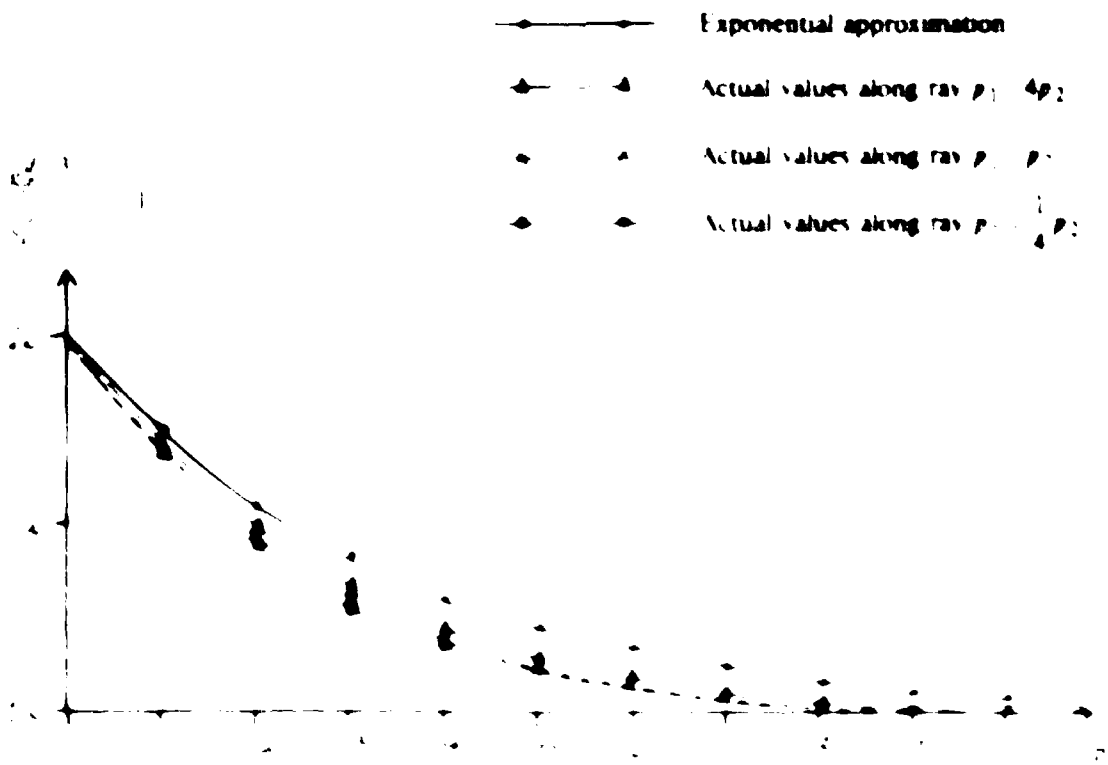


Figure 3.15: Exponential approximation and actual values of optimum gain along some rays for $d = 3$ and 4 and $S = 4$



For $d = 3$ the approximation overestimates $\frac{g_p^{d>1}}{g_p^{d}}$ for about $p_0 > .1$. For $d = 4$ the approximation seems to slightly underestimate $\frac{g_p^{d>1}}{g_p^{d}}$ for about $p_0 < .1$ and otherwise it slightly overestimates the ratio.

We do not know what performance to expect of the approximation in equation 3.18 for larger values of S and d. However, it is also clear that the approximation is roughly correct: $\frac{g_p^{d>1}}{g_p^{d}}$ must decrease from d to 1 as $1 - p_0$ increases from 0 to 1. The approximation has two key advantages. First, it is very simple to calculate. Second, it reduces the determination of $g_p^{d>1}$ to the determination of g_p^{d}, a much simpler problem. This second advantage cannot be overstated: for large S it is difficult enough to determine $g_p^{d=1}$, as discussed later in this chapter, let alone $g_p^{d>1}$. Indeed, in the remainder of this chapter (excluding the next two subsections 3.4.3 and 3.4.4) we only consider the Ringbus with $d = 1$ and point to equation 3.18 for treatment of arbitrary deterministic grant durations.

3.4.3 Geometric Grant Duration

For investigating the optimal throughput of the four slice Symmetric Ringbus with geometric grant durations we used the same state description as discussed earlier for a deterministic grant duration of 1 rounds. However, instead of interpreting d_i ($= 0$ or 1) as the number of rounds that slice i's request has been granted, we interpreted d_i as a boolean value indicating whether or not slice i's request was granted in the preceding round. If a slice's request was granted in the preceding round, then it remained granted in the current round with probability *prbcont* and it terminated immediately prior to the current round with probability $1 - prbcont$. Thus the grant duration of a request was a geometric random variable with mean $\frac{1}{1 - prbcont}$.

Using the iteration and Odoni's bounds we obtained estimates of the optimal average number of grants in progress and estimates of the optimal policy for *prbcont* = .001, .5, .75, .9. Figures 3.[illegible] show the estimated optimal number of grants in progress per round for these four cases with selected request probabilities. All these estimates are within ±.005 of optimal.

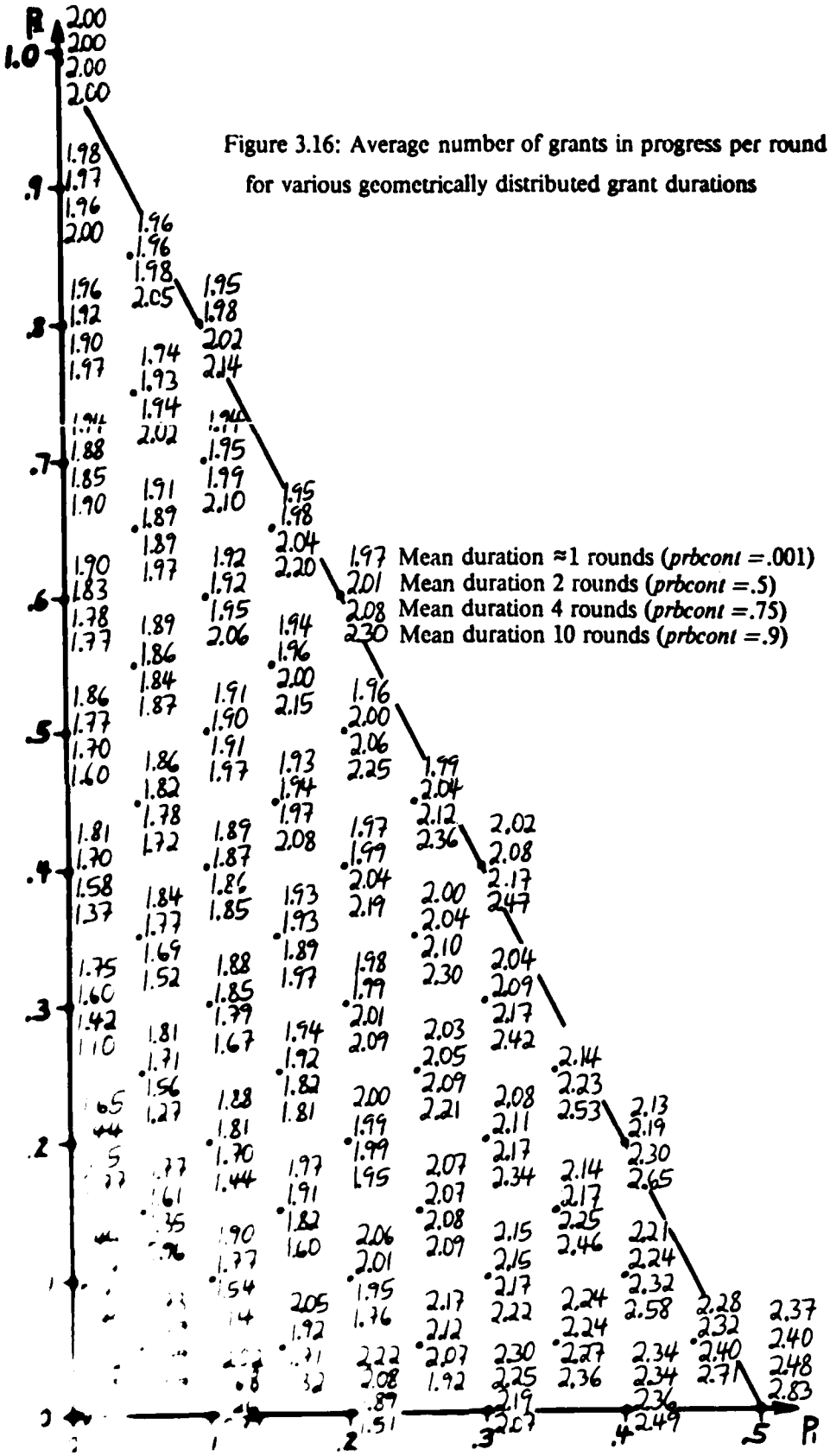


Figure 3.16: Average number of grants in progress per round for various geometrically distributed grant durations

As we found for [illegible] grant durations [illegible] progress [illegible] value of p_0. W[illegible] closely with that [illegible] value of t (which [illegible] tion of d rounds (which [illegible] g^d and g^t increased to [illegible] makes sense. Since the grant duration [illegible] tional constraints on which requests [illegible] decision is made. These constraints generally reduce [illegible] durations.

As with deterministic grant durations, the estimated optimal [illegible] the maximum reward in some states. The states in which this phenomenon [illegible] the pattern of having from one to three nonnull requests of which at most [illegible] granted so far. Included in these states were those in the first of the two main classes described for deterministic grant durations but not those in the second of the two main classes. An explanation for this exclusion of the second main class is that the time required until all grants currently in progress terminate is unknown and possibly very large. The number of states in which the estimated optimal decision is to grant less than the maximum reward seems to increase with both p_0 and d (i.e. *pbcomt*).

3.4.4 Other Grant Duration Distributions

We cannot say much about the effect of other grant duration distributions without further study. However, we can give the following generalities about the optimum throughput with any grant duration distribution:

1) Along any ray for which the nonnull probabilities have some fixed ratio, t^{opt} is monotonic in $1-p_0$ everywhere along the ray and convex down in $1-p_0$ within any subregion (i.e. within any region in which one particular policy is optimal) along the ray. The argument to support these two conclusions is the same as that given in section 3.4.2.1.

2) If $t_{\vec{p}}^{d}$ denotes the optimal throughput with some grant duration distribution with mean $\bar{d}$ and some request probabilities $p_0, p_1, \cdots, p_{S/2}$ denoted by $\vec{p}$ and if $t_{\vec{p}}^{d=1}$ denotes the optimal throughput with a deterministic grant duration of one round and the same request probabilities denoted by $\vec{p}$, then

$$t_{\vec{p}}^{d} \leq t_{\vec{p}}^{d=1}$$

3.5 Optimal Arbitrator for Six Slices and Current Duration of One Second

[illegible] the state description is

$$[illegible] \quad (6)$$

where [illegible] as discussed in section [illegible]. This state description yields [illegible] states. By utilizing the rotational and flip symmetries discussed in section 3.3, the number of states can be reduced to 4003. We were able to develop a relatively fast program to solve the Markov decision problem on these 4003 states using value iteration and Odoni's bound as discussed in section 3.4. On a DEC VAX† 11/780 super minicomputer with one user, a single iteration of the value iteration algorithm through all 4003 states took about one minute. For each set of probabilities we ran the program until the optimal throughput was determined to within ± .005 (i.e. until $U(\alpha) - L(\alpha) < .01$ where $U(\alpha)$ and $L(\alpha)$ are the upper and lower bounds respectively on t^{opt} after n iterations). This required an average of about 9 iterations for each set of probabilities. Figure 3.17 shows the optimal throughput (to within ± .005) obtained for selected probabilities.

Figure 3.17 also shows the throughput estimated from the relation $t^{est} = \sum_i p_{(i)} q_i^{max}$ where q_i^{max} is the maximum reward in state i. In most cases t^{est} is a surprisingly good estimate of the optimal throughput t^{opt}. In every case investigated we found $.11 \leq t^{opt} - t^{est} \leq .35$. (Not all of these results are shown in Figure 3.17 to avoid cluttering the figure). In all cases, except for four along the edge $p_0 = 0$ and $p_2 = 0$ and one at $p_0 = 0$, $p_1 = .100$, $p_2 = .400$, and $p_3 = 0$, we found $t^{est} < t^{opt}$.

There are far too many states to determine and analyze the optimal decision regions as we did for four slices in section 3.3. Furthermore, the decisions determined by the value iteration do not necessarily comprise an optimal policy - they only comprise an estimate of the optimal policy‡ - so it is best not to examine them too closely. Thus we will only discuss the main trends in the decisions. We will also discuss the performance of some rule of thumb policies.

† DEC and VAX are trademarks of the Digital Equipment Corporation.

‡ As discussed in section 3.4, a policy determined via value iteration is optimal only in the sense that the throughput with that policy is within some interval, given by the Odoni bounds, of the optimal throughput.

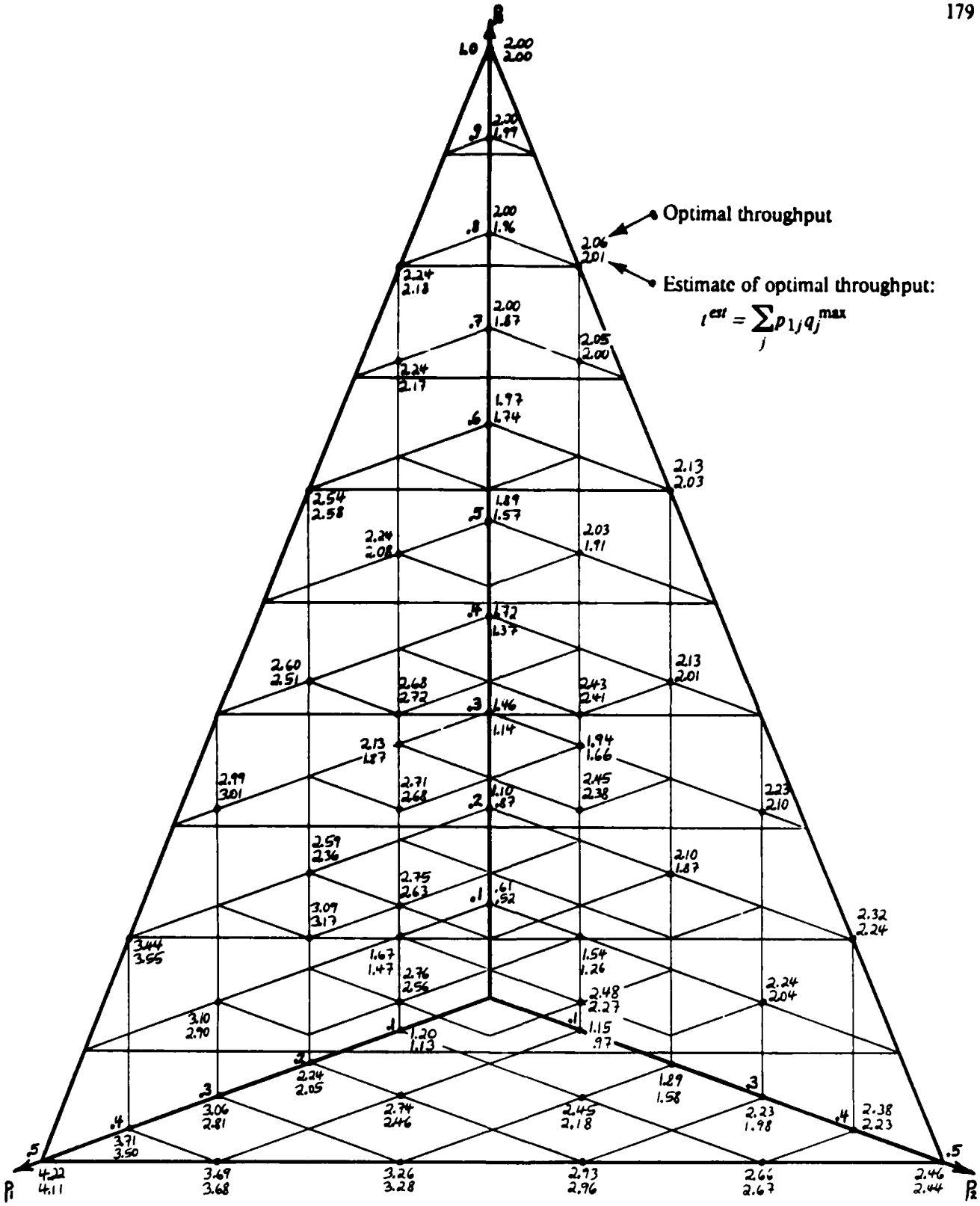


Figure 3.17: Optimal throughput (average number of grants per round) for Ringbus with six slices and one round grant durations

The most interesting trend in the decisions is that, unlike the case for four slices, it is not always best to grant the request subset with the maximum number of requests (i.e. reward). For every set of probabilities considered, we found at least one state in which the estimated optimal decision is to grant some request subset having less than the maximum reward. The number of states with such estimated optimal decisions is small for p_0 large (i.e. light traffic) and increases rapidly as p_0 decreases (i.e. as traffic increases). The most rapid increase of the number of these states as p_0 decreases occurs for probabilities in the p_2 - p_3 plane - i.e. for $p_1 = 0$.

One state in which we found the estimated optimal decision to grant less than the maximum reward is (-2, 3, -1, -1, 1, 1). The subset with maximum reward is (0, 0, -1, -1, 1, 1). However, for every set of probabilities we considered, the estimated optimal decision is to grant the subset (0, 3, 0, 0, 1, 1). The request set and these two subsets are pictured in the diagrams in Figure 3.18.

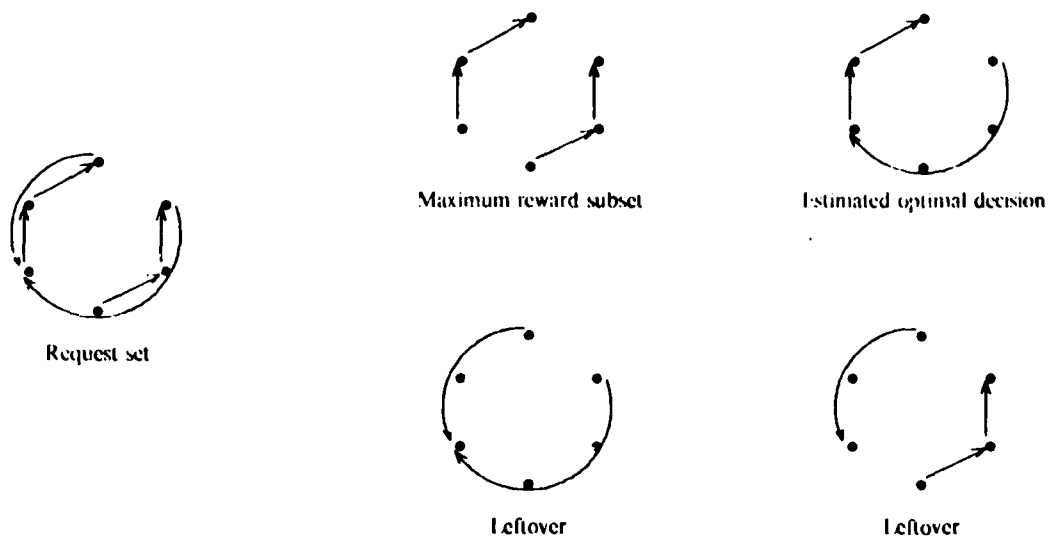


Figure 3.18: An example of a request set for which the estimated optimal decision is to grant less than the maximum number of requests

Note that the requests of length 2 and 3 conflict. Evidently this conflict reduces the value of the leftover of the maximum reward subset compared to the value of the leftover of the estimated optimal decision to a degree that cannot be overcome by the larger reward. An additional factor is that both requests in the leftover of the maximum reward subset are long requests. In heavy traffic long requests "cost" more than short requests to grant since they involve blocking a request from each of the one or two slices along the route which the long request is granted. This factor

seems to predominate in the states in which the estimated optimal decision is to grant less than the maximum reward: all such states have a mix of long and short requests. In states in which all nonnull requests are of the same length the estimated optimal decision is always to grant one of the request subsets with maximum reward. A state typical of those in which the estimated optimal decision is to grant less than the maximum reward is

$$(\ \ 2, 1, 3, 2, 3, 1).$$

The subset with maximum reward is $(0, 1, 0, 2, 0, 1)$ but for many sets of probabilities the estimated optimal decision is the subset $(\ \ 2, 0, 0, 0, 3, 0)$. A more glaring example is the state

$$(\ \ 1, \ \ 1, 3, \ \ 1, \ \ 1, 3).$$

The subset with maximum reward is $(\ \ 1, \ \ 1, 0, 0, \ \ 1, \ \ 1, 0)$, yet $(0, 0, 3, 0, 0, 3)$ is often estimated as the best subset. Note that the leftovers for both these subsets are immediately grantable.

To determine the significance of the fact that the estimated optimal decision often indicates that the request subset with maximum reward is not the best to grant, we modified our value iteration program to find the optimal throughput of the Ringbus with the additional constraint that the request subset granted in each state must have the maximum reward possible for that state. We call this the maximum reward constraint. Figure 3.19 shows the optimal throughput (to within $\pm.005$) of the Ringbus with this constraint for selected probabilities. The amount by which this throughput is less than that without the maximum reward constraint for a particular set of probabilities is indicated (to within 3 decimal places) by the quantity in the brackets.†

For most of the sets of probabilities investigated, and especially for light traffic (i.e. p_0 large), the optimal throughput of the Ringbus is not significantly affected by the maximum reward constraint. The most significant reduction caused by this constraint occurs mostly on the face $p_2 = 0$ and near the face $p_0 = 1$ for p_1 large. Another way to describe this region is that p_0 is rather small, p_1 is large, and p_2 is very small. In other words, traffic is fairly heavy and there is mainly short and long requests. Of the probability sets considered, the largest reduction in throughput - at least .057 - occurred at $p_1 = .4$, $p_2 = 0$, and $p_3 = .2$. The fact that the maximum reward strategy is not optimal in this region with heavy traffic and mainly short and long requests is easy to see.

† The quantity in the brackets is actually $L - U^{\max}$ where $U^{\max}$ is the upper bound on the optimal throughput with the maximum reward constraint and L is the lower bound on the optimal throughput without this constraint. Thus the actual difference in throughputs exceeds that indicated. Nothing is indicated inside the brackets if $L < U^{\max}$.

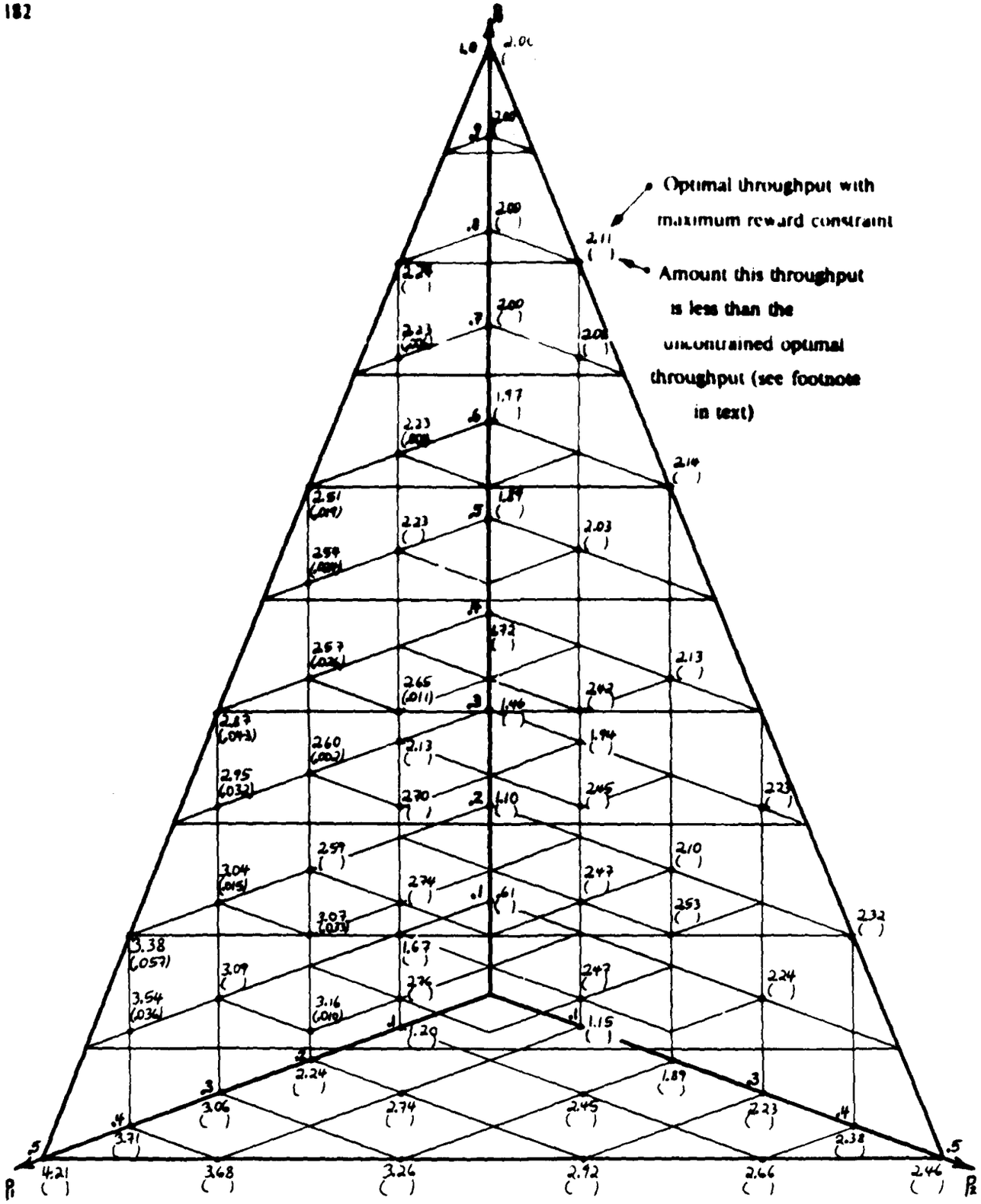


Figure 3.19: Optimal throughput, subject to the maximum reward constraint, with six slices and one round grant durations

Request subsets with only short requests will usually have a larger reward than subsets with a long request so a long request will tend to remain ungranted for a long time (until most of the other requests are also long). The slice which submitted the long request is prevented from submitting further requests. Thus a six slice Ringbus could be effectively reduced to five operating slices. This effect can be especially severe if $p_0 \approx 0$ and the long request is very rare compared to the short requests. In this case the long request will remain ungranted for a very long time and will prevent many potential short requests from its originating slice from being generated and contributing to the throughput. For $p_0 \approx 0$, $\frac{p_1}{1-p_0} \approx .5$, $p_2 = 0$, and $\frac{p_4}{1-p_0} \approx 0$, the optimal decision is obviously to grant any long requests first.‡ On the other hand, for $p_0 \approx 1$, $\frac{p_1}{1-p_0} \approx .5$, $p_2 = 0$, and $p_4 \approx 0$, any long request can just wait (a short time on average) until there are no other requests to conflict with the long request and thus the optimal decision is to grant the maximum number of requests.

An examination of the estimated optimal decisions revealed that the request subset chosen often utilized the maximum number of segments. This tendency seemed particularly strong in those states for which the estimated optimal decision was not the maximum reward request subset. To establish the merit of the maximum number of segments strategy, we modified our value iteration program to find the optimal throughput of our Ringbus model with the additional constraint that the request subset granted in each state must utilize the maximum number of segments possible for that state. In computing the number of segments a request subset requires, we use the number of segments that each request would require if it were granted in the shortest direction around the Ringbus. Thus the number of segments that a request subset utilizes is equal to the sum of the request lengths for those requests granted. Figure 3.20 shows the optimal throughput (to within ±.005) of the Ringbus with the maximum number of segments constraint for various probabilities. As for Figure 3.19, the amount that this throughput is less than the unconstrained optimal throughput (displayed in Figure 3.17) for a particular set of probabilities is indicated (to within 3 decimal points) by the quantity in the brackets.†

‡ The throughput listed beside the point $p_1 = .5$, $p_2 = p_3 = 0$ in Figures 3.17 and 3.19 is actually for the point $p_1 = .498$, $p_2 = p_3 = .001$. (All zero probabilities were replaced with very small probabilities so that the same state space and same program could be used to calculate all throughputs for six slices without possible problems caused by noncommunicating states. All states communicate if $p_i \geq 0$ for $1 \leq i < S/2$.) This accounts for the apparent contradiction between our earlier observation that the estimated optimal decision is to grant the maximum reward request subset in states with all nonnull requests of the same length and the fact that the throughput listed in Figure 3.19 for the point $p_1 = .5$, $p_2 = p_3 = 0$ with the maximum reward constraint is less than optimal. A separate analysis confirmed that exactly at the point $p_1 = .5$, $p_2 = p_3 = 0$, the optimal policy grants the maximum reward subset in every state. (Of course, exactly at the point $p_1 = .5$, $p_2 = p_3 = 0$, all states have all requests of the same length.)

† The quantity in the brackets is actually $L - U^{max}$, where U^{max} is the upper bound on the optimal throughput with the maximum segment constraint and L is the lower bound on the optimal throughput without this constraint. Thus the actual difference in throughputs exceeds that indicated. Nothing is indicated inside the brackets if $L < U^{max}$.

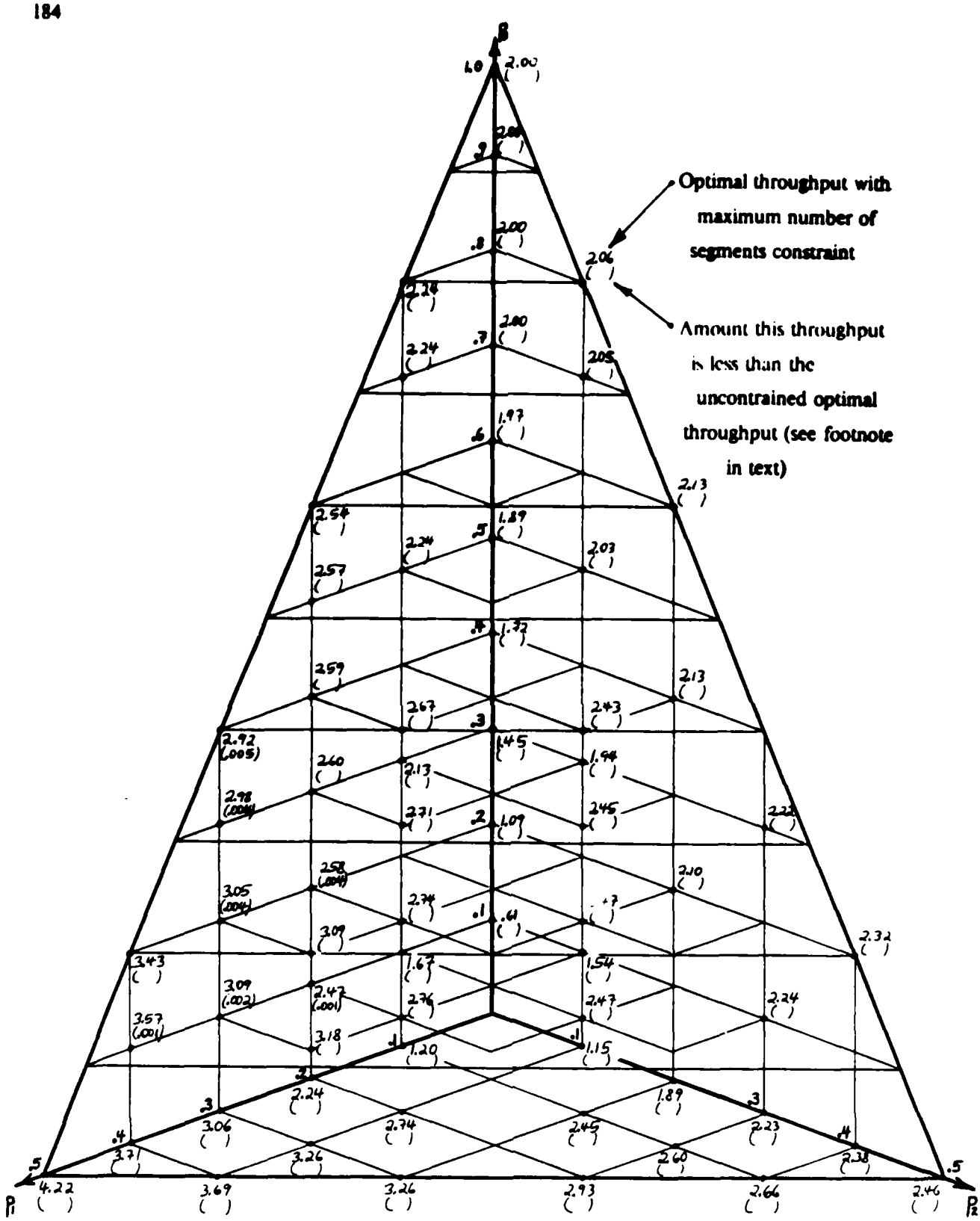
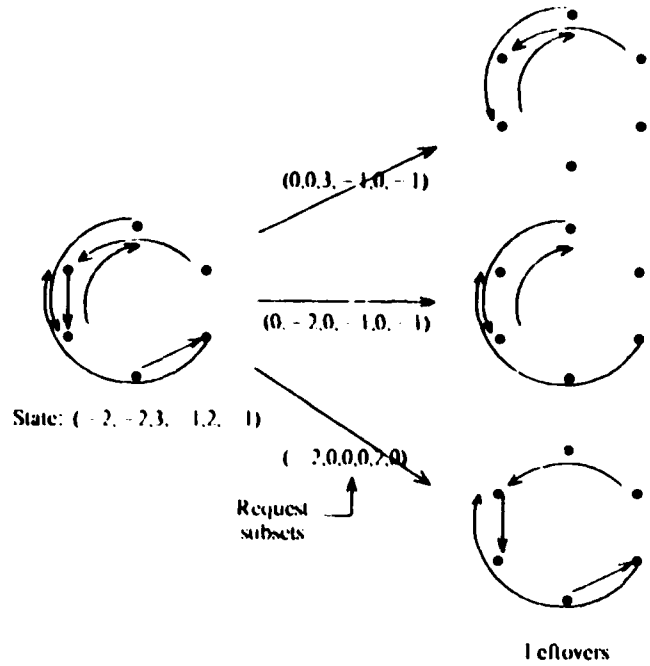


Figure 3.20: Optimal throughput, subject to the maximum number of segments constraint, with six slices and one round grant durations

For all the sets of probabilities investigated, the maximum number of segments constraint only caused a notable reduction in throughput for p_1 large, $p_2 \approx 0$, and p_3 small to medium large. This is the same region (heavy traffic, mainly short and long requests) for which the maximum reward constraint caused the most significant reduction in throughput. However, the maximum number of segments constraint never caused as large a reduction in throughput as the maximum reward constraint. In fact, the reduction in throughput with the maximum number of segments constraint provides an efficient compromise between the conflicting desires to grant the maximum number of requests in a state and minimize the waiting time of long requests.

One might conjecture that the optimal policy grants the request subset in each state with either the maximum reward or the maximum number of segments. However, this conjecture seems to be false in general. It is indeed true that for most states and for most probabilities, the estimated optimal decisions correspond to either the maximum reward or the maximum number of segments (or both) request subsets. As p_0 decreases and p_3 increases, the number of states in which the estimated optimal decision corresponds to neither maximum reward or maximum number of segments increases, but it never exceeds about 90 states. Two typical states in which the estimated optimum decision is often neither the maximum reward nor maximum number of segments subsets are $(-2, -2, 3, -1, 2, -1)$ and $(-2, 3, -1, -1, 1, -1)$. For the former state, the request subsets $(0, 0, 3, -1, 0, -1)$ and $(0, -2, 0, -1, 0, -1)$ achieve the maximum reward and the request subset $(0, 0, 3, -1, 0, -1)$ uniquely achieves the maximum number of segments. For the latter state, the request subsets $(0, 3, -1, -1, 0, 0)$ and $(2, 0, -1, -1, 0, 0)$ achieve the maximum reward and the request subset $(0, 3, -1, -1, 0, 0)$ uniquely achieves the maximum number of segments. However, the estimated optimum decision in these two states is often $(-2, 0, 0, 0, 2, 0)$ and $(0, 3, 0, 0, 1, 0)$ respectively. Figure 3.21 depicts diagrams of these various possible decisions in the two states.



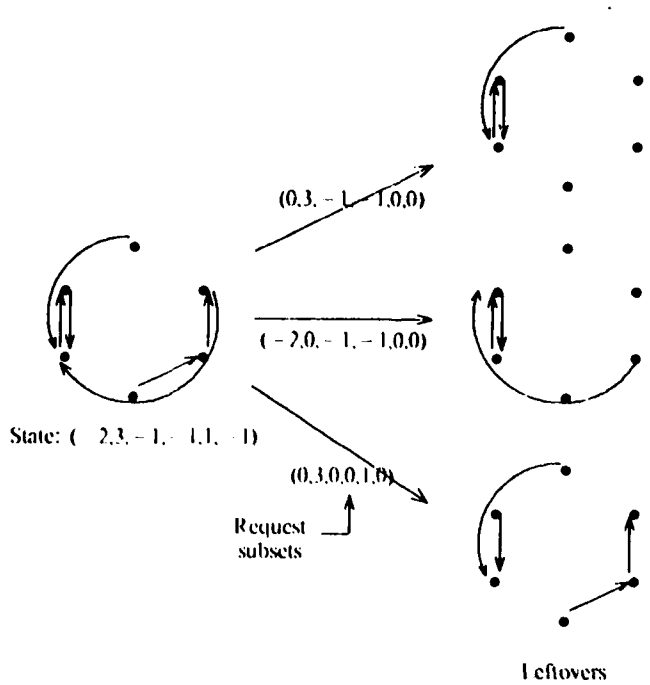


Figure 3.21: Some possible decisions

The request subsets (0, 2, 0, 0, 0, 2, 0) and (0, 3, 0, 0, 1, 0) allow at least 3 requests to be granted in the round following the states (0, 2, 2, 3, 1, 2, 1) and (0, 2, 3, 1, 1, 1, 1) respectively. All the other request subsets allow at least 2 requests. Thus depending on the probabilities of the various requests, the effect of not granting the maximum number of requests in the subset (0, 2, 0, 0, 0, 2, 0) and (0, 2, 0, 0, 1, 0) can be compensated to some degree by possibly granting more requests in the following round. Of course, the relative values $v_j - v_1$ of Howard's policy iteration algorithm (which can also be estimated by $V_j(n) - V_1(n)$ in the value iteration algorithm) give the exact degree to which one request subset is preferable over another.

A possible rule of thumb for the decision in each state so as to achieve near-optimal throughput of the Ringbus is to grant some request subset utilizing the maximum number of segments. As we discussed earlier, the maximum number of segments constraint only slightly affects the optimal throughput. A more precise rule of thumb policy that we investigated is the following. In each state grant some request subset that:

1. utilizes the maximum number of segments,
2. has the maximum number of requests subject to 1, and
3. has the maximum number of the longest requests subject to 1 and 2 (i.e. a request subset with requests of length 1, 2, and 3 is preferable to one with requests of length 2, 2, and 2).

Constraint 2 serves mainly to reduce the number of eligible request subsets in each state while keeping the reward large. Constraint 3 ensures that long requests are granted before shorter ones (for subsets meeting constraints 1 and 2).

We investigated this rule of thumb policy by determining the estimated optimal throughput subject to these three constraints for the 91 sets of probabilities with p_1, p_2, and p_3 some integral multiple of .1. (We used these same sets of probabilities whenever we calculated the throughput for any variation of the Ringbus model with six slices). For every set of probabilities considered, the estimated optimal throughput with these constraints was close (within $= -.009$) to the estimated optimal throughput with just the maximum number of segments constraint. Furthermore, in the vast majority of states there is only one request subset that meets constraints 1, 2, and 3. Thus, these constraints function well in reducing the number of possible decisions in each state without affecting the throughput by much. Quite a few states remain, however, for which there is still more than one request subset meeting the three constraints. An examination of these states revealed that for most states these remaining subsets are either related by symmetry or nearly identical. We believe that the throughput would remain essentially the same if for each state, the request subset is selected arbitrarily from those meeting all three constraints. For that matter, we suspect that the throughput would remain approximately the same if for each state the request subset is selected arbitrarily from all those meeting the maximum number of segments constraint.

3.5.1 Bounds on the Optimal Throughput

We now discuss three different bounds (all upper bounds) on the optimal throughput of the Ringbus. Caveats are:

1) the Ringbus has symmetrical access paths - i.e. it is a Symmetric Ringbus,

2) all slices have identical request probabilities and geometrically distributed processing times (as assumed in section 3.1), and

3) the duration of all grants is a single round.

All of the bounds can be extended to deal remove these restrictions. However, all these extensions (except from a symmetric to a non-symmetric Ringbus), complicate the calculation of the bounds and thus makes the bounds less attractive.

3.5.1.1 Flow Model Bound

Denote the rate at which requests - null and nonnull - arrive at the Ringbus (in number of requests per round) from slice i by λ_i. Because of our symmetry assumptions, λ_i is the same for all slices, thus we simply denote the rate by λ. The rate at which nonnull requests arrive at the Ringbus from a slice is $(1-p_0)\lambda$. Therefore the throughput of the Ringbus is $S(1-p_0)\lambda$ where S is the number of slices.

We now consider the rate at which requests are granted from a slice for various destinations. This rate may be likened to a flow: nonnull requests flow into the Ringbus from one slice at the rate $(1-p_0)\lambda$. The flow from a slice to a destination i segments away is $p_i\lambda$.† We assume that all requests of length $0<i<S/2$ are granted in the shortest direction and that requests of length $S/2$ are granted in the clockwise direction. Thus this flow divides in accordance with the clockwise or counterclockwise position of the destination relative to the source.

The total clockwise flow over a particular segment is $(1-p_0)\lambda\sum_{i=1}^{S/2}\frac{i\,p_i}{(1-p_0)}=\lambda\sum_{i=1}^{S/2} i\,p_i$. Similarly the total counterclockwise flow over a particular segment is $(1-p_0)\lambda\sum_{i=1}^{S/2-1}\frac{i\,p_i}{(1-p_0)}=\lambda\sum_{i=1}^{S/2-1} i\,p_i$. Thus the total flow over a particular segment is

$$(1-p_0)\lambda\left[\sum_{i=1}^{S/2-1}\frac{2\,i\,p_i}{(1-p_0)}+\frac{S/2\,p_{S/2}}{(1-p_0)}\right]=(1-p_0)\lambda\bar{l}$$

where $\bar{l}$ is the average length of a request (in terms of the number of hops or segments required)

† The probability that a request is for a destination i segments away from the source, given that the request is nonnull, is $\frac{p_i}{(1-p_0)}$, yielding a flow of $\frac{p_i}{(1-p_0)}(1-p_0)\lambda=p_i\lambda$.

given that the request is nonnull. By symmetry arguments, the flow is identical on all segments.

The total flow on any segment must not exceed 1 (i.e. one grant per round). Thus $(1-p_0)\lambda l \leq 1$ and therefore

$$t^{opt} \leq \frac{S}{l}.$$

This bound is best for heavy traffic - i.e. $p_0 \approx 0$ - but even then it is not that good.

The throughput of the Ringbus can be written as

$$t = \frac{S}{\frac{p_0}{1-p_0} + \bar{w} + 1}$$

where $\frac{p_0}{1-p_0}$ is the average processing time (in rounds) and $\bar{w}$ is the average waiting time of a request (again in rounds). Since $\bar{w} \geq 0$, we have

$$t \leq S(1-p_0),$$

yielding a tighter bound for light traffic, i.e. $p_0 \approx 1$. Thus

$$t^{opt} \leq S \min\left[\frac{1}{l}, (1-p_0)\right] \tag{3.19}$$

The effects of segment and/or destination conflicts must be included to get more useful bounds.

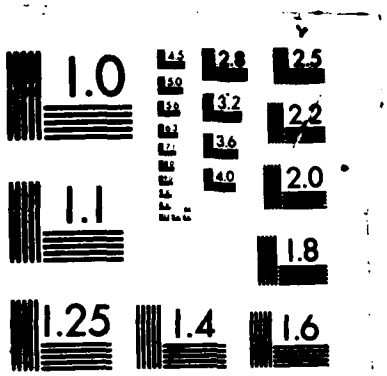
3.5.1.2 Crossbar Bound

An alternative way to obtain an upper bound on the throughput of the Ringbus is to consider a simpler model. One such simpler model is to consider only the destination of a request - in other words, ignore the segments that a request requires. For S slices, single round grant [illegible] tions, and ignoring request waiting times, the state description of such a model is

$$(r_1, r_2, \cdots, r_S)$$

where r_i is the destination (1, 2, $\cdots$, or S) of the request at slice [illegible] null request at slice i. Alternatively, the destination may be expr[illegible] slices the destination slice is around the Ringbus [illegible] r_i [illegible] $(S/2-1), \cdots, -1, 0, 1,$ [illegible] or S where [illegible] wise direction and a positive quantity [illegible] request, wh[illegible] is not the same [illegible] resources. Since each of [illegible]

AD-A103 619 MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR 3/4
(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
COMPUTER SCIENCE R B OSBORNE MAY 87 MIT/LCS/TR-375
UNCLASSIFIED N00014-83-K-0125 F/G 12/7 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

can be only destination conflicts, this simpler model can be viewed as $S \times S$ nondiagonal crossbar interconnection. (Nondiagonal means that there are no crosspoint switches along the major diagonal.) This crossbar model has the same state description as the Ringbus model discussed in the beginning of section 3.5. The only difference between the two models is in the constraints. The Ringbus model has segment and destination requirements for each request and the crossbar model has only destination requirements. Thus the crossbar model has fewer constraints on which requests may be granted simultaneously i.e. it has more immediately grantable request sets and fewer request conflicts.

Therefore merely by changing what constitutes a grantable request subset (a request subset in which all requests are grantable), the same computer program can be used to determine the optimal throughput for both the Ringbus and crossbar models. Figure 3.22 shows the optimal throughput for selected probabilities for the Ringbus and crossbar.

The optimal throughput of the Ringbus is close to that for the crossbar when p_0 is large (i.e. light loading) and when p_1 is large. For most other probability sets, and especially for large p_3, the throughput of the crossbar exceeds that of the Ringbus by a great deal. This is to be expected since the crossbar does not have any of the segment conflicts which comprise the majority of the conflicts in the Ringbus.

The chief value of the crossbar bound is to allow a comparison between the performance of the Ringbus interconnection scheme and that of a crossbar interconnection, which has the best performance achievable.† The crossbar bound is, of course, a bound on the optimal throughput of the Ringbus, but it is as difficult to compute as the optimal throughput of the Ringbus itself (since both the Ringbus and crossbar models have the same large state space).

† Where the interconnection must be circuit-switched with S sources and S destinations.

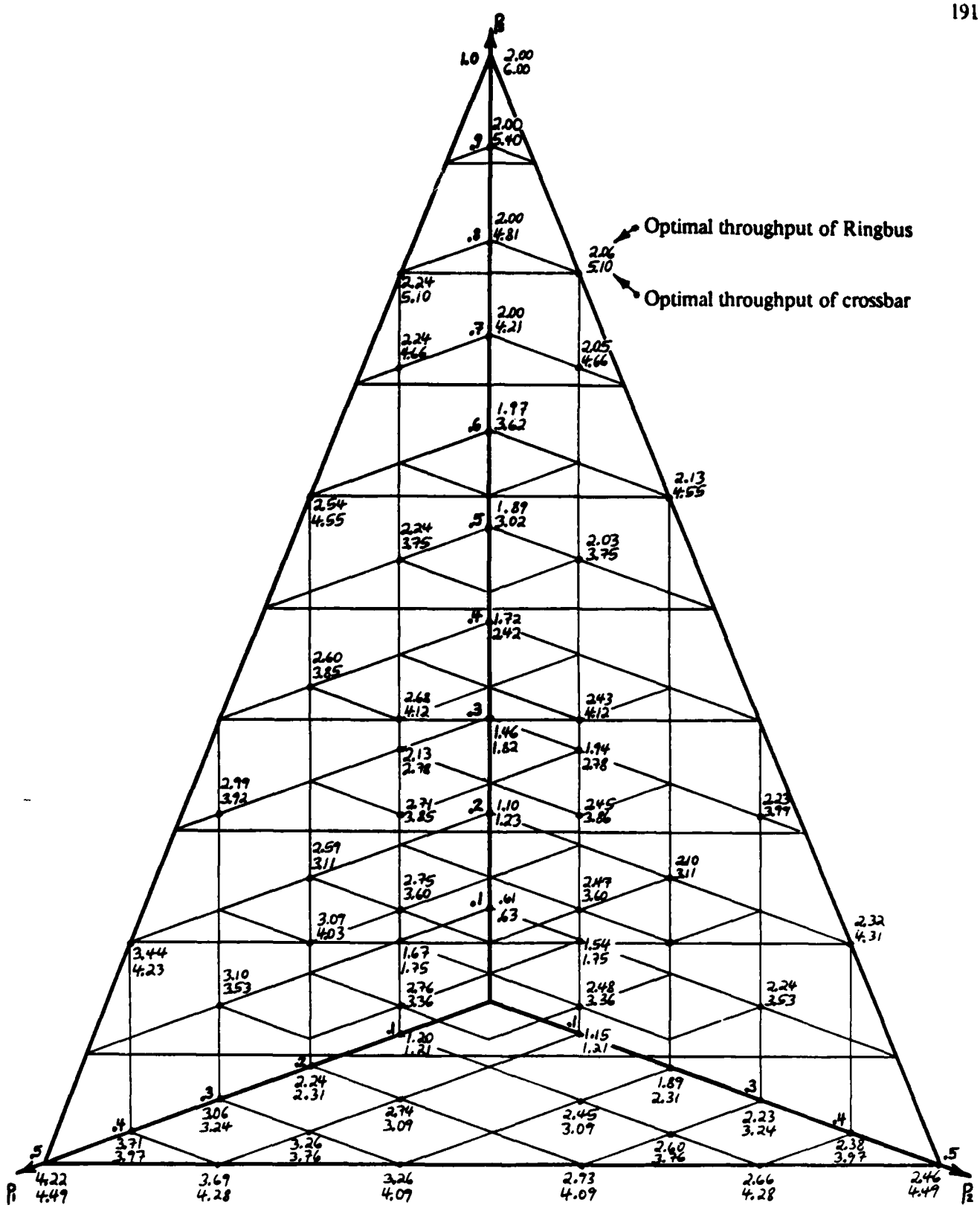


Figure 3.22: Optimal throughput for crossbar and Ringbus, each with six slices and one round grant durations

3.5.1.3 Number of Segments Bound

Another simple model of the Ringbus is to consider only the segments required by each request and ignore the destination of each request. This model captures the essence of the Ringbus better than the crossbar model but it still has the same large state space and thus is useless for obtaining a practical bound. In order to reduce the size of the state space we consider an even simpler model of the Ringbus. Now we consider only the number of segments required by each request and ignore the particular segments and destination required by each request. For S slices, single round grant durations, and ignoring request waiting times, the state description of this model reduces to

$$(m_0, m_1, m_2, \cdots, m_{S/2})$$

where m_0 is the number of null requests, m_i, for $1 \leq i \leq S/2$, is the number of requests requiring i segments, $0 \leq m_i \leq S$, for $0 \leq i \leq S/2$, and $\sum_{i=0}^{S/2} m_i = S$. The only constraint on granting requests is that the total number of segments required by the requests not exceed the number of segments S. Thus a state is immediately grantable if $\sum_{i=1}^{S/2} i\, m_i \leq S$. The total number of states is

$$\binom{1 + S/2 + S - 1}{S} = \binom{S + S/2}{S}.$$

For $S = 6$ this model has 84 states as compared to the 4003 states of the original Ringbus model (after symmetry is removed).

Figure 3.23 shows the optimal throughput of this model, which we call the number of segments model, and the optimal throughput of the Ringbus for various request probabilities. The number of segments model yields an excellent upper bound on the optimal throughput for light traffic (i.e. p_0 large) and for $p_3 \geq .8$. The quality of the bound degrades as p_2 and especially as p_1 increases. This performance is to be expected since the number of segments model ignores destination conflicts and the particular segments required by each request. These two factors dominate the performance of the Ringbus for heavy traffic and short request lengths. The bound is worst for $p_1 = .5$, $p_2 = p_3 = 0$. At this point, the number of segments model gives a bound of 6.0 on the optimal throughput, whereas the optimal throughput of the Ringbus at this point is 4.22 (grants per round).

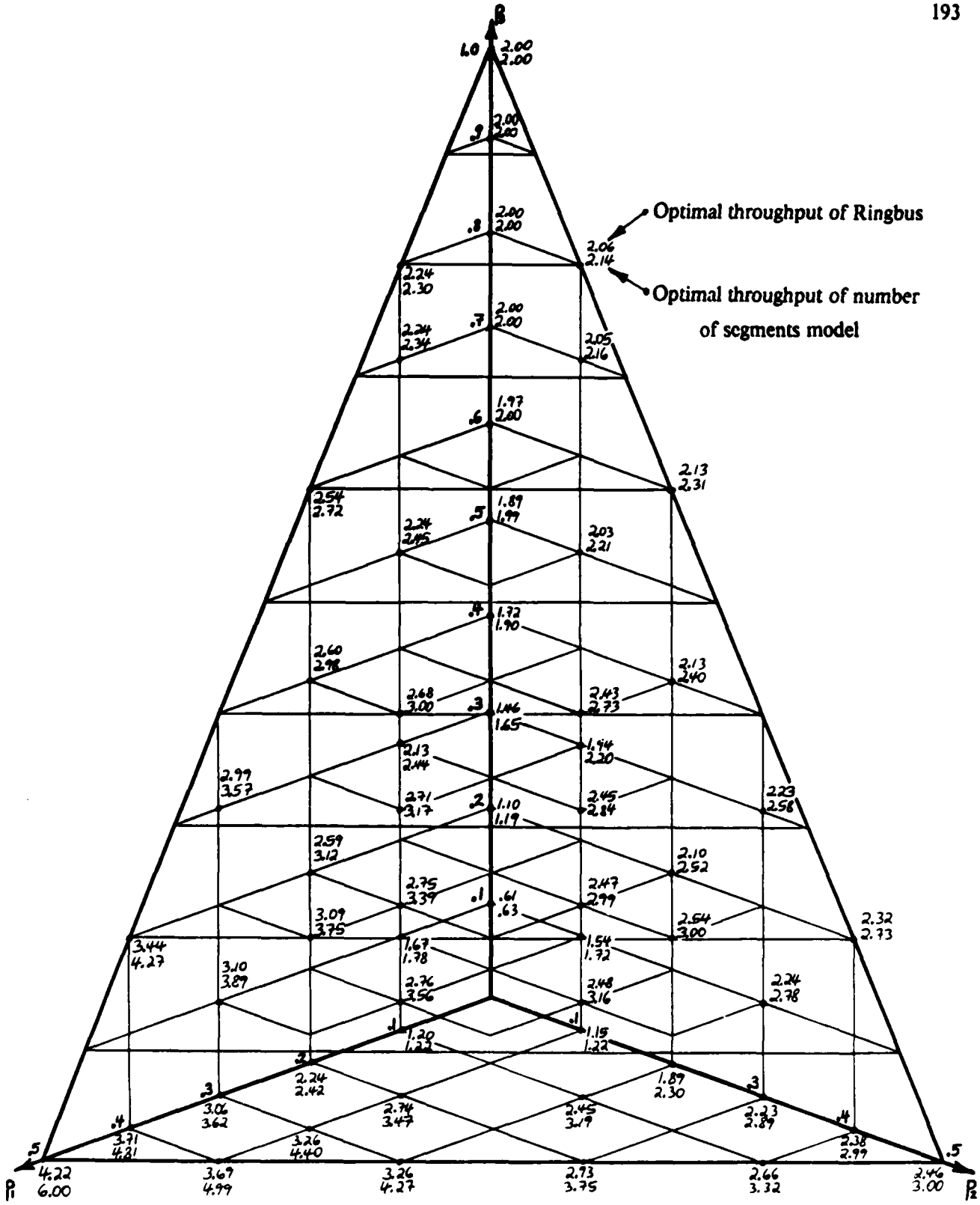


Figure 3.23: Optimal throughput of the number of segments model

An examination of the estimated optimal decisions in each state of the number of segments model revealed the same general trend as those in the Ringbus model: request subsets with long requests (i.e. requests requiring many segments) were increasingly favoured over ones with only shorter requests, as the traffic increased (i.e. as $p_0 \rightarrow 0$). This trend was most pronounced when p_1 was large, $p_2 = 0$, and p_3 small.

We computed the optimal throughput of the number of segments model subject to the two different constraints investigated earlier for the Ringbus model: the maximum reward and maximum number of segments constraints. Our findings again parallel that discussed earlier for the Ringbus model. The optimal throughput with the maximum number of segments constraint was indistinguishable (within the $\pm .005$ tolerance range on the optimum from the value iteration algorithm) from the unconstrained optimal throughput. The optimal throughput with the maximum reward constraint was less than the unconstrained optimal throughput in about the same region for which the optimal throughput of the Ringbus model with the maximum reward constraint was less than the unconstrained optimal throughput of the Ringbus model. (See Figure 3.19 for this latter region.)

3.5.1.4 Discussion

There is usually a tradeoff between the tightness of a bound and the ease of its calculation. Tight bounds tend to be complex and difficult to calculate while loose bounds tend to be simple and easy to calculate. Unfortunately, the Ringbus model is very complex as evidenced by its large number of states. This suggests that any really tight bounds on the throughput of the Ringbus in all cases will also be very complex and difficult to calculate.

The bounds we investigated are examples of the spectrum of the tradeoff between tightness of a bound and its ease of calculation. The average number of segments bound is simple but not very accurate. The crossbar bound is extremely difficult to calculate (as difficult as the optimum Ringbus throughput itself). The main purpose of the crossbar bound is to provide the performance of the best possible interconnection network for comparison with the performance of the Ringbus. The number of segments bound is the best of the three different bounds investigated, except when p_1 is large, in which case it is the worst bound.

The number of segments bound has a further significant advantage over the other bounds: it yields some idea of the optimal decisions in the Ringbus model.

3.6 Optimal Arbiter for Eight Slices

In this case the state description with grant durations of one round is

$$(r_1, r_2, r_3, \ldots, r_8)$$

where $r_i = -3, -2, -1, 0, 1, 2, 3$, or 4 as discussed in section 3.2. This yields $8^8 = 16{,}777{,}216$ states. By utilizing rotational and flip symmetry in the state description, the number of states can be reduced by a factor of less than 16†, which still yields over 1,000,000 states. Needless to say, this huge number of states makes the pursuit of the optimum throughput and corresponding optimum policy very difficult for general request probabilities. Based on our experience with the value iteration algorithm for determining the optimum throughput with six slices, we concluded that such an algorithm would be impractical for eight slices with the computational resources available to us. The optimum throughput can still be determined rather easily for some special cases with a small number of states.

One special case that we investigated is the optimum throughput along the axes of the feasible probability region (i.e. only one request probability nonzero). Figure 3.24 shows the optimum throughput along each axis of the feasible probability region. Another special case is the optimum throughput on a face of the feasible probability region (i.e. with only two request probabilities nonzero). We did not investigate this case.

Bounds and approximations are the only practical methods to obtain some idea of the optimum throughput for general request probabilities. However, some idea of the general characteristics of the throughput is also useful. We discuss such characteristics in section 3.6.1. Any of the bounds discussed in section 3.5.1 can be applied, although the Markovian decision formulation bounds and the crossbar bound are not very practical due to their large computational requirements. We examine the number of segments bound in section 3.6.2. One simple approximation is to replace all nonnull requests by requests of a single length closest to the mean request length (given that a nonnull request occurs) $\bar{l} = \dfrac{2p_1 + 4p_2 + 6p_3 + 4p_4}{(1 - p_0)}$. Another approximation is $t^{opt} \approx \sum_j p_{1j} q_j^{\max}$. We expect this to be an excellent approximation again but it is rather difficult to calculate. The difficulty is in determining $q_j^{\max}$ in each of the 8^8 states; p_{1j} is trivial to determine.

† At best, 16 states - corresponding to 8 rotations and 2 flips - can be reduced to one state. This reduction factor can only be attained for certain states with zero nonnull requests and zero requests of length 4.

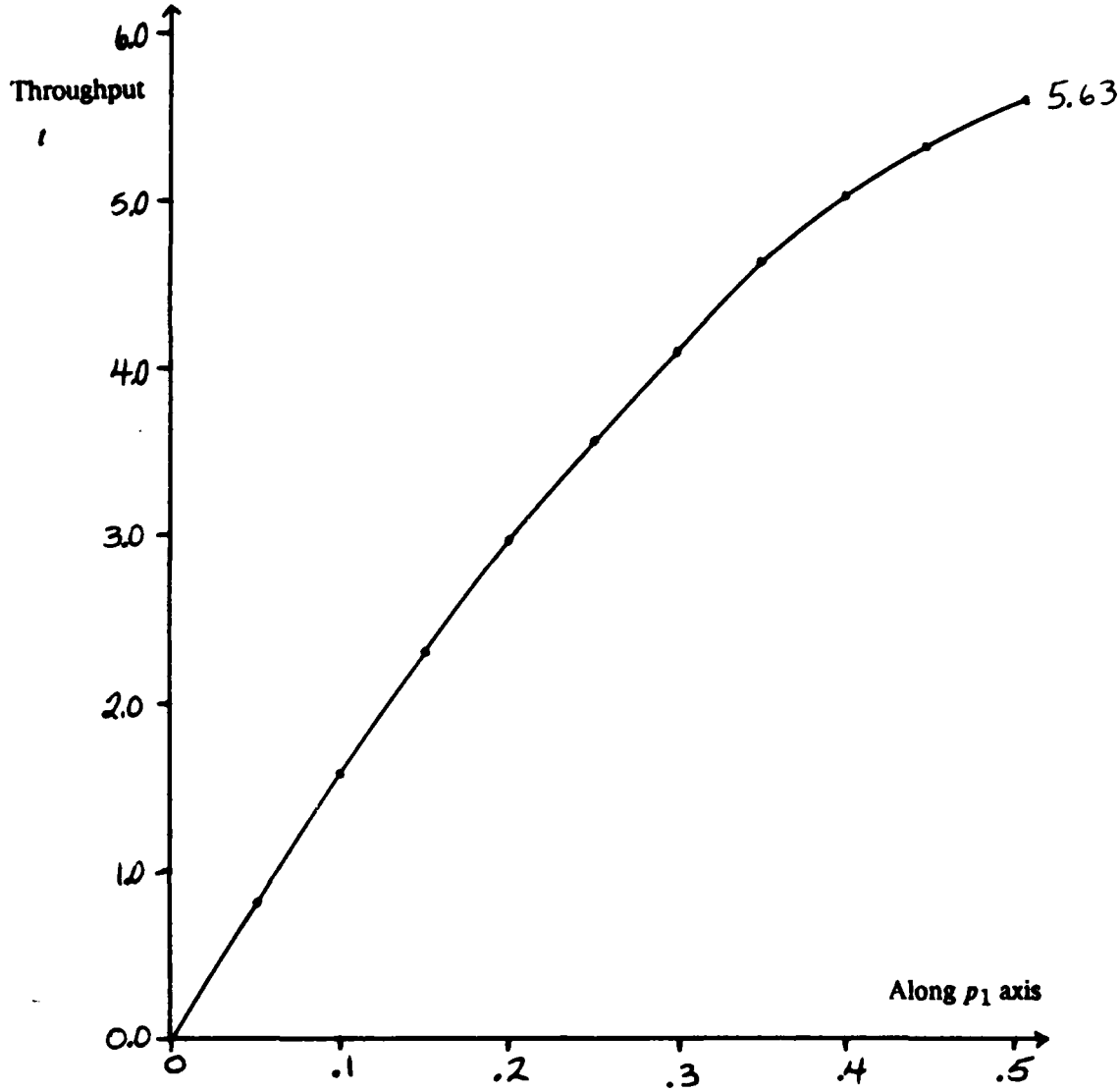
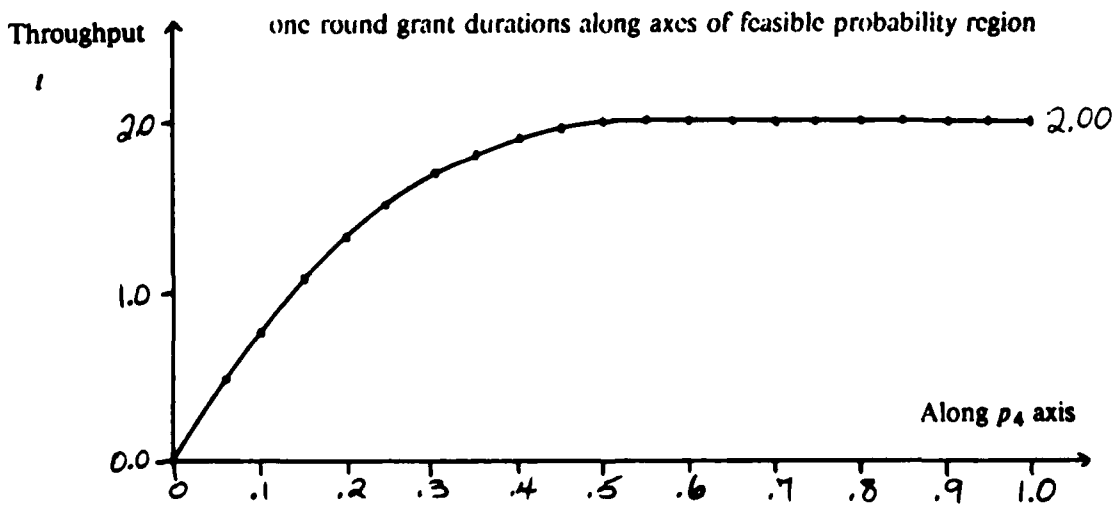
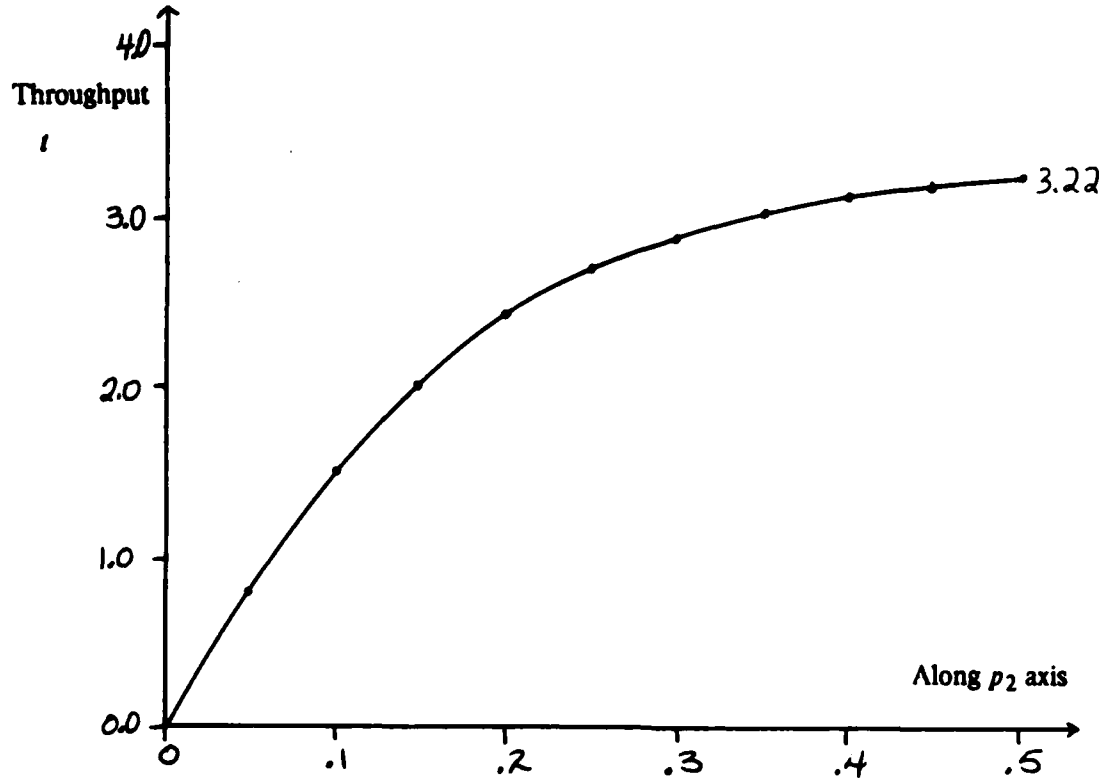
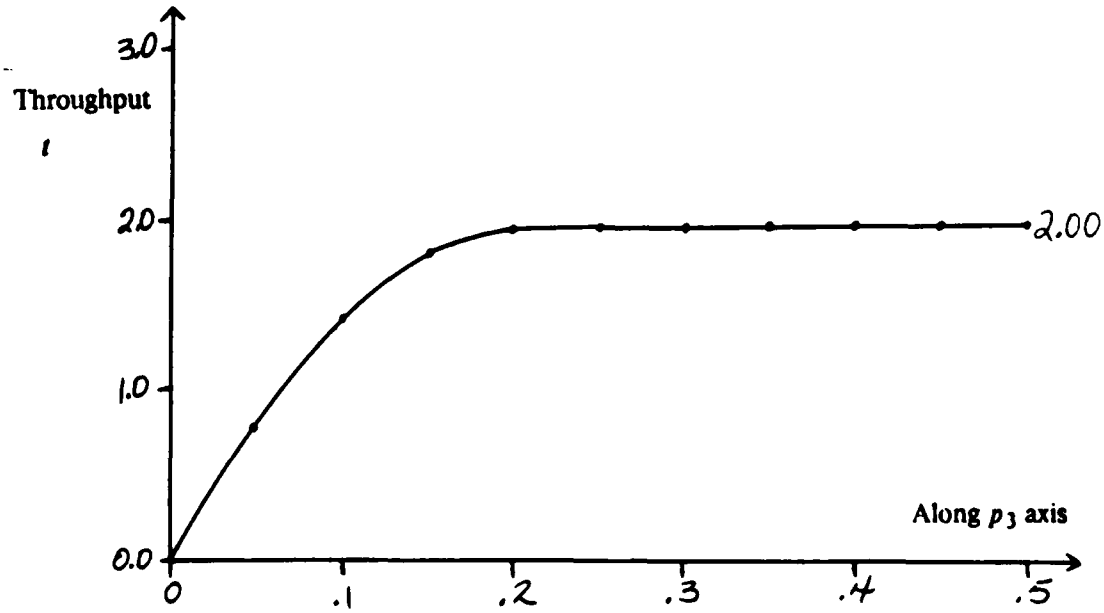


Figure 3.24: Optimal throughput of Ringbus with eight slices and one round grant durations along axes of feasible probability region




4.0
Throughput
3.0
2.0
1.0
0.0
3.22
Along p_2 axis
0
.1
.2
.3
.4
.5


3.0
Throughput
2.0
1.0
0.0
2.00
Along p_3 axis
0
.1
.2
.3
.4
.5

3.6.1 General Characteristics of the Optimum Throughput

The optimum throughput is a function of the request probabilities i.e. $t^{opt}(p_1,p_2,p_3,p_4)$. In this section we consider the general shape of this function.

3.6.1.1 Slope for Very Light Traffic

From equation 3.10 we have $t^{opt} \approx \sum_k p_{1k}(v_k^{opt} - v_1^{opt})$. For very light traffic, i.e. $p_0 \approx 1$, $v_k^{opt} - v_1^{opt} \approx n_k$ where n_k is the number of nonnull requests in state k. This can be seen from equation 3.13:

$$v_k^{opt} - v_1^{opt} = q_k^{opt} + \sum_{m=0}^{\infty} \sum_j (p_{kj}^{opt} - p_{1j}) \sum_l \varphi^{opt}{}_{jl}(m) q_l^{opt}.$$

For $p_0 \approx 1$, $p_{kj}^{opt} \approx \begin{cases} 1 & \text{if } j \text{ is a leftover of state } k \\ 0 & \text{otherwise} \end{cases}$ and $p_{1j} \approx \begin{cases} 1 & j=1 \\ 0 & j \neq 1 \end{cases}$ (where state 1 is the state with all null requests). Of course $q_1^{opt} = 0$. Thus

$$v_k^{opt} - v_1^{opt} \approx q_k^{opt} + q_l^{opt} + q_m^{opt} + \cdots = n_k$$

where state l is the leftover of state k, state m is the leftover of state l, etc. until the leftover is state 1 with all null requests. Therefore for $p_0 \approx 1$ we have

$$t^{opt} \approx \sum_k p_{1k} n_k$$

Now if $p_i = \delta$ for some $1 \leq i < S/2$ where δ is very small and positive and $p_j = 0$ for all $j \neq 0$ and $j \neq i$, then

$$p_{1k} = \begin{bmatrix} S \\ n_k \end{bmatrix} (2\delta)^{n_k} (1-2\delta)^{S-n_k}$$

Therefore $t^{opt} \approx 2S\delta$ and thus $\frac{\partial t^{opt}}{\partial p_i} \approx 2S$. Taking the limit as $\delta \to 0$, we have $\frac{\partial t^{opt}}{\partial p_i}\Big|_{p_0=1} = 2S$ for $1 \leq i \leq S/2$.

If $p_{S/2} = \delta$ where δ is very small and positive and $p_j = 0$ for all $j \neq 0$ and $j \neq i$, then

$$p_{1k} = \begin{bmatrix} S \\ n_k \end{bmatrix} \delta^{n_k} (1-\delta)^{S-n_k}$$

Therefore $t^{opt} \approx S\delta(1-\delta)^{S-1} \approx S\delta$ and thus $\frac{\partial t^{opt}}{\partial p_{S/2}} \approx S$. Taking the limit as $\delta \to 0$, we have $\frac{\partial t^{opt}}{\partial p_{S/2}}\Big|_{p_0=1} = S$.

Note that these slopes are reflected in the drawings in Figure 3.24.

3.6.1.2 Shape Along a Ray with Fixed Ratio of Nonnull Probabilities

For any arbitrary value of S the characteristics of the shape along a ray are similar to those discussed in section 3.4.1.2 for four slices.

3.6.1.3 Maximum Points

At any point in the feasible probability region, the throughput increases if p_1 increases by some positive amount δ. (This may require that the probability of other request lengths decrease.) Thus there are no maxima in the interior of the feasible probability region; the maximum must occur on the boundary. Obviously, the unique maximum occurs at $p_1=.5$ and the unique minimum occurs at $p_0=1.0$.

3.6.1.4 Shape Along Cross Sections

The throughput increases monotonically along any cross section parallel to the p_1 axis since $\frac{\partial t}{\partial p_1}>0$. ($t$ is the throughput.) Along other cross sections, such as parallel to the p_4 axis, the throughput may both increase and decrease. (For example, in Figure 3.17 the throughput decreases as p_3 increases for $p_1=.2$ and $p_2=.1$.)

3.6.2 Number of Segments Bound

To obtain some idea of the optimum throughput of the Ringbus model with $S=8$ and grant durations of one round for general request probabilities, we calculated the optimum throughput of the number of segments model for $S=8$ with selected request probabilities. Table 3.3 lists the results, which we obtained via value iteration, to within $\pm.005$ of optimum. For comparison, Table 3.3 also lists the optimum throughput of the Ringbus model for the request probabilities in Table 3.3 for which it is known. These request probabilities (for which t^{opt} is known) all correspond to points along the axes of the feasible probability region. Note that $t^{number\ of\ segments}$ is a poor bound for t^{opt} for large p_1, as observed for $S=6$ in section 3.5.1.4. Otherwise, we expect that $t^{number\ of\ segments}$ is a reasonable bound for t^{opt}, as observed for $S=6$.

Request Probabilities				Number of Segments Model	Ringbus Model
p_1	p_2	p_3	p_4	$t^{number\ of\ segments}$	t^{opt}
0.2	0.0	0.0	0.0	3.20	2.96
0.4	0.0	0.0	0.0	6.40	4.94
0.5	0.0	0.0	0.0	8.00	5.63
0.0	0.2	0.0	0.0	3.09	2.43
0.2	0.2	0.0	0.0	5.23	?
0.0	0.4	0.0	0.0	3.99	3.10
0.0	0.5	0.0	0.0	4.00	3.22
0.0	0.0	0.2	0.0	1.99	1.96
0.2	0.0	0.2	0.0	3.86	?
0.0	0.0	0.4	0.0	2.00	2.00
0.0	0.0	0.5	0.0	2.00	2.00
0.0	0.0	0.0	0.2	1.51	1.32
0.2	0.0	0.0	0.2	3.75	?
0.4	0.0	0.0	0.2	4.99	?
0.0	0.2	0.0	0.2	3.00	?
0.2	0.2	0.0	0.2	4.00	?
0.0	0.0	0.2	0.2	2.00	?
0.2	0.0	0.2	0.2	3.29	?
0.0	0.2	0.2	0.2	2.86	?
0.0	0.0	0.4	0.2	2.00	?
0.0	0.0	0.0	0.4	1.99	1.90
0.2	0.0	0.0	0.4	3.20	?
0.0	0.2	0.0	0.4	2.67	?
0.0	0.0	0.2	0.4	2.00	?
0.0	0.0	0.0	0.6	2.00	2.00
0.2	0.0	0.0	0.6	2.85	?
0.0	0.2	0.0	0.6	2.50	?
0.0	0.0	0.2	0.6	2.00	?
0.0	0.0	0.0	1.0	2.00	2.00

Table 3.3: Results from number of segments model for eight slices

An examination of the estimated optimal decision in each state of the number of segments model revealed that the number of states with non-maximum reward decisions increased as the request probabilities became dominated by short (i.e. length 1) and long (i.e. length 3 and 4) requests. Otherwise the number of states with less than the maximum reward was quite small. In fact, as long as p_1 and p_3 were both small, the estimated optimal decision in each state almost always gave the maximum reward. The optimal throughput of the number of segments model with a maximum reward constraint was very close to the unconstrained optimal throughput except when there were mostly short and long requests. Of the request probabilities listed in Table 3.3, the degradation caused by the maximum reward constraint was greatest (0.40) for $p_1 = .4$, $p_2 = 0$,

$p_3 = 0$, and $p_4 = .2$). On the other hand, the optimum throughput of the number of segments model with a maximum number of segments constraint was indistinguishable from the unconstrained optimal throughput $t^{number\ of\ segments}$ for all the request probabilities listed in Table 3.3 except for $p_1 = .4$, $p_2 = 0$, $p_3 = 0$, and $p_4 = .2$. This comes as no surprise since the estimated optimal decision in each state in the unconstrained case almost always utilized the maximum number of segments.

These observations suggest that the trends in the optimal decisions for the Ringbus model for $S = 6$, discussed in section 3.5, continue for $S = 8$. In particular, these observations suggest that the maximum reward constraint has even a greater effect on the optimum throughput of the Ringbus for $S = 8$ than for $S = 6$, reflecting the sharper contrast between short and long request for $S = 8$.

3.7 The Symmetric Ringbus With More Than Eight Slices

Any pursuit of the optimum throughput and/or optimum policy for more than eight slices and general request probabilities seems hopeless. As the number of slices increases much past eight, there even begin to be too many states to compute the optimum throughput on the faces and along the axes representing requests of length less than $S/2$. (The number of states along these axes is 3^S where S is the number of slices. Only 2^S states are required to compute the optimum along the axis representing requests of length $S/2$. This number can be reduced further as we discuss in section 3.7.1.) Of course, the general characteristics of the throughput as discussed in section 3.6.1 remain the same for more than eight slices. In addition, the bounds discussed previously, particularly the number of segments bound, can still be effectively applied (although the number of states increases rapidly above eight slices for the number of segments bound).

3.7.1 Throughput as a Function of the Number of Slices for Some Special Cases

Two special cases for which it is easy to determine the optimum throughput of the Ringbus for a large number of slices are

(i) at an extreme point of the feasible probability region i.e. at a point where $p_i = .5$ and $p_j = 0$ for $j \neq i$ for some $0 < i < S/2$, and

(ii) along the axis corresponding to requests of length $S/2$ i.e. $p_i = 0$ for $i = 1, 2, \cdots, S/2 - 1$. Using rotational and flip symmetry, the 2^S states in case (i) can be reduced by a significant fraction.

One extreme point of particular interest in case (i) is $p_1 = .5$, where the maximum throughput occurs. We can easily obtain bounds on this maximum throughput for a large number of slices as follows.

Let the number of requests in a round in the clockwise direction be denoted by n_{cw} and the number of requests in a round in the counterclockwise direction be denoted by n_{ccw} ($n_{cw} + n_{ccw} = S$). Since all the requests are nonnull and of length one, we can grant at least $max(n_{cw}, n_{ccw})$ requests in a round. Imagine an arbiter which operates by granting exactly $max(n_{cw}, n_{ccw})$ requests in every round. Since an optimal arbiter can grant at least this number of requests in every round, the throughput of the Ringbus with this clockwise-counterclockwise arbiter (which we term the cw-ccw arbiter) thus gives a lower bound on the optimum throughput of the Ringbus for $p_1 = .5$.

An obvious state description of the Ringbus with the cw-ccw arbiter is

$$(n_{cw}, n_{ccw}).$$

However, we can reduce the number of states by utilizing the symmetry between the clockwise

and counterclockwise requests. Thus we consider instead the state description

$$(m,n)$$

where $m = max(n_{cw}, n_{ccw})$ (i.e. $m \geq S/2$) and $m + n = S$. It is convenient to number the states with n, $n = 0, 1, 2, \cdots, S/2$. The reward in each state is m. The one step transition probability from state (m,n) to state (m',n') is given by

$$p_{n,n'} = \begin{cases} \left(\frac{1}{2}\right)^{S-n} \left[\binom{S-n}{n'-n} + \binom{S-n}{n'}\right], & n' < S/2 \\ \left(\frac{1}{2}\right)^{S-n} \binom{S-n}{S/2}, & n' = S/2 \end{cases}$$

where

$$\binom{a}{b} = \begin{cases} \frac{a!}{b!(a-b)!} & \text{if } a \text{ and } b \text{ are integers and } b = 0, 1, \ldots, a \\ 0, & \text{otherwise} \end{cases}$$

This expression for $p_{n,n'}$ may be understood as follows. The reward in state (m,n) is $S-n$; hence the next state has $S-n$ new requests. There are two ways for this next state to be (m',n') if $n' < S/2$:

1) $n'-n$ (where $n' > n$) of the $S-n$ new requests are in the same direction as the n old requests inherited from state (m,n) and the state is not "flipped" (i.e. $S-n' \geq n'$).

2) n' of the $S-n$ new requests are in the opposite direction as the n old requests inherited from state (m,n) and the state is "flipped" (i.e. $n' < S-n'$).

There is only one way for the next state to be m',n' if $n' = S/2$ since the state is never "flipped" in this case.

The throughput of the Ringbus with the cw-ccw arbiter is given by

$$t^{cw-ccw} = \sum_{n=0}^{S/2} \pi_n (S-n)$$

where π_n is the steady state probability of being in state n. The π_n satisfy $\pi_n = \sum_{n'=0}^{S/2} \pi_{n'} p_{n',n}$, $n = 0, 1, 2, \ldots, S/2$, and $\sum_{n'=0}^{S/2} \pi_{n'} = 1$.

We computed t^{cw-ccw} for various values of S; the results are listed in Table 3.4 along with the optimum throughput of the Ringbus for 4, 6, and 8 slices. Note that the lower bound given by

t^{cw-ccw} is equal to the optimum throughput for 4 slices. The lower bound is progressively less tight for 6 and 8 slices. We expect that this trend continues as the number of slices increases further. As $S \rightarrow \infty$, an average of 2/3 of the requests are granted†, hence $\frac{t^{cw-ccw}}{S} \rightarrow \frac{2}{3}$ as the figures indicate in Table 3.4.

S	t^{cw-ccw}	t^{opt}	$\frac{t^{cw-ccw}}{S}$	$\frac{t^{opt}}{S}$
4	2.833	2.833	0.708	0.708
6	4.154	4.23	0.692	0.705
8	5.473	5.63	0.684	0.704
10	6.792	?	0.679	?
12	8.112	?	0.676	?
14	9.433	?	0.674	?
16	10.755	?	0.672	?
18	12.078	?	0.671	?
20	13.403	?	0.670	?
22	14.729	?	0.669	?
24	16.055	?	0.669	?
26	17.382	?	0.669	?
28	18.709	?	0.668	?
30	20.038	?	0.668	?
32	21.367	?	0.668	?

Table 3.4: t^{cw-ccw} for various values of S

We can obtain an upper bound on the optimum throughput of the Ringbus for $p_1=.5$ and S even by considering only destination conflicts. Number the slices from 1 to S in the clockwise direction around the Ringbus. Odd numbered slices only request even numbered slices and even numbered slices only request odd numbered slices. Thus, ignoring the segment conflicts, the Ringbus is equivalent for $p_1=.5$ to two $S/2 \times S/2$ crossbars - one connecting odd sources to even destinations and the other connecting even sources to odd destinations. Each of these crossbars consist of $S/4$ cells as depicted in Figure 3.25.

† For large S, if m requests are granted in the current round, then the average number of requests granted in the next round, m', is the number of leftovers from the current round plus half of the new requests that arrive in the next round i.e. $m' = (S-m) + \frac{m}{2}$. In steady state $m' = m$, hence $m = \frac{2}{3}S$.

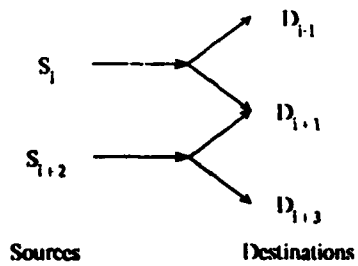


Figure 3.25: A crossbar cell

Suppose we ignore the interactions between cells. (The cells interact via conflicts at the destinations in common with adjacent cells.) Each cell is thus independent of all the others. Under this condition, it is easy to establish that the throughput of a cell is $t_{cell} = \frac{7}{4}$. The total number of cells is $S/2$, hence

$$t^{opt} \leq \frac{S}{2}\frac{7}{4} = \frac{7}{8}S.$$

Considering that this is also a bound on the throughput of a crossbar with $p_1 = .5$ and S slices, this is a poor upper bound for the Ringbus. Examining Table 3.4, it appears that $\frac{t^{opt}}{S}$ decreases monotonically with S. This leads us to make the following conjecture.

Conjecture

Let t_S^{opt} denote the optimum throughput of a S slice Ringbus with request probabilities p_i^S, $i = -(S/2-1), \cdots, S/2$ if S is even, and $i = -(S-1)/2, \cdots, (S-1)/2$ if S is odd. Let t_{S+1}^{opt} denote the optimum throughput of a $S+1$ slice Ringbus with request probabilities

$$p_i^{S+1} = \begin{cases} p_i^S, & |i| = 0, 1, 2, \cdots, S/2-1 \text{ if } S \text{ is even} \\ p_{S/2}^S, & i = \pm S/2 \\ 0, & i = S/2+1 \end{cases}$$

and

$$p_i^{S+1} = \begin{cases} p_i^S, & |i| = 0, 1, 2, \cdots, (S-1)/2, \quad \text{if } S \text{ is odd} \\ 0, & \text{otherwise.} \end{cases}$$

Then, assuming grant durations of a single round, $\frac{t_{S+1}^{opt}}{S+1} \leq \frac{t_S^{opt}}{S}$. Note that this generalizes to

$$t_{S+n}^{opt} \leq \left\lceil \frac{S+n}{S} \right\rceil t_S^{opt}.$$

Unfortunately, this conjecture seems difficult to prove. If it is true, then a much better bound on the optimum throughput of the Ringbus for $p_1 = .5$ is

$$t_{4+n}^{opt} \leq \left\lceil \frac{4+n}{4} \right\rceil t_4^{opt} = \left\lceil 1 + \frac{n}{4} \right\rceil 2.833$$

for $n = 1, 2, 3, \cdots$. For large S the conjecture leads to $\frac{t_S^{opt}}{S} \leq \frac{t_8^{opt}}{8} = .704 \leq \frac{t_6^{opt}}{6} = .705 \leq \frac{t_4^{opt}}{4} \leq .708.$

It is not easy to obtain good lower bounds for the other extreme points $p_i = .5$ $(i \neq S/2)$ since the possibility of requests in the same direction conflicting introduces additional complexity. One way to obtain a lower bound on the optimum throughput for $p_i = .5$, $0 < i < S/2$, is to construct a Ringbus in which all requests are of length 1 by deleting the $i-1$ slices between every i^{th} slice. Of course, this is only successful (although it can be modified) if $\frac{S}{i}$ is an integer. As an example, consider $S = 8$ and $p_2 = .5$. After deleting every second slice, we obtain a 4 slice Ringbus with all requests of length 1. The throughput for such a Ringbus is 2.833, hence a lower bound on the throughput of the eight slice Ringbus with $p_2 = .5$ is 2.833. The exact throughput in this case is 3.22 (see Figure 3.24).

For some extreme points $t^{opt} = 2$. This is obviously true, for example, for $p_{S/2} = 1$. It is also true for $p_i = .5$ when $\left\lceil \frac{S}{i} \right\rceil = 2$ (S even).

The optimum throughput along the $S/2$ axis is easy to calculate for large S since the number of states can be greatly reduced from the 2^S mentioned earlier. If the only nonnull requests are of length $S/2$, it suffices for a state description to merely describe the number of pairs of slices with zero, one, and two nonnull requests, where two slices 180° apart on the Ringbus comprise a pair. Thus the state description is

$$(n_0, n_1, n_2)$$

where n_i, $i = 0$, 1, or 2 is the number of pairs with i nonnull requests and $\sum_{i=0}^{2} n_i = S/2$ (S even). The total number of states is $\binom{S/2+2}{S/2} = \frac{(S/2+2)(S/2+1)}{2}$.

A lower bound on the optimum throughput along the $S/2$ axis can be obtained easily by ignoring leftover requests. Certainly,

$$t^{opt} \geq 2 \cdot Prob(\text{at least 1 pair has 2 requests}) +$$

$$1 \cdot Prob(\text{at least 1 pair has 1 request and no pair has 2 requests})$$

$$t^{opt} \geq 2 \cdot Prob(\text{at least 1 pair has 2 requests}) +$$

$$1 - Prob(\text{at least 1 pair has 2 requests}) - Prob(\text{no pair has any requests})$$

Now $Prob(\text{at least 1 pair has 2 requests}) = 1 - Prob(\text{no pair has 2 requests}) = 1 - (1 - p_{S/2}{}^2)^{S/2}$ and $Prob(\text{no pair has any requests}) = ((1 - p_{S/2})^2)^{S/2}$, thus

$$t^{opt} \geq 2 - (1 - p_{S/2}{}^2)^{S/2} - (1 - p_{S/2})^S.$$

The throughput as a function of S gives some idea of the scalability of the Ringbus. The throughput varies with the traffic, as reflected by the values of the p_i. This latter factor affects the throughput the most: with $p_0 = 0$, the throughput can vary from 2 to somewhere between $\frac{2}{3}S$ and $\frac{7}{8}S$. The sensitivity of the throughput to the distribution of request lengths is perhaps best illustrated by the bound $t \leq \frac{S}{\bar{l}}$ from section 3.5.1.1.

3.8 The Performance of the Concert Ringbus

In this section we investigate the performance of the Concert Ringbus and compare its performance with that of the Symmetric Ringbus. (We, of course, have to specify some arbitration scheme for the Symmetric Ringbus. We do this shortly.) The Concert Ringbus has asymmetrical access paths, and a rotating priority arbitration algorithm, as discussed in section 3.1. The investigation and comparison consist of three parts:

1) We determine the effect of the asymmetrical access paths by comparing the optimum throughput with asymmetrical access paths (i.e. the Asymmetrical Ringbus) to the optimum throughput with symmetrical access paths (i.e. the Symmetrical Ringbus).

2) We determine the effect of the rotating priority arbitration algorithm by comparing the throughput with this algorithm for the Symmetric Ringbus with the optimum throughput for the Symmetrical Ringbus.

3) We determine the effect of both the asymmetrical access paths and the rotating priority arbitration algorithm (i.e. the Concert Ringbus) by comparing the throughput with these to the optimum throughput with symmetrical access paths.

We consider only four slice Ringbuses. There are, unfortunately, too many states to consider Markov chain models for six or more slices. The state description with the asymmetrical access paths in part 1 remains

$$(r_1, r_2, \cdots, r_S)$$

where $r_i = -(S/2-1), \cdots, -1, 0, 1, \cdots, S/2$, but flip symmetry can no longer be utilized to reduce the number of states because a request in the counterclockwise direction requires more segments than a request of a similar number of hops in the clockwise direction. Thus, for $S = 4$ the number of states is 70, an increase of about 86% above the 43 states for the Symmetric Ringbus. A similar increase for $S = 6$ would put the number of states at about 7400. This number may not seem all that unreasonable. However, we felt it was not worth pursuing part 1 for $S = 6$ if we could not also pursue parts 2 and 3 for $S = 6$. The state description with rotating priority is

$$(r_1, r_2, \cdots, r_S, p)$$

where $p_i = (p + k) \bmod S$ is the priority of the request at slice i and r_i is the same as before. For $S = 4$, the number of states for symmetrical access paths is 129 and the number of states for asymmetrical access paths is 214. Similar increases for $S = 6$ would put the number of states above 10,000, which we consider to be too many states.

We could have pursued parts 2 and 3 for larger values of S via simulation. In fact, we did do this for $S = 8$; the results are reported in Chapter 4. However, the simulations reported in

Chapter 4 are for the entire Concert model, not just the Ringbus as is our focus here. For example, the simulations reported in Chapter 4 assume grant durations of nine rounds and arbitration times of two rounds; we assume single round grant durations and instantaneous arbitration here. Part 1 cannot be carried out via simulation since optimization is impractical via simulation.

3.8.1 The Effect of Asymmetrical Access Paths

One factor complicating the comparison of the optimum throughput with asymmetrical access paths with the optimum throughput with symmetrical access paths is that users may adapt their programs to suit the topology. As a result, requests may be biased in favour of the clockwise direction in the former case and unbiased in direction in the latter case. To avoid biasing the comparison, we present the results for various asymmetrically weighted and symmetrically weighted request probabilities for both asymmetrical and symmetrical access paths.

Figures 3.26, 3.27, and 3.28 show the optimum throughput with asymmetrical and symmetrical access paths for $p_{-1}=p_1$, $p_{-1}=.5p_1$, and $p_{-1}=0$ respectively. (p_{-1} is the probability of a request of one hop in the counterclockwise direction.) Note that the optimum throughput with asymmetrical and symmetrical access paths is identical for $p_{-1}=0$; hence only one set of points is shown in Figure 3.28. As expected, the difference in the optimum throughputs for asymmetrical and symmetrical access paths increases as p_{-1} decreases.

3.8.2 The Effect of the Rotating Priority Arbitration Algorithm

Figure 3.29 shows the optimum throughput of the Symmetric Ringbus and the throughput of the Symmetric Ringbus with the rotating priority arbitration algorithm used in the Concert Ringbus. For very light traffic, the throughput with rotating priority is close to the optimum. For all other traffic, the throughput with rotating priority quickly deteriorates with respect to the optimum. The maximum throughput, at $p_1=.5$, with rotating priority is .42 less than the optimum throughput. For $p_2=1$ the deterioration is especially severe. Even though two requests can be granted without conflicting, the rotating priority algorithm only grants one request. The reason for this stupidity is that slices are assigned consecutively decreasing priorities in the clockwise direction from the highest priority slice. Since no request can be granted which may conflict with one at a higher priority, a long request currently blocked by a request granted by a higher priority slice can nevertheless prevent an otherwise grantable request from being granted due to a conflict with the higher priority long request. An example of such a situation is shown in Figure 3.30.

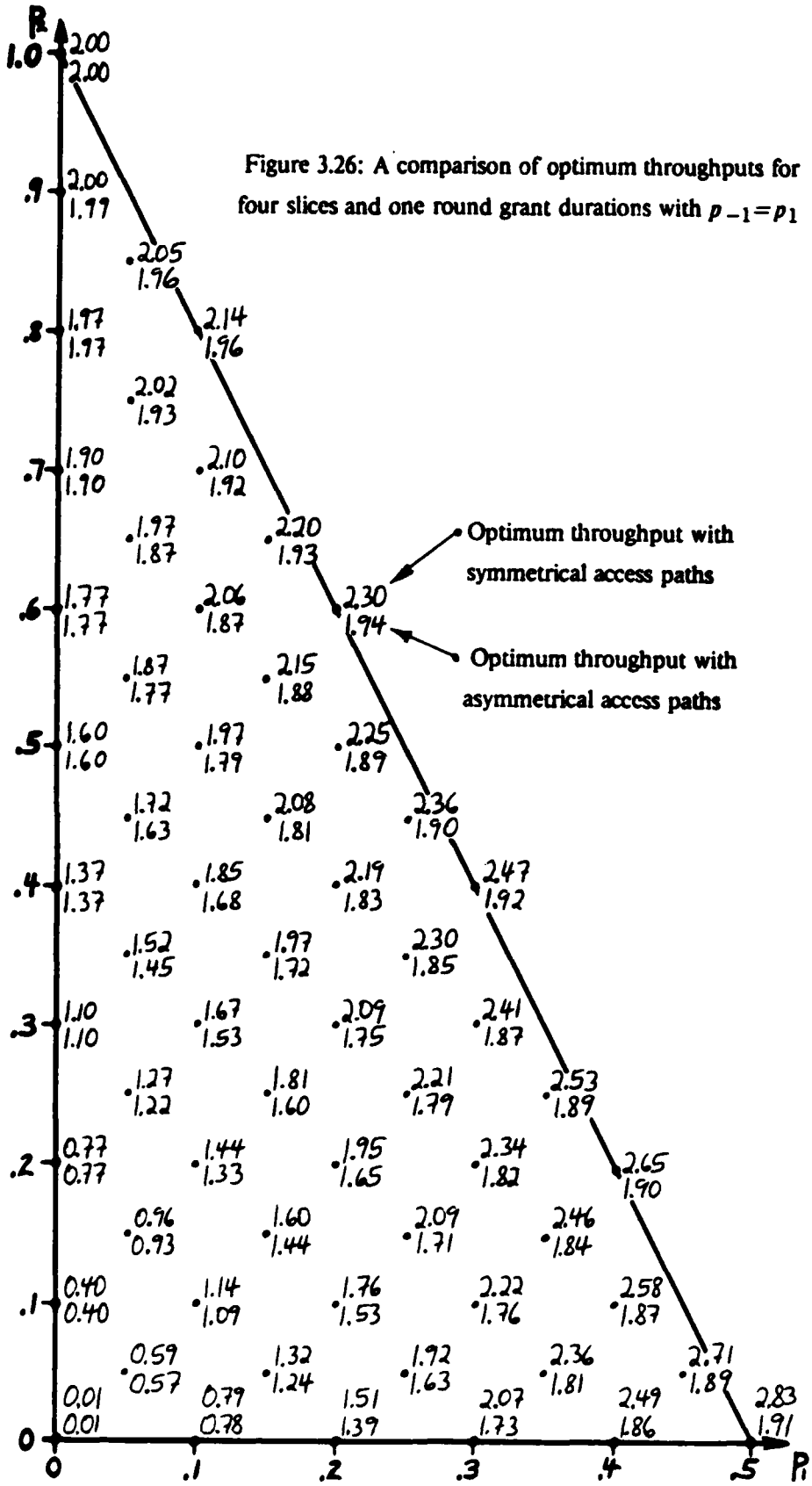


Figure 3.26: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1}=p_1$

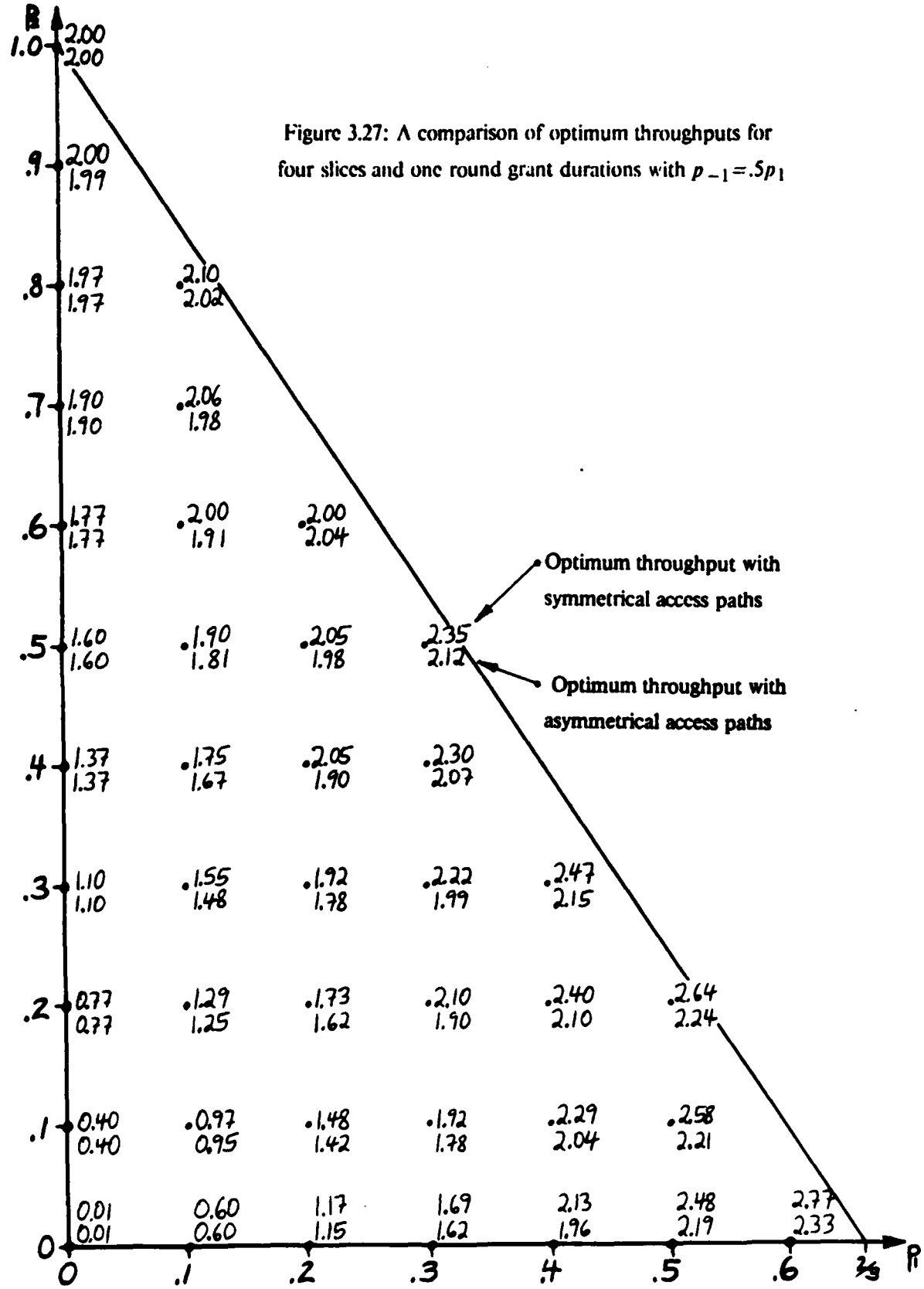


Figure 3.27: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1}=.5p_1$

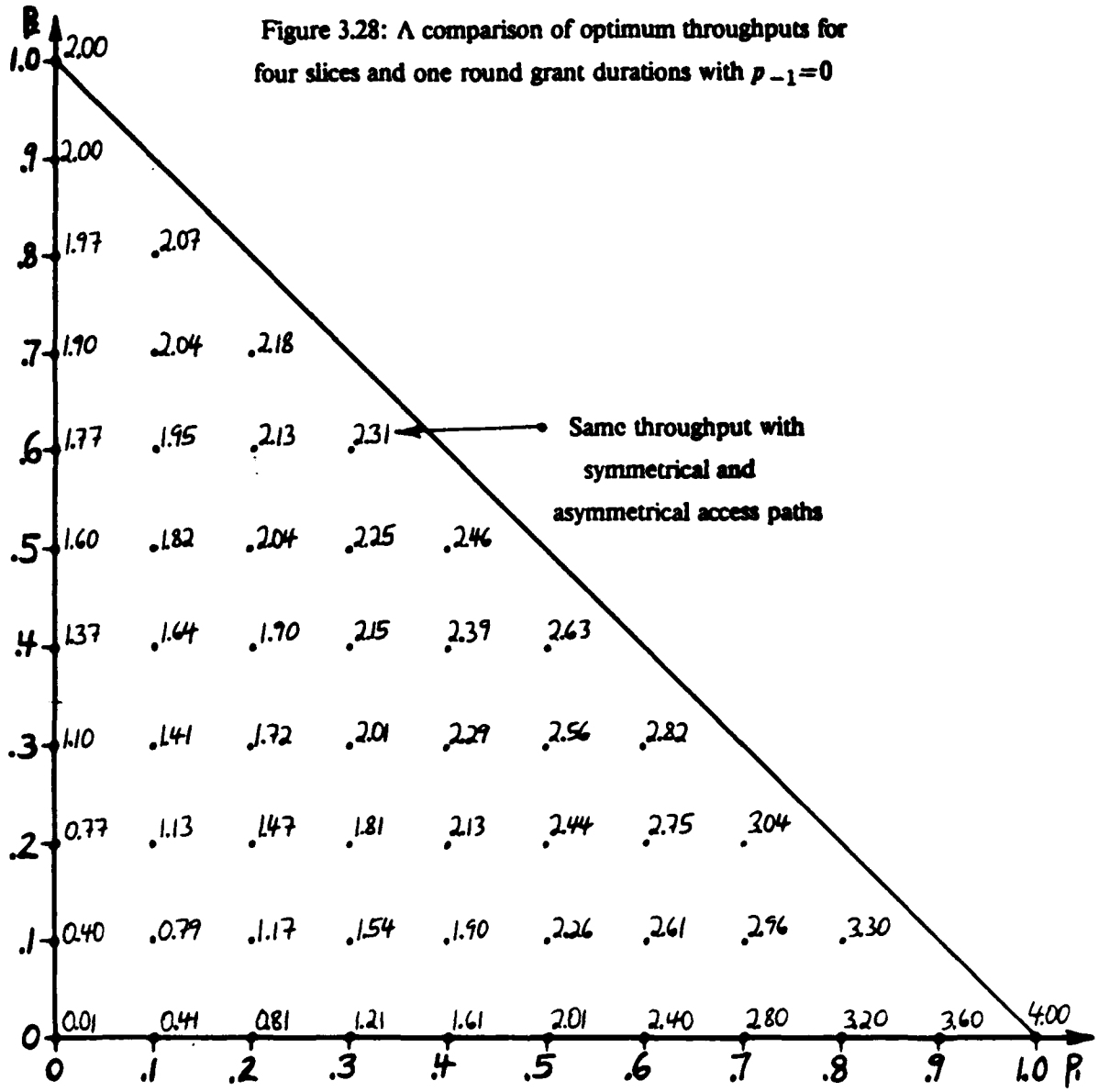


Figure 3.28: A comparison of optimum throughputs for four slices and one round grant durations with $p_{-1}=0$

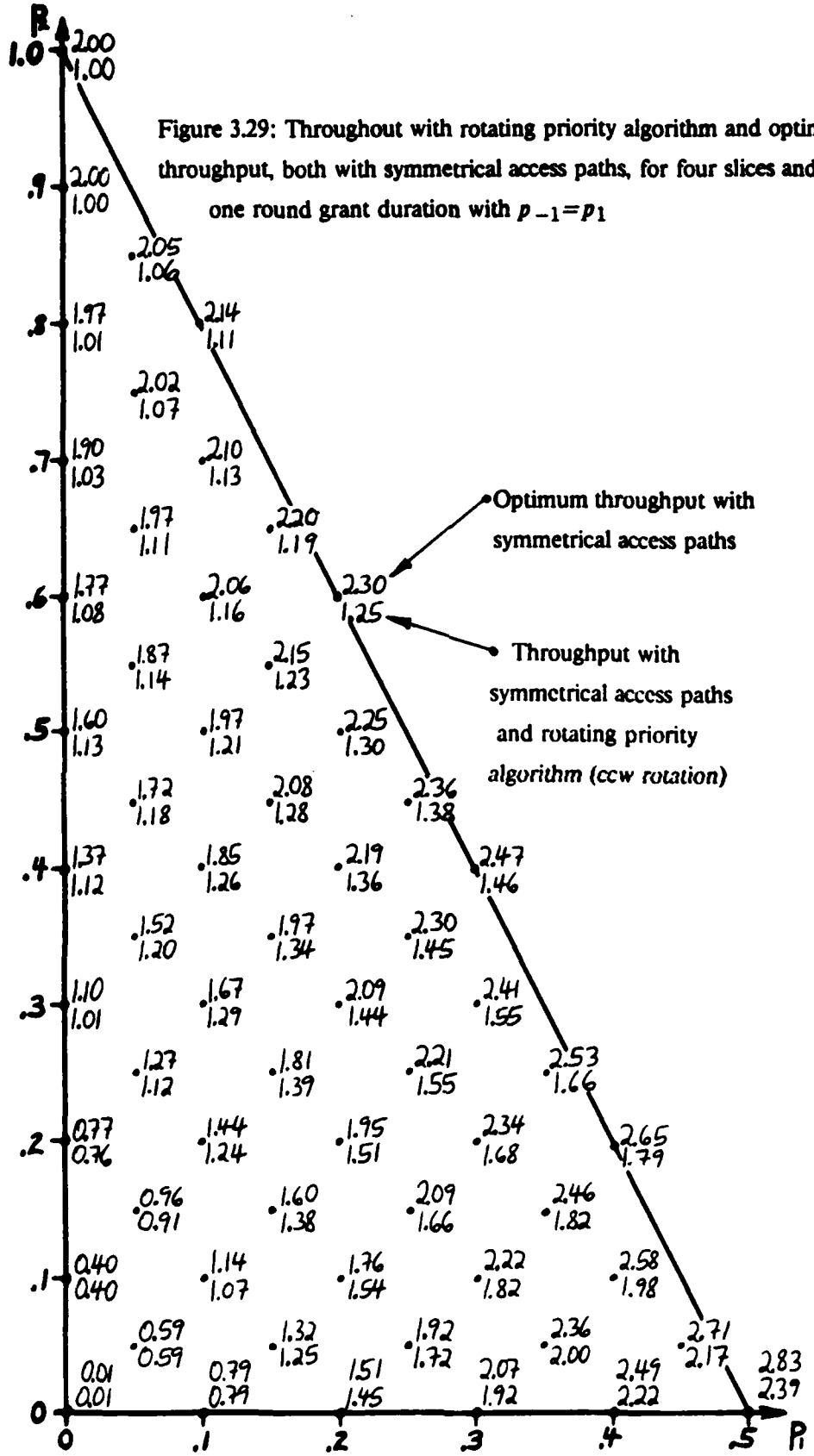


Figure 3.29: Throughout with rotating priority algorithm and optimum throughput, both with symmetrical access paths, for four slices and one round grant duration with $p_{-1}=p_1$

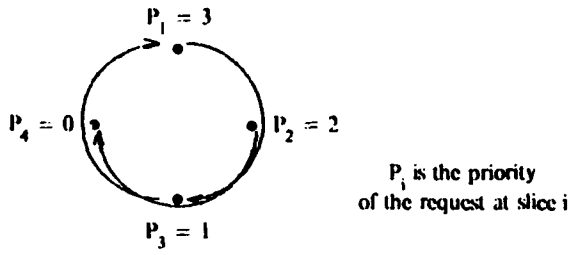


Figure 3.30: Example of a disadvantage of the rotating priority algorithm

Slice 1's request, which may be granted because it has the highest priority, conflicts with the lower priority slice 2's request, hence slice 2's request cannot be granted. However, slice 2's request conflicts with the lower priority slice 3's request and thus slice 3's request cannot be granted either, even though it is otherwise grantable. An obvious fix to the problem is to stagger the slice priorities as shown in Figure 3.31.

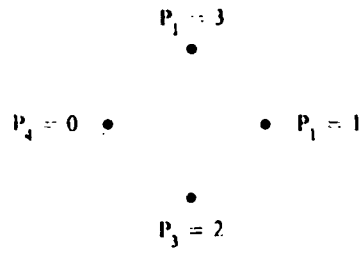


Figure 3.31: Staggered request probabilities

The consecutive assignment of slice priorities around the Ringbus will also obviously lead to throughput degradation for a larger number of slices, such as $S = 8$, and for cases in which clockwise requests of greater than one hop predominate. The priorities can be staggered in a manner similar to that in Figure 3.31 to reduce this degradation. Interestingly, it is easy to modify the Concert Ringbus to effect such a change to the assignment of the slice priorities. A new arbiter priority ROM (a $2K \times 8$ ROM) is all that is required.

A different, but still simple, improvement to the throughput of the Symmetric Ringbus with the rotating priority algorithm is to change the direction of the rotation. When the priorities are updated in the Concert Ringbus arbiter, the highest priority is assigned to the next slice with a pending request in the counterclockwise direction from the current highest priority slice. Clockwise

rotation of the priority yields better throughput (assuming the slices are assigned consecutively decreasing priorities in the clockwise direction from the highest priority slice). The maximum improvement in throughput by reversing the priority rotation from counterclockwise to clockwise is .10, which is attained at $p_1 = .5$. As before, a new arbiter priority ROM is all that is required to implement clockwise priority rotation.

3.8.3 The Effect of Asymmetrical Access Paths and the Rotating Priority Arbitration Algorithm

Figures 3.32, 3.33, and 3.34 show the throughput with asymmetrical access paths and the rotating priority algorithm, the optimum throughput with asymmetrical access paths, and the optimum throughput with symmetrical access paths for $p_{-1} = p_1$, $p_{-1} = .5p_1$, and $p_{-1} = 0$ respectively. As in Figure 3.29, the rotating priority algorithm imposes a degradation in throughput (as compared with the optimum throughput with asymmetrical access paths) that increases as p_1 or p_2 or both increase. For $p_1 = p_{-1} = .5$, the degradation is .30 or 16%. Again, the degradation is especially severe for $p_2 = 1.0$.

The throughput degradation is mostly attributable to the rotating priority algorithm if p_2 is large and is mostly attributable to the asymmetrical access paths if p_1 and p_{-1} are both large and if the request probabilities are the same with symmetrical and asymmetrical access paths. (This comparison can be misleading since the request probabilities would probably have a strong clockwise bias in direction in any Ringbus with asymmetrical access paths and would probably be relatively unbiased in any Ringbus with symmetrical access paths. See the paragraph at the beginning of section 3.8.1.) The throughput degradation attributable to the asymmetrical access paths diminishes as $p_{-1} \rightarrow 0$ if the request probabilities are the same with symmetrical and asymmetrical access paths. (The same parenthetical note applies to this statement too.)

We expect all the trends observable in Figures 3.32, 3.33, and 3.34 to be accentuated with larger values of S.

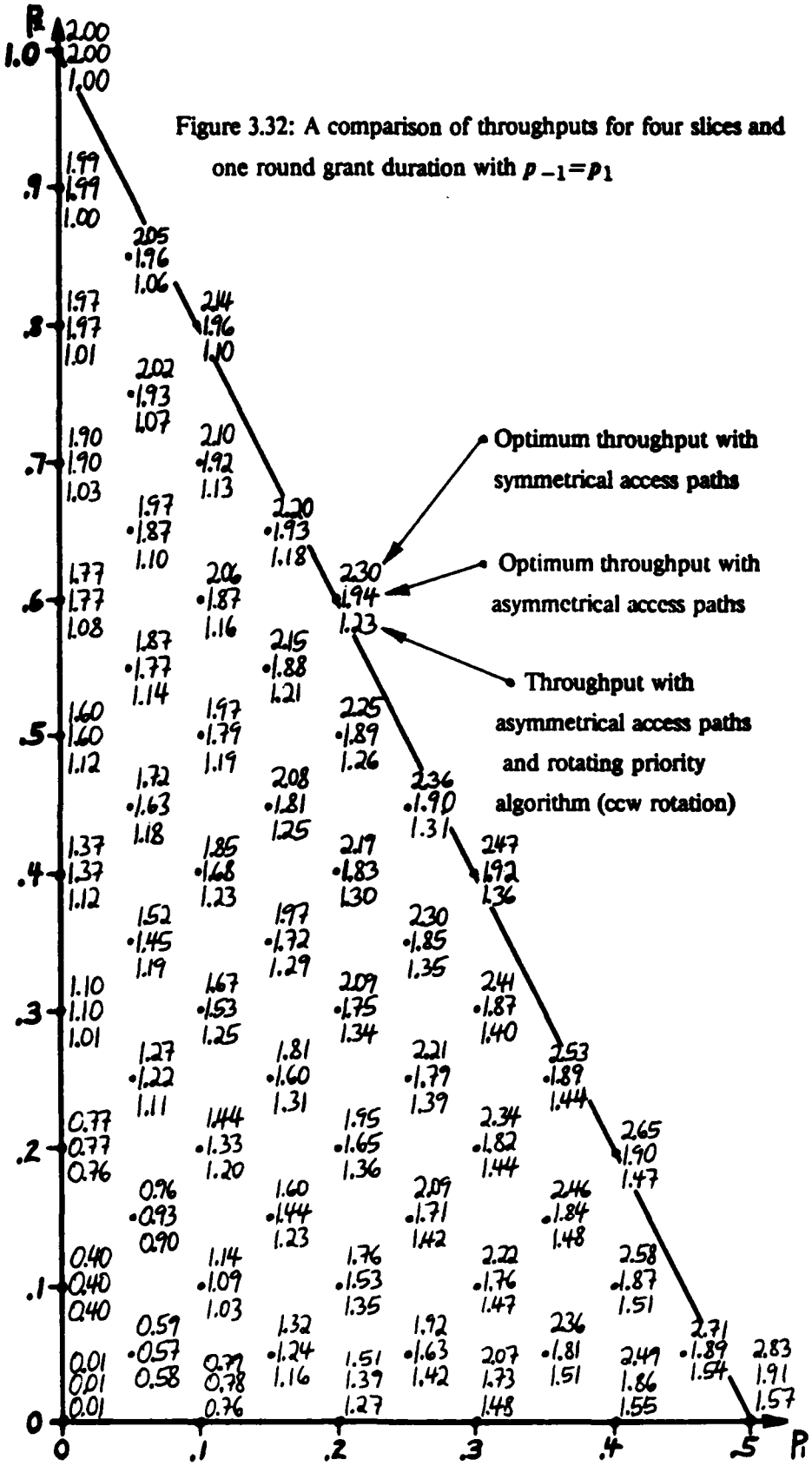


Figure 3.32: A comparison of throughputs for four slices and one round grant duration with $p_{-1}=p_1$

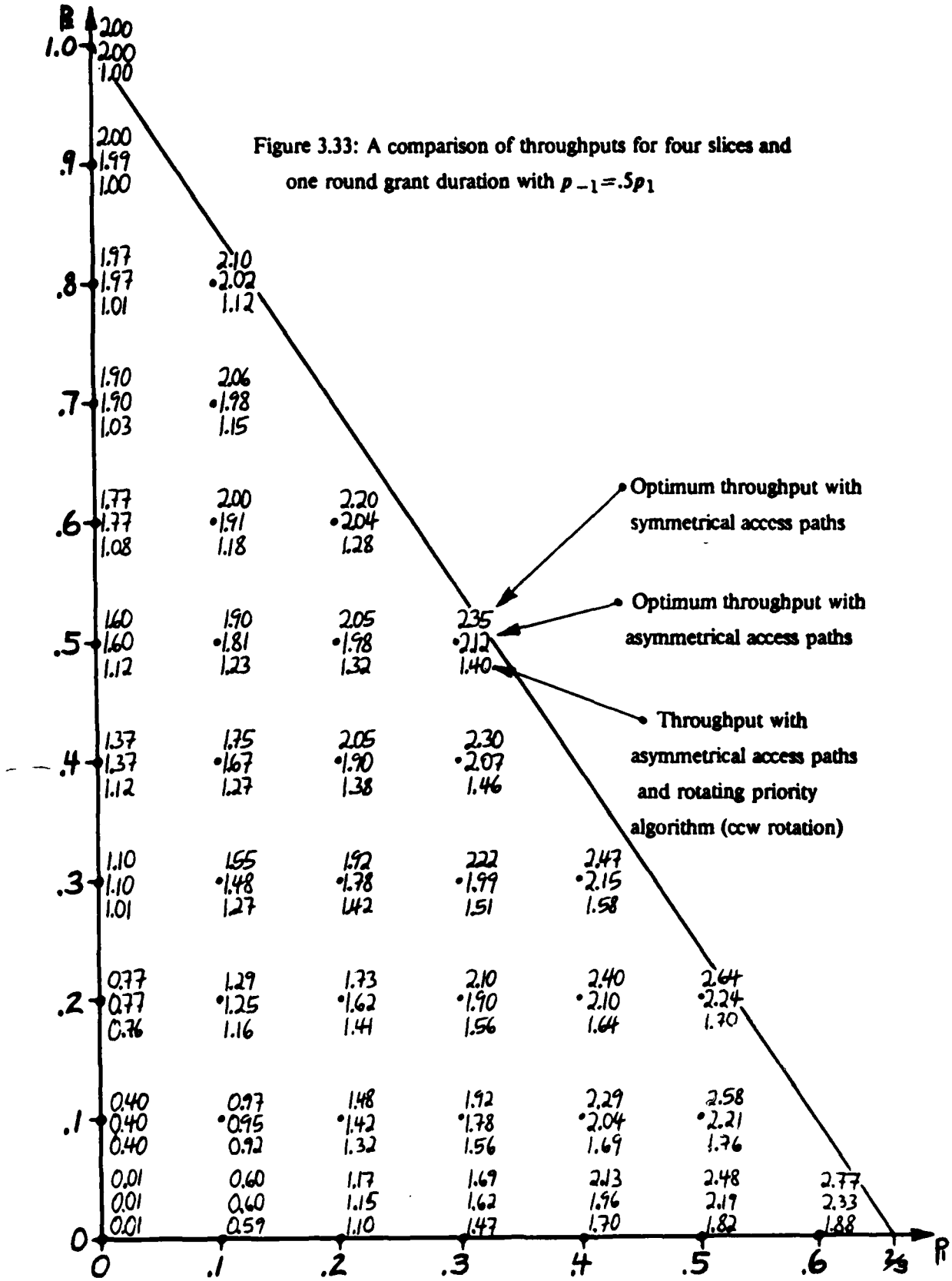


Figure 3.33: A comparison of throughputs for four slices and one round grant duration with $p_{-1}=.5p_1$

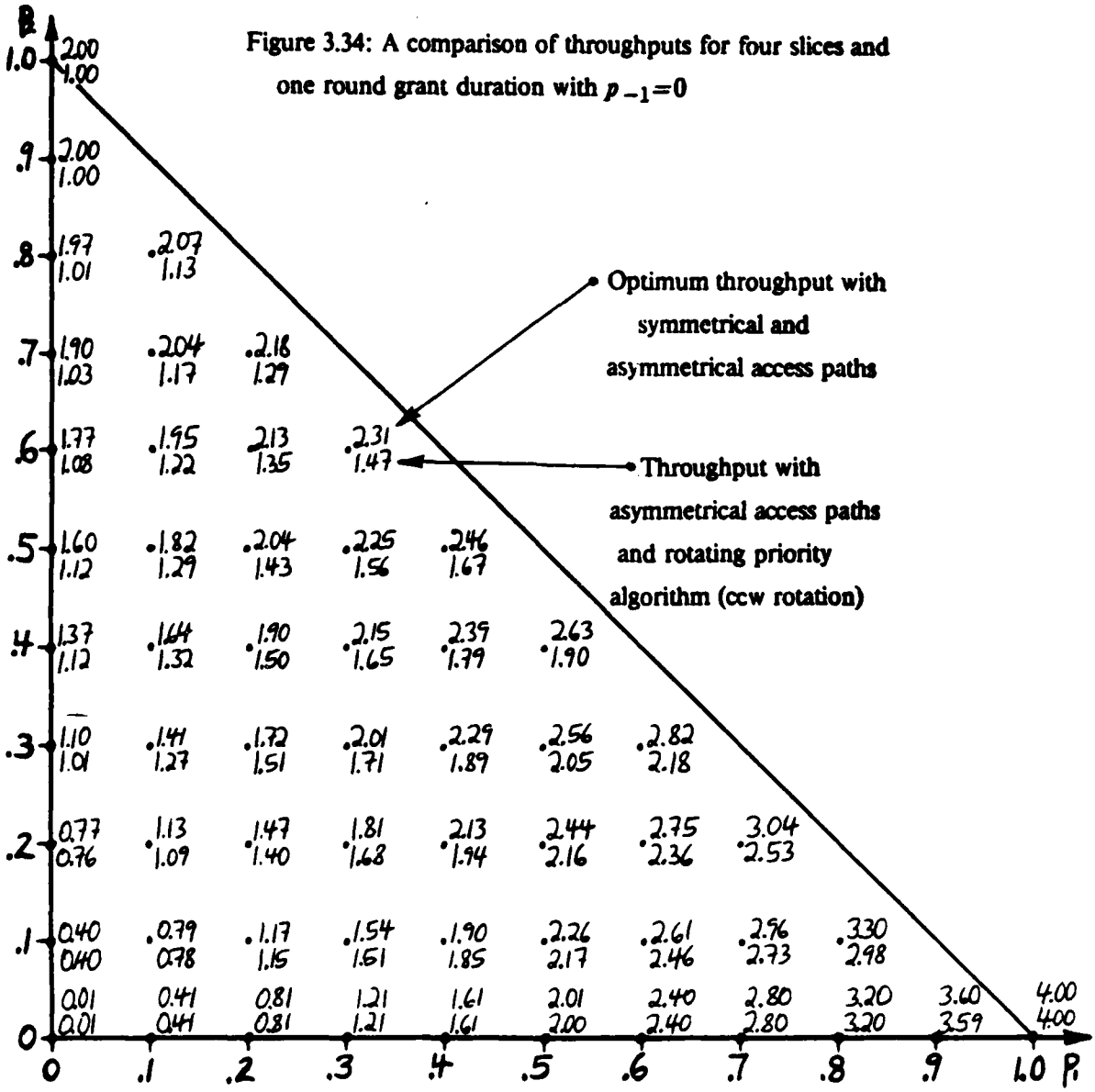


Figure 3.34: A comparison of throughputs for four slices and one round grant duration with $p_{-1}=0$

3.9 The Ringbus in the Concert Environment

So far in this chapter we have considered the Ringbus model in isolation. Now we consider some of the differences between this artificial environment and the Concert environment. We discuss the effects that these differences have on the operation and performance of the Ringbus. In section 3.9.1 we discuss the details of the Multibus-Ringbus connection and develop the hooks for the integration of the Ringbus model with the Multibus models in Chapter 4.

The major differences between the artificial environment of the isolated Ringbus and the Concert environment are:

1) the duration of the grants,

2) the arbitration time,

3) the *dead* time between successive Ringbus requests, and

4) global register accesses.

The duration of a grant is the total duration for which Ringbus segments are allocated to a request. As reported in section 3.3.2 of Appendix A, this duration is 9 or 10 arbiter clock cycles - i.e. 9 or 10 rounds - for reads and write accesses when the arbiter clock period is 200nsec. Other than for a geometrically distributed grant duration with a mean of 10 rounds, we did not investigate the isolated Ringbus model for such long grant durations. Furthermore, this case with a mean grant duration of 10 rounds applied for $S=4$ and symmetric access paths. Thus grant durations in the Concert environment are much longer than we considered for the isolated Ringbus model except in one special case.

As discussed in section 3.4, we expect that the effect of the long grant durations on the optimum performance of the Ringbus can be estimated fairly well from the optimum throughput with a deterministic grant duration of one round and equation 3.18. It should be possible to estimate the effect of long grant durations on the throughput for arbitration algorithms other than the optimum, by similar means. We expect then that the performance of the Ringbus is initially quite sensitive to the duration of grants and decreases rapidly as the duration of grants increases.

The arbitration time (or more precisely, the arbitration delay) can be divided into two components. At some point during the arbitration time the arbiter decides (or can be regarded as deciding) whether or not to grant a request. The rest of the time is a delay gathering request information before the decision and a delay communicating and implementing the decision. Thus the arbitration time may be treated by assuming instantaneous arbitration time and adding the appropriate grant implementation delay to the request interarrival time and the appropriate grant implementation delay to the grant duration. Increasing the request interarrival time (i.e. increasing p_0) and increasing the grant duration decreases the throughput. The exact effect of adding these delays depends on the magnitudes of the delays and the parameter values for the requests and

grant durations. The arbitration delay in Concert is two rounds - one round of request gathering delay and one round of grant implementation delay. For light to medium loading the resultant additional clock cycle of request interarrival time will cause little change in p_0 and hence will have little effect on performance. Likewise, the effect of the additional clock cycle of grant duration will be small since grant durations are already quite long in Concert.

The dead time between successive Ringbus requests is the minimum time between the end of a Ringbus grant and the next nonnull request generated from the same slice. In our isolated Ringbus model we assumed a dead time of zero. However, in Concert there is a dead time of 2 or 3 rounds. (The dead time corresponds to the minimum value of t_p^{RBeqv} which is reported in section 3.3.2 of Appendix A. We define t_p^{RBeqv} and discuss the details of the Multibus-Ringbus interaction in section 3.9.1.) Since the dead time is relatively small compared to the total duration of a grant, we do not expect the dead time to have a large direct effect on the performance of the Ringbus as compared to that predicted by our isolated Ringbus models. Of course, there will be an indirect effect since the dead time portion of the processing time is not well approximated by the geometric distribution which we assume for the processing time in our isolated Ringbus models. The consequence of the dead time is that the mean processing time must be at least 2 or 3 rounds, and thus $p_0 \geq \frac{2}{3}$. This corresponds to light traffic in our isolated Ringbus models.

We have already discussed global register accesses. Accesses to global registers on a slice different from the slice originating the access can be treated as special global memory requests. Accesses to global registers on the same slice originating the access cannot be treated in this manner. Instead, we simply ignore such accesses. We expect global register accesses to be infrequent in normal operation, so the effect of ignoring such accesses in our isolated Ringbus models to be minimal in most cases.

Note that there is additional information available in the Concert environment which could conceivably allow the Ringbus arbiter to achieve better performance. In Concert, the only information available to the Ringbus arbiter is the type of request or grant present at each slice. The arbiter is able to infer from this information the duration that the request has been pending or that the grant has been in progress at each slice. Other information available in the Concert environment, but not available to the arbiter, is the number and type (i.e. Multibus or Ringbus) of requests in each Multibus queue and the waiting time so far of each request.

Since all other Multibus activity is blocked during the entire duration of a Ringbus access - even during the period which the access waits for use of the Ringbus - the arbiter could conceivably give priority to Ringbus accesses blocking a large number of Multibus accesses and thereby improve the overall throughput of Concert.

Finally, note that although the arbiter clock period does affect the performance of the

Ringbus, the effect is not as large as one may expect. The reason is that a considerable fraction of the duration of a Ringbus grant is approximately constant independent of the arbiter clock period.

3.9.1 The Equivalent Model of the Ringbus

As discussed in section 1.3.5, the Ringbus can be replaced by an equivalent model for each slice-Ringbus connection. The equivalent model for each slice-Ringbus connection is the Ringbus access time distribution as seen by that slice. In determining these equivalent models of the Ringbus, we assume that each slice has been replaced by its single processor equivalent with some processing time distribution, with mean $\bar{t}_p^{MBeqv}$, and some Ringbus destination probabilities p_i^{MBeqv}, $i = -(S/2-1), \cdots, -1, 1, 2, \cdots$, or $S/2$. (S is the number of slices.) We assume that all of the single processor equivalent models are identical and that the Ringbus is symmetric with respect to each slice. Under these latter two assumptions, the equivalent models of the Ringbus are identical for each slice-Ringbus connection and thus the Ringbus is completely characterized by one equivalent model. As noted in section 1.3.5, this means that only one Multibus-Ringbus connection need be considered during integration.

The single processor equivalent of the Multibus and the Ringbus each perceive a Ringbus access cycle in a different way. From the point of view of the single processor equivalent, a Ringbus access cycle consists of a processing time, denoted by t_p^{MBeqv}, and an access time, denoted by t_{aRB}. t_{aRB} includes the waiting time, if any, of the Ringbus request generated by the access. The probability distribution of t_p^{MBeqv} incorporates the Multibus waiting time. Figure 3.35 depicts the point of view of the single processor equivalent.

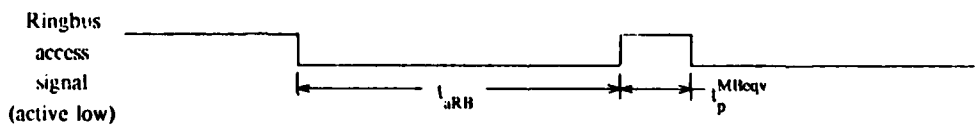


Figure 3.35: Point of view of single processor equivalent

From the point of view of the Ringbus, a Ringbus access cycle consists of a processing time, a waiting time, and a grant duration, all defined relative to the arbiter clock. We define the grant duration as the total duration for which Ringbus segments are allocated to the Ringbus request generated by the access; we denote the grant duration by d. We define the processing time as the interval between the termination of a grant and the commencement of the following grant in the absence of contention in the Ringbus. We denote this interval by t_p^{RBeqv} to indicate the processing time as seen by the Ringbus. Finally, we define the waiting time to be the duration that a grant is

delayed due to Ringbus contention; we denote it by w_{RB}. We measure t_p^{RBreq}, w_{RB}, and d synchronous to the rising edges of the arbiter clock. Figure 3.36 depicts the point of view of the Ringbus.

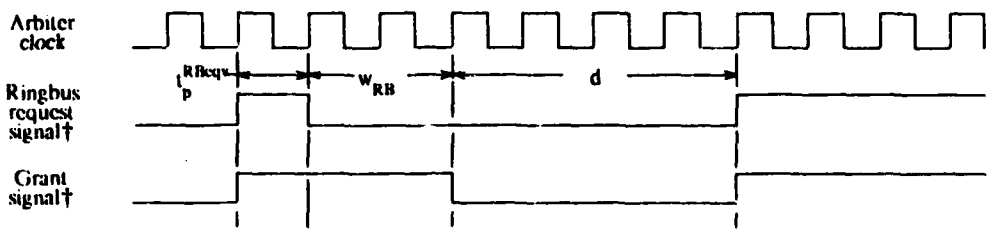


Figure 3.36: Point of view of Ringbus

We now combine the points of view of the single processor equivalent of the Multibus and the point of view of the Ringbus. Central to this combination are the facts that 1) the Multibus operates asynchronously with respect to the Ringbus arbiter and 2) the arbitration for the Ringbus takes some nonzero time. We define t_{latch} as the time required to synchronize a Multibus request for a Ringbus access with the arbiter clock. More precisely, t_{latch} is the interval between the arrival of a request at the Ringbus arbiter and the next rising edge of the arbiter clock.‡ We define t_{arb} as the arbitration delay of the Ringbus arbiter. (t_{arb} is some integral multiple of the arbiter clock period.) In addition, we define t_{start} as the interval between the initiation of a Ringbus access on the Multibus and the arrival of the corresponding Ringbus request at the Ringbus arbiter. t_{start} reflects the time that a processor takes to put valid signals on the Multibus once it has seized control of the Multibus and the time that the RIB takes to decode these signals. (We consider an access on the Multibus to initiate when a processor seizes control of the Multibus and to terminate when the processor releases control of the Multibus. See section 2 of Appendix A for details.) Through various quirks in the timing of Multibus and Ringbus signals, the termination of an access and the disassertion of the Ringbus request at the Ringbus arbiter occur at approximately the same time. (See section 3.3.2 of Appendix A.) We assume this to be the case here and thus we do not introduce a corresponding "t_{end}".

The combined points of view of the single processor equivalent and the Ringbus are pictured in Figure 3.37 along with the quantities just defined.

† These signals are drawn as active low to parallel the signals in the actual Concert system.
‡ In this section we ignore delays that would normally be introduced to mitigate metastability problems and assume that the arbiter inputs are sampled on every rising edge of the arbiter clock.

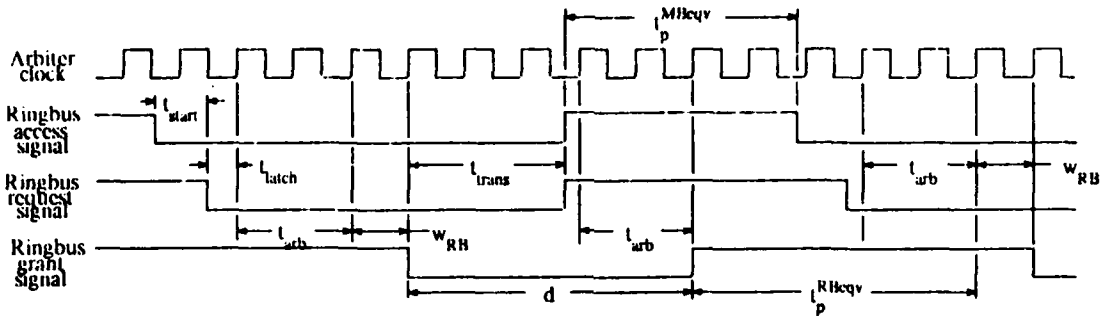


Figure 3.37: Combined points of view of single processor equivalent and Ringbus

Note that the access time of the single processor equivalent and the duration for which segments are allocated in the Ringbus - i.e. the grant duration - are out of phase. Of course, the actual period for which the data transfer occurs is the same for the single processor equivalent and the Ringbus. We denote this time by t_{trans}. The arbitration delay skews the total time allocated to the access in the respective worlds of the single processor equivalent and the Ringbus.

Taking means, we have $\bar{t}_p^{MBeqv} + \bar{t}_{aRB} = \bar{t}_p^{RBeqv} + \bar{w}_{RB} + \bar{d}$ and thus $\bar{t}_p^{RBeqv} - \bar{t}_p^{MBeqv} + \bar{t}_{aRB} - \bar{w}_{RB} - \bar{d} = \bar{t}_p^{MBeqv} + \bar{t}_{aRB}^{(norm)} - \bar{d}$,† where $\bar{t}_{aRB}^{(norm)}$ is the mean Ringbus access time when there is no contention on the Ringbus i.e. when $w_{RB} = 0$. Note that $\bar{t}_{aRB}^{(norm)} = \bar{t}_{start} + \bar{t}_{latch} + \bar{t}_{arb} + \bar{t}_{trans}$.

The inputs to the equivalent model of the Ringbus are the probability distribution of t_p^{MBeqv} and the Ringbus destination probabilities p_i^{MBeqv}. The output is the probability distribution of t_{aRB}. The inputs to the actual Ringbus model are the probability distribution of t_p^{RBeqv} and the Ringbus destination probabilities, p_i^{MBeqv}. The output of the actual Ringbus model is the

† More precisely, we have $t_p^{RBeqv} = \begin{cases} t_p^{MBeqv} + t_{aRB}^{(norm)} - d^{prev} & \text{if } t_p^{MBeqv} + t_{start} \geq d^{prev} - t_{trans}^{prev} \\ t_{arb} + c & \text{if } t_p^{MBeqv} + t_{start} < d^{prev} - t_{trans}^{prev} \end{cases}$

where c is the arbiter clock period and the superscript $prev$ denotes the quantities from the previous Ringbus access. If $t_p^{MBeqv} + t_{start} < d^{prev} - t_{trans}^{prev}$, then a new Ringbus request arrives at the Ringbus arbiter before the grant of the previous request has been disasserted. This previous grant must be disasserted before the new request can be accepted by the arbiter, hence the new grant follows the old by the arbitration time and one entire arbiter clock cycle for latching. $t_{arb} + c$ is the dead time mentioned earlier.

throughput, or alternatively, w_{RB}. The two are related by

$$\frac{S}{\bar{t}_p^{RBeqv} + \bar{w}_{RB} + \bar{d}} = \frac{g}{\bar{d}}$$

where g is the average number of new or continuing grants per clock period (i.e. round). Now to say anything more about the relation between $\bar{t}_p^{RBeqv}$, p_i^{MBeqv}, and $\bar{w}_{RB}$ we need to consider a specific Ringbus model. We, of course, assume the Ringbus model discussed in this chapter. Specifically, we do the following:

1) We approximate the probability distribution of t_p^{RBeqv} by a discrete geometric distribution with the same mean. Thus p_0 in our Ringbus model can be computed from the relation $\frac{p_0}{1-p_0} = \frac{\bar{t}_p^{RBeqv}}{c}$ where c is the arbiter clock period.

2) We set $p_i = p_i^{MBeqv}$, $i = -(S/2-1), \cdots, -1, 1, \cdots$, or $S/2$ for the other Ringbus request probabilities.

3) We set the grant duration equal to $\bar{d}$. We could just as easily allow geometric or arbitrary discrete probability distributions for the grant distribution provided that the Ringbus model allowed such distributions. We assume a deterministic grant duration for simplicity and because observed grant durations in Concert are very nearly deterministic for reads and writes. (See section 3.3.2 of Appendix A.)

The Ringbus model can now be solved for g and $\bar{w}_{RB}$ computed from

$$\frac{S}{\frac{p_0}{1-p_0} + \bar{w}_{RB} + \bar{d}} = \frac{g}{\bar{d}}.$$

Finally, we can obtain $\bar{t}_{aRB}$.

Note that because of our approximation of the distribution of t_p^{RBeqv} by a discrete geometric distribution, we only need $\bar{t}_p^{RBeqv}$, as an input to the Ringbus model. Recall that $t_p^{RBeqv} = t_p^{MBeqv} + \bar{t}_{aRB}^{(norm)} - \bar{d} = \bar{t}_p^{MBeqv} + \bar{t}_{start} + \bar{t}_{latch} + \bar{t}_{arb} + \bar{t}_{trans} - \bar{d}$. t_p^{MBeqv} is an input to the Ringbus model and $\bar{t}_{start}$, $\bar{t}_{arb}$, $\bar{t}_{trans}$, and $\bar{d}$ are constants that can be determined by empirical observations. Such observations are reported in Section 3.3 of Appendix A. $\bar{t}_{latch}$, however, actually depends on $\bar{t}_p^{MBeqv}$, $\bar{t}_{trans}$, $\bar{d}$, and $\bar{t}_{arb}$. We define $t_{aRB}^{endtoclock}$ as the interval from the completion of a Ringbus access (from the point of view of the single processor equivalent) to the next rising edge of the arbiter clock. Thus $t_{aRB}^{endtoclock} = d - t_{arb} - t_{trans}$.

For $\bar{t}_p^{MBeqv}$ large, $\bar{t}_{aRB}^{endtoclock}$ is irrelevant and since the Multibus model (and the single processor equivalent model) is asynchronous with respect to the arbiter clock, we have $\bar{t}_{latch} = .5c$. This situation is depicted in Figure 3.38.

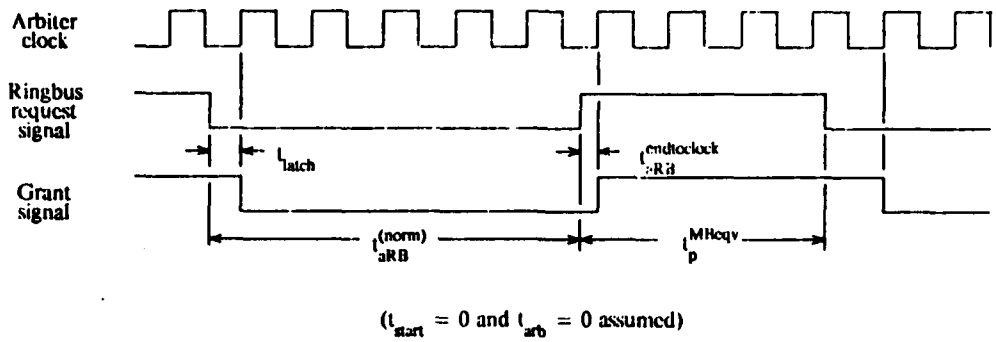


Figure 3.38: Signals when $\bar{t}_p^{MBeqv}$ large

For $\bar{t}_p^{MBeqv}$ small, $\bar{t}_{aRB}^{endtoclock}$ becomes important. The reason for this is that the Multibus model (and the single processor equivalent model) may generate a request at any time after the completion of a Ringbus access, but the arbiter cannot accept and act on the request until at least the next rising edge of the arbiter clock after the previous access. (Of course, acceptance of the request must also wait until after the previous grant. See section 3.3.2 of Appendix A for details. In the figures we assume $t_{arb} = 0$ for clarity of presentation, so the grant terminates at the next clock edge after the request terminates.) That is, $t_{latch} \geq t_p^{MBeqv}$. In fact, for $\bar{t}_p^{MBeqv} = 0$, a condition that can occur with many processors on a Multibus all accessing the Ringbus, we have $\bar{t}_{latch} = \bar{t}_{aRB}^{endtoclock}$. Thus $\bar{t}_{latch}$ can vary from 0 to c (ignoring setup and hold times on the arbiter input devices). Figure 3.39 depicts an example with $t_{latch} = .9c$.

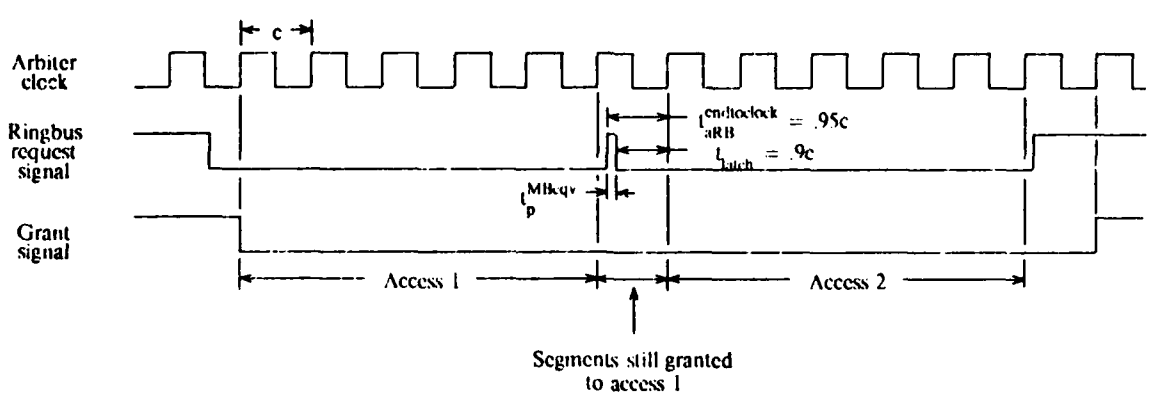


Figure 3.39: Signals when $\bar{t}_p^{MBeqv} = 0$

An approximate relation for t_{latch} is

$$t_{latch} = \begin{cases} t_{aRB}^{endtoclock} & \text{if a Ringbus access is followed immediately by another Ringbus request} \\ .5c & \text{otherwise} \end{cases}$$

Therefore $t_{latch} \approx .5c(1 - P_{RB}) + P_{RB}\bar{t}_{aRB}^{endtoclock} = .5c + P_{RB}(\bar{d} - \bar{t}_{trans} - \bar{t}_{arb} - .5c)$ where P_{RB} is the probability that a Ringbus access is followed immediately by a Ringbus request. In other words, P_{RB} is the probability that the Multibus queue is nonempty at the termination of the present access and that the next request in the Multibus queue is for the Ringbus given that the present access is a Ringbus access. P_{RB} can be determined from the Multibus model: it is another output of the single processor equivalent model of the Multibus.

In summary, we have three inputs to the Ringbus equivalent model from the single processor equivalent model: $\bar{t}_p^{MBeqv}$, p_i^{MBeqv} (for $i = -(S/2-1), \cdots, -1, 1, \cdots$, or $S/2$), and P_{RB}. In addition, we have four other inputs to the Ringbus equivalent model: $\bar{t}_{start}$, $\bar{t}_{arb}$, $\bar{t}_{trans}$, and $\bar{d}$. Note that only means are required for the inputs (except for p_i^{MBeqv} and P_{RB}) to the Ringbus equivalent model. Formally, the output of the Ringbus equivalent model is the probability distribution of t_{aRB}. However, in section 2.9.2 we assumed an exponential probability distribution for t_{aRB} in the single processor equivalent model, which is completely characterized by $\bar{t}_{aRB}$. Thus we only require $\bar{t}_{aRB}$ as an output of the Ringbus equivalent model.

3.10 Conclusions

Conclusions 1 to 5 pertain to the definition of the Ringbus given in section 3.1 and the various assumptions that we made. These assumptions are listed below:

1) even number of slices
2) no propagation delays or metastability settling delays
3) memoryless i.e. geometric - probability distribution for nonnull request arrivals
4) symmetric request probabilities
5) no global registers
6) all slices identical in all respects
7) all probability distributions stationary and all processes in steady-state
8) no bound on request waiting time
9) deterministic or geometrically distributed grant durations of an integral number of arbiter clock periods
10) instantaneous arbitration time (i.e. no arbitration time)
11) no start up time, no end time, and no dead time

1. For six or more slices, the optimum performance of the Ringbus is difficult to determine and analyze - because of the large number of states - even with all the simplifying assumptions.

2. The optimal arbiter algorithm depends strongly on the request probabilities; no one arbiter algorithm is best. In addition, the optimum performance of the Ringbus depends very strongly on the request probabilities. The maximum throughput for requests of length one is between $\frac{2}{3}S$ and $\frac{7}{8}S$ (where S is the number of slices) and the maximum throughput for requests of length $S/2$ is 2. A first order approximation of the dependence of the optimum throughput on the request probabilities is given by $\frac{S}{\bar{l}}$ where $\bar{l}$ is the average request length:

$$\bar{l} = \sum_{i=1}^{S/2-1} \frac{2\, i\, p_i}{(1-p_0)} + \frac{S/2\, p_{S/2}}{(1-p_0)}$$

(Note that $\frac{S}{\bar{l}}$ is part of the upper bound on g^{opt} developed in section 3.5.1.1.)

3. For four slices the optimal arbiter algorithm always grants the maximum number of requests possible in every state, independent of the request probabilities. For six or more slices, the optimal arbiter algorithm does not always grant the maximum number of requests possible in every state. However, for six slices the optimal throughput is not degraded significantly for light to medium loading by restricting the algorithm to grant the maximum number of requests in every state. For heavy loading with mainly very short and very long requests, the optimal throughput is significantly degraded with this maximum request restriction. We expect that this degradation increases with the number of slices.

   For six slices, the optimal arbiter algorithm does not always grant the request set utilizing the maximum number of segments possible in every state either, although the maximum number of segments decision seemed favoured in those states in which the maximum number of requests decision was not favoured. For all request probabilities, the optimal throughput subject to the maximum number of segments in every state is always greater than or equal to the optimal throughput subject to the maximum number of requests restriction in every state. We expect that this result also holds for more than six slices.

   A reasonable sub-optimal arbiter algorithm for six slices is the following:

   In each state select a request subset to grant by choosing arbitrarily from all the request subsets in a state that:

   1. utilize the maximum number of segments
   2. have the maximum number of requests subject to 1, and
   3. have the maximum number of longest requests subject to 1 and 2.

   We expect that this algorithm is also a reasonable sub-optimal arbiter algorithm for more than six slices.

4. For deterministic grant durations of $d>1$ rounds, the optimal arbiter algorithm tends to grant requests immediately for very light loading ($p_0 \approx 1$) and tends to delay and align requests so that they can be granted at intervals of d rounds for very heavy loading ($p_0 \approx 0$). In fact, for $p_0 \approx 0$ the optimal arbiter algorithm is the optimal interval algorithm - i.e. the optimal algorithm subject to the restriction that requests can only be granted at intervals of d rounds. The optimal interval algorithm is the same as the optimal algorithm for $d=1$ and the equivalent request probabilities. The optimal algorithm in between the extremes $p_0 \approx 1$ and $p_0 \approx 0$ is a complex function of the request probabilities and grant duration d. For four slices, the optimum throughput can be estimated fairly closely by the exponential approximation of equation 3.19, which depends only on the optimum throughput for $d=1$. We expect that equation 3.19 also yields a reasonable approximation to the optimum

throughput for more than four slices.

5. The performance of the Concert Ringbus can be improved by making the access paths symmetrical and by modifying the arbiter algorithm. Results for four slices suggest that when counterclockwise requests predominate, the greatest improvement in performance is achieved by making the access paths symmetrical, and when long requests predominate, the greatest improvement in performance is achieved by modifying the arbiter algorithm.

   The performance advantage of symmetrical access paths over asymmetrical access paths is difficult to quantify since users may adapt their behaviour to suit the topology, and thus the request probabilities may change with the topology.

   Symmetrical access paths require three additional set of drivers per slice (see Figure 3.1)† and a more complex arbiter since arbitration must also be performed for request destinations (unlike with asymmetrical access paths). As discussed in section 3.1, the Concert Ringbus arbiter is easily modified to perform this arbitration for destinations but the number of parts required doubles.

   It must be cautioned that modifying the arbiter algorithm may not improve the performance to the degree suggested by the results in this chapter since we have ignored two important issues. These are 1) the realizability of the optimum arbiter algorithm in a reasonable amount of hardware and 2) the arbitration time required by a realization. The arbitration time obviously degrades performance and if sufficiently large, it may negate any possible gain in performance. We have also ignored the practical requirement for a bounded request waiting time. However, provided that the maximum permissible waiting time may be sufficiently large, the degradation that this requirement imposes is minimal.

   The performance of the Concert Ringbus arbiter can be improved by either of two trivial changes (or possibly both) to the arbiter priority ROM. Results for four slices indicate these changes yield only minor improvements in performance. However, the magnitude of these improvements should increase with the number of slices.

6. Since a crossbar interconnection has the best performance achievable (where the interconnection must be circuit-switched with S sources and S destinations) and is popular and well known, it is interesting to compare the Ringbus and a crossbar interconnection. We make such a comparison on the next page, dividing the comparison into the following three areas: performance, hardware costs, and arbitration costs.

† One set of drivers is required for each unidirectional switch and two sets are required for each bidirectional switch.

Performance

The optimal throughput of the Ringbus is close to that for a crossbar when either the loading is light or short requests predominate (or both). Otherwise, the optimal throughput of the Ringbus is significantly less than that of a crossbar. This degradation in throughput relative to that of a crossbar is especially severe in heavy loading when long requests predominate.

Hardware Costs

To connect S sources to S destinations, the crossbar interconnection requires S^2 drivers whereas the Symmetric Ringbus requires $6S$ drivers and the Concert Ringbus requires $3S$ drivers. The Ringbus also requires more hardware for arbitration than a crossbar does, but the difference is difficult to quantify.

Arbitration Costs

Arbitration for the Ringbus must be centralized whereas arbitration for a crossbar may be distributed amongst the destinations. Consequently, an arbiter for the Ringbus - especially an optimal arbiter - can be much more complex than an arbiter for a crossbar.

Any final conclusion in comparing the Ringbus and crossbar interconnections (or any other interconnection) depends on the number of slices, the expected operating point (i.e. the request probabilities), and the relative importance of performance versus cost. Certainly, the Ringbus seems well suited for predominantly short requests and unattractive for predominantly long requests.

7. The scalability of the Ringbus past eight or so slices is doubtful because of the complexities of the centralized arbitration and control.

3.11 Suggestions for Future Work

The following suggestions are listed in order of perceived importance.

1. Explore the performance, hardware cost/arbitration time, and maximum waiting time tradeoffs of various algorithms and implementations in an attempt to identify an ideal arbitration algorithm and implementation. At least investigate various implementations for optimal or near-optimal arbitration algorithms (such as the algorithm mentioned in conclusion 3 of section 3.10).

2. Remove as many of the eleven assumptions listed in section 3.10 as possible. The most important assumption to remove is that of zero dead time. In the Concert Ringbus arbiter, as in any other arbiter implementation†, there must be at least one round between successive nonnull requests in order to identify new requests. Other factors, such as the minimum processing time of processors and the Ringbus arbitration time (since a new request cannot be granted until after the grant from the previous request, delayed by the arbitration time, terminates) contribute to a nonzero dead time in practice. We feel that a nonzero dead time is an important addition to make to improve the accuracy of our Ringbus model, especially in heavy loading.

Removal of assumptions 3 and 9 to consider arbitrary nonnull request interarrival time and grant duration probability distributions, would be ideal. Such a generalization of our Ringbus model would not only lead to more accurate modeling of request arrivals and grant durations, but also allow the removal of other assumptions. As discussed in section 3.9, a nonzero arbitration time can be treated by assuming instantaneous arbitration and suitably apportioning the arbitration time between request interarrival time and grant duration. Any start up time, end time, or propagation delays can be treated by a similar apportioning between request interarrival time and grant duration. Unfortunately, arbitrary request interarrival and grant duration probability distributions would seem to make the Ringbus unreasonably difficult to analyze. Hence any practical generalization in this direction is likely to be just an extension of our treatment by special cases.

It would be worthwhile to consider more slices in the Ringbus model but the large number of states required makes an exact analysis difficult and costly.

Conceptually, there is no difficulty in removing assumptions 1,4,5, and 6 (see list of assumptions in section 3.10). However, there is the practical difficulty that the analysis becomes complicated. This is especially true for the removal of assumptions 4 and 6 since the symmetry that we exploited so heavily and successfully to ease the analysis will not exist. It would probably be best to have a specific situation in mind before pursuing the removal of any of the assumptions 1, 4, 5, and 6.

† In making this statement, we assume that the only information available to the arbiter from a slice is whether or not a request is present and if so, the destination of the request. This is the only information available to the arbiter in Concert.

3. Investigate the degree to which the performance of the Ringbus may be improved by making additional information - such as the number and type of requests in each Multibus queue and the waiting time so far of each request - available to the Ringbus arbiter.

4. Consider other metrics for the performance of the Ringbus such as minimizing the maximum waiting time of requests.

5. Establish the validity of the conjecture in section 3.4.2.2 that when $p_0=0$ $g_{\vec{p}}^{d>1}=g_{\vec{p}}^{d=1}$, i.e. when $p_0=0$ the optimal average number of grants per round with deterministic grant durations of d rounds equals the optimal average number of grants per round with grant durations of 1 round, assuming the nonnull request probabilities are the same in each case.

Chapter 4

Integration and Simulation

4.1 Introduction

In this chapter we consider the integration of the Multibus submodel, discussed in Chapter 2, and the Ringbus submodel, discussed in Chapter 3. We describe the results of the integration for a few example cases and compare these results to those obtained via simulation of the overall Concert model. In the rest of the chapter we present and discuss the results of two different sets of simulations of the overall Concert model with eight slices. The purpose of the first set is to assess the performance of the Ringbus with different access paths and arbiter algorithms and to compare this performance with that of other interconnection architectures in an environment close to that in the actual Concert system. Such a comparison would be too computationally expensive to perform by solving the associated Markovian decision problems. The purpose of the second set is to determine the expected performance of the actual Concert system for various parameter values. The variables considered in these simulations are the number of processors in a slice, the mean processing time, and the request destination probabilities.

4.2 Integration

Summarizing the results of sections 2.9.2 and 3.9.1 we have:

a) The Single Processor Equivalent Model

Input: $\bar{t}_{aRB}$ (the mean Ringbus access time)

Exogenous Inputs: N (the number of processors on a Multibus), t_p (the mean processing time), $\bar{t}_r$ (the mean recovery time), $\bar{t}_{aMB}$ (the mean Multibus access time), p_i^{RB} (the Ringbus destination probabilities), β (the probability of a long word access), and ψ (the probability of a Ringbus access).

Outputs: $\bar{t}_p^{MBeqv}$, p_i^{MBeqv} (the mean processing time and destination probabilities, respectively, of the single processor equivalent model of the Multibus), P_{RB} (the conditional probability that, given a Ringbus access, that access is immediately followed by another Ringbus access)

Computation: $p_i^{MBeqv} = p_i^{RB}$ for $i = -(S/2-1), \cdots, -1, 1, \cdots$, or $S/2$

$$\bar{t}_p^{MBeqv} = \frac{\bar{t}_p + \beta \bar{t}_r}{(1+\beta)N\psi} + ((1-\psi)\bar{t}_{aMB} + \psi \bar{t}_{aRB})\frac{(\bar{t}_w/\bar{t}_a + 1)}{N\psi} - \bar{t}_{aRB}$$

where

$$\frac{\bar{t}_w}{\bar{t}_a} = \frac{\mu}{\lambda} \cdot \frac{\sum_{k=2}^{N} \frac{N!(k-1)}{(N-k)!}\left[\frac{\mu}{\lambda}\right]^{-k}}{\sum_{k=0}^{N-1} \frac{N!}{(N-k-1)!}\left[\frac{\mu}{\lambda}\right]^{-k}},$$

$\frac{\mu}{\lambda} = \frac{\bar{t}_p + \beta \bar{t}_r}{\bar{t}_a}$, and $\bar{t}_a = (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi \bar{t}_{aRB})$.

$$P_{RB} = \frac{\psi\zeta}{1-\psi+\psi\zeta}\rho^{N-1}$$

where $\zeta = \frac{\bar{t}_{aRB}}{\bar{t}_{aMB}}$, and $\rho^{N-1} = \frac{1}{\sum_{k=0}^{N-1} \frac{N!}{(N-k)!}\left[\frac{\mu}{\lambda}\right]^{-k}}$.

b) The Ringbus Equivalent Model

Inputs: $\bar{t}_p^{MBeqv}$, p_i^{MBeqv}, P_{RB}

Exogenous Inputs: S (the number of slices), $\bar{t}_{start}$ (the mean start up overhead), $\bar{t}_{arb}$ (the mean Ringbus arbitration time), $\bar{t}_{trans}$ (the mean Ringbus data transfer time), $\bar{d}$ (the mean duration for which Ringbus segments are allocated to a request and related to $\bar{t}_{trans}$ by $d = t_{arb} + \left\lceil \frac{t_{trans}}{c} \right\rceil c$), c (the Ringbus arbiter clock period), type of Ringbus access paths, and the Ringbus arbitration algorithm.

Output: $\bar{t}_{aRB}$

Computation: $\bar{t}_{aRB} = \bar{t}_{aRB}^{(norm)} + \bar{w}_{RB}$ where $\bar{t}_{aRB}^{(norm)} = \bar{t}_{start} + \bar{t}_{latch} + \bar{t}_{arb} + \bar{t}_{trans}$.

$$\bar{t}_{latch} = .5c + P_{RB}(\bar{d} - \bar{t}_{trans} - \bar{t}_{arb} - .5c)$$

$\bar{w}_{RB}$ is determined from

$$\frac{S}{\frac{p_0}{1-p_0} + \bar{w}_{RB} + \bar{d}} = \frac{g}{\bar{d}}.$$

g, the average number of new or continuing grants per round, is found by solving the Ringbus model with parameters $p_i = \frac{p_i^{MBeqv}}{1-p_0}$ and p_0 where

$$\frac{c \cdot p_0}{1-p_0} = \bar{t}_p^{MBeqv} + \bar{t}_{aRB}^{(norm)} - \bar{d}.$$

This is an approximation - see the footnote in section 2.9.2. Assuming all the quantities are deterministic, we have $\frac{c \cdot p_0}{1-p_0} = \begin{cases} \bar{t}_p^{MBeqv} + \bar{t}_{aRB}^{(norm)} - \bar{d}, & \text{if } \bar{t}_p^{MBeqv} + \bar{t}_{start} \geq \bar{d} - \bar{t}_{trans} \\ \bar{t}_{arb} + c, & \text{otherwise} \end{cases}$

For a given set of exogenous inputs in (a) and (b), integration consists of matching the input in (a) with the output in (b) and matching the outputs in (a) with the inputs in (b). This can be done iteratively, as outlined in the following steps. The subscripts k on $\bar{t}_{aRB}$, $\bar{t}_p^{MBeqv}$, and p_i^{MBeqv} denote successive estimates of the true values of these respective quantities.

1) $k \leftarrow 0$. Assume some initial value for $\bar{t}_{aRB}$; denote it by $(\bar{t}_{aRB})_0$.
2) Using $(\bar{t}_{aRB})_k$, determine $(\bar{t}_p^{MBeqv})_k$ and $(p_i^{MBeqv})_k$ for the single processor model of the Multibus.
3) Using $(\bar{t}_p^{MBeqv})_k$ and $(p_i^{MBeqv})_k$, determine $(\bar{t}_{aRB})_{k+1}$ for the Ringbus equivalent model.
4) $k \leftarrow k+1$. If the estimates of $\bar{t}_{aRB}$, $\bar{t}_p^{MBeqv}$, and p_i^{MBeqv} are satisfactory, stop. Otherwise go to step 2.

The iteration can begin instead by assuming some initial values for $\bar{t}_p^{MBeqv}$ and p_i^{MBeqv} and then estimating $\bar{t}_{aRB}$. Note that since we employ various approximations in obtaining the equivalent models of the Multibus and the Ringbus (principally approximating the interaction between the two models by first moments), the final estimates for $\bar{t}_{aRB}$ and $\bar{t}_p^{MBeqv}$, will not necessarily equal their true values. We did not investigate the convergence properties of the above iterative procedure. However, we found that the estimates converged rapidly whenever we used it.

There are two cases for which the Multibus and Ringbus models can be integrated without resorting to iteration:

Case 1: Very light Ringbus traffic

This case can arise in two ways:

i) $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ large, ψ irrelevant, or

ii) ψ small (i.e. $\psi \approx 0$), $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ irrelevant.

The first way corresponds to very light utilization of the Multibus, which leads to very light utilization of the Ringbus regardless of ψ. The second way corresponds to very minor coupling between the Multibus and Ringbus, which leads to very light utilization of the Ringbus regardless of the utilization of the Multibus. Of course, very light Ringbus traffic can be achieved both ways simultaneously. However, in our treatment below we choose to consider each way as a distinct subcase.

Case 1(i): $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ large, ψ irrelevant

For $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ sufficiently large, we have $\bar{t}_w \approx 0$, $\rho^{N-1} \approx 0$ (and hence $\bar{t}_{latch} \approx .5c$), $\bar{t}_p^{MBeqv} \approx \frac{\bar{t}_p + \beta \bar{t}_r}{(1+\beta)N\psi}$, $\bar{w}_{RB} \approx 0$, and

$\bar{t}_{aRB} \approx \bar{t}_{aRB}^{(norm)} \approx \bar{t}_{start} + .5c + \bar{t}_{arb} + \bar{t}_{trans}$. Thus $\bar{t}_p^{MBeqv}$ and $\bar{t}_{aRB}$ and all the other quantities of interest can be found without resorting to iteration.

Case 1(ii): ψ small (i.e. $\psi \approx 0$) and $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ irrelevant

In this case $\bar{t}_p^{MBeqv}$ is very large, $P_{RB} \approx 0$ (and hence $\bar{t}_{latch} \approx .5c$), $\bar{w}_{RB} \approx 0$, and $\bar{t}_{aRB} \approx \bar{t}_{aRB}^{(norm)} \approx \bar{t}_{start} + .5c + \bar{t}_{arb} + \bar{t}_{trans}$. $\bar{t}_w$ may be computed by taking $\bar{t}_a \approx (1+\beta)\bar{t}_{aMB}$. Note that it is the possibly large value of $\bar{t}_w$ that differentiates this subcase from the previous one.

Case 2: Very heavy Ringbus traffic

This case arises when both $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ is small and ψ is large (i.e. $\psi \approx 1$). For $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ sufficiently small, the Multibus is saturated (i.e. $N \gg N^*$ where N^* is the saturation point of the Multibus) and hence $\bar{t}_w \approx (N - N^*)\bar{t}_a$, yielding $\bar{t}_p^{MBeqv} \approx \bar{t}_{aMB} \frac{1-\psi}{\psi}$ as in section 2.9.2. In addition $\rho^{N-1} \approx 1$ and thus $P_{RB} \approx \frac{\psi\zeta}{1-\psi+\psi\zeta}$. With $\psi \approx 1$, $\bar{t}_p^{MBeqv} \approx 0$ and $P_{RB} \approx 1$, hence $\bar{t}_{latch} \approx d - \bar{t}_{arb} - \bar{t}_{trans}$. Therefore $\bar{t}_p^{RBeqv}$ is a constant.

Assuming d, t_{arb}, and t_{trans} are deterministic random variables

$$\bar{t}_p^{RBeqv} = \begin{cases} 0 + \bar{t}_{start} + \bar{t}_{latch} + \bar{t}_{arb} + \bar{t}_{trans} - \bar{d} = \bar{t}_{start}, & \text{if } \bar{t}_{start} \geq \bar{d} - \bar{t}_{trans} \\ \bar{t}_{arb} + c, & \text{otherwise.} \end{cases}$$

Once p_0 and the corresponding g are determined, $\bar{w}_{RB}$ is given by

$$\frac{S}{\bar{t}_p^{RBeqv} + \bar{w}_{RB} + \bar{d}} = \frac{g}{\bar{d}}.$$

Finally, $\bar{t}_{aRB} = \bar{t}_{start} + \bar{t}_{latch} + \bar{t}_{arb} + \bar{t}_{trans} + \bar{w}_{RB} = \bar{t}_{start} + \bar{d} + \bar{w}_{RB}$. Thus $\bar{t}_p^{MBeqv}$ and $\bar{t}_{aRB}$ and all the other quantities of interest can be found without resorting to iteration.

Note that for small enough $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ and ψ close enough to 1, $\bar{t}_p^{MBeqv} \approx 0$ regardless of the various probability distributions in the Multibus and Ringbus models. In this case the probability distribution of t_p^{MBeqv} is given very accurately by $\bar{t}_p^{MBeqv}$. (It in fact becomes exact for $\psi = 1$ as $\frac{\bar{t}_p + \beta \bar{t}_r}{N} \rightarrow 0$ since $t_p^{MBeqv} \rightarrow 0$.) Hence our first moment approximation of the Multibus to Ringbus interaction is very accurate for $\frac{\bar{t}_p + \beta \bar{t}_r}{N}$ small and $\psi \approx 1$.

We now consider some example cases. In each case we determine t_p^{MBeqv} and $\bar{t}_w$ for the Multibus model and $\bar{t}_{aRB}$ and $\bar{w}_{RB}$ for the Ringbus model.

All the simulations reported in this section and in this chapter are simulations of the overall Concert model. As discussed in section 1.3.5, this overall model is comprised of a model for each Multibus and a model for the Ringbus. As a model for each Multibus we choose the Multibus model with long word and Ringbus accesses discussed in section 2.9. As a model for the Ringbus, we choose the basic model discussed in section 3.1. This model depends, of course, on the particular arbitration algorithm and access paths desired. In simulating the overall Concert model, we simulate each Multibus model, the Ringbus model, and the interaction between Multibus models and the Ringbus model. Since our Multibus models are continuous time models, we simulate them in continuous time and since our Ringbus models are discrete time models, we simulate them in discrete time. We simulate the Multibus models as operating asynchronously with respect to the Ringbus model; thus our simulations include the effect of synchronizing the Multibus signals with the Ringbus arbiter clock. The parameters of our simulations are as follows:

Multibus model:

- the number of processors on a Multibus, N.
- the processing time distribution, with mean $\bar{t}_p$.
- the recovery time distribution, with mean $\bar{t}_r$.
- the Multibus access time distribution, with mean $\bar{t}_{aMB}$.
- the probability of a long word access, β.
- the probability of a Ringbus access, ψ.
- the Ringbus destination probabilities, p_i^{RB}.

Ringbus model:

- the number of slices, S.
- the arbiter algorithm.
- the Ringbus access paths.
- the arbiter clock period, c.
- the start up overhead, $\bar{t}_{start}$. (Taken as a constant.)
- the Ringbus arbitration time, $\bar{t}_{arb}$. (Taken as a constant and an integral multiple of c.)
- the probability distribution of the Ringbus data transfer time, with mean t_{trans}. (Note that the duration, d for which segments are allocated to a Ringbus request is related to t_{trans} by $d = t_{arb} + \left\lceil \frac{t_{trans}}{c} \right\rceil c$.)

Other:

- the block size, B. Each simulation was run until processor 1 on Multibus 1 (this numbering is arbitrary) completed $30 + B$ processor cycles (i.e. processing time, waiting, access time for word or long word). To remove the effect of transients, statistic gathering did not begin until processor 1 on Multibus 1 completed 30 accesses.
- the number of block repetitions, R. The statistics reported are based on R repetitions of each simulation.

We remind the reader of our basic assumptions, which apply to our simulations as well:

- We assume that all the random variables t_p, t_r, t_{aMB}, and t_{trans} and all the probabilities β, ψ, and p_i^{RB} are mutually independent and stationary.
- We assume each Multibus model has the exactly the same parameters, so all Multibus models are identical in every respect.
- We assume that the Ringbus model is completely symmetric with respect to each Multibus interconnection.

In all our simulations we assume in addition that:

1) the processing time is exponentially distributed,
2) the recovery time is deterministic, and
3) the Multibus access time is deterministic.

We caution that the following examples were chosen for purposes of illustration. They do not represent the actual Concert system and they do not represent an in-depth study or analysis of integration. In each case we assume that the Ringbus arbiter has zero arbitration time (i.e. $t_{arb}=0$) and that there are no long word accesses (i.e. $\beta=0$, and hence we take $\bar{t}_r=0$ and $\gamma=0$). In addition, we take $\bar{t}_{start}=0$ and $\bar{t}_{aMB}=1.0c$.

Example 1: $\bar{t}_p=1.0c$, $S=4$, deterministic grant duration of one round i.e. $\bar{d}=c$, optimal arbiter for deterministic grant duration of one round, symmetrical access paths, $p_{-1}^{RB}=p_1^{RB}=.4$, and $p_2^{RB}=.2$.

Table 4.1 presents the integration and simulation results for various values of N, ψ, and $\bar{t}_{trans}$ (which is a deterministic value here).

N	ψ	$\bar{t}_{trans}/c$		$\bar{t}_p^{MBeqv}/c$	$\bar{t}_w/c$	$\bar{t}_{latch}/c$	$\bar{t}_{aRB}/c$	w_{RB}/c	g
1	0.5	0.01	Integration	3.00	0.0	0.50	0.64	0.13	1.10
			Simulation	2.95±.20	0.0	0.54±.01	0.68†	0.13±.03	1.10±.06
1	0.5	0.98	Integration	3.00	0.0	0.50	1.55	0.07	0.88
			Simulation	3.00±.16	0.0	0.53±.02	1.60	0.09±.02	0.87±.03
1	1.0	0.01	Integration	1.00	0.0	0.50	0.87	0.36	2.14
			Simulation	1.00±.06	0.0	0.58±.01	0.94	0.35±.01	2.05±.07
1	1.0	0.98	Integration	1.00	0.0	0.50	1.74	0.26	1.46
			Simulation	1.00±.03	0.0	0.58±.02	1.73	0.17±.02	1.47±.02
2	0.5	0.01	Integration	1.53	0.42	0.62	0.78	0.16	1.73
			Simulation	1.40±.09	0.40±.02	0.77±.01	1.02	0.24±.02	1.64±.06
2	0.5	0.98	Integration	1.44	0.69	0.37	1.50	0.15	1.36
			Simulation	1.34±.03	0.54±.02	0.29±.01	1.46	0.19±.03	1.43±.03
2	1.0	0.01	Integration	0.22	0.73	0.78	1.30	0.51	2.64
			Simulation	0.17±.01	0.70±.01	0.88±.01	1.38	0.49±.01	2.58±.04
2	1.0	0.98	Integration	0.19	0.99	0.21	1.60	0.42	2.23
			Simulation	0.13±.01	0.84±.04	0.18±.01	1.59	0.43±.02	2.32±.02
4	0.5	0.98	Integration	1.02	2.76	0.27	1.48	0.23	1.60
			Simulation	1.00±.05	2.41±.05	0.03±.01	1.28	0.27±.03	1.75±.03
4	1.0	0.98	Integration	0.01	3.55	0.03	1.51	0.50	2.64
			Simulation	0.00±.001	3.51±.07	0.02±.04	1.50	0.50±.02	2.66±.03
6	0.5	0.98	Integration	1.00	5.17	0.26	1.47	0.23	1.62
			Simulation	1.00±.04	4.67±.04	0.02±.001	1.27	0.27±.01	1.76±.03
6	1.0	0.98	Integration	0.00	6.58	0.02	1.52	0.52	2.64
			Simulation	0.00±.001	6.54±.06	0.02±.001	1.51	0.51±.01	2.64±.02

Table 4.1: $\bar{t}_p = 1.0c$, $S = 4$, deterministic grant duration of one round i.e. $\bar{d} = c$, optimal arbiter for deterministic grant duration of one round, symmetrical access paths, $p_{-1}^{RB} = p_1^{RB} = .4$, and $p_2^{RB} = .2$.

In general, the integration and simulation results agree rather closely. The results are closest for light Ringbus loading ($N = 1$, $\psi = .5$) and very heavy Ringbus loading ($N = 4$ and 6, $\psi = 1.0$). This is to be expected since

1) the analytical formulae describing the Multibus model are the most accurate for general

† $\bar{t}_{aRB}$ was not one of the statistics gathered by the simulations. In each row corresponding to a simulation in this table and in the other tables, $\bar{t}_{aRB}$ was computed from the relation $\bar{t}_{aRB} = \bar{t}_{start} + \bar{t}_{latch} + \bar{w}_{RB} + \bar{t}_{trans}$ where $\bar{t}_{start} = 0$.

probability distributions for $N \ll N^*$ and $N \gg N^*$ where N^* is the Multibus saturation point, and

2) the first moment approximation of the Multibus-Ringbus interaction is fairly accurate for light Ringbus loading (and $N = 1$) and heavy Ringbus loading (and $N \gg N^*$). In the first case, since Ringbus traffic is light, $\bar{w}_{RB} \approx 0$ and $\bar{t}_{latch} \approx .5c$. Thus $\bar{t}_{aRB} \approx \bar{t}_{start} + .5c + \bar{t}_{arb} + \bar{t}_{trans}$. Since $N = 1$, the Multibus queue (in the Multibus model) is quasi-reversible regardless of the Multibus and Ringbus access time distributions and thus the analytical formulae of the Multibus are exact with only the mean Ringbus access time, $\bar{t}_{aRB}$. In the second case, both the Multibus and Ringbus are saturated. In saturation only the means of the various quantities are required to determine $\bar{t}_w$, $\bar{w}_{RB}$, and $\bar{t}_{aRB}$.

Note that light Ringbus loading and very heavy Ringbus loading are two cases - as discussed earlier - for which integration can be performed without iteration.

The results for various values of $\bar{t}_{trans}$ (for $N = 1$ and 2) are presented in Table 4.1 to determine the effect of $\bar{t}_{trans}$ on the accuracy of the integration results. In all of the Ringbus models that we investigated in detail in Chapter 3 (i.e. the models in section 3.3, 3.4, 3.5, and 3.8) - including the optimal arbiter with a deterministic grant duration of one round, as in Example 1 - we assumed that the probability, p_0, of a null Ringbus request was independent of all other requests on the Ringbus. However, the probability of a null request at the Ringbus in our Concert model can depend on the previous requests at the Ringbus. The reason is as follows.

First we introduce some terminology. We term a request latched by the Ringbus arbiter a latched Ringbus request or a LRB request for short. In addition, we call the arrival of a nonnull Ringbus request from a Multibus an arrival event. Now, if the previous LRB request at a slice is a null request then the next LRB request at that slice will also be a null request if there is no arrival event at that slice in the arbiter clock period following the latching of the previous null request. On the other hand, if the previous LRB request at a slice is a nonnull request, then the next LRB request at that slice will be a null request if there is no arrival event at that slice in the interval between the termination of the Ringbus access (the data transfer, not the interval for which segments are allocated) of the previous LRB request and the next latching instant. These two situations are depicted in Figure 4.10. (Remember that $\bar{t}_{start} = 0$ and $\bar{t}_{arb} = 0$ here.)

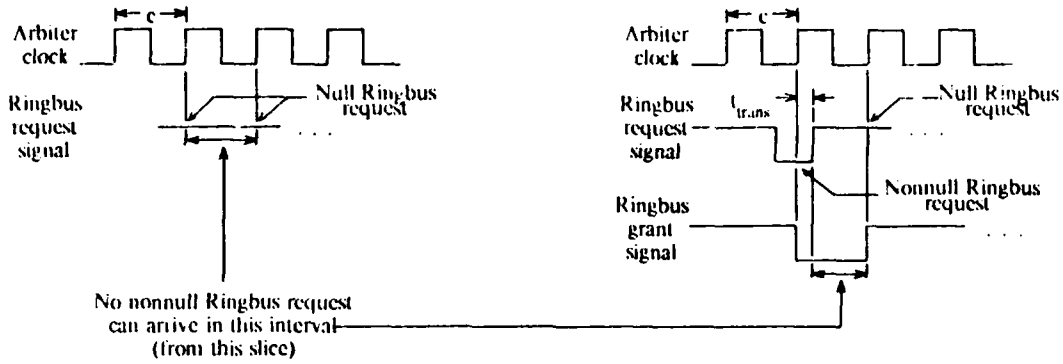


Figure 4.1: Two situations leading to a null request

Thus a null Ringbus request follows a null Ringbus request if no Ringbus request arrives from the Multibus in an interval c and a null Ringbus request follows a nonnull Ringbus request if no Ringbus request arrives from the Multibus in an interval $d - t_{trans} < c$. Note that if $t_{trans} = d$, then a null Ringbus request must follow every nonnull Ringbus request. To avoid this - since the Ringbus model in this example (and all the other examples) does not incorporate a null request after every nonnull request - we take $t_{trans} = d - e$ for some constant e, $0 < e < c$ in all the cases in this section.

If $N = 1$, then with our assumption of exponential processing time, the probability of a null LRB request following a null LRB request is proportional to c and the probability of a null LRB request following a nonnull LRB is proportional to $d - \bar{t}_{trans}$. (t_{trans} and d are deterministic in this example.) By taking $\bar{t}_{trans}$ very small ($.01c$), we minimize the dependency of the probability of a null LRB request on the previous LRB request at the same slice. By taking $\bar{t}_{trans}$ large ($.98c$) we increase this dependency.

If N is large and ψ large so that the Multibus queue is nearly always nonempty with Ringbus requests, then $t_p^{MBequ} \approx 0$ and the likelihood of a nonnull Ringbus request arriving in c or $d - t_{trans}$ is about the same (as long as $d > t_{trans}$). Thus the interval $d - t_{trans}$ has small effect on the probability of a null LRB request for large N and $\psi \approx 1.0$.

Thus we expect the probability of a null LRB request to depend quite heavily on $d - t_{trans}$ for light Ringbus traffic and diminish as the Ringbus traffic increases. As we stated earlier, the Ringbus model used in the integration does not incorporate the dependency of null request probabilities on $d - t_{trans}$. It might be expected that the integration results would be most accurate when $d - t_{trans}$ is adjusted to reduce this dependency. Indeed, this does seem to be the case for $N = 1$ and $\psi = 1.0$: the integration and simulation results for $\bar{t}_{trans} = .01c$ are closer than those for

$\bar{t}_{trans} = .98c$. For other values of N and ψ there seems to be no conclusive link. In fact, for $N = 4$ and 6, the value of $\bar{t}_{trans}$ seemed to make no difference (as long as $d - t_{trans} < c$), as expected. (Hence only the results for $\bar{t}_{trans} = .98c$ are shown in Table 4.1.)

Example 2: $\bar{t}_p = 1.0c$, $S = 4$, deterministic grant duration of one round i.e. $\bar{d} = c$, rotating priority with counterclockwise rotation, asymmetrical access paths, $p_{-1}^{RB} = p_1^{RB} = .25$, and $p_2^{RB} = .5$.

The integration and simulation results are contained in Table 4.2. Again the results agree rather closely and again the results are closest for light and very heavy Ringbus loading.

N	ψ	$\bar{t}_{trans}/c$		$\bar{t}_p^{MBeqv}/c$	$\bar{t}_w/c$	$\bar{t}_{latch}/c$	$\bar{t}_{aRB}/c$	$\bar{w}_{RB}/c$	g
1	0.5	0.01	Integration	3.00	0.0	0.50	0.88	0.37	1.03
			Simulation	2.98±.70	0.0	0.54±.04	0.91	0.36±.12	1.03±.17
1	0.5	0.98	Integration	3.00	0.0	0.50	1.76	0.28	0.84
			Simulation	2.84±.37	0.0	0.54±.02	1.77	0.25±.13	0.86±.06
1	1.0	0.01	Integration	1.00	0.0	0.50	1.48	0.97	1.61
			Simulation	0.97±.12	0.0	0.58±.02	1.56	0.97±.14	1.57±.03
1	1.0	0.98	Integration	1.00	0.0	0.50	1.82	0.34	1.42
			Simulation	0.99±.04	0.0	0.57±.01	1.96	0.41±.09	1.35±.04
2	0.5	0.01	Integration	1.47	0.61	0.63	1.29	0.65	1.45
			Simulation	1.37±.08	0.56±.04	0.81±.04	1.42	0.60±.07	1.43±.04
2	0.5	0.98	Integration	1.42	0.82	0.36	1.81	0.47	1.24
			Simulation	1.31±.28	0.69±.08	0.27±.02	1.80	0.55±.14	1.29±.10
2	1.0	0.01	Integration	0.15	1.56	0.84	2.26	1.41	1.66
			Simulation	0.08±.03	1.52±.14	0.94±.02	2.32	1.37±.08	1.66±.05
2	1.0	0.98	Integration	0.15	1.59	0.17	2.29	1.14	1.64
			Simulation	0.08±.02	1.54±.03	0.12±.02	2.37	1.27±.09	1.63±.05
4	0.5	0.98	Integration	1.02	3.08	0.27	1.67	0.42	1.49
			Simulation	0.98±.01	3.15±.20	0.03±.01	1.75	0.74±.14	1.46±.07
4	1.0	0.98	Integration	0.00	6.23	0.03	2.41	1.40	1.66
			Simulation	0.00±.01	6.14±.19	0.02±.01	2.39	1.39±.06	1.67±.05

Table 4.2: $\bar{t}_p = 1.0c$, $S = 4$, deterministic grant duration of one round i.e. $\bar{d} = c$, rotating priority with counterclockwise rotation, asymmetrical access paths, $p_{-1}^{RB} = p_1^{RB} = .25$, and $p_2^{RB} = .5$.

Example 3: $\bar{t}_p = 1.0c$, $S = 4$, geometrically distributed grant duration with mean $\bar{d} = 4c$, optimal arbiter for geometric duration with mean $\bar{d} = 4c$, symmetrical access paths, $p_{-1}^{RB} = p_1^{RB} = .4$, and $p_2^{RB} = .2$.

The integration and simulation results are shown in Table 4.3. In obtaining these results we took $t_{trans} = d - .02c$, hence $\bar{t}_{latch} = .02c$ for heavy Ringbus loading. Note that once again the integration and simulation results agree rather closely.

N	ψ		$\bar{t}_p^{MBeqv}/c$	$\bar{t}_w/c$	$\bar{t}_{latch}/c$	$\bar{t}_{aRB}/c$	$\bar{w}_{RB}/c$	g
1	0.5	Integration	3.00	0.0	0.50	6.58	2.10	1.67
		Simulation	2.95±.40	0.0	0.51±.03	6.50	2.01±.35	1.66±.09
1	1.0	Integration	1.00	0.0	0.50	7.21	2.73	1.95
		Simulation	1.00±.10	0.0	0.57±.06	7.20	2.65±.44	1.92±.04
2	0.5	Integration	1.20	3.20	0.31	7.01	2.72	1.95
		Simulation	1.18±.12	3.22±.26	0.27	7.06	2.81±.53	1.94±.14
2	1.0	Integration	0.06	6.43	0.08	7.31	3.26	2.17
		Simulation	0.03±.01	6.28±.78	0.07±.01	7.30	3.25±.57	2.16±.12
4	0.5	Integration	1.00	11.12	0.26	7.08	2.84	1.98
		Simulation	1.03±.08	10.86±.74	0.20±.01	7.11	2.93±.33	2.02±.09
4	1.0	Integration	0.00	20.92	0.02	7.31	3.31	2.19
		Simulation	0.00±.01	20.99±.82	0.02±.01	7.36	3.36±.12	2.17±.03

Table 4.3: $\bar{t}_p = 1.0c$, $S = 4$, geometrically distributed grant duration with mean $\bar{d} = 4c$, optimal arbiter for geometric duration with mean $\bar{d} = 4c$, symmetrical access paths, $p_{-1}^{RB} = p_1^{RB} = .4$, and $p_2^{RB} = .2$.

Example 4: $S = 4$, deterministic grant duration of one round i.e. $d = c$, optimal arbiter for deterministic grant duration of one round, symmetrical access paths, $p_{-1} = p_1 = .4$, $p_2 = .2$, and $t_{trans} = .98c$ (i.e. t_{trans} deterministic).

This example is the same as Example 1 except for the value of $\bar{t}_p$. The object of this example is to examine the accuracy of the integration results when the Multibus is operating in the knee region i.e. for $N \approx N^*$. We have already seen in the previous examples and have discussed that the integration results are the most accurate for light Ringbus loading and very heavy Ringbus loading.

We attempted to keep $\frac{\bar{t}_p + \beta \bar{t}_r}{\bar{t}_a}(\bar{t}_a - (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi \bar{t}_{aRB})$ approximately equal to 5.

(This corresponds to $\alpha \approx 5$ in the M/M/1//N model discussed in section 2.4.) For this value of $\frac{\bar{t}_p + \beta \bar{t}_r}{\bar{t}_a}$, N^* is approximately 6; thus we consider $N = 2$, 4, 6, and 8. For $\psi = .5$, we took $\bar{t}_p = 6.0c$ and for $\psi = 1.0$ we took $\bar{t}_p = 7.5c$. (Recall that $\bar{t}_{aMB} = 0$ and $\beta = 0$.) The corresponding integration and simulation results are shown in Table 4.4.

N	ψ	$\bar{t}_{trans}/c$		$\bar{t}_p^{MBeqv}/c$	$\bar{t}_w/c$	$\bar{t}_{latch}/c$	$\bar{t}_{aRB}/c$	$\bar{w}_{RB}/c$	g
2	0.5	0.98	Integration	6.05	0.26	0.46	1.50	0.06	0.53
			Simulation	5.81±.48	0.13±.01	0.46±.46	1.50	0.06±.01	0.55±.55
4	0.5	0.98	Integration	2.58	0.79	0.39	1.42	0.06	1.00
			Simulation	2.42±.07	0.52±.02	0.33±.01	1.43	0.12±.01	1.04±.02
6	0.5	0.98	Integration	1.54	1.85	0.33	1.49	0.19	1.32
			Simulation	1.45±.03	1.23±.02	0.19±.01	1.36	0.19±.01	1.42±.02
8	0.5	0.98	Integration	1.16	3.41	0.29	1.51	0.24	1.50
			Simulation	1.10±.02	2.44±.06	0.08±.01	1.31	0.25±.01	1.66±.01
2	1.0	0.98	Integration	3.12	0.26	0.42	1.54	0.14	0.86
			Simulation	3.08±.13	0.15±.02	0.43±.01	1.51	0.10±.02	0.87±.02
4	1.0	0.98	Integration	1.00	0.96	0.28	1.49	0.23	1.61
			Simulation	0.91±.04	0.65±.03	0.27±.01	1.50	0.25±.01	1.66±.03
6	1.0	0.98	Integration	0.35	2.19	0.15	1.52	0.39	2.14
			Simulation	0.27±.01	1.75±.06	0.14±.01	1.53	0.41±.01	2.23±.01
8	1.0	0.98	Integration	0.11	4.04	0.08	1.53	0.47	2.45
			Simulation	0.06±.01	3.63±.06	0.06±.01	1.53	0.49±.01	2.53±.02

Table 4.4: $S = 4$, deterministic grant duration of one round i.e. $d = c$, optimal arbiter for deterministic grant duration of one round, symmetrical access paths, $p_{-1} = p_1 = .4$, $p_2 = .2$, and $t_{trans} = .98c$ (i.e. t_{trans} deterministic).

In every case listed in Table 4.4, $\bar{t}_w^{\,integration} > \bar{t}_w^{\,simulation}$. (The superscripts denote how the quantities were obtained.) This is not surprising, especially for $\psi = .5$ since the access times have a large deterministic component. We have already seen in Chapter 2 that the Multibus model with a server-sharing queue overestimates $\bar{t}_w$ if the access time distributions are deterministic. Here, the Multibus access time distribution is entirely deterministic and the Ringbus access time has a large deterministic component: the Ringbus access time is at least $\bar{t}_{trans}$, where t_{trans} is deterministic.

In addition, $\bar{t}_p^{MBeqv\ integration} > \bar{t}_p^{MBeqv\ simulation}$ and $g^{integration} < g^{simulation}$ in every case listed. These are obviously related. A larger value of $\bar{t}_p^{MBeqv}$ implies a smaller value of p_0 and hence a smaller value of g. Also, $\bar{t}_p^{MBeqv\ integration}$ is related to $\bar{t}_w^{\,integration}$. If $t_w^{\,integration}$ is larger

than it should be, then $\bar{t}_p^{MBeqv\ integration}$ is likely to be larger than it should be. In addition, it is likely that the probability distribution of $\bar{t}_p^{RBeqv}$ is skewed more towards shorter times than that predicted by our geometric approximation of it. This would cause the Ringbus to be more heavily loaded in actuality - i.e. in the simulation - than predicted by integration; hence the actual throughput of the Ringbus would be greater than predicted by integration. Since the probability distribution of $\bar{t}_p^{RBeqv}$ would be more skewed towards shorter times for larger N, this effect might explain why the difference $g^{simulation} - g^{integration}$ increases with N.

Discussion

The results predicted by integration of the Multibus and Ringbus models agree fairly closely with simulation results of the overall Concert model for the four examples considered. We observed that the integration results were most accurate for light Ringbus loading ($N \ll N^*$, small ψ) and very heavy Ringbus loading ($N \gg N^*$, $\psi \approx 1$). This is in fact a general result for integration, as we discussed earlier, and can be justified analytically. We performed the integration for several other examples with $S=4$ and observed the same general trends as in the four examples reported. We did not perform any integration for $S>4$, for which we expect the same general trends.

The accuracy of the integration results in the knee areas (i.e. for $N \approx N^*$) will depend strongly on the various probability distributions, as we saw with the Multibus models in Chapter 2. A great deal of further work is required to clarify and characterize the accuracy of our integration technique in the knee area.

Certainly, our four examples demonstrate that our integration technique works and that it is a viable approach if accuracy is not paramount. If greater accuracy is desired from the integration, then the interactions between the Multibus and Ringbus models will have to be approximated by more than just first moments. However, this will be difficult, and probably infeasible, in most cases when dealing with analytical models for the Multibus and Ringbus.

4.3 Simulation I: The Ringbus in the Concert Environment

In this section we present and discuss the results of a series of simulations to assess the performance of the Ringbus - with eight slices - in the Concert environment.

We have already discussed our simulation model and its parameters in conjunction with the simulations reported in section 4.2. To recap, our simulation model is the overall Concert model comprised of a Multibus model (one for each Multibus) and a Ringbus model. We assume the Multibus model with long word and Ringbus accesses, discussed in section 2.9, for the Multibus. The Ringbus model depends on the arbitration algorithm and the Ringbus access paths. Once again, our standing assumptions are:

- each Multibus model has exactly the same parameters so all Multibus models are identical in every respect
- all the random variables t_p, t_r, t_{aMB}, and t_{trans} and all the probabilities β, ψ, and p_i^{RB} are mutually independent and stationary
- the Ringbus model is completely symmetric with respect to each Multibus interconnection.

The simulations include the Multibus-Ringbus interaction. In particular, the simulation model faithfully incorporates the fact that a request from a Multibus cannot be latched by the Ringbus arbiter until the grant from the previous request from that Multibus has terminated, as is the case in the actual Concert system.

In these simulations we assume in addition to the previous assumptions that:

1) the processing time is exponentially distributed
2) there are no long word accesses i.e. $\beta = 0$ (hence the recovery time distribution is irrelevant)
3) there are only Ringbus accesses i.e. $\psi = 1$ (hence the Multibus access time distribution is irrelevant)
4) the start up time is zero i.e. $t_{start} = 0$
5) the Ringbus data transfer time t_{trans} is deterministic and hence the duration d for which segments are allocated to a Ringbus request is constant ($d = t_{arb} + \left\lceil \frac{t_{trans}}{c} \right\rceil c$ †) (We have already assumed in section 4.2 that t_{arb} is a deterministic integral multiple of c.)

The restrictions of $\beta = 0$ and $\psi = 1$ may seem restrictive, but we make them because of space and time constraints. To some degree, the effect of ψ can be determined by varying $\bar{t}_p$ with ψ held constant at 1.0. We make the assumption of $\psi = 1$ in particular because we are chiefly interested in the performance of the Ringbus.

† c is the Ringbus arbiter clock period.

The parameters in the simulations are as follows:

1) The number of processors on a Multibus, N. We take $N = 1$, 2, and 4.

2) The mean processing time, $\bar{t}_p$. We take $\bar{t}_p = 5.0c$, $10.0c$, $20.0c$, $50.0c$, and sometimes $100.0c$ and $200.0c$.

3) The Ringbus destination probabilities, p_i^{RB}. We consider three different sets of Ringbus destination probabilities: asymmetrical, symmetrical, and uniform, as listed in Table 4.5.

Distribution	p_1^{RB}	p_2^{RB}	p_3^{RB}	p_4^{RB}	p_{-1}^{RB}	p_{-2}^{RB}	p_{-3}^{RB}
Asymmetrical	.4324	.2162	.1081	.0541	.0270	.0541	.1081
Symmetrical	.2759	.1379	.0690	.0345	.0690	.1379	.2759
Uniform	.1429	.1429	.1429	.1429	.1429	.1429	.1429

Table 4.5: Ringbus destination probabilities

Both the asymmetrical and symmetrical Ringbus destination probability distributions are negative binary exponential distributions where the exponent is the smallest number of segments required to connect the source and destination. That is, for both the asymmetrical and symmetrical distributions, $p_i^{RB} = C\,2^{-seg(i)}$ where $seg(i)$ is the smallest number of segments required to connect the source slice to the destination slice i slices away from the source slice and C is a normalizing constant. (Recall that the sign of i denotes the direction around the Ringbus). For the asymmetrical distribution, $seg(i)$ is computed assuming asymmetrical Ringbus access paths and for the symmetrical distribution, $seg(i)$ is computed assuming symmetrical Ringbus access paths. For example, the minimum number of segments required to connect a slice to its neighbouring slice in the clockwise direction is one for both the asymmetrical and symmetrical access paths. Thus $p_1^{RB(asym)} = C^{asym}2^{-1}$ and $p_1^{RB(sym)} = C^{sym}2^{-1}$. On the other hand, the minimum number of segments required to connect a slice to its neighbouring slice in the counterclockwise direction is three for asymmetrical access paths and one for symmetrical access paths. Thus $p_{-1}^{RB(asym)} = C^{asym}2^{-3}$ and $p_{-1}^{RB(sym)} = C^{sym}2^{-1}$.

The asymmetrical and symmetrical access paths are intended to reflect the different distribution of accesses that would be plausible with the respective asymmetrical and symmetrical access paths if the accesses exhibited locality. The uniform distribution is intended to reflect the distribution of accesses if the accesses exhibited no particular locality.

4) The Ringbus arbiter algorithm. We consider five different arbiter algorithms:

   i) The rotating priority (with counter-clockwise priority rotation) algorithm discussed in Chapter 3 and section 1.2.3. This is the algorithm employed in the actual Concert system.

ii) The greedy algorithm. This algorithm pursues a maximum reward strategy - in every arbiter clock cycle it grants the maximum number of requests that it can. Ties between request sets with the same reward are broken in favour of the request set with the greatest number of the largest requests. Any ties remaining after this point are broken arbitrarily.

iii) The two phase greedy *interval* algorithm. This algorithm is best described by first considering a single phase greedy interval algorithm. Such an algorithm alternates between an idle interval and a grant interval. No nonnull requests are granted during the idle interval. The idle interval terminates when the first nonnull request arrives at the Ringbus arbiter, if there currently are no pending nonnull requests latched by the arbiter, or it terminates $t_{arb} + c$ after the end of the previous grant interval, if there is at least one nonnull Ringbus request ungranted from the previous grant interval. This minimum idle interval of $t_{arb} + c$ corresponds to the minimum time between the termination of a Ringbus grant and the initiation of the next Ringbus grant from the same slice.

As the name implies, nonnull requests are granted only during the grant interval which extends from the termination of the idle interval until the Ringbus access corresponding to each granted request has completed. The actual arbitration - i.e. deciding which request set to grant - is done only at the beginning of a grant interval. The same greedy algorithm discussed in 4(ii) performs the arbitration at this point. All grants remain in effect unchanged until their respective Ringbus accesses terminate.

Thus the duration of a grant interval is determined by the longest access time of those requests granted. This could be a problem if there was a high variability in the Ringbus data transfer time, t_{trans}. However, we assume that t_{trans} is deterministic (see 6)). This, of course, ignores read-modify-write accesses, for which t_{trans} would be much greater than for reads or writes. The arbiter algorithm can be modified to deal with such accesses. One such way is to terminate a grant interval when all non-read-modify-write accesses terminate and allow grants corresponding to read-modify-write accesses to carry on into the next grant interval.

Now a two phase greedy interval algorithm consists of one single phase greedy interval algorithm, which we call the primary phase, and a second single phase greedy interval algorithm, delayed by $t_{arb} + c$ with respect to the primary phase. We call this second phase the secondary phase.

The single phase greedy interval algorithm was motivated by the finding in section 3.4 that in heavy traffic the optimal arbiter algorithm for four slices and deterministic grant durations of d rounds tends to align the requests so that they are granted at intervals

of d rounds. Thus we expected the single phase greedy interval algorithm to yield good performance in heavy traffic. We found, however, that it actually yielded performance that was usually worse than the rotating priority algorithm (with symmetrical access paths). Presumably, this was due to the idle interval of duration $t_{arb} + c$ during which no request are granted. We added the secondary phase in an attempt to improve the utilization of the Ringbus segments and hence improve the throughput.

iv) The crossbar algorithm. With this algorithm the Ringbus is transformed into a crossbar interconnection.

v) The commonbus algorithm. With this algorithm the Ringbus is transformed into a single time-shared common bus.

5) The Ringbus access paths. For the rotating priority arbiter algorithm, we consider both asymmetrical and symmetrical access paths. For the greedy and the greedy interval algorithms we consider only symmetrical access paths. The issue of asymmetrical or symmetrical access paths is irrelevant for the crossbar and commonbus algorithms.

6) The Ringbus data transfer time, t_{trans}. In all cases we take $t_{trans} = 7c$, as a rough approximation of the case in the actual Concert system (when $c = 200$nsec - see section 3.3.2 of Appendix A).

(Note: there is no point to taking $t_{trans} = 6.8c$ here as we would have done in section 4.2. The reason is that no new requests can be latched by the Ringbus arbiter until $\geq t_{arb}$ after the Ringbus access - i.e. data transfer - has terminated [since the grant corresponding to this access continues for t_{arb} past the termination of the access]. Since $t_{arb} \geq c$ here [see below], the minimum interval between the termination of an access and the latching of the next request from the same Multibus is always $\geq c$. For $t_{trans} = 7c$ this interval is t_{arb}, and for $t_{trans} = 6.98c$ this interval is $t_{arb} + .02c$. The difference between the probability of a request arriving from a Multibus in an interval of t_{arb} and an interval of $t_{arb} + .02c$ is negligible.)

7) The Ringbus arbitration delay, t_{arb}. We take $t_{arb} = 2c$ as in the actual Concert system for the rotating priority, greedy, and greedy interval algorithms. (Hence $d = 9c$ for these three algorithms.) For the crossbar and commonbus algorithms, we take $t_{arb} = c$ to reflect the greater simplicity inherent in the arbiter algorithm in these cases. (Hence $d = 8c$ for these two algorithms.)

8) The block size B and the run size R. In all cases we took $B = 100$ and $R = 10$.

The following tables contain the simulation results. The statistics reported are the mean processor cycle time, t_{cycle} (the reciprocal of the throughput of the processor), the Multibus waiting

time per access, $\bar{t}_w$, the Ringbus waiting time per Ringbus access, $\bar{w}_{RB}$, and the mean number of Ringbus grants in progress per arbiter clock period, g. A grant is considered in progress for the total time that at least one Ringbus segment is allocated to the grant. Since segments remain allocated to a grant for a period t_{arb} after the termination of the Ringbus data transfer time, a grant is in progress for a total time of $t_{trans} + t_{arb}$, which equals $9c$ for the rotating priority, greedy, and greedy interval algorithms and $8c$ for the crossbar and common bus interconnections. The ± figures associated with each statistic indicate the corresponding 95% confidence intervals.

Destination Probs: asymmetrical							$N = 1$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	29.7±2.1	28.3±1.4	23.75±.69	24.81±.56	16.13±.37	64.05±.03
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	14.5±2.1	13.1±1.1	8.59±.80	9.64±.48	2.27±.19	50.02±.33
	g	2.42±.17	2.55±.13	3.03±.09	2.90±.07	3.98±.09	.9992±.0005
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	30.7±1.6	29.6±1.1	26.21±.88	28.15±.55	20.53±.94	64.09±.11
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	10.8±1.7	9.57±1.19	6.32±.76	8.28±.27	1.67±.30	45.10±.57
	g	2.35±.12	2.43±.09	2.75±.09	2.56±.05	3.13±.14	.998±.002
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	36.5±1.9	35.1±1.2	33.9±1.6	36.42±.78	29.9±1.3	64.17±.16
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	6.18±.89	5.33±.75	3.93±.76	6.81±.44	1.00±.18	34.83±.58
	g	1.97±.11	2.05±.07	2.12±.10	1.98±.04	2.15±.09	.997±.002
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	61.1±3.3	62.3±4.8	61.4±4.1	64.59±3.4	58.6±2.7	70.9±2.8
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	2.19±.49	1.86±.22	1.51±.35	4.40±.34	.44±.10	11.6±1.7
	g	1.18±.06	1.16±.09	1.17±.08	1.12±.06	1.09±.05†	.90±.04

Table 4.6(a): Ringbus simulation results

† Remember, here g represents the average number of grants in progress per round, by which we mean the average number of grants per round to which one or more segments are allocated, not the average number of grants per round utilizing segments. (Here we consider a grant to be in progress for the total time that at least one Ringbus segment is allocated to the grant. Since segments remain allocated to a grant for a period t_{arb} after the termination of the Ringbus data transfer time, a grant is in progress for a total time of $t_{trans} + t_{arb}$.)

Destination Probs: symmetrical							$N=1$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	t_{cycle}/c	38.5±1.4	29.6±1.4	23.80±.40	25.38±.64	16.35±.10	64.05±.03
	t_w/c	0.0	0.0	0.0	0.0	0.0	0.0
	w_{RB}/c	23.2±1.6	14.5±1.5	8.67±.33	10.22±.58	2.54±.26	50.02±.33
	g	1.87±.07	2.43±.11	3.03±.05	2.84±.07	3.92±.10	.9992±.0005
$\bar{t}_p = 10.0c$	t_{cycle}/c	39.14±.74	30.6±1.2	26.75±.43	28.76±.58	20.63±.64	64.09±.11
	t_w/c	0.0	0.0	0.0	0.0	0.0	0.0
	w_{RB}/c	19.3±1.1	10.55±.95	6.90±.66	8.73±.50	1.81±.26	45.10±.57
	g	1.84±.03	2.36±.10	2.69±.04	2.50±.05	3.11±.10	.998±.002
$\bar{t}_p = 20.0c$	t_{cycle}/c	41.9±2.5	36.35±.70	34.4±2.2	37.2±1.3	30.17±.71	64.17±.16
	t_w/c	0.0	0.0	0.0	0.0	0.0	0.0
	w_{RB}/c	12.4±2.1	6.38±.79	4.20±.72	7.05±.48	1.07±.23	34.83±.58
	g	1.72±.10	1.98±.04	2.10±.13	1.94±.07	2.12±.05	.997±.002
$\bar{t}_p = 50.0c$	t_{cycle}/c	63.7±4.7	62.9±3.9	62.3±3.1	65.0±3.2	59.8±3.5	70.9±2.8
	t_w/c	0.0	0.0	0.0	0.0	0.0	0.0
	w_{RB}/c	4.04±.99	2.19±.47	1.59±.24	4.58±.41	.46±.11	11.6±1.7
	g	1.13±.08	1.15±.07	1.16±.06	1.11±.05	1.07±.06	.90±.04

Table 4.6(b): Ringbus simulation results

Destination Probs: uniform							$N = 1$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	47.9±1.9	42.9±2.1	29.18±.86	29.26±.53	16.69±.39	64.05±.03
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	32.5±2.1	27.6±2.1	13.95±.78	14.01±.54	2.89±.35	50.02±.33
	g	1.50±.06	1.68±.08	2.47±.07	2.46±.04	3.84±.09	.9992±.0005
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	48.3±1.9	42.7±2.1	31.19±.71	31.81±.49	20.77±.62	64.09±.11
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	28.2±2.2	22.7±2.1	11.31±.64	11.89±.70	1.92±.31	45.10±.57
	g	1.49±.06	1.69±.08	2.31±.05	2.26±.04	3.09±.09	.998±.002
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	49.0±1.4	43.8±2.5	37.4±1.6	39.3±1.1	29.5±1.1	64.17±.16
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	19.2±2.2	13.8±3.1	7.31±.63	9.23±.36	1.25±.28	34.83±.58
	g	1.47±.04	1.64±.09	1.92±.08	1.83±.05	2.17±.08	.997±.002
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	65.9±2.6	65.4±5.4	62.6±3.4	65.7±3.8	59.9±3.3	70.9±2.8
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0	0.0	0.0
	w_{RB}/c	6.46±.90	4.55±1.22	2.96±.64	5.29±.47	.51±.18	11.6±1.7
	g	1.09±.04	1.10±.09	1.15±.06	1.10±.06	1.07±.06	.90±.04

Table 4.6(c): Ringbus simulation results

Destination Probs: asymmetrical							$N = 2$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	60.2±2.2	55.8±2.7	44.1±1.4	47.4±1.0	26.79±.45	127.80±.17
	$\bar{t}_w/c$	25.1±1.2	22.9±1.4	17.09±.76	18.70±.58	8.55±.34	58.71±.30
	$\bar{w}_{RB}/c$	18.1±1.1	15.8±1.4	9.96±.69	11.66±.52	3.23±.20	53.84±.02
	g	2.39±.09	2.58±.13	3.26±.10	3.03±.07	4.77±.08	.9998±.0002
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	59.3±2.8	55.9±2.1	44.79±.97	47.46±.48	28.29±.47	127.77±.18
	$\bar{t}_w/c$	20.1±1.5	18.42±.87	13.21±.48	14.17±.38	5.58±.20	53.71±.51
	$\bar{w}_{RB}/c$	17.3±1.4	15.5±1.0	9.80±.52	11.24±.25	2.90±.18	53.80±.04
	g	2.42±.11	2.57±.09	3.21±.07	3.03±.03	4.52±.08	.9996±.0003
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	58.9±2.0	56.0±2.2	47.5±1.0	50.18±.82	34.3±1.1	127.78±.30
	$\bar{t}_w/c$	12.5±1.0	11.2±1.0	7.76±.69	8.70±.34	2.93±.20	44.04±.57
	$\bar{w}_{RB}/c$	14.8±1.1	13.2±1.2	8.32±.36	10.08±.30	2.19±.23	53.40±.13
	g	2.44±.08	2.57±.10	3.03±.07	2.86±.05	3.72±.12	.9994±.0006
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	71.4±2.9	70.2±2.2	66.5±2.2	70.8±1.7	61.7±1.9	128.05±.34
	$\bar{t}_w/c$	3.10±.51	2.89±.59	2.12±.43	3.01±.42	.93±.15	22.1±1.4
	$\bar{w}_{RB}/c$	6.86±1.0	6.04±1.08	4.12±.44	7.26±.26	.99±.17	45.78±1.04
	g	2.01±.08	2.05±.07	2.16±.07	2.03±.05	2.07±.06	.998±.002
$\bar{t}_p = 100.0c$	$\bar{t}_{cycle}/c$	114.9±5.3	114.6±6.2	112.0±5.6	117.4±5.3	110.5±3.3	133.3±1.8
	$\bar{t}_w/c$	.84±.17	.78±.17	.75±.12	1.14±.24	.43±.08	4.16±1.3
	$\bar{w}_{RB}/c$	2.74±.46	2.25±.37	1.82±.28	4.88±.39	.49±.08	19.4±2.8
	g	1.25±.06	1.25±.07	1.28±.06	1.22±.05	1.16±.04	.958±.013

Table 4.6(d): Ringbus simulation results

Destination Probs: symmetrical							$N = 2$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	t_{cycle}/c	77.3±3.9	56.6±1.7	45.26±.67	47.92±.90	27.36±.42	127.80±.17
	t_w/c	33.6±1.8	23.3±.84	17.72±.43	18.97±.61	8.82±.27	58.71±.30
	w_{RB}/c	26.6±1.9	16.26±.83	10.56±.32	11.93±.46	3.51±.25	53.84±.02
	g	1.86±.09	2.54±.07	3.18±.05	3.00±.06	4.67±.08	.9998±.0002
$\bar{t}_p = 10.0c$	t_{cycle}/c	76.4±3.5	56.9±1.1	45.27±.92	48.48±.75	28.82±.54	127.77±.18
	t_w/c	28.3±1.7	18.97±.68	13.32±.60	14.91±.51	5.79±.38	53.71±.51
	w_{RB}/c	26.0±1.8	16.04±.71	10.06±.42	11.79±.35	3.21±.33	53.80±.04
	g	1.88±.09	2.53±.05	3.18±.06	2.96±.04	4.43±.09	.9996±.0003
$\bar{t}_p = 20.0c$	t_{cycle}/c	78.0±3.2	58.7±2.3	48.30±.54	51.51±.57	34.52±.74	127.78±.30
	t_w/c	21.1±1.6	12.4±1.4	8.02±.41	9.23±.56	2.97±.33	44.04±.59
	w_{RB}/c	25.4±1.7	14.8±1.4	8.84±.19	10.74±.40	2.31±.23	53.40±.13
	g	1.84±.07	2.45±.10	2.98±.03	2.79±.03	3.70±.08	.9994±.0006
$\bar{t}_p = 50.0c$	t_{cycle}/c	82.3±1.8	71.2±1.3	67.2±1.7	71.7±1.3	61.4±1.8	128.05±.34
	t_w/c	6.19±.82	3.11±.52	2.29±.36	3.18±.19	.93±.08	22.1±1.4
	w_{RB}/c	14.6±1.0	6.89±.74	4.52±.51	7.76±.44	1.03±.11	45.78±1.04
	g	1.75±.04	2.02±.04	2.14±.05	2.00±.04	2.08±.06	.998±.002
$\bar{t}_p = 100.0c$	t_{cycle}/c	117.0±4.9	114.2±2.6	112.6±3.6	116.4±4.2	110.8±5.8	133.3±1.8
	t_w/c	1.17±.26	.81±.12	.72±.13	1.17±.18	.42±.07	4.16±1.3
	w_{RB}/c	4.70±.68	2.76±.45	1.91±.08	5.08±.33	.52±.10	19.4±2.8
	g	1.23±.05	1.26±.03	1.28±.04	1.23±.04	1.15±.06	.958±.013

Table 4.6(c): Ringbus simulation results

Destination Probs: uniform							$N = 2$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	t_{cycle}/c	96.5±2.8	85.2±2.4	56.1±1.2	57.1±1.0	28.01±.69	127.80±.17
	t_w/c	43.2±1.4	37.5±1.3	23.03±.78	23.48±.54	9.18±.34	58.71±.30
	w_{RB}/c	36.2±1.4	30.6±1.2	15.99±.61	16.46±.47	3.84±.32	53.84±.02
	g	1.49±.04	1.69±.05	2.56±.05	2.52±.05	4.57±.11	.9998±.0002
$\bar{t}_p = 10.0c$	t_{cycle}/c	97.2±3.7	84.8±3.3	56.48±.83	57.47±.85	29.35±.70	127.77±.18
	t_w/c	38.6±1.9	32.4±2.1	18.79±.60	19.12±.92	6.13±.42	53.71±.51
	w_{RB}/c	36.5±1.9	30.2±1.8	15.76±.47	16.32±.44	3.54±.30	53.80±.04
	g	1.48±.06	1.70±.07	2.54±.04	2.50±.03	4.36±.10	.9996±.0003
$\bar{t}_p = 20.0c$	t_{cycle}/c	97.5±2.1	84.6±3.4	57.82±.85	58.61±.63	35.1±1.1	127.78±.30
	t_w/c	29.8±1.4	24.1±1.5	12.37±.70	12.53±.68	3.14±.35	44.04±.59
	w_{RB}/c	35.6±1.3	28.9±1.9	14.11±.46	14.61±.44	2.58±.30	53.40±.13
	g	1.47±.03	1.70±.07	2.49±.04	2.45±.03	3.64±.11	.9994±.0006
$\bar{t}_p = 50.0c$	t_{cycle}/c	97.7±2.5	86.9±2.5	73.0±1.3	74.8±1.2	60.7±2.1	128.05±.34
	t_w/c	11.4±1.7	7.92±.85	3.68±.54	4.13±.47	1.01±.14	22.1±1.4
	w_{RB}/c	25.2±2.4	17.9±2.0	8.15±.57	9.96±.22	1.19±.14	45.78±1.04
	g	1.47±.04	1.65±.05	1.97±.03	1.92±.03	2.11±.07	.998±.002
$\bar{t}_p = 100.0c$	t_{cycle}/c	121.3±5.3	118.1±4.6	114.2±3.3	118.7±5.2	109.7±5.5	133.3±1.8
	t_w/c	1.83±.49	1.38±.24	1.00±.25	1.45±.30	.42±.08	4.16±1.3
	w_{RB}/c	8.0±1.5	5.71±.61	3.57±.23	6.25±.38	.59±.13	19.4±2.8
	g	1.19±.05	1.22±.05	1.26±.04	1.21±.06	1.17±.06	.958±.013

Table 4.6(f): Ringbus simulation results

Destination Probs: asymmetrical							N 4
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	118.4±3.1	112.1±2.3	88.4±1.4	94.1±1.0	53.34±.77	255.11±.30
	$\bar{t}_w/c$	83.5±2.3	78.7±1.8	61.1±1.1	65.24±.80	34.82±.56	185.58±.25
	$\bar{w}_{RB}/c$	17.65±.77	16.09±.61	10.14±.35	11.56±.25	3.36±.19	53.93±.01
	g	2.43±.06	2.56±.05	3.25±.05	3.05±.03	4.78±.07	.9999±.0001
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	117.9±4.1	111.2±3.9	88.0±1.6	93.7±1.5	53.21±.65	254.99±.42
	$\bar{t}_w/c$	78.1±3.1	72.9±3.0	55.7±1.3	60.0±1.1	29.73±.60	180.45±.53
	$\bar{w}_{RB}/c$	17.6±1.0	15.9±1.0	10.04±.39	11.49±.37	3.32±.16	53.92±.01
	g	2.43±.08	2.58±.09	3.26±.06	3.06±.05	4.79±.06	.9998±.0002
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	118.2±4.6	111.1±2.9	88.5±1.7	94.0±1.2	53.78±.86	254.95±.45
	$\bar{t}_w/c$	68.4±3.5	62.8±2.3	46.1±1.5	50.13±.86	20.28±.88	170.27±.69
	$\bar{w}_{RB}/c$	17.6±1.2	15.78±.71	10.11±.43	11.52±.29	3.24±.20	53.90±.02
	g	2.43±.10	2.58±.07	3.24±.06	3.05±.04	4.74±.07	.9998±.0002
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	118.7±3.5	112.2±3.1	92.33±.68	98.0±1.4	67.9±1.2	255.03±.22
	$\bar{t}_w/c$	39.9±2.2	35.3±2.9	21.7±1.1	25.00±.85	6.33±.60	140.3±2.2
	$\bar{w}_{RB}/c$	16.55±.82	14.73±.98	8.90±.31	10.77±.32	2.26±.18	53.83±.05
	g	2.42±.07	2.56±.07	3.11±.02	2.93±.04	3.76±.07	.9994±.0004
$\bar{t}_p = 100.0c$	$\bar{t}_{cycle}/c$	131.7±2.6	127.9±2.3	122.1±2.9	127.5±2.9	112.5±2.5	255.3±.71
	$\bar{t}_w/c$	10.3±2.0	8.51±1.2	5.55±.77	8.06±.94	1.86±.26	91.6±3.6
	$\bar{w}_{RB}/c$	9.39±1.33	7.77±.99	4.94±.37	8.24±.25	1.09±.15	52.56±.39
	g	2.18±.05	2.24±.04	2.35±.06	2.25±.05	2.27±.05	.998±.001
$\bar{t}_p = 200.0c$	$\bar{t}_{cycle}/c$	214.5±7.6	214.8±8.2	214.2±8.7	218.6±4.7	216.9±9.2	259.0±2.7
	$\bar{t}_w/c$	1.70±.21	1.52±.27	1.37±.19	2.32±.30	.72±.07	17.7±3.2
	$\bar{w}_{RB}/c$	2.99±.42	2.60±.45	1.95±.22	5.29±.31	.51±.04	30.1±3.0
	g	1.34±.05	1.34±.05	1.34±.05	1.31±.03	1.21±.05	.985±.010

Table 4.6(g): Ringbus simulation results

Destination Probs: symmetrical							$N = 4$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	154.1±2.9	113.8±3.5	89.8±1.2	95.8±1.1	54.19±.58	255.11±.30
	$\bar{t}_w/c$	110.1±2.2	80.1±2.6	62.02±.84	66.58±.84	35.44±.45	185.58±.25
	$\bar{w}_{RB}/c$	26.63±.71	16.52±.85	10.48±.28	12.00±.28	3.57±.14	53.93±.01
	g	1.86±.03	2.52±.07	3.20±.04	3.00±.04	4.71±.05	.9999±.0001
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	152.9±5.1	114.3±3.5	89.6±1.0	95.9±1.3	54.3±1.2	254.99±.42
	$\bar{t}_w/c$	104.1±3.7	75.3±2.6	56.9±.68	61.6±1.2	30.52±.78	180.45±.53
	$\bar{w}_{RB}/c$	26.3±1.3	16.65±.84	10.43±.26	12.02±.31	3.61±.29	53.92±.01
	g	1.88±.06	2.51±.08	3.20±.04	2.99±.04	4.69±.10	.9998±.0002
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	154.2±4.4	114.9±2.6	89.9±1.3	96.24±.99	54.99±.64	254.95±.45
	$\bar{t}_w/c$	95.1±3.3	65.7±2.1	47.05±.72	51.66±.78	21.06±.81	170.27±.69
	$\bar{w}_{RB}/c$	26.6±1.1	16.77±.64	10.49±.33	12.08±.24	3.54±.18	53.90±.02
	g	1.86±.05	2.50±.05	3.19±.05	2.98±.03	4.64±.06	.9998±.0002
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	154.3±3.4	116.0±4.5	94.0±1.2	99.9±1.3	68.5±1.1	255.03±.22
	$\bar{t}_w/c$	65.3±2.5	37.7±3.5	22.5±1.1	26.8±1.3	6.44±.54	140.3±2.2
	$\bar{w}_{RB}/c$	26.1±.91	15.8±1.3	9.44±.27	11.44±.28	2.44±.16	53.83±.05
	g	1.86±.04	2.47±.09	3.05±.04	2.87±.04	3.73±.06	.9994±.0004
$\bar{t}_p = 100.0c$	$\bar{t}_{cycle}/c$	157.9±2.7	130.2±3.1	123.3±2.9	128.4±2.8	112.6±2.5	255.3±.71
	$\bar{t}_w/c$	25.8±3.4	9.95±1.7	6.13±.72	8.53±.65	1.98±.18	91.6±3.6
	$\bar{w}_{RB}/c$	20.4±1.2	9.30±1.2	5.58±.53	8.67±.19	1.19±.09	52.56±.39
	g	1.82±.03	2.20±.05	2.33±.06	2.23±.05	2.27±.05	.989±.001
$\bar{t}_p = 200.0c$	$\bar{t}_{cycle}/c$	219.2±5.5	215.3±7.8	215.7±9.0	218.6±7.3	211.4±6.3	259.0±2.7
	$\bar{t}_w/c$	2.84±.70	1.62±.30	1.37±.24	2.35±.27	.71±.08	17.7±3.2
	$\bar{w}_{RB}/c$	5.95±.83	3.00±.30	2.11±.20	5.56±.27	.55±.07	30.1±3.0
	g	1.31±.03	1.33±.05	1.33±.06	1.31±.04	1.21±.04	.985±.010

Table 4.6(h): Ringbus simulation results

Destination Probs: uniform							$N=4$
Arbiter Algorithm		Rotating	Rotating	Greedy	Interval	Cross-	Common
Access Paths		Asym.	Sym.	Sym.	Sym.	bar	Bus
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	193.5±4.1	170.3±2.8	111.9±1.5	114.20±.95	55.6±1.1	255.11±.30
	$\bar{t}_w/c$	139.5±3.0	122.3±2.2	78.5±1.1	80.26±.64	36.53±.74	185.58±.25
	$\bar{w}_{RB}/c$	36.5±1.0	30.68±.69	16.02±.39	16.61±.21	3.93±.26	53.93±.01
	g	1.48±.03	1.68±.03	2.57±.04	2.51±.02	4.59±.09	.9999±.0001
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	193.7±3.3	170.0±4.7	112.2±1.1	114.0±1.2	55.7±.99	254.99±.42
	$\bar{t}_w/c$	134.8±2.5	116.9±3.5	73.66±.94	75.05±.91	31.6±1.0	180.45±.53
	$\bar{w}_{RB}/c$	36.57±.82	30.6±1.1	16.10±.24	16.57±.28	3.94±.25	53.92±.01
	g	1.48±.02	1.69±.04	2.56±.02	2.52±.02	4.58±.08	.9998±.0002
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	193.6±2.3	171.4±3.1	112.1±1.5	114.1±1.0	56.28±.80	254.95±.45
	$\bar{t}_w/c$	124.4±2.0	108.0±2.4	63.5±.81	65.2±1.2	22.0±1.0	170.27±.69
	$\bar{w}_{RB}/c$	36.48±.61	31.0±.74	16.06±.35	16.59±.23	3.89±.20	53.90±.02
	g	1.48±.02	1.67±.03	2.56±.03	2.51±.02	4.53±.06	.9998±.0002
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	193.5±4.1	172.1±3.9	113.9±1.4	115.9±1.7	69.3±1.6	255.03±.22
	$\bar{t}_w/c$	94.3±4.4	78.7±3.3	36.6±1.8	37.8±1.8	6.62±.60	140.3±2.2
	$\bar{w}_{RB}/c$	36.3±1.1	30.8±1.1	15.22±.48	15.79±.47	2.58±.24	53.83±.05
	g	1.48±.03	1.67±.04	2.58±.03	2.47±.03	3.68±.09	.9994±.0004
$\bar{t}_p = 100.0c$	$\bar{t}_{cycle}/c$	193.0±4.9	171.2±6.4	132.3±1.8	135.8±2.2	113.2±2.9	255.3±.71
	$\bar{t}_w/c$	48.4±4.0	34.7±4.4	10.94±1.3	12.67±.99	2.03±.13	91.6±3.6
	$\bar{w}_{RB}/c$	32.2±1.6	25.2±1.9	10.02±.42	11.65±.27	1.32±.10	52.56±.39
	g	1.49±.04	1.68±.06	2.17±.03	2.11±.04	2.26±.06	.998±.001
$\bar{t}_p = 200.0c$	$\bar{t}_{cycle}/c$	228.0±5.1	220.8±6.8	216.1±5.8	221.3±5.6	210.6±11.3	259.0±2.7
	$\bar{t}_w/c$	5.15±1.5	3.43±.66	2.05±.39	2.94±.40	.73±.13	17.7±3.2
	$\bar{w}_{RB}/c$	10.7±1.7	7.00±.82	3.93±.47	6.82±.23	.60±.07	30.1±3.0
	g	1.26±.03	1.30±.04	1.33±.04	1.30±.03	1.21±.06	.985±.010

Table 4.6(i): Ringbus simulation results

The results in Tables 4.6(a) through (i) indicate little variation in the performance with different access paths and arbiter algorithms for light loading, as one would expect, and large variation in the performance for heavy loading. These variations in performance for heavy loading are illustrated in the following table of the throughput with $\bar{t}_p = 5.0c$ relative to that with rotating priority and asymmetrical access paths.

Dest. Probs.		Arbiter algorithm (Sym. access paths)			Cross-bar
		Rotating	Interval	Greedy	
$N=1$	Asym.	5%	17%	20%	46%
	Sym.	23	34	38	58
	Uni.	10	39	39	65
$N=2$	Asym.	7	21	27	56
	Sym.	27	38	41	65
	Uni.	12	41	42	71
$N=4$	Asym.	5	21	25	55
	Sym.	26	38	42	65
	Uni.	12	41	42	71

Table 4.7: Percent increase in throughput for $\bar{t}_p = 5.0c$ relative to that for rotating priority arbiter algorithm with asymmetrical access paths

The figures in Table 4.7 indicate that the Ringbus throughput can be increased 20 to 40% in heavy loading relative to the throughput with the rotating priority arbiter algorithm and asymmetrical access paths. In other words, the throughput of the actual Concert system can be improved in heavy loading by at least 20 to 40% by using a better arbiter algorithm and symmetrical access paths. By comparing the improvement in throughput with rotating priority and symmetrical access paths with the improvement in throughput with the interval or greedy algorithm (both of which yield about the same performance) and symmetrical access paths, we can see that the change from asymmetrical to symmetrical access paths accounts for 1/5 to 1/4 of the improvement with asymmetrical destination probabilities, about 1/4 of the improvement with uniform destination probabilities, and over 1/2 of the improvement with symmetrical destination probabilities. Interestingly, the improvement in throughput with the interval and greedy algorithms (with symmetrical access paths) remained about the same for both uniform and symmetrical destination probabilities, indicating that the improvement in throughput contributed by these algorithms also changes with the destination probabilities but in an opposite manner to that contributed by the symmetrical access paths.

For general destination probabilities we cannot draw too many conclusions from Table 4.7 besides that the throughput can be improved by at least 20 to 40% and that both the degree of improvement and the relative contribution of the arbiter algorithm and symmetrical access paths depend strongly on the destination probabilities. Table 4.7 does give some idea though - which of course must be balanced with the costs - of the relative advantage of different arbiter algorithms and access paths.

Another way to characterize and compare the performance of the various arbiter algorithms

is by their saturation throughput i.e. the maximum throughput achievable. This is a particularly useful and convenient way to characterize the performance because the saturation point depends only on the arbiter algorithm and the destination probabilities. Table 4.8 lists the saturation throughput g^{sat} (in mean number of grants in progress per arbiter clock period†) with the various arbiter algorithms and access paths considered in this section.

Algorithm	Access Path	Destination Probs.		
		Asym.	Sym.	Uni.
Commonbus	n/a	1.0	1.0	1.0
Rotating	Asym.	2.4	1.9	1.5
Rotating	Sym.	2.5	2.5	1.7
Interval	Sym.	3.1	3.0	2.5
Greedy	Sym.	3.3	3.2	2.6
Crossbar	n/a	4.8	4.7	4.6

Table 4.8: Saturation throughput for various algorithms and destination probabilities

Table 4.8 shows clearly the relative ordering in terms of throughput in saturation of the various arbiter algorithms and access paths considered. Note that Table 4.8 also shows clearly that the saturation throughput decreases as the destination probabilities change from asymmetrical to symmetrical to uniform.

In all the simulations the greedy arbiter algorithm yielded better performance - although not by much - than the two phase interval algorithm. This was a slightly surprising result considering that, extrapolating from our finding with four slices and $p_0 = 0$ in section 3.4, one would expect an interval algorithm to be optimal for heavy traffic. On closer examination this result is not so surprising. Presumably, the result is a consequence of the nonzero arbitration time. As already mentioned, the single phase interval algorithm yielded poor performance due to the idle interval during which no requests are granted. The two phase interval algorithm is a simple attempt to utilize the Ringbus during the idle period, but it has the consequence of causing additional request conflicts because one phase follows the other by less than the duration of the grants. Ideally one would like the phases to be nonoverlapping but this has the drawback of imposing a minimum wait of one phase (the duration of a grant) until the next request can be granted at a slice after the previous grant at that slice terminates. Thus there seems to be no way to avoid some sort of performance penalty due to the nonzero arbitration time when implementing an interval-type

† As before, a grant is considered to be in progress for the total time that at least one Ringbus segment is allocated to the grant.

algorithm. This suggests that it is important to include the effect of nonzero arbitration time when trying to determine an optimum arbiter algorithm for the actual Concert system.

The interval algorithm suffers from another disadvantage: in light traffic the synchronization of the requests with the phases adds to the total waiting time of a request. In some cases with light loading, the throughput with the two phase interval algorithm is actually less than that of the rotating priority algorithm with symmetrical access paths. This suggests the obvious: for best performance the arbiter should be able to change algorithms to adapt to changing load conditions.

The overall throughput of the Concert system is $\frac{NS}{\bar{t}_{cycle}}$ where $\bar{t}_{cycle}$ is the mean cycle time of a single processor. (Recall that $\bar{t}_{cycle} = \bar{t}_p + \beta\bar{t}_r + \bar{t}_{w_T} + (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB})$.) As a function of $\bar{t}_p$, the overall throughput is maximum at $\bar{t}_p = 0$, monotonically decreases as $\bar{t}_p$ decreases, and is asymptotic to a curve in the family $\frac{1}{\bar{t}_p}$. Because of the nonlinear asymptote of the overall throughput it is more convenient to deal with the mean cycle time, for which an equivalent statement is: As a function of $\bar{t}_p$, $\bar{t}_{cycle}$ is minimum at $\bar{t}_p = 0$, monotonically increases as $\bar{t}_p$ increases, and is asymptotic to $\bar{t}_{cycle} = \bar{t}_p + \beta\bar{t}_r + (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}^{(norm)})$. This leads to the following simple first order approximation of the overall throughput as a function of $\bar{t}_p$:

$$\bar{t}_{cycle} = \begin{cases} \bar{t}_{cycle}^{\min} & \text{for } \bar{t}_p + \beta\bar{t}_r + (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}^{(norm)}) < \bar{t}_{cycle}^{\min} \\ \bar{t}_p + \beta\bar{t}_r + (1+\beta)((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}^{(norm)}) & \text{otherwise} \end{cases} \tag{4.1}$$

$\bar{t}_{cycle}^{\min}$ is the value of $\bar{t}_{cycle}$ when $\bar{t}_p = 0$ (and all other parameters fixed). Equation 4.1 is a convenient approximation since it depends only on one parameter, $\bar{t}_{cycle}^{\min}$, aside from the fixed input parameters. Furthermore, $\bar{t}_{cycle}^{\min}$ can be related to the Ringbus throughput when $\bar{t}_p = 0$ - which we denote by $g^{\bar{t}_p = 0}$ - as follows.

First, when $\bar{t}_p = 0$ a request from a processor must wait for the requests of each of the other $N-1$ processors to complete before it must proceed. Hence

$$\bar{t}_{cycle}^{\min} = \beta\bar{t}_r + N(1+\beta)((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB})$$

Second, recall that $\bar{t}_{aRB} = \bar{w}_{RB} + \bar{d}$, and thus

$$\frac{g^{\bar{t}_p = 0}}{\bar{d}} = \frac{S}{\frac{cp_0}{1-p_0} + \bar{w}_{RB} + \bar{d}} = \frac{S}{\frac{cp_0}{1-p_0} + \bar{t}_{aRB}}$$

(Note that $\bar{t}_{aRB}$ is not the same as $\bar{t}_{aRB}^{(norm)}$ since $\bar{t}_{aRB}$ is a function of the Ringbus loading.)

Third, the mean spacing between the termination of one Ringbus request and the arrival of

the next Ringbus request at the same slice, $\frac{cp_0}{1-p_0}$ is equal to $\frac{(1-\psi)}{\psi}\bar{t}_{aMB}$. Therefore $\frac{g^{\bar{t}_p=0}}{\bar{d}} = \frac{S}{\frac{(1-\psi)}{\psi}\bar{t}_{aMB}+\bar{t}_{aRB}} = \frac{S\psi}{(1-\psi)\bar{t}_{aMB}+\psi\bar{t}_{aRB}}$ from which it follows that

$$\bar{t}_{cycle}^{\min} = \beta\bar{t}_r + \frac{(1+\beta)\psi NS\bar{d}}{g^{\bar{t}_p=0}} \tag{4.2}$$

(provided that $g^{\bar{t}_p=0} \neq 0$).

If the Ringbus throughput is saturated when $\bar{t}_p = 0$ (note that it need not be saturated for small enough ψ), then $g^{\bar{t}_p=0} = g^{sat}$ and

$$\bar{t}_{cycle}^{\min} = \beta\bar{t}_r + \frac{(1+\beta)\psi NS\bar{d}}{g^{sat}} \tag{4.3}$$

Note that while $g^{\bar{t}_p=0}$ may depend on β and ψ, g^{sat} does not. Hence equation 4.3 allows the determination of $\bar{t}_{cycle}^{\min}$ as a function of β, ψ, and N provided that the Ringbus remains saturated for $\bar{t}_p = 0$.

Note that equations 4.1 and 4.2 also allow the results obtained in this section for $\beta = 0$ and $\psi = 1$ to be extrapolated for other values of β and ψ.

4.4 Simulation II: The Actual Concert System

In this section we present the results of a series of simulations of the actual Concert system, as implemented, in order to give some idea of the performance of this system and how it varies under the influence of various parameters. The simulation model and the manner in which the simulations were performed is the same as for the simulations in section 4.2 and 4.3. All the assumptions and parameters are the same as those listed in section 4.3 except for the following:

Ringbus arbitration algorithm: rotating priority (counterclockwise priority rotation) as in Concert

Access paths: asymmetrical as in Concert

Ringbus destination probabilities: asymmetrical (as listed in Table 4.5). We take these probabilities as asymmetrical to show the Ringbus (with asymmetrical access paths) in its best light and to correspond to the expected asymmetrical bias in the request probabilities. We expect that most applications will be structured to take advantage of the more favourable clockwise direction for accesses, implying an asymmetrical bias in the request probabilities.

Ringbus access probability: We take $\psi = .2$, .4 , .6, and .8 to illustrate a range of operating conditions. Note that the performance with $\psi = 0$ (no Ringbus accesses) is given by the isolated Ringbus model of section 2.9 and the performance with $\psi = 1$ (only Ringbus accesses) is given by the results in section 4.3.

Arbiter clock period: $c = 200$nsec.

Multibus access time distribution: We assume a deterministic access time with duration 1.10μsec $= 5.5c$. (We arrived at this duration by assuming that all the Multibus accesses of a slice are directed towards the slice global memory and that the Ringbus port of this global memory is lightly loaded. In the actual Concert system, the mean Multibus access time of slice global memory is about 1.10μsec when the Ringbus port is heavily loaded and about 1.05μsec when the Ringbus is lightly loaded. (See section 3.3 of Appendix A.) Thus our assumed 1.10μsec duration is slightly pessimistic for most cases.)

As before, we assume the start up time is zero i.e. $\bar{t}_{start} = 0$, there are no long word accesses i.e. $\beta = 0$, the Ringbus data transfer time is deterministic with duration $d = 7c$, and the Ringbus arbitration time is deterministic with duration $t_{arb} = 2c$.

The simulation results are listed in Tables 4.9(a), (b), and (c).

$N=1$					
ψ		.2	.4	.6	.8
$\bar{t}_p = 5.0c$	$\bar{t}_{cycle}/c$	11.88±.47	14.52±.64	19.35±.98	24.3±1.6
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	2.50±1.3	5.73±1.26	9.84±1.40	12.4±1.8
	g	1.22±.15	1.94±.09	2.28±.12	2.39±.12
$\bar{t}_p = 10.0c$	$\bar{t}_{cycle}/c$	17.0±1.4	18.82±.88	22.06±.61	26.2±1.2
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	1.49±.88	3.71±1.03	6.26±.94	8.55±1.0
	g	.83±.17	1.52±.14	1.98±.11	2.22±.09
$\bar{t}_p = 20.0c$	$\bar{t}_{cycle}/c$	26.5±1.2	27.83±.89	30.1±1.4	32.9±1.0
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	.84±.56	1.80±.63	3.03±.79	4.56±1.4
	g	.56±.06	1.03±.09	1.44±.07	1.76±.07
$\bar{t}_p = 50.0c$	$\bar{t}_{cycle}/c$	56.3±4.6	58.8±3.2	59.2±4.0	61.4±3.1
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	.30±.20	.73±.48	1.12±.22	1.62±.24
	g	.24±.03	.48±.05	.74±.06	.94±.05
$\bar{t}_p = 100.0c$	$\bar{t}_{cycle}/c$	105.5±10.7	109.5±8.3	111.0±7.3	110.1±8.0
	$\bar{t}_w/c$	0.0	0.0	0.0	0.0
	$\bar{w}_{RB}/c$	.16±.15	.28±.10	.52±.22	.79±.22
	g	.14±.02	.26±.04	.39±.04	.53±.05

Table 4.9(a): Concert simulation results

$N = 2$					
ψ		.2	.4	.6	.8
$\bar{l}_p = 5.0c$	$\bar{l}_{cycle}/c$	16.33±.97	24.6±2.2	35.3±1.8	48.2±4.1
	$\bar{l}_w/c$	3.78±.40	7.74±1.13	12.98±.90	19.2±2.1
	$\bar{w}_{RB}/c$	4.66±1.16	10.4±2.4	14.0±1.6	16.8±1.8
	g	1.79±.09	2.32±.16	2.45±.16	2.39±.14
$\bar{l}_p = 10.0c$	$\bar{l}_{cycle}/c$	19.69±.54	25.8±1.4	36.2±1.7	47.8±3.8
	$\bar{l}_w/c$	2.27±.22	4.94±.68	9.65±.67	14.9±1.8
	$\bar{w}_{RB}/c$	3.50±.92	8.09±1.56	12.9±1.1	15.6±2.4
	g	1.47±.11	2.20±.10	2.37±.11	2.42±.20
$\bar{l}_p = 20.0c$	$\bar{l}_{cycle}/c$	27.99±.78	32.2±1.2	39.1±2.0	49.1±1.4
	$\bar{l}_w/c$	1.18±.09	2.34±.26	4.85±1.13	8.78±.69
	$\bar{w}_{RB}/c$	1.88±.44	4.95±1.19	8.80±1.48	12.7±1.0
	g	1.03±.07	1.77±.06	2.19±.08	2.35±.07
$\bar{l}_p = 50.0c$	$\bar{l}_{cycle}/c$	57.9±2.5	58.5±2.8	61.5±3.0	65.8±2.1
	$\bar{l}_w/c$	.42±.05	.69±.09	1.15±.16	1.97±.44
	$\bar{w}_{RB}/c$	.68±.27	1.60±.27	3.11±.66	4.93±.97
	g	.50±.07	.98±.06	1.40±.07	1.74±.05
$\bar{l}_p = 100.0c$	$\bar{l}_{cycle}/c$	107.3±5.8	108.2±5.2	109.6±5.7	111.6±6.1
	$\bar{l}_w/c$	.22±.05	.31±.07	.44±.10	.61±.10
	$\bar{w}_{RB}/c$	.33±.17	.84±.31	1.29±.17	1.96±.21
	g	.28±.03	.53±.04	.79±.06	1.03±.05
$\bar{l}_p = 200.0c$	$\bar{l}_{cycle}/c$	210.9±9.8	208.1±10.2	210.4±16.0	210.2±4.5
	$\bar{l}_w/c$	.10±.03	.15±.07	.20±.04	.23±.06
	$\bar{w}_{RB}/c$	.20±.07	.39±.14	.63±.22	.86±.13
	g	.14±.01	.28±.02	.41±.04	.55±.01

Table 4.9(b): Concert simulation results

$N=4$					
ψ		.2	.4	.6	.8
$\bar{t}_p=5.0c$	$\bar{t}_{cycle}/c$	30.50±.94	49.4±2.9	73.1±2.5	95.6±1.8
	$\bar{t}_w/c$	17.67±.63	31.8±2.2	49.5±1.9	66.3±1.2
	$\bar{w}_{RB}/c$	5.65±.90	11.75±.95	15.49±.95	16.94±.58
	g	1.88±.06	2.34±.07	2.38±.08	2.40±.05
$\bar{t}_p=10.0c$	$\bar{t}_{cycle}/c$	30.7±1.2	49.2±3.1	71.7±4.5	94.7±3.3
	$\bar{t}_w/c$	12.92±.75	26.7±2.4	43.5±3.5	60.6±2.4
	$\bar{w}_{RB}/c$	5.26±.99	11.5±1.5	15.0±1.3	16.64±.93
	g	1.89±.05	2.35±.09	2.40±.08	2.43±.08
$\bar{t}_p=20.0c$	$\bar{t}_{cycle}/c$	34.3±1.1	49.7±2.4	71.5±3.9	95.3±3.4
	$\bar{t}_w/c$	6.76±.74	17.5±1.6	33.5±3.0	51.3±3.1
	$\bar{w}_{RB}/c$	4.16±.85	10.3±1.0	14.5±1.2	16.73±.93
	g	1.68±.07	2.32±.07	2.41±.08	2.41±.07
$\bar{t}_p=50.0c$	$\bar{t}_{cycle}/c$	59.3±1.9	64.5±2.4	77.3±2.1	97.1±3.7
	$\bar{t}_w/c$	1.94±.24	4.73±.41	12.3±1.0	25.2±2.9
	$\bar{w}_{RB}/c$	1.85±.40	5.00±.64	9.86±.97	14.2±1.2
	g	.97±.08	1.79±.06	2.23±.05	2.36±.06
$\bar{t}_p=100.0c$	$\bar{t}_{cycle}/c$	107.7±5.0	110.9±2.5	112.6±3.5	119.9±4.4
	$\bar{t}_w/c$	.80±.11	1.34±.15	2.59±.32	5.32±.98
	$\bar{w}_{RB}/c$	.80±.16	1.90±.37	3.75±.37	6.33±.79
	g	.54±.03	1.03±.03	1.53±.05	1.91±.06
$\bar{t}_p=200.0c$	$\bar{t}_{cycle}/c$	207.6±6.3	208.1±5.5	212.0±9.2	212.0±9.4
	$\bar{t}_w/c$	.34±.07	.52±.08	.78±.13	1.08±.14
	$\bar{w}_{RB}/c$	.36±.14	.84±.21	1.43±.26	2.03±.20
	g	.27±.03	.55±.03	.82±.04	1.08±.05
$\bar{t}_p=500.0c$	$\bar{t}_{cycle}/c$	507.7±13.5	511.9±18.5	509.9±15.6	511.7±26.2
	$\bar{t}_w/c$	.13±.04	.18±.04	.22±.03	.28±.04
	$\bar{w}_{RB}/c$	.15±.06	.28±.04	.47±.06	.64±.08
	g	.11±.01	.22±.01	.34±.01	.45±.02

Table 4.9(c): Concert simulation results

Examining Tables 4.9(a), (b), and (c) we can see that the mean total waiting time (or wasted time) per cycle - given by $\bar{t}_w + \psi \bar{t}_{aMB}$ - can be quite large. This waiting time is largest, naturally, for a given set of parameters when the overall throughput, and the Ringbus throughput in particular, is saturated. We can derive a necessary condition for the saturation of the Ringbus throughput as follows.

First, the overall throughput must be above the "knee point". Referring to the approximation in equation 4.1, the overall throughput is above the knee point when

$$\bar{t}_p + ((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}^{(norm)}) < \bar{t}_{cycle}^{\min} = \frac{\psi NS\bar{d}}{g^{t_p=0}} \tag{4.4}$$

(Recall that $\bar{t}_{cycle} = \bar{t}_p + \bar{t}_{w_T} + ((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB})$ and $\bar{t}_{aRB} \geq \bar{t}_{aRB}^{(norm)}$.) Second, $g^{t_p=0}$ must equal g^{sat}. Therefore a necessary condition for the saturation of the Ringbus is

$$\bar{t}_p + ((1-\psi)\bar{t}_{aMB} + \psi\bar{t}_{aRB}^{(norm)}) < \frac{\psi NS\bar{d}}{g^{sat}} \tag{4.5}$$

or on rearranging

$$\bar{t}_p + \bar{t}_{aMB} < \psi(\frac{NS\bar{d}}{g^{sat}} + \bar{t}_{aMB} - \bar{t}_{aRB}^{(norm)}) \tag{4.6}$$

In Table 4.10 we list this inequality for various values of N and destination probabilities in the actual Concert system (i.e. 8 slices, rotating priority algorithm, and asymmetrical access paths) with $\beta = 0$. Note that $\bar{d} = 9c$, $\bar{t}_{aMB} = 5.5c$, and $\bar{t}_{aRB}^{(norm)} = 10.5c$ (from Appendix A) independent of the destination probabilities.

Destination Probs.	N	g^{sat}	Necessary condition for saturation
Asymmetrical	1	~2.4	$\frac{\bar{t}_p}{c} + 5.5 < 25\psi$
	2	~2.4	$\frac{\bar{t}_p}{c} + 5.5 < 55\psi$
	4	~2.4	$\frac{\bar{t}_p}{c} + 5.5 < 115\psi$
Symmetrical	1	~1.9	$\frac{\bar{t}_p}{c} + 5.5 < 33\psi$
	2	~1.9	$\frac{\bar{t}_p}{c} + 5.5 < 71\psi$
	4	~1.9	$\frac{\bar{t}_p}{c} + 5.5 < 146\psi$
Uniform	1	~1.5	$\frac{\bar{t}_p}{c} + 5.5 < 43\psi$
	2	~1.5	$\frac{\bar{t}_p}{c} + 5.5 < 91\psi$
	4	~1.5	$\frac{\bar{t}_p}{c} + 5.5 < 187\psi$

Table 4.10: Necessary conditions for Ringbus saturation in the actual Concert system

Operation in the saturated region of throughput is undesirable because of the associated large waiting times. Inequality 4.6 provides a means to adjust parameters to possibly avoid operation in this region.

Chapter 5

Conclusions

Since conclusions specific to the Multibus and Ringbus have already been presented, this chapter covers the general conclusions that can be drawn from the research in this tome.

We can now address the three questions raised in the Introduction.

What is the performance of the Concert Multiprocessor?

We can still not answer this question directly because the performance depends on the models employed (which may be dictated by the application programs) and the model parameters (which certainly depend on the application programs). However, we have developed techniques to determine the performance. Assuming the simple processor model presented in Chapter 1, we have shown how to determine analytically the performance, using throughput as the metric, for any Concert-like system. This analytical approach involves decomposing the overall system into Multibus and Ringbus subsystems, which may be modeled in isolation using the models formulated in Chapter 2 and 3, and then integrating these models, using the procedure in section 4.22, to determine the throughput. The integration procedure is in fact an approximation based on matching the first moments of the interactions between the Multibus and Ringbus models. More accurate results than this procedure yields can be obtained via simulation. Simulation is also the preferred method to include features which are difficult or cumbersome to handle in the analytical models and to allow sizes - such as eight slices - that are too complex for the analytical approach.

The performance of the actual Concert system with eight slices has been established for some different parameter sets by the simulation results presented in section 4.4.

Why is the performance as it is? What factors influence the performance?

The performance of Concert, as modeled in this thesis, depends critically on the parameters

of the simple processor model. The performance is especially sensitive to the mean processing time, $\bar{t}_p$, and the probability of a Ringbus access, ψ.

The effect of different Ringbus architectures and different Ringbus arbiter algorithms on the performance is small except when the Ringbus is heavily loaded, in which case these factors can be significant.

How can the performance be improved?

There are two orthogonal ways in which the performance can be improved:

1) change the physical structure, or

2) change the input parameters i.e. change the characteristics of the application programs.

The more obvious changes in physical structure have already been discussed in the conclusions of Chapter 2 and 3. An important part of the work in this thesis has been establishing the ultimate performance that can be attained with Ringbus-like schemes.

The desirable changes in the input parameters are again rather obvious: localize the processing as much as possible. However, the work in this thesis enables the quantification of the performance improvement resulting from any change in the input parameters. Such quantification is important: it serves as a directional derivative in the performance-action space.

One activity is still required to complete the first cycle in the iterative process of performance modeling: a comparison of the predicted performance, based on the simple processor with parameters obtained from actual programs, with the actual performance obtained with the same programs. The purpose of such a comparison is to establish where the processor model and other models need the most improvement and perhaps how to improve them. Certainly, the processor model needs to be more specific and more oriented to the application program. As discussed in Chapter 2, higher level models should be considered in future cycles of the modeling effort.

Appendix A

Measurement Details

This appendix describes how actual measurements of processing, recovery, access, and waiting times were obtained. These terms are defined in section 2 (as well as in the main text) for convenience. Measured access times under different conditions are given in section 4.

1. Background

Three types of Multibus and Ringbus accesses may occur: byte (8 bits), word (16 bits), and long word (32 bits). A word access consists of two simultaneous byte accesses (a high byte and a low byte). Consequently, byte and word accesses are indistinguishable to an observer of the Multibus or Ringbus unless the observer examines the *BHEN** (byte high enable) signal on the Multibus (see the Multibus 796 specification [P4] for details) or the *BYTE/$\overline{WORD}$* signal on the Ringbus (see Anderson [A2] for details). In particular, a byte and a word have the same access time distribution.† A long word access consists of two consecutive word accesses (since the Multibus and Ringbus are 16 bits wide).

Timing diagrams for the three types of accesses are given in Figures A.1 and A.2. The diagrams depict the essential features of the Multibus operation from the point of view of the processor originating the access. The relative duration and timing of the signals shown is only approximate. *BREQ** and *BPRN** refer to the Multibus request and grant signals for the originating processor; *MRDC** and *MWTC** refer to the Multibus read and write signals respectively; and *XACK** refers to the Multibus acknowledge signal.

† We assume that the time difference between decoding a byte and a word access is negligible. Measurements made on the Multibus and Ringbus with only a single port of the dual port memory loaded support this assumption (see section 4). We see no reason for our findings in this matter to change when both memory ports are loaded.

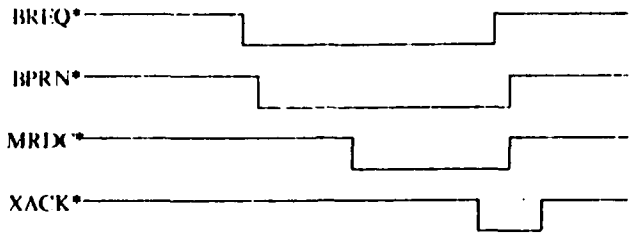


Figure A.1(a): Byte and word access - read cycle

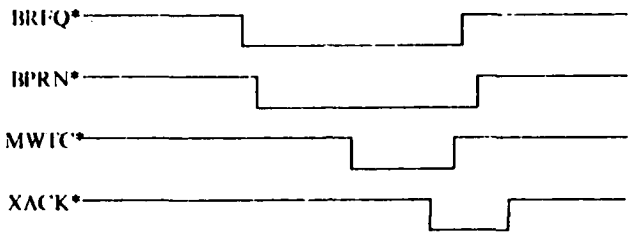


Figure A.1(b): Byte and word access - write cycle

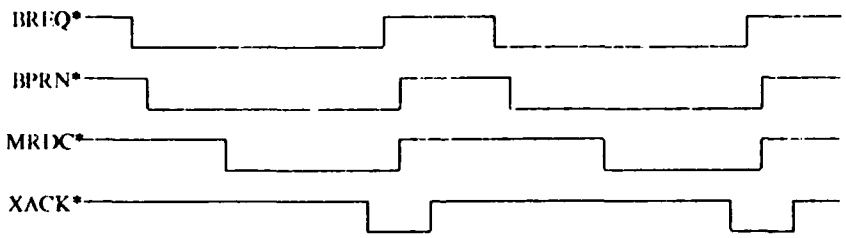


Figure A.2(a): Long word access - read cycle

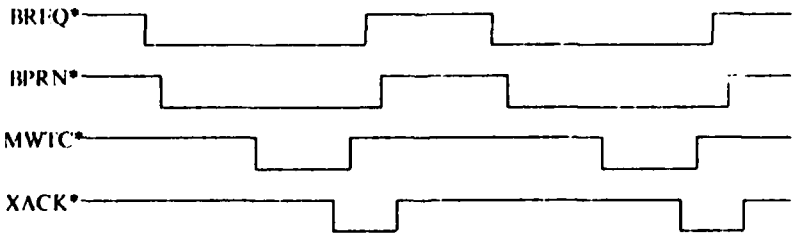


Figure A.2(b): Long word access - write cycle

Note from Figures A.2(a) and A.2(b) that control of the Multibus (and hence of the Ringbus) is relinquished between the successive word access of a long word access.

2. Definitions

All the following quantities are defined with respect to the rising edges of $BCLK*$ (the Multibus clock signal). Let the Multibus request and grant signals for processor i be denoted by $BREQ_i*$ and $BPRN_i*$ respectively.

The processing time, denoted by t_p, for some processor i is the interval between the first rising edge of $BCLK*$ after $BREQ_i*$ goes high at the end of an access to the first rising edge of $BCLK*$ after $BREQ_i*$ next goes low. In the case of a long word access, the end of the second word access is interpreted as the end of the long word access. Thus the interval between the two successive word accesses of a long word is not called a processing time.

The access time, denoted by t_a, for the access of some processor i is the interval between the first rising edge before $BPRN_i*$ goes low to the first rising edge of $BCLK*$ after $BREQ_i*$ goes high.

The recovery time, denoted by t_r, for the long word access of some processor i is the interval between the first rising edge of $BCLK*$ after $BREQ_i*$ goes high at the end of the first word access to the first rising edge of $BCLK*$ after $BREQ_i*$ goes low for the second word of the long word access.

The waiting time, denoted by t_w, for any request for use of the Multibus by processor i is the interval between the first rising edge of $BCLK*$ after $BREQ_i*$ goes low to the first rising edge of $BCLK*$ before $BPRN_i*$ goes low. In the absence of any other traffic on the Multibus, there is always exactly one rising edge of $BCLK*$ after $BREQ_i*$ goes low and before $BPRN_i*$ goes low, yielding $t_w = 0$.

The above definitions were chosen so as to meet the following two constraints: 1) the access time must include the total time that Multibus resources are allocated to a particular processor, and 2) the waiting time must be zero for a single processor on a Multibus. The time that Multibus resources are allocated to a processor is determined by the Multibus arbiter which is a small finite state machine clocked on the rising edge of $BCLK*$. We chose to regard the allocation of Multibus resources to be decided on the rising edge of $BCLK*$. This view is not unique: we could have just as well chosen the Multibus resources to be allocated on the edges of $BPRN_i*$. However, our choice has three advantages: 1) the allocation instants are easily demarcated by $BCLK*$, 2) the waiting time can be defined so that it is easily demarcated by $BCLK*$ (so it is easy to measure) and it is zero for a single processor, and 3) our hardware monitor (the DSD, see the next section) also samples all signals on the rising edge of $BCLK*$. Note that our definition of

access time includes the delay of the Multibus arbiter. This must be the case if we are to meet our first constraint since any delay effectively increases the duration of any allocation of resources.

The previous definitions are depicted in Figure A.3.

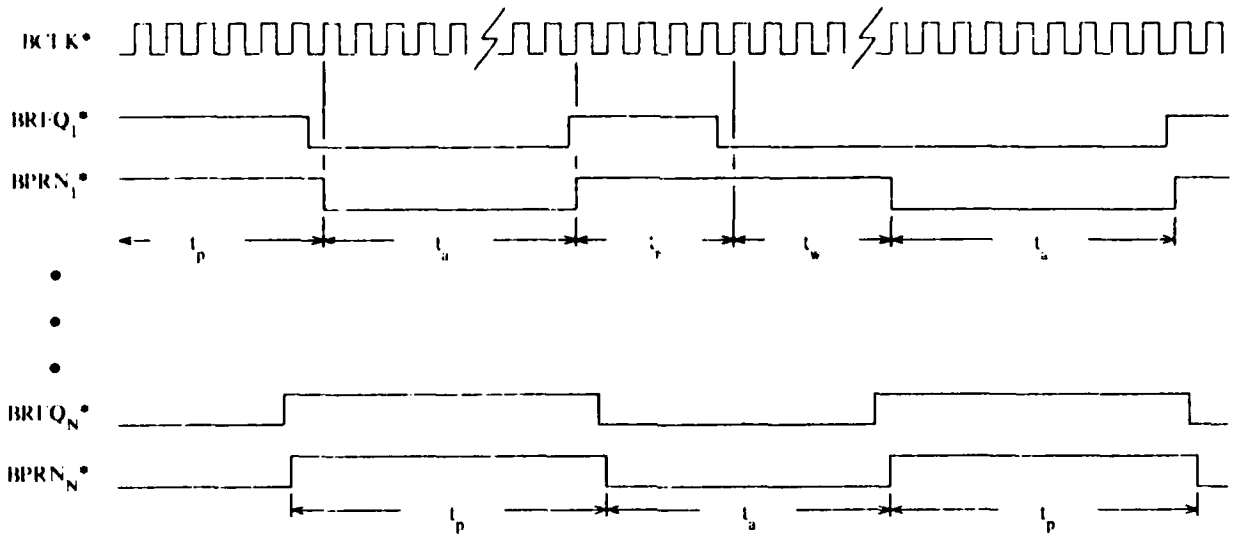


Figure A.3: *Illustration of definitions*

The measurements presented in section 4 indicate that there is little difference in access times for reads and writes; thus the *MRDC** and *MWTC** lines are omitted from Figure A.3.

3. Time Measurements

All the measurements reported in this section were taken with a digital logic analyzer with 10 nsec clock resolution† according to the definitions given in section 2. The measurements were performed on three slices of the Concert system connected by the Ringbus with a Ringbus arbiter clock (*LCLK*) period of 200 nsec. All the processor and memory boards were Microbar DBC68K and DBR50 models respectively with all options set as listed in Appendix C. All the measurements were repeated for several different processor-memory pairs on different slices. No noticeable differences in the measurements for the different repetitions were observed, thus we present the following measurements as if only one set of measurements were taken for each case.

† A Gould Biomation K100-D Digital Logic Analyzer

3.1 Minimum Processing Time

Executing the assembly language program

```
loop: bra loop
```

(corresponding to the single instruction word 60fe) from a non-local memory gives the smallest possible processing time. The processing time in this case is the time it takes to decode the instruction word 60fe and initiate the fetch for the next instruction. The minimum observed processing time in this case was 600 nsec; the processing times varied almost uniformly from 600 nsec to 900 nsec (in 100 nsec steps since the time is measured with respect to *BCLK** rising edges).

To determine the smallest possible processing time for a program executing out of local memory we ran the following assembly language program:

```
loop: movb a4@, a5@
        .
        .
      movb a4@, a5@
        .
        .
      bra loop
```

The movb a4@, a5@ instruction reads the byte at the address stored in address register a4 and writes the byte at the address stored in address register a5. We stored the loop containing the movb instructions in a processor's local HSB memory, installed non-local addresses in address registers a4 and a5, and measured the minimum processing time of the movb instruction. There are actually two different processing times associated with the movb a4@,a5@ instruction: the interval between completion of the byte read and initiation of the byte write within one movb a4@, a5@ instruction and also the interval between the completion of the byte write of one movb a5@, a4@ instruction and the initiation of the byte read of the following movb a4@, a5@ instruction. The intra-instruction processing time (i.e. the former of the two processing times just mentioned) was 600 nsec about half the time and 700 nsec the other half. The inter-instruction processing time (i.e. the latter of the two processing times) varied from 1.20 to 1.40 μsec.

We also considered the minimum processing time of a program executing out of non-local memory subject to the restriction of one non-local memory access per instruction. To determine this minimum, we ran the following assembly language program:

```
loop: movb d7, a5@
        .
        .
      movb d7, a5@
        .
```

bra loop

The movb d7, a5@ instruction writes the byte in data register d7 to the address contained in address register a5. We stored the loop containing the movb instructions in a processor's local HSB memory, installed a non-local address in address register a5, and measured the processing time of the movb instruction. This processing time consists of the time to fetch the single word movb instruction, decode it, and initiate the byte write on the Multibus. The processing time of the movb d7, a5@ instruction was consistently 1.50 μsec. We also tried the movb a5@, d7 instruction corresponding to a byte read, and also measured 1.50 μsec.

3.2 Recovery Time

The distribution of recovery time between the successive word accesses of a long word access was the same for reads and writes: approximately half of the time the recovery time was 600 nsec and the other half of the time it was 700 nsec, yielding a mean of 650 nsec.

3.3 Access Time

Since all the memory boards are dual ported we have to consider the effect of traffic on one port of a memory board on the access time via the other port. In all cases we found no difference in the access time distributions for bytes and words and in the access time distributions for the two words of a long word access.

3.3.1 Multibus Access Time

3.3.1.1 Multibus Access Time with Other Memory Port Unloaded

In this case the access time distribution was approximately the same for reads and writes, with a minimum access time of 1.00 μsec and a maximum of 1.30 μsec. The actual observed distributions are given in Figure A.4 below.

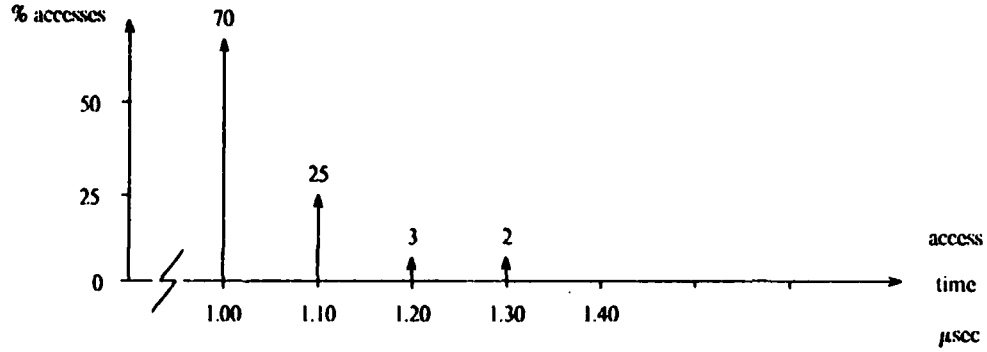


Figure A.4(a): Multibus read access time - other port unloaded

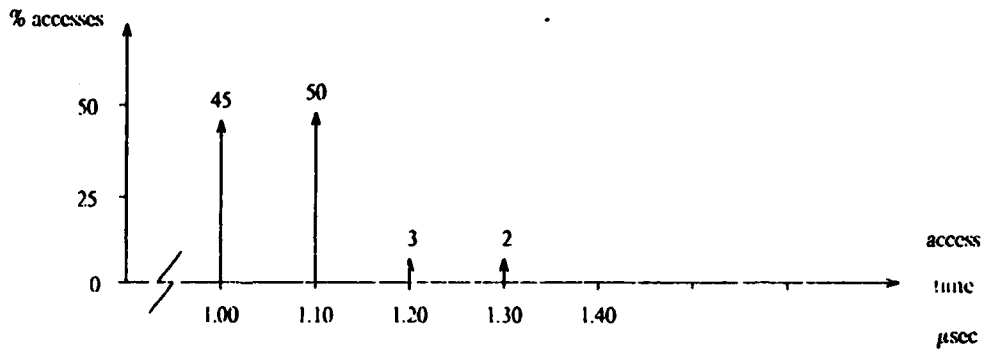


Figure A.4(b): Multibus write access time - other port unloaded

3.3.1.2 Multibus Access Time with Other Memory Port Loaded

We considered two situations: 1) accessing the local memory of another processor via the Multibus while that processor is loading the HSB port of the memory, and 2) accessing the global memory of a slice via the Multibus while other processors access it via the Ringbus.

1) Accessing the local memory of another processor:

We loaded the HSB port of the local memory by having the associated processor execute

```
loop: bra loop
```

out of the local memory. We observed no noticeable difference between the access time distribution for reads and writes via the Multibus. As indicated in Figure A.5, the access times varied from 1.00 μsec to 1.80 μsec.

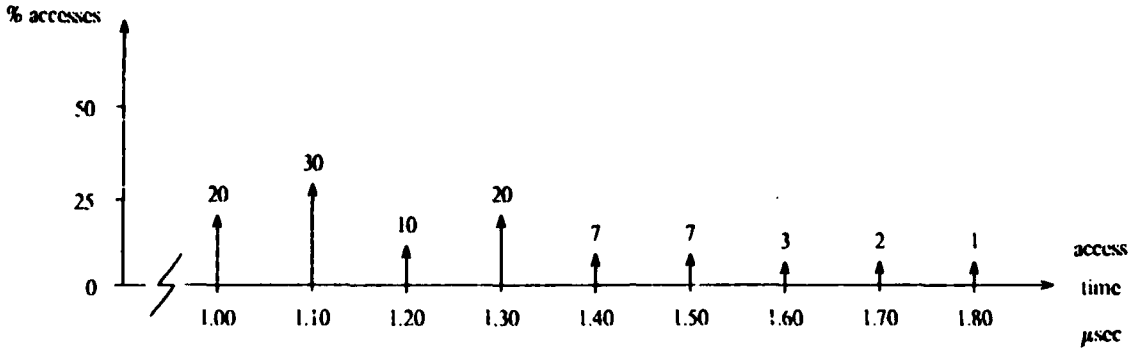


Figure A.5: Multibus access time - HSB port loaded

2) Accessing the global memory of the slice:

We loaded the Ringbus port of the slice global memory with 3 processors on another slice and 2 processors on yet another slice all executing

loop: bra loop

out of the first slice's global memory (i.e. over the Ringbus). Figure A.6 shows the resulting access time distribution for Multibus accesses to the slice global memory. We observed no noticeable difference in the distribution between reads and writes.

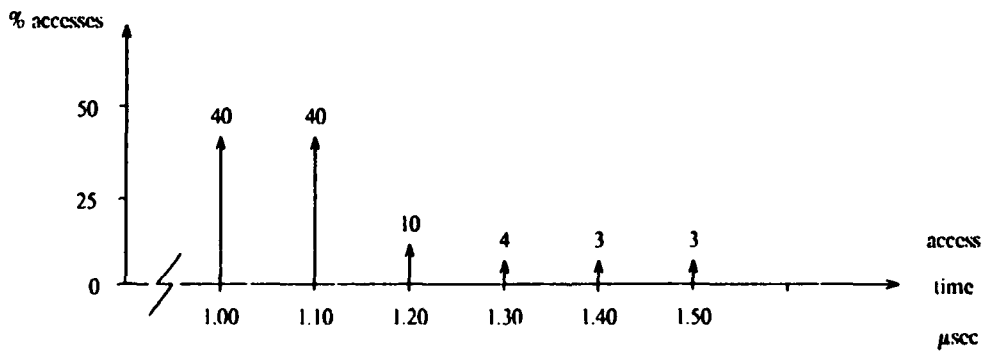


Figure A.6: Multibus access time of slice global memory - Ringbus port loaded

3.3.2 Ringbus Access Time

Figure A.7 depicts a Ringbus read access (byte or word) combining the points of view of the originating processor and the Ringbus, for a single processor in a slice.

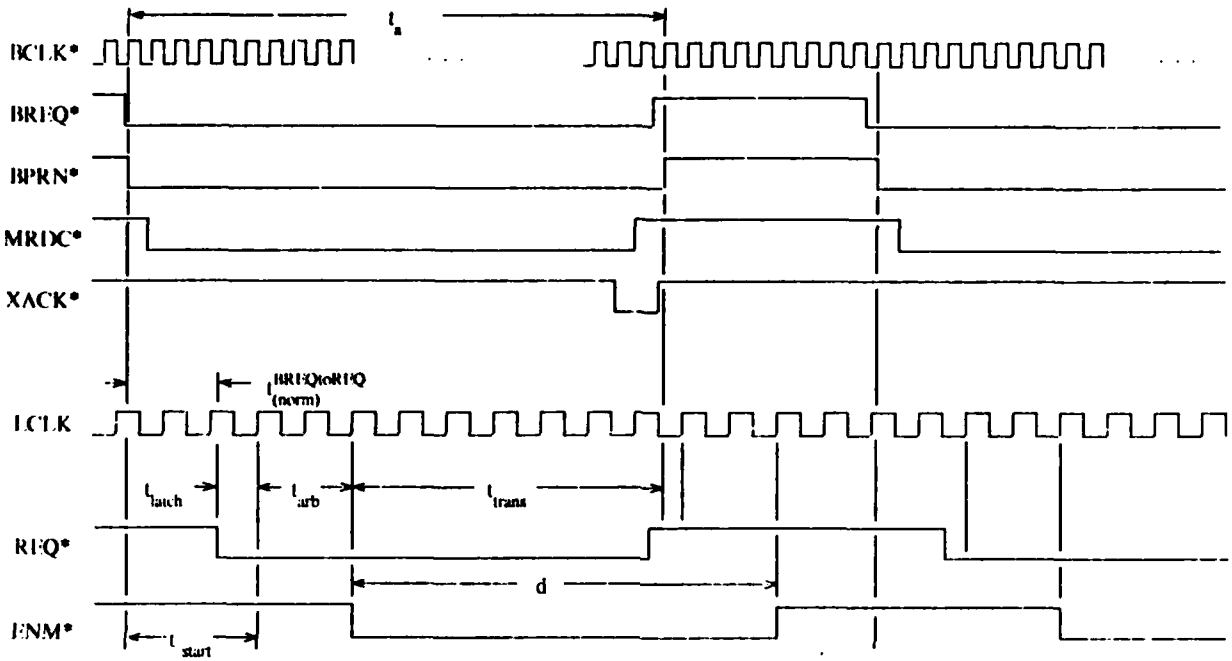


Figure A.7: Typical Ringbus read access

REQ* is the Ringbus request signal for the slice; it indicates that the RIB has detected an access that requires the Ringbus. ENM* (short for enable Multibus) is the Ringbus grant signal for the slice; it indicates that the Ringbus request has been allocated the necessary Ringbus segments. LCLK is the Ringbus arbiter clock signal. The Multibus and the Ringbus operate asynchronously with respect to each other, thus BCLK* and LCLK are not synchronized. Since REQ* is generated from Multibus signals, it is not synchronous with LCLK. On the other hand, ENM* is generated by the arbiter so it is synchronous with LCLK.

We define a number of quantities with respect to the diagram in Figure A.7 as follows:

t_a is the Ringbus access time (as defined earlier)

$t_{(norm)}^{BPRNtoREQ}$ is the normal time from initiation of a Ringbus read access† to generation of a Ringbus request. We discuss shortly what normal means in this context.

t_{latch} is the interval between the generation of a Ringbus request and the arbiter latching in on a rising LCLK edge.

t_{start} is the overhead associated with the start of a Ringbus access. It is the interval from the

initiation of the access† to the latching of the Ringbus request by the arbiter.

t_{arb} is the arbitration delay.

t_{trans} is the data transfer time. It is the interval from the end of the arbitration delay to the termination of the access.† Note that in actuality, data transfers on the Ringbus occur in the interval between the end of the arbitration delay and the falling edge of XACK*. Thus t_{trans} should be interpreted as the total interval in which a data transfer *could* occur, not as the interval in which it does actually occur.

Finally, d is the total duration for which segments are allocated to a Ringbus request.

The observed means of these quantities with one processor in a slice are as follows:

$\bar{t}_a = 2.17$ μsec (We present a histogram of the access time in section 3.3.2.1.)

$\bar{t}_{(norm)}^{BPRNtoREQ} = 230$ nsec

$t_{latch} = 150$ nsec

$\bar{t}_{start} = 380$ nsec

$t_{trans} = 1.38$ μsec

$\bar{d} = 9.1$ arbiter clock periods i.e. 1.82μsec. (The arbiter clock period was 200 nsec for all the measurements reported in this Appendix, as mentioned earlier.) d was either 9 or 10 arbiter clock periods.

Since the Multibus signals are asynchronous with respect to the arbiter clock, one would expect $\bar{t}_{latch}$ to be half an arbiter clock period, i.e. 100 nsec. In actual fact it is a little more than this (as can be seen above) due to the delay contributed by a preliminary sampling stage incorporated in the arbiter to inhibit metastability in the final sampling of REQ*. (t_{latch} is measured with respect to this final sampling.)

The start overhead, t_{start}, and the access time, t_a, vary with the spacing between the termination and initiation of successive Ringbus accesses generated by the slice in which the processor is located. (Of course, t_a also varies with the rate and type of Ringbus accesses generated by other slices.) In section 2.9.2 we defined this spacing to be the processing time of the single processor equivalent of this slice and we denoted it by t_p^{MBequ}. Figure A.8 depicts the same signals as in Figure A.7 for a Ringbus read access with $t_p^{MBequ} = 0$. (This was achieved with only two processors on a slice, each executing

```
loop:  bra loop
```

† By initiation and termination of the access we mean the first rising edge of BCLK* before BPRN* goes low and the first rising edge of BCLK* before BPRN goes high respectively, as defined in section 2 of this Appendix.

out of the global memory located in some other slice.)

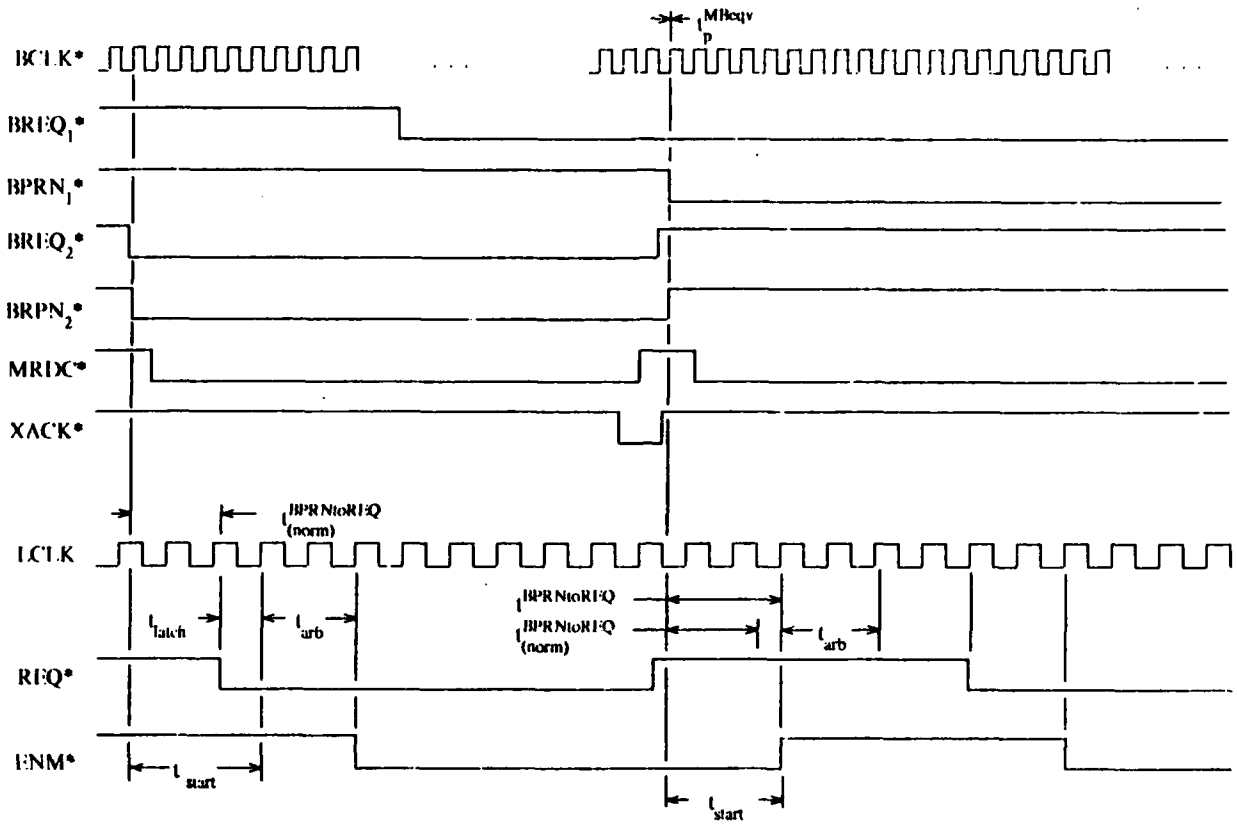


Figure A.8: Typical Ringbus read access with $t_p^{MBeqv}=0$

The reason that t_{start} and t_a vary with t_p^{MBeqv} is that the ENM* signal remains active for a period after the termination of a Ringbus access. If t_p^{MBeqv} is small enough, as in Figure A.8, the ENM* signal remains active past the initiation of the next Ringbus access and causes a delay in the assertion of the REQ* signal (since the REQ* signal for the present accesses cannot be asserted until the ENM* signal for the previous access has been disasserted). We define $t_{(norm)}^{BPRNtoREQ}$ to be the time from the initiation of a Ringbus access to the assertion of REQ* if REQ* is not delayed by the ENM* from the previous cycle. Thus $t^{BPRNtoREQ} = t_{(norm)}^{BPRNtoREQ} + t_{delay}$. The duration for which ENM* remains active past the termination of the previous access is $t_{start}^{prev} + t_{arb}^{prev} + d^{prev} - t_a^{prev} = d^{prev} - t_{trans}^{prev}$, where the superscript *prev* denotes the quantity in the previous access. Thus

$$t_{delay} = max(0,\ d^{prev} - t_{trans}^{prev} - t_p^{MBeqv} - t_{(norm)}^{BPRNtoREQ}).$$

If $t_{delay} > 0$ then, ignoring a small propagation delay, REQ* is asserted at the same time that ENM* from the previous access is disasserted. Thus t_{start} equals $t^{BPRNtoREQ}$ plus one arbiter period. Therefore

$$t_{start} = \begin{cases} t_{(norm)}^{BPRNtoREQ} + t_{latch} & \text{if } t_{delay} = 0 \\ t_{(norm)}^{BPRNtoREQ} + t_{delay} + 200nsec & \text{if } t_{delay} > 0 \end{cases}$$

where t_{latch} is the latch time when $t_{delay} = 0$. Finally $t_a = t_{start} + t_{arb} + t_{trans}$. Measurements revealed that the distribution of t_{trans} is approximately the same (fairly uniform over the interval 1.30 to 1.45 μsec) regardless of t_p^{MBeqv} (although its mean is slightly different for reads and writes). Thus t_{start} is the sole contributor to the change in t_a as t_p^{MBeqv} varies. Thus

$$t_a = \begin{cases} t_{(norm)}^{BPRNtoREQ} + t_{latch} + t_{arb} + t_{trans} & \text{if } t_{delay} = 0 \\ t_{(norm)}^{BPRNtoREQ} + t_{arb} + t_{trans} + t_{delay} + 200nsec & \text{if } t_{delay} > 0 \end{cases}$$

or simply

$$t_a = \begin{cases} t_a^{(norm)} & \text{if } t_{delay} = 0 \\ t_a^{(norm)} - t_{latch} + t_{delay} + 200nsec & \text{if } t_{delay} > 0 \end{cases}$$

where $t_a^{(norm)}$ is the access time if $t_{delay} = 0$.

In section 3.9 we defined the spacing between the completion of a Ringbus grant and the initiation of the next Ringbus grant from the same slice, excluding the waiting time of the Ringbus requests, to be the Ringbus equivalent processing time which we denoted by t_p^{RBeqv}.

If $t_{delay} = 0$, t_p^{RBeqv} is t_p^{MBeqv} plus t_{start} and t_{arb} and less the duration for which *ENM** remains active past the termination of the previous access. If $t_{delay} > 0$, t_p^{RBeqv} is three arbiter clock periods, i.e.

$$t_p^{RBeqv} = \begin{cases} t_p^{MBeqv} + t_{start} + t_{arb} - (t_{start}^{prev} + t_{arb} + d^{prev} + t_a^{prev}) & \text{if } t_{delay} = 0 \\ 600nsec & \text{if } t_{delay} > 0 \end{cases}$$

Note that if $t_{delay} = 0$, then $t_{start}^{prev} + t_{arb} + d^{prev} + t_a^{prev} \leq t_p^{MBeqv} + t_{(norm)}^{BPRNtoREQ}$ and $t_{start} = t_{(norm)}^{BPRNtoREQ} + t_{latch}$, yielding

$$t_p^{RBeqv} \geq \begin{cases} t_{latch} + t_{arb} & \text{if } t_{delay} = 0 \\ 600nsec & \text{if } t_{delay} > 0. \end{cases}$$

Now $t_{latch} \geq 0$ and $t_{arb} = 2$ arbiter clock periods, thus $t_p^{RBeqv} \geq 2$ or 3 arbiter clock periods.

The observed means of the quantities in Figure A.8 (read accesses with $t_p^{MBeqv} = 0$) are

$\bar{t}_a = 2.47\mu sec$

$\bar{t}_{start} = 690$ nsec

$t_{trans} = 1.38\ \mu sec$

$\bar{d} = 9$ arbiter clock periods i.e. $1.80 \mu sec$ (the duration was consistently 9 clock periods.)

The situation just discussed for Ringbus read accesses is similar for Ringbus write accesses. The observed means for write accesses are summarized in Table A.1.

	t_p^{MBeqv} large	$t_p^{MBeqv} = 0$
$\bar{t}_a$	2.13 μsec	2.58 μsec
$\bar{t}_{(norm)}^{BPRNtoREQ}$	230 nsec	n/a
$\bar{t}_{start}$	380 nsec	830 nsec
$\bar{t}_{trans}$	1.35 μsec	1.35 μsec
$\bar{d}$	9.6×200 nsec	9.5×200 nsec

Table A.1

These figures reveal several things. First, for large t_p^{MBeqv}, $\bar{t}_{(norm)}^{BPRNtoREQ}$, and $\bar{t}_{start}$ are the same for reads and writes while $\bar{t}_a$ is slightly less for writes than for reads. Second, for $t_p^{MBeqv} = 0$, both $\bar{t}_{start}$ and $\bar{t}_a$ are larger for successive write accesses than for successive read accesses. This is due to the fact that ENM^* remains active for a longer interval after the termination of a write access than after the termination of a read access. Thus the access time of a read or write depends on the type of access preceding it. We only investigated cases with reads preceding reads and writes preceding writes. Third, $\bar{d}$ is slightly greater for write accesses than for read accesses for both large t_p^{MBeqv} and $t_p^{MBeqv} = 0$.

3.3.2.1 Ringbus Access Time with other Memory Port Unloaded

The observed access time distribution for this case for reads and writes and for large t_p^{MBeqv} and $t_p^{MBeqv} = 0$ are given in Figure A.9.

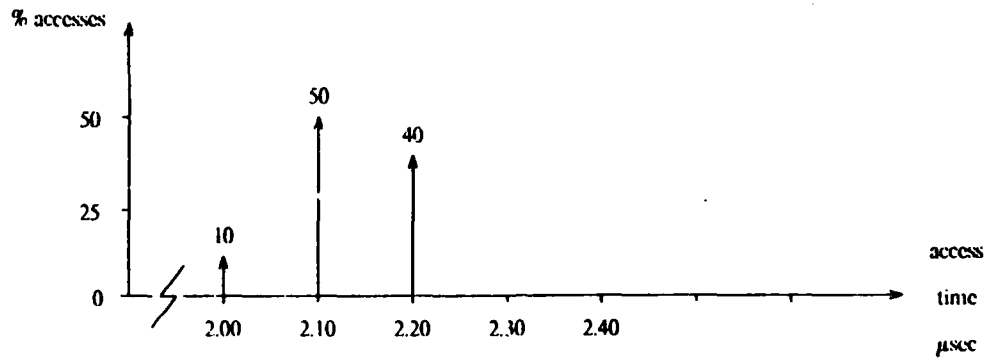


Figure A.9(a): Ringbus read access time distribution - t_p^{MBeqv} large

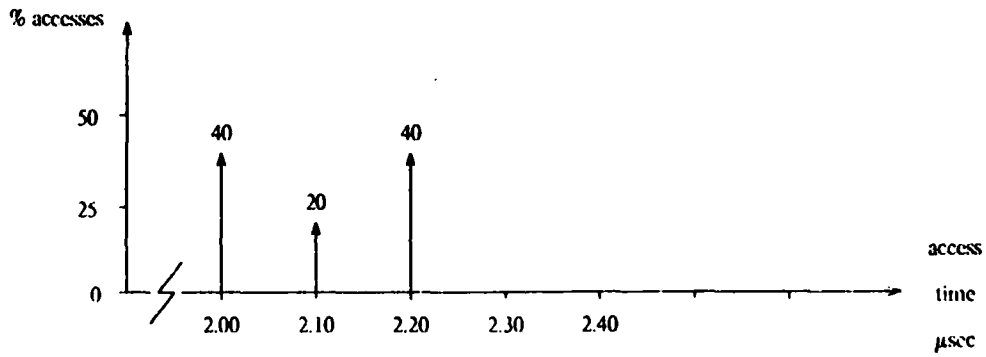


Figure A.9(b): Ringbus write access time distribution - t_p^{MBeqv} large

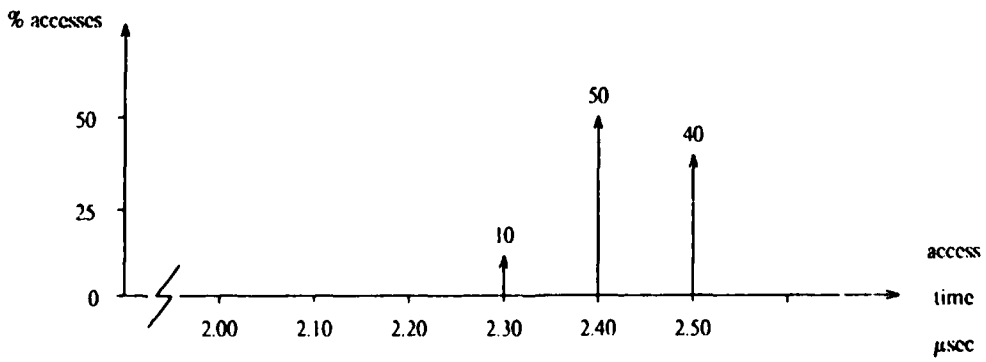


Figure A.9(c): Ringbus read access time distribution - $t_p^{MBeqv}=0$

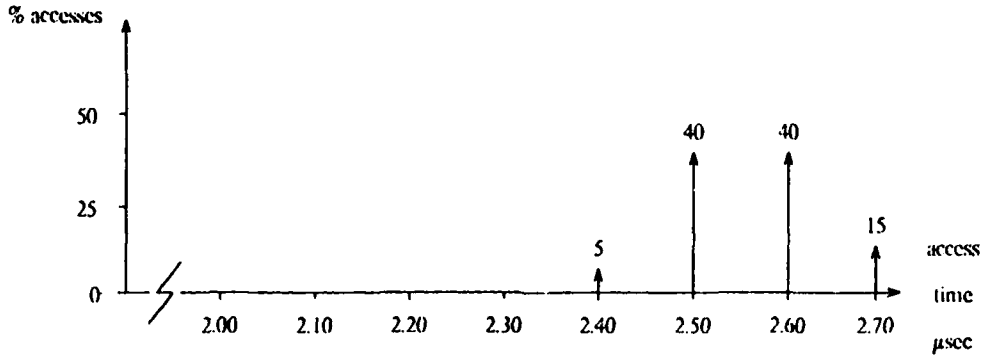


Figure A.9(d): Ringbus write access time distribution - $t_p^{MBeqv}=0$

Note that t_p^{MBeqv} does not have much effect on the distributions except for a horizontal shift reflecting the larger mean. The horizontal shift is indicative of the duration for which ENM* remains active past the termination of the previous access. As mentioned earlier, this duration, and hence the mean, depends on the type of access preceding the observed access. It seems that most of the randomness in the Ringbus access time, as least for reads preceded by reads and writes preceded by writes, is due to the random arrivals of the REQ* signal with respect to the arbiter clock.

3.3.2.2 Ringbus Access Time with Other Memory Port Loaded

We loaded the Multibus port of a slice global memory with four processors executing

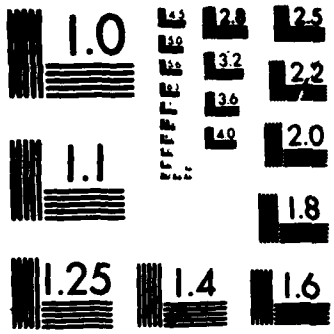
```
loop: bra loop
```

out of the slice global memory on the Multibus. We observed the access times for accesses to that same global memory over the Ringbus from another slice. No significant difference was observed in the access time distribution for reads and writes for large t_p^{MBeqv}. Our observations for large t_p^{MBeqv} are summarized in Figure A.10.

AD-A183 619 MODELING THE PERFORMANCE OF THE CONCERT MULTIPROCESSOR (U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE R B OSBORNE MAY 87 MIT/LCS/TR-375 4/4

UNCLASSIFIED N00014-83-K-0125 F/G 12/7 NL

END
9-87
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

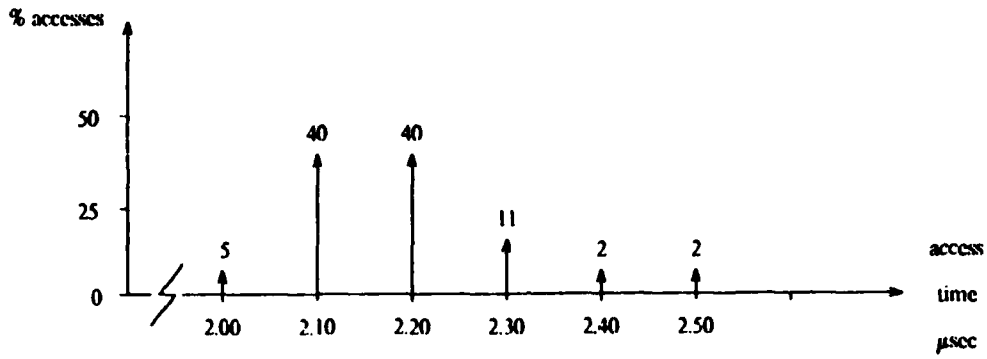


Figure A.10: Ringbus access time distribution - t_p^{MBeqv} large and other port loaded

For $t_p^{MBeqv}=0$, the access time distributions are similar, except for a horizontal shift. Note that for small enough t_p^{MBeqv}, the distribution, through the mean, does vary between the type of access observed and the type of access preceding it.

3.4 Access Times: General Observations

- The access time distribution is approximately the same for Multibus read and write accesses.
- The mean Multibus access time varies from 1.05 μsec to about 1.2 μsec depending on the loading on the other memory port.
- The access time distribution for Ringbus accesses depends on four factors:
  1) the type of access.
  2) the type of the preceding access.
  3) the value of t_p^{MBeqv}, and
  4) the loading on the other port of the access's memory.

The second factor is only relevant when t_p^{MBeqv} is small.

- The mean Ringbus access time varies from about 2.13 μsec to about 2.58 μsec depending on the above four factors.
- The tail of the access time distribution increases as the loading on the other port of the accessed memory increases.
- Reads and writes have a Ringbus grant duration (i.e. duration for which segments are allocated) of 9 or 10 arbiter clock periods.
- t_p^{MBeqv} cannot be less than 2 or 3 arbiter clock periods.

3.5 Read-Modify-Write Access Time

A test and set instruction has an access time of about 2.60 μsec to 2.70 μsec on the Multibus and an access time of about 4.30 μsec on the Ringbus. (These figures are with the other memory port unloaded in each case). We did not determine distributions for these two cases. For a Ringbus access, the segments are allocated to the access for about 19 arbiter clock periods.

Appendix B

In this appendix we present the proofs for the various Lemmas and Theorems which would have hindered the flow of presentation if they had been included in the main text.

Theorem 2.1

With independent identical processors with deterministic processing time t_p and deterministic access time t_a served by a single bus in FCFS order, the waiting time per request after at most two cycles of every processor is the same for every request. Moreover, after at most two cycles of every processor the FCFS queue is either always empty or always nonempty at the instant a request arrives at the queue.

Proof:

Let there be N processors denoted by $0, 1, 2, 3, \cdots, N-1$. Let $t_i(n)$ denote the time at which processor i makes its nth request for the bus (i.e. the instant that processor i's n^{th} request arrives at the end of the queue). Let $w_i(n)$ denote the waiting time of processor i's n^{th} request. To simplify the presentation we choose to interpret $t_i(n)$ and $w_i(n)$ as $t_{i \bmod N}(n + \lfloor i/N \rfloor)$ and $w_{i \bmod N}(n + \lfloor i/N \rfloor)$ respectively whenever $i<0$ or $i \geq N$. We then have

$$t_i(n+1) = t_i(n) + w_i(n) + t_a + t_p \tag{2.1.1}$$

Without loss of generality start counting the requests made by each processor at the first instant at which all N processors have made at least one request for the bus and let the initial condition be

$$t_0(1) \leq t_1(1) \leq t_2(1) \leq \cdots \leq t_{N-1}(1),$$

with ties being broken by the FCFS service discipline in favor of the smallest numbered processor.

Let the interval between the requests of processor i and processor $(i+1)\ mod\ N$ be denoted by $\Delta t_i(n)$ where

$$\Delta t_i(n) = t_{i+1}(n) - t_i(n)$$

where again we interpret $\Delta t_i(n)$ as $\Delta t_{i\ mod\ N}(n + \lfloor i/N \rfloor)$. From equation 2.1.1 we have

$$\Delta t_i(n+1) = \Delta t_i(n) + w_{i+1}(n) - w_i(n). \tag{2.1.2}$$

Because of the deterministic processing and access times, requests remain in their initial ordering for all $n \geq 1$; thus the FCFS queue enforces

$$w_{i+1}(n) = max(0,\ w_i(n) + t_a - \Delta t_i(n)). \tag{2.1.3}$$

With equations 2.1.2 and 2.1.3 we obtain

$$\Delta t_i(n+1) = \begin{cases} t_a & \text{if } w_{i+1}(n) > 0 \\ w_i(n) + \Delta t_i(n) & \text{if } w_{i+1}(n) = 0 \end{cases}$$

But if $w_{i+1}(n) = 0$, then $-w_i(n) + \Delta t_i(n) \geq t_a$. Thus $\Delta t_i(n+1) \geq t_a$, or more specifically

$$\Delta t_i(n) \geq t_a, \quad i \geq 0,\ n \geq 2 \tag{2.1.4}$$

i.e. after the first cycle of every processor, the arrival of successive requests must occur at intervals of at least the access time t_a. Equations 2.1.3 and 2.1.4 imply that $w_{i+1}(n) \leq w_i(n)$ for $n \geq 2$, with equality if and only if $w_i(n) = 0$ or $\Delta t_i(n) = t_a$ for $n \geq 2$ (or both).

Therefore if $w_i(n) = 0$ for any $i \geq 0$ and any $n \geq 2$, then $w_i(n) = 0$ for every $i \geq 0$ and every n past that point, and if $w_i(n) > 0$ then $\Delta t_{i-1}(n+1) = t_a$, implying that $w_i(n+1) = w_{i-1}(n+1)$.

Now either $w_i(2) = 0$ for some $i \geq 0$, in which case $w_i(n) = 0$ for all $i \geq 0$ and $n \geq 3$, or $w_i(2) > 0$ for all $i \geq 0$, in which case $\Delta t_{i-1}(3) = t_a$ and $w_i(3) = w_{i-1}(3)$ for all $i \geq 0$ which in turn implies that $\Delta t_i(n) = t_a$ and $w_i(n) = w_{N-1}(2)$ (by equations 2.1.2 and 2.1.3 respectively) for all $i \geq 0$ and $n \geq 3$. Therefore $w_i(n) = C$ for all $i \geq 0$ and $n \geq 3$ where $C = 0$ or $C > 0$.

Thus the waiting time per request is the same for every request for $n \geq 3$. Moreover, the waiting time per request for $n \geq 3$ is either always zero or always strictly positive, implying that the FCFS queue is either always empty or always nonempty respectively at the instant a request arrives at the queue.

Existence and Ergodicity Assumptions of Section 2.3

1. We assume that a stationary probability distribution exists for t_w.
2. We assume that the waiting time process is ergodic.
3. We assume that the time averages necessary for any application of Little's Law to the queueing system described in section 2.1 exist.

Some Conditions Guaranteeing the Validity of these Assumptions

We first describe the basic G/G/1//N queueing system as a Markov process and then consider some conditions on this Markov process to show the validity of the above assumptions. We assume throughout that:

1) the processing time, t_p, at each processor is a random variable with a stationary distribution and $E[t_p]<\infty$
2) the access time, t_a, is a random variable with a stationary distribution and $E[t_a]<\infty$
3) the processing time random variables (one for each processor) and the access time random variable are mutually independent.

Let $\vec{q}_n$ be a $N-1 \times 1$ column vector i.e. $\vec{q}_n = \begin{bmatrix} q_1 \\ q_2 \\ \cdot \\ \cdot \\ \cdot \\ q_{N-1} \end{bmatrix}$ whose elements indicate the time that a request enters the queue (i.e. the time that a processor makes a request) relative to the time at which the request presently in service began its service. Let the elements of $\vec{q}_n$ be ordered so that $q_1 \leq q_2 \leq \cdots \leq q_{N-1}$. Consider the vector $\vec{x}_n = \begin{bmatrix} w_n \\ \vec{q}_n \end{bmatrix}$ where w_n is the waiting time of the nth request to arrive at the queue. We then have $\vec{x}_{n+1} = f(\vec{x}_n, t_{a_n}, t_{p_n})$ where t_{a_n} is the service time (access time) of the nth request to arrive at the queue, t_{p_n} is the processing time of the nth request, and $f(\cdot)$ is a deterministic operation on its arguments defined as below.

The operation $f(\cdot)$:

1. Insert $t_{a_n} + t_{p_n}$ in the ordered list defined by $\vec{q}_n$ so that the elements remain in nondecreasing order with respect to the element indices. The list now contains N elements $q'_1, q'_2, \cdots, q'_N$ obeying $q'_1 \leq q'_2 \leq \cdots \leq q'_N$. The inserted request represents the time at which the next request from the processor whose request is presently in service again enters the queue, relative to the time at which its present request began service.

2. $w_{n+1} = max(0, w_n + t_{a_n} - q'_1)$, the waiting time of the next request to arrive at the queue.

3. Subtract q'_1 from all N entries in the ordered list $q'_1, q'_2, \cdots, q'_N$. Discard the zero value i.e. $q_{(n+1)_i} = q'_{i+1} - q'_1$, $1 \leq i \leq N-1$ where $q_{(n+1)_i}$ is the ith element of $\vec{q}_{n+1}$. This subtraction updates the request arrival times relative to the time at which the next - i.e. the $n+1^{th}$ - request began service.

The sequence $\{\vec{x}_n\}$, $n \geq 1$, with some initial probability distribution $Pr(\vec{x}_0 \leq \vec{y})^{\dagger}$ describes a discrete time continuous state Markov process with stationary transition probabilities (since $f(\cdot)$ is deterministic and t_{a_n} and t_{p_n} are stationary random variables).

Let $p^{(v)}(\vec{x}, A)$ denote the v step transition probabilities i.e. $p^{(v)}(\vec{x}, A) = Pr(\vec{x}$ in set $A \subset R^N$ after v transitions). If we define $p^{(1)}(\vec{x}, A) = p(\vec{x}, A)$ and $p(A) = Pr(\vec{x}_0 \in A)$ then we have $p^{(v+1)}(\vec{x}, A) = \int_{R^N} p^{(v)}(\vec{\eta}, A) p(\vec{x}, d\vec{\eta})$ for $v \geq 1$ and

$$Pr(\vec{x}_n \in A) = \begin{cases} p(A), & n = 1 \\ \int_{R^N} p^{(n-1)}(\vec{\eta}, A) p(d\vec{\eta}), & n > 1 \end{cases}$$

If $Pr(\vec{x}_n \in A)$ is independent of n then the process described by $\{\vec{x}_n\}$ is strictly stationary and $p(\cdot)$ is called a stationary probability distribution.

We define the sequence $\{\vec{x}_n\}$ to be periodic with period M if $\vec{x}_n = \vec{x}_{n+M}$ for $n \geq m$ for some integer $m > 0$ and some $M < \infty$.

We are now prepared to consider some conditions guaranteeing the validity of the Existence and Ergodicity Assumptions.

Case 0: $t_{w_n} = 0$ for every $n \geq m$ for some $m > 0$. In this trivial case $\lim_{n \to \infty} Pr(t_{w_n} \leq y)$ certainly exists and $\bar{t}_w = \lim_{n \to \infty} \frac{1}{n} \sum_{i=m}^{m+n} t_{w_i} = 0$. Thus a stationary probability distribution exists for t_w and t_w is ergodic. The application of Little's Law in this case is just an academic exercise since the majority of useful information has already been conveyed by the fact that $t_{w_n} = 0$ for $n \geq m$. We note that if $t_{w_n} = 0$ for $n \geq m$, then the N processor system is really N independent subsystems.

† Here $\vec{x} \leq \vec{y}$ means less than or equal element-wise.

Since each of these simple subsystems is closed, the time averages must be finite. Furthermore, due to the extremely simple structure and the stationarity of the probability distributions, the time averages cannot fail to exist due to periodicities. Therefore, the time averages for each subsystem must exist. And since all the subsystems are independent, we conclude that the time averages for the entire system must exist.

We assume in the following that we do not have $t_{w_n}=0$ for every $n \geq m$ for some $m>0$.

Case 1: The Markov process $\{\vec{x}_n\}$ satisfies Hypothesis D of Doob [p.192 of Ref. D1] which roughly stated is the following:

Hypothesis D

There is a probability assignment of sets $A \subset R^N$, an integer $v \geq 1$, and a positive ϵ, such that $p^{(v)}(\vec{x},A) \leq 1-\epsilon$ if $Pr(A) \leq \epsilon$.

(A more precise statement in terms of Borel sets and measures is given by Doob). This hypothesis basically says that if $Pr(A)$ is small then $p^{(v)}(\vec{x},A)$ is uniformly bounded away from 1. In particular this means that $\{\vec{x}_n\}$ cannot be periodic since then $p^{(v)}(\vec{x}_n,\vec{x}_m)=1$ for all $v \geq 1$ and $m>n$. If a density function $p_0(\vec{x},\vec{\eta})$ exists (i.e. $p_0(\vec{x},\vec{\eta}) \geq 0$, $\int_{R^N} p_0(\vec{x},\vec{\eta})d\vec{\eta}=1$, and $p(\vec{x},A)=\int_A p_0(\vec{x},\vec{\eta})d\vec{\eta}$) and is bounded, then Hypothesis D is satisfied [Ref. D1, p.193]. This condition is somewhat stronger than Hypothesis D and excludes impulses in $p_0(\vec{x},\vec{\eta})$ (i.e. discontinuities in $p(\vec{x},A)$). Hypothesis D does not exclude discontinuities in $p(\vec{x},A)$ as long as $p(\vec{x},A)<1$ for all $\vec{x}$, and all A for which $Pr(A)$ is small.

Now since we occasionally have t_{w_n} for $n \geq m$ for some m (by assumption), all N subsystems must communicate, hence $\{\vec{x}_n\}$ consists of a single communicating class (or ergodic set, as Doob calls it). Doob's Theorem 5.7 [Ref. D1, p.214] then asserts that under Hypothesis D there exists a unique stationary probability distribution for $\vec{x}_n$ independent of $\vec{x}_0$. This implies that a stationary probability distribution exists for t_w i.e. $\lim_{n\to\infty} Pr(t_{w_n} \leq y)$ exists. Furthermore, Doob's Theorem 2.1 [Ref. D1, p.465] (see also Theorem 6.1 and its proof on p.219) asserts that $\bar{t}_w = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} t_{w_i}$.

All the time averages necessary for any application of Little's Law to the queueing system described by the Markov process $\{\vec{x}_n\}$ can be derived from this Markov process. Any particular time average of interest can be expressed as the time average of some random variable which is a deterministic function of $\vec{x}_n$, t_{a_n}, and t_{p_n}. For example, if a_n represents the

interval between the arrival of the nth and the $n+1^{th}$ request to be served in the G/G/1//N system, then the time average reciprocal of the arrival rate (if it exists) is $\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}a_i$ where $a_n = min(q_{n-1}, t_{a_n} + t_{p_n})$. As another example, if n_n represents the number of requests in the G/G/1//N queue waiting for service when the nth begins service, then the time average queue length (if it exists) is $\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}n_i t_{a_i}$ where n_n is computed in a straightforward manner from $\vec{q}_n$.* Since the Markov process $\{\vec{x}_n\}$ has a unique stationary probability distribution and t_a and t_p have stationary probability distributions, any random variable which is a deterministic function of these quantities will also have a stationary probability distribution. Doob's Theorem 2.1 [Ref D1, p.465] then implies that the time average of such a random variable exists (since it must equal its mean, which exists since a stationary probability distribution exists).

Case 2: The Markov process $\{\vec{x}_n\}$ is periodic.

Lemma B.1

The Markov process $\{\vec{x}_n\}$ is periodic if and only if t_{a_n} and t_{p_n} are deterministic random variables - i.e. constants for all $n \geq 0$.

Proof:

The "if" part: If t_{a_n} and t_{p_n} are deterministic random variables, then by Theorem 2.1 t_{w_n} is a constant for $n \geq 3N$ where N is the number of processors. Now t_{w_n}, t_{a_n}, and t_{p_n} constants for $n \geq 3N$ implies that $\{\vec{x}_n\}$ is periodic with period at most N.

The "only if" part: Suppose that $\{\vec{x}_n\}$ was periodic and t_{a_n} and t_{p_n} were not both deterministic random variables. Consider some state $\vec{x}_n$ of the periodic portion of the sequence $\{\vec{x}_n\}$. Then the next state depends on t_{a_n} and t_{p_n}. But because $\{\vec{x}_n\}$ is periodic, this next state, $\vec{x}_{n+1}$, is already known with probability 1. Since t_{a_n} and t_{p_n} are not both deterministic random variables (and since both are stationary random variables), there is some positive probability of the sum $t_{a_n} + t_{p_n}$ being such that some element in the ordered list obtained via the $f(\cdot)$ operation is

* Note that these time averages are different from but equivalent to those in the statement of Little's Law in section 2.3 if they exist.

different from that in the known next state. This contradicts the hypothesis that $\{\vec{x}_n\}$ is periodic.

Corollary B.1

If the Markov process $\{\vec{x}_n\}$ is periodic, then

1. t_{w_n} is a constant for $n \geq m$ for $m > 0$ large enough and thus $\lim_{n \to \infty} Pr(t_{w_n} \leq y)$ exists i.e. a stationary probability distribution exists for t_{w_n}.
2. $\bar{t}_w = \lim_{n \to \infty} \frac{1}{n} \sum_{i=m}^{m+n} t_{w_i}$
3. the time averages necessary for any application of Little's Law to the queueing system described by $\{\vec{x}_n\}$ exist.

Proof:

Points 1 and 2 follow immediately from Lemma B.1 and Theorem 2.1. Since the Markov process $\{\vec{x}_n\}$ is periodic, any time average derived from $\{\vec{x}_n\}$ is equal to the same average over one period of $\{\vec{x}_n\}$. Since by hypothesis the period of $\{\vec{x}_n\}$ is finite, all possible averages derived from $\{\vec{x}_n\}$ must exist and hence all possible time averages derived from $\{\vec{x}_n\}$ must also exist. Point 3 now follows since the set of time averages necessary for any application of Little's Law to the queueing system described by $\{\vec{x}_n\}$ is a subset of all possible time averages derived from $\{\vec{x}_n\}$.

Remark:

The above three cases are not necessarily exhaustive.

Theorem 2.2

Consider the queueing model described in section 2.1 with stationary processing and access time distributions with means $t_p < \infty$ and $t_a < \infty$ respectively and subject to the assumptions in section 2.3. Then $w(N+1) - w(N) \leq t_a$ where $w(N)$ denotes the mean waiting time in a N processor model.

Proof:

The N processor model is a G/G/1//N queueing system as described in section 2.1. Let the processing and access time distributions of this G/G/1//N system be denoted by $F_{t_p}(x)$ and $F_{t_a}(x)$ respectively. The $N+1$ processor model is a G/G/1//N+1 queueing system with the same processing and access time distributions - $F_{t_p}(x)$ and $F_{t_a}(x)$ respectively - as for the G/G/1//N system. In the remainder of the proof the G/G/1//N system and the G/G/1//N+1 system are referred to as the G/G/1//N/P system and the G/G/1//N+1/P system respectively to emphasize the special relationship between the two systems. The additional P denotes "pair".

Let the state of the G/G/1//N system at time t be described by:

$$X(t,N) = (n, x, t_{p_1}, t_{p_2}, \ldots, t_{p_N})$$

where N denotes the number of processors, n indicates the number of requests queued for service and presently in service, x is the residual access time, and t_{p_i}, $1 \leq i \leq N$, is the residual processing time at processor i.

It is not necessary to include t_{p_i} in the state description when processor i is not processing - i.e. when processor i is waiting for a request to complete; indeed, t_{p_i} has no meaning in this case. However, we choose to include t_{p_i} in the state for this type of situation for notational convenience (i.e. so we can always write $X(t,N)$ in the same way independent of which processors are processing). To ensure that t_{p_i} is always well defined, we let $t_{p_i} = 0$ when processor i is not processing. The analogous situation occurs with x when there are no outstanding requests. In the following we refer to the N-tuple $t_{p_1}, t_{p_2}, \cdots, t_{p_N}$ by the vector $\vec{t_p}$.

Similarly, let the state of the G/G/1//N+1/P system at time t' be described by $X(t,N+1) = (n, x, t_{p_1}, \cdots, t_{p_N}, t_{p_{N+1}})$. The interpretation of each quantity in this state description is the same as for the G/G/1/N/P system.

The proof is based on an argument that the behaviour of a G/G/1//N/P system with n requests queued for and in service and the behaviour of a G/G/1//N+1/P system $n+1$ requests queued for and in service are probabilistically identical. The details of the argument are as follows.

Consider a state $X(t,N)=(n,x,\vec{t_p})$. For some t' and $n\geq 1$, there exists a state $X(t',N+1)=(n+1,x',\vec{t_p}')$ where $x'=x$ and $\vec{t_p}'=(\vec{t_p},t_{p_{N+1}})$, $t_{p_{N+1}}=0$, since the G/G/1//N+1 processor system is identical to the G/G/1//N system aside from an extra processor. In fact, for each $n\geq 1$ and for each state $X(t,N)=(n,x,\vec{t_p})$, there exists a corresponding state $X(t',N+1)=(n+1,x',\vec{t_p}')$ for some t' with $x'=x$ and $\vec{t_p}'=(\vec{t_p},t_{p_{N+1}})$, $t_{p_{N+1}}=0$.

The state $X(t',N+1)=(n+1,x,(\vec{t_p},0))$ differs from the state $X(t,N)=(n,x,\vec{t_p})$ (aside from the possible difference in times t and t') only in that there is one more request in the queue. But for $n\geq 1$, this additional request in the queue cannot be receiving service (without loss of generality we can consider the additional request to be the last request in the queue). Furthermore, the processing times at each processor and the access time of each request are independent of each other and everything else in the system including the additional request in the queue. Therefore, for $n\geq 1$, the system operation cannot depend on the fact that there is an additional request in the queue. Thus, for $n\geq 1$, the probabilistic behaviour in states $X(t,N)=(n,x,\vec{t_p})$ and $X(t',N+1)=(n+1,x,(\vec{t_p},0))$ must be identical. Since given any state $X(t,N)=(n,x,\vec{t_p})$ with $n\geq 1$, there exists some state $X(t',N+1)=(n+1,x,(\vec{t_p},0))$ and since these two states must have the same probabilistic behaviour, the G/G/1//N/P and the G/G/1//N+1/P systems have the same probabilistic behaviour, as long as $n\geq 1$. In particular, if one request in the queue of the G/G/1//N+1/P system was hidden from view, an observer would be unable to distinguish the G/G/1//N/P queueing system from the G/G/1//N+1/P system so long as $n\geq 1$.

We now introduce some notation. Let π_i^N denote the fraction of time that the G/G/1//N/P system has i requests in the queue and in service. Let n_q^N denote the time average of the number of requests queued for but not in service in the G/G/1//N/P system. Finally, let ρ^N denote the fraction of time that the server is busy. We have

$$n_q^N=\sum_{i=2}^{N}(i-1)\pi_i^N \qquad \rho^N=\sum_{i=1}^{N}\pi_i^N$$

Let π_i^{N+1}, n_q^{N+1}, and ρ^{N+1} be the analogous quantities for the G/G/1//N+1/P system.

Lemma B.2

$\pi_{i+1}^{N+1} = C\pi_i^N$ for $i \geq 1$

Proof:

As argued earlier, the G/G/1//N/P system with i requests in the queue is probabilistically identical to the G/G/1/N+1/P system with $i+1$ requests in the queue and in service if $i \geq 1$. It follows that $\pi_{i+1}^{N+1} = C\pi_i^N$ for some constant C.

Proceeding with the proof of Theorem 2.2, there are two cases to consider.

Case 1: $\bar{w}(N) = 0$

Lemma B.3

If $\bar{w}(N) = 0$ then $\pi_i^{N+1} = 0$ for $i > 2$.

Proof:

If $\bar{w}(N) = 0$, then $\pi_i^N = 0$ for $i > 1$. It follows from Lemma B.2 that $\pi_i^{N+1} = 0$ for $i > 2$.

Thus in case 1 $\bar{n}_q^{N+1} = \pi_2^{N+1} = \rho^{N+1} - \pi_1^{N+1} \leq \rho^{N+1}$. From Little's Law we have $\bar{w}(N+1) = \dfrac{\bar{n}_q^{N+1}\,\bar{t}_a}{\rho^{N+1}} \leq \bar{t}_a$. Therefore $\bar{w}(N+1) - \bar{w}(N) \leq \bar{t}_a$.

Case 2: $\bar{w}(N) > 0$

If $\bar{w}(N) > 0$, then $\pi_i^N > 0$ for some $i > 1$. By Little's Law we have $\bar{w}(N) = \dfrac{\bar{n}_q^N\,\bar{t}_a}{\rho^N}$ and $\bar{w}(N+1) = \dfrac{\bar{n}_q^{N+1}\,\bar{t}_a}{\rho^{N+1}}$.

$$\bar{n}_q^{N+1} = \sum_{i=2}^{N+1}(i-1)\pi_i^{N+1} = \sum_{i=2}^{N}(i-1)\pi_{i+1}^{N+1} + \sum_{i=1}^{N}\pi_{i+1}^{N+1}$$

and $\rho^{N+1} = \pi_1^{N+1} + \sum_{i=2}^{N+1}\pi_i^{N+1}$. Combining these two relations with Lemma B.2 yields:

$$\bar{n}_q^{N+1} = C\bar{n}_q^N + \rho^{N+1} - \pi_1^{N+1}$$

Therefore

$$\bar{w}(N+1) - \bar{w}(N) = \bar{t}_a\left[\frac{\bar{n}_q^{N+1}}{\rho^{N+1}} - \frac{\bar{n}_q^N}{\rho^N}\right] = \bar{t}_a\left[1 + \frac{\bar{n}_q^N(C\rho^N - \rho^{N+1})}{\rho^{N+1}\rho^N} - \frac{\pi_1^{N+1}}{\rho^{N+1}}\right]$$

Applying Lemma B.2 again yields $\rho^{N+1} = \pi_1^{N+1} + C\sum_{i=1}^{N}\pi_i^N = \pi_1^{N+1} + C\rho^N$. Thus finally,

$$\bar{w}(N+1) - \bar{w}(N) = \bar{t}_a\left\{1 - \frac{\bar{n}_q^N \pi_1^{N+1}}{\rho^{N+1}\rho^N} - \frac{\pi_1^{N+1}}{\rho^{N+1}}\right\} \le \bar{t}_a$$

Theorem 2.4

Let $F^*_{ab}(s)$ denote the Laplace transform of the probability density function $f_{ab}(x)$ with mean $\bar{x}$ where $f_{ab}(x)=0$ for $x \notin [a,b]$; $0<a$ and $b<2a$. Let $F^*(s)_{M/M/\infty//N}$ denote the Laplace transform of the exponential density function with the same mean $\bar{x}$. Then $F^*(s)_{M/M/\infty//N} \geq F^*_{ab}(s)$ for s real and $s \geq 0$.

Proof

We are to show that $F^*(s)_{M/M/\infty//N} = \dfrac{1}{s\bar{x}+1} \geq F^*_{ab}(s) = \int_a^b e^{-sx} f_{ab}(x)dx$ for s real and $s \geq 0$ where $\bar{x} = \int_a^b x f_{ab}(x)dx$. This is equivalent to showing that $\epsilon \leq 1$ where $\epsilon = (1+s\bar{x})\int_a^b e^{-sx} f_{ab}(x)dx$. We note that ϵ and all its derivatives with respect to s exist and are continuous in s. In addition we note that $\epsilon = 1$ at $s=0$. It thus suffices to show that $\dfrac{\partial \epsilon}{\partial s} \leq 0$ for $s \geq 0$.

$$\frac{\partial \epsilon}{\partial s} = \bar{x}\int_a^b e^{-sx} f_{ab}(x)dx - (1+s\bar{x})\int_a^b x e^{-sx} f_{ab}(x)dx$$

$$\frac{\partial \epsilon}{\partial s}\Big|_{s=0} = 0$$

$$\frac{\partial^2 \epsilon}{\partial s^2} = -2\bar{x}\int_a^b x e^{-sx} f_{ab}(x)dx + (1+s\bar{x})\int_a^b x^2 e^{-sx} f_{ab}(x)dx$$

Now $\dfrac{\partial^2 \epsilon}{\partial s^2} \leq (-2\bar{x} + (1+s\bar{x})b)\int_a^b x e^{-sx} f_{ab}(x)dx$. Since $\int_a^b x e^{-sx} f_{ab}(x)dx > 0$ and $-2\bar{x} + (1+s\bar{x})b < 0$ for s real and $s < \dfrac{2\bar{x}-b}{b\bar{x}}$, we must have

$\dfrac{\partial^2 \epsilon}{\partial s^2} < 0$ for $0 \leq s \leq \dfrac{2\bar{x}-b}{b\bar{x}}$,

implying that $\dfrac{\partial \epsilon}{\partial s} \leq 0$ for $0 \leq s \leq \dfrac{2\bar{x}-b}{b\bar{x}}$.

Furthermore, $\dfrac{\partial \epsilon}{\partial s} \leq (\bar{x} - (1+s\bar{x})a)\int_a^b e^{-sx} f_{ab}(x)dx$.

Since $\int_a^b e^{-sx} f_{ab}(x)dx > 0$ for s real and $\bar{x} - (1+s\bar{x})a < 0$ for $s > \dfrac{\bar{x}-a}{a\bar{x}}$,

we must have $\frac{\partial e}{\partial s} < 0$ for $s > \frac{\bar{x} - a}{a\bar{x}}$.

But $0 < b < 2a$ implies $\bar{x}b - ab < 2a\bar{x} - ab$ which implies $\frac{\bar{x} - a}{a\bar{x}} < \frac{2\bar{x} - b}{b\bar{x}}$.

Therefore $\frac{\partial e}{\partial s} \leq 0$ for $s \geq 0$.

Theorem 3.1

Preamble:

Consider the following two state descriptions of a Ringbus with S slices, request probabilities p_i, $i = -(S/2-1), \cdots, -1, 0, 1, \cdots, S/2$, and subject to the assumptions in chapter 3:

State description A: $(r_1,d_1;r_2,d_2;\cdots;r_S,d_S)$

State description B: $(r_1,w_1,d_1;r_2,w_2,d_2;\cdots;r_S,w_S,d_S)$

r_i and d_i denote the request at slice i and the duration of the grant at slice i, as discussed in section 3.2.1. w_i denotes the interval for which the request at slice i has waited so far without being granted. We adopt the convention that $w_i = 0$ whenever $d_i \neq 0$. The arbitration problem relative to state description A is to find a policy $\mathbf{D}_A$ which maximizes the throughput $g^{\mathbf{D}_A}$ given by

$$g^{\mathbf{D}_A} = \sum_{(\underline{r},\underline{d})} q_{(\underline{r},\underline{d})}^{d(\underline{r},\underline{d})} \pi_{(\underline{r},\underline{d})}^{\mathbf{D}_A}$$

where

$(\underline{r},\underline{d})$ denotes a particular state (using vector notation),

$d(\underline{r},\underline{d})$ is the decision in state $(\underline{r},\underline{d})$,

$q_{(\underline{r},\underline{d})}^{d(\underline{r},\underline{d})}$ is the reward in state $(\underline{r},\underline{d})$ under decision $d(\underline{r},\underline{d})$,

$p_{(\underline{r},\underline{d}),(\underline{r}',\underline{d}')}^{d(\underline{r},\underline{d})}$ is the one step transition probability from state $(\underline{r},\underline{d})$ to state $(\underline{r}',\underline{d}')$ under decision $d(\underline{r},\underline{d})$, and

$\pi_{(\underline{r},\underline{d})}^{\mathbf{D}_A}$ is the steady-state probability of being in state $(\underline{r},\underline{d})$ under policy $\mathbf{D}_A$.

The arbitration problem relative to state description B is to find a policy $\mathbf{D}_A$ which maximizes the throughput $g^{\mathbf{D}_B}$ given by

$$g^{\mathbf{D}_B} = \sum_{(\underline{r},\underline{w},\underline{d})} q_{(\underline{r},\underline{w},\underline{d})}^{d(\underline{r},\underline{w},\underline{d})} \pi_{(\underline{r},\underline{w},\underline{d})}^{\mathbf{D}_B}$$

where

$(\underline{r},\underline{w},\underline{d})$ denotes a particular state (using vector notation),

$d(\underline{r},\underline{w},\underline{d})$ is the decision in state $(\underline{r},\underline{w},\underline{d})$,

$q_{(\underline{r},\underline{w},\underline{d})}^{d(\underline{r},\underline{w},\underline{d})}$ is the reward in state $(\underline{r},\underline{w},\underline{d})$ under decision $d(\underline{r},\underline{w},\underline{d})$,

$p_{(\underline{r},\underline{w},\underline{d}),(\underline{r}',\underline{w}',\underline{d}')}^{d(\underline{r},\underline{w},\underline{d})}$ is the one step transition probability from state $(\underline{r},\underline{w},\underline{d})$ to state $(\underline{r}',\underline{w}',\underline{d}')$ under decision $d(\underline{r},\underline{w},\underline{d})$, and

$\pi_{(\underline{r},\underline{w},\underline{d})}^{\mathbf{D}_B}$ is the steady-state probability of being in state $(\underline{r},\underline{w},\underline{d})$ under policy $\mathbf{D}_B$.

Note that if $(\underline{r}',\underline{w}',\underline{d}')$ is the immediate successor of some state $(\underline{r},\underline{w},\underline{d})$ then $w'_i = 0$ if request r_i was granted and $w'_i = w_i + 1$ otherwise. Thus $p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(\underline{r}',\underline{w}',\underline{d}')} = 0$ if for any i either a) request r_i is granted and $w'_i \neq 0$, or b) request r_i is not granted and $w'_i \neq w_i + 1$. This can be expressed more concisely as $p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(\underline{r}',\underline{w}',\underline{d}')} = 0$ if $(\underline{r}',\underline{w}',\underline{d}') \notin W((\underline{r},\underline{w},\underline{d}))$ where $(\underline{r}',\underline{w}',\underline{d}') \in W((\underline{r},\underline{w},\underline{d}))$ if for each $i = -(S/2-1), \cdots, -1, 0, 1, \cdots, S/2$ either $w_i' = 0$ if request r_i is granted or $w_i' = w_i + 1$ if request r_i is not granted.

Statement:

Let $\mathbf{D}^{opt}_A$ be any optimum policy for the arbitration problem relative to state description A. Then, if there is no upper bound constraint on the waiting times w_i, an optimum stationary policy $\mathbf{D}^{opt}_B$ for the arbitration problem relative to state description B is the following policy $\mathbf{D}^{*}_B$:

Choose $d^*(\underline{r},\underline{w},\underline{d})$ in each state $(\underline{r},\underline{w},\underline{d})$ such that $d^*(\underline{r},\underline{w},\underline{d}) = d^{opt}(\underline{r},\underline{d})$ for all $\underline{w}$. Consequently

$$p^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(\underline{r}',\underline{w}',\underline{d}')} = \begin{cases} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(\underline{r}',\underline{d}')} & \text{for all } \underline{w} \text{ and all } \underline{w}' \text{ such that} (\underline{r}',\underline{w}',\underline{d}') \in W((\underline{r},\underline{w},\underline{d})) \\ 0 & \text{otherwise} \end{cases}$$

and

$$q^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} = q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} \quad \text{for all } \underline{w}.$$

Furthermore, $g^{\mathbf{D}^{opt}_B} = g^{\mathbf{D}^{opt}_A}$. Thus the waiting time information $\underline{w}$ is irrelevant in determining the optimum throughput.

Proof:

For the arbitration problem relative to state description A, let $v^{\mathbf{D}^{opt}_A}_{(\underline{r},\underline{d})}$ denote the value of being in state $(\underline{r},\underline{d})$ under policy $\mathbf{D}^{opt}_A$ and let $v_1^{\mathbf{D}^{opt}_A}$ denote the value of being in state $1 \equiv (\underline{0},\underline{0})$. Then from equation 3.6 we have

$$g^{\mathbf{D}^{opt}_A} + v^{\mathbf{D}^{opt}_A}_{(\underline{r},\underline{d})} - v_1^{\mathbf{D}^{opt}_A} = q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(\underline{r}',\underline{d}')} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(\underline{r}',\underline{d}')} (v^{\mathbf{D}^{opt}_A}_{(\underline{r}',\underline{d}')} - v_1^{\mathbf{D}^{opt}_A})$$

For the arbitration problem relative to state description B, let $V_{(\underline{r},\underline{w},\underline{d})}(n)$ denote the optimal expected total reward accumulated over n rounds if the process started in state $(\underline{r},\underline{w},\underline{d})$ with terminal reward $V_{(\underline{r},\underline{w},\underline{d})}(0)$. Then from equation 3.16 we have

$$V_{(\underline{r},\underline{w},\underline{d})}(n+1) = \max_{d(\underline{r},\underline{w},\underline{d})} \left(q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(\underline{r}',\underline{w}',\underline{d}')} p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(\underline{r}',\underline{w}',\underline{d}')} V_{(\underline{r}',\underline{w}',\underline{d}')}(n) \right)$$

$$= max[\, q^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(r',w',d')} p^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} V_{(r',w',d')}(n),$$

$$\max_{d(\underline{r},\underline{w},\underline{d}) \neq d^*(\underline{r},\underline{w},\underline{d})} (\quad q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(r',w',d')} p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} V_{(r',w',d')}(n))]$$

Substituting in for $p^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')}$ and $q^{d^*(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})}$, we have

$$V_{(\underline{r},\underline{w},\underline{d})}(n+1) = max[\, q^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(r',w',d') \in W((\underline{r},\underline{w},\underline{d}))} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')} V_{(r',w',d')}(n),$$

$$\max_{d(\underline{r},\underline{w},\underline{d}) \neq d^*(\underline{r},\underline{w},\underline{d})} (\quad q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(r',w',d')} p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} V_{(r',w',d')}(n))]$$

Let the terminal rewards be $V_{(\underline{r},\underline{w},\underline{d})}(0) = v^{D^{opt}_A}_{(\underline{r},\underline{d})} - v^{D^{opt}_A}_1$ for every $(\underline{r},\underline{w},\underline{d})$. Then

$$V_{(\underline{r},\underline{w},\underline{d})}(1) = max[\, q^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(r',d')} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1),$$

$$\max_{d(\underline{r},\underline{w},\underline{d}) \neq d^*(\underline{r},\underline{w},\underline{d})} (\quad q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(r',w',d')} p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1))]$$

Now every decision $d(\underline{r},\underline{w},\underline{d})$ in state $(\underline{r},\underline{w},\underline{d})$ is the same as some decision $d(\underline{r},\underline{d})$ in state $(\underline{r},\underline{d})$. Thus $p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} = p^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')}$ and $q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} = q^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})}$ for some $d(\underline{r},\underline{d})$ for every $d(\underline{r},\underline{w},\underline{d})$ for every state $(\underline{r},\underline{w},\underline{d})$. Since D^{opt}_A is an optimal policy,

$$q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(r',d')} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1) \geq q^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(r',d')} p^{d(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1)$$

for every $d(\underline{r},\underline{d})$ and thus

$$q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} + \sum_{(r',d')} p^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})(r',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1) \geq q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} + \sum_{(r',w',d') \in W((\underline{r},\underline{w},\underline{d}))} p^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})(r',w',d')} (v^{D^{opt}_A}_{(r',d')} - v^{D^{opt}_A}_1)$$

for every $d(\underline{r},\underline{w},\underline{d})$ and every state $(\underline{r},\underline{w},\underline{d})$. Therefore $V_{(\underline{r},\underline{w},\underline{d})}(1) = g^{D_A} + v^{D^{opt}_A}_{(\underline{r},\underline{d})} - v^{D^{opt}_A}_1$ for every state $(\underline{r},\underline{w},\underline{d})$. Thus the policy D^*_B is an optimal stationary policy. It follows that $g^{D^{opt}_B} = g^{D^*_B}$.

Under policy D^*_B, $q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} = q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})}$ for all $\underline{w}$ and $\sum_{\underline{w}} \pi^{D^*_B}_{(\underline{r},\underline{w},\underline{d})} = \pi^{D_A}_{(\underline{r},\underline{d})}$. Thus

$$g^{D^*_B} = \sum_{(\underline{r},\underline{w},\underline{d})} \pi^{D^*_B}_{(\underline{r},\underline{w},\underline{d})} q^{d(\underline{r},\underline{w},\underline{d})}_{(\underline{r},\underline{w},\underline{d})} = \sum_{(\underline{r},\underline{d})} \left\{ \sum_{\underline{w}} \pi^{D^*_B}_{(\underline{r},\underline{w},\underline{d})} \right\} q^{d^{opt}(\underline{r},\underline{d})}_{(\underline{r},\underline{d})} = \sum_{(\underline{r},\underline{d})} \pi^{D^{opt}_A}_{(\underline{r},\underline{d})} q_{(\underline{r},\underline{d})} = g^{D^{opt}_A}.$$

Therefore $g^{D^{opt}_B} = g^{D^{opt}_A}$.

References

[A1] Ali, S.V., "Performance Modeling of the Concert Multibus", SB Thesis, Dept. of Elec. Eng. and Comp. Science, MIT, 1985

[A2] Anderson, T.L., "The Design of a Multiprocessor Development System", Report # TR-279, Laboratory for Computer Science, MIT, 1982

[A3] Ashcroft, H., "The Productivity of Several Machines Under the Care of One Operator", *J. Roy. Stat. Soc.*, Series B, Vol. 12, 1950, p.145-151

[B1] Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G., "Open, closed, and Mixed Networks of Queues with Different Classes of Customers", *JACM*, Vol. 22, No. 2, April 1975, p. 248-260

[B2] Benson, F., and Cox, D.R., "The Productivity of Machines Requiring Attention at Random Intervals", *J. Roy. Stat. Soc.*, Series B, Vol. 131, 1951, p. 65-82

[B3] Buzen, J.P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers", *Comm. ACM*, Vol. 16, 1973, p. 527-531

[C1] Carroll et al, "Solutions of M/G/1//N-type Loops with Extensions to M/G/1 and GI/M/1 Queues", *Operations Research*, Vol. 30, No. 3, p. 490

[C2] Chandy, K.M., Herzog, U., and Woo, L.S., "Parametric Analysis of Queueing Networks", *IBM J. Res. and Develop.*, Vol. 19, Jan. 1975, p. 43-49

[C3] Chandy, K.M. and Sauer, C.H., "Computational Algorithms for Product Form Queueing Networks", *Comm. ACM*, Vol. 23, 1980, p. 573-583

[C4] Cox, D.R., "A Use of Complex Probabilities in the Theory of Stochastic Processes", *Proc. Cambridge Phil. Soc.*, Vol. 51, p. 313

[C5] Courtois, P.J., Decomposability: Queueing and Computer System Applications, Academic Press, 1977

[D1] Doob, J.L., Stochastic Processes, Wiley, 1953

[F1] Flynn, M.J., "Very High Speed Computing Systems", *Proc. IEEE*, Vol. 54, No. 12, December 1966, p. 1901-1909

[H1] Halachmi, B. and Franta, W.R., "A Closed, Cyclic, Two Stage Multiprogrammed System Model and Its Diffusion Approximation Solution", Technical Report #74-18, Computer,

Information, and Control Sciences, University of Minnesota, July 1974

[H2] Herzog, U., Woo, L., and Chandy, K.M., "Solution of Queueing Problems by a Recursive Technique", *IBM J. Res. Develop.*, Vol. 19, No. 3, May 1975, p. 295-300

[H3] Hildebrand, F.B., Methods of Applied Mathematics, Prentice-Hall, 1965

[H4] Howard, R., Dynamic Programming and Markov Processes, MIT Press, 1960

[J1] Jaiswal, N.K., Priority Queues, Academic Press, 1968

[K1] Kelly, F.P., Reversibility and Stochastic Networks, Wiley, 1979

[K2] Khintchine, A.J., *Matematiceski Sbornick*, T40, 2, 1936

[K3] Kleinrock, L., Queueing Systems Volume I: Theory, Wiley, 1975

[L1] Lam, S.S., "Dynamic Scaling and Growth Behavior of Queueing Network Normalization Constants", *JACM*, Vol. 29, 1982, p. 492-513,

[L2] Lam, S.S., and Lien, Y.L., "A Tree Convolution Algorithm for the Solutions of Queueing Networks", *Comm. ACM*, Vol. 26, 1983, p. 203-215,

[M1] Merchant, S.S., "The Design and Performance Analysis of an Arbiter for a Multiprocessor Shared-Memory System", Report #LIDS-TH-1396, Laboratory for Information and Decision Systems, MIT, 1984

[M2] Molloy, M., "On the Integration of Delay and Throughput Measures in Distributed Processing Models", Ph.D. dissertation, Dept. of Computer Science, UCLA, 1981. Available as report #CSD-810921.

[N1] Neuts, M.F., Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach, John Hopkins University Press, Baltimore, 1981

[O1] Odoni, A., "Alternative Schemes for Investigating Markov Decision Processes", Technical Report 28, Operations Research Center, MIT, 1967

[O2] Odoni, A., "On Finding the Maximal Gain for Markov Decision Processes", *Operations Research*, Vol. 17, No. 5, 1969

[P1] Parzen, E., Modern Probability Theory, Wiley, 1960

[P2] Peterson, J.L., Petri Net Theory and the Modeling of Systems, Prentice-Hall, 1981

[P3] Price, T.G., "A Note on the Effect of the Central Processor Service Time Distribution in Processor Utilization in Multiprogrammed Computer Systems", *JACM*, Vol. 23, No. 2, April 1976, p. 342-346

[P4] "Proposed Microcomputer System 796 Bus Standard", *I.E.E.E. Computer Magazine*, Oct. 1980, p. 89-105

[R1] Raskin, L., "Performance Evaluation of Multiple Processor Systems", Ph.D. dissertation,

Dept. of Electrical Engineering, Carnegie-Mellon University

[R2] Reiser, M. and Kobayashi, H., "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms", *IBM J. Res. and Develop.*, Vol. 19, 1975, p. 283-294

[R3] Reiser, M. and Sauer, C.H., "Queueing Network Models: Methods of Solution and Their Program Implementation", in Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance, Chandy, K.M. and Yeh, R.T. editors, Prentice-Hall, 1978, p.115-167

[S1] Sauer, C.H. and Chandy, K.M., Computer Systems Performance Modeling, Prentice-Hall, 1981

[S2] Stidham Jr., Shaler, "A Last Word on $L = \lambda W$", *Operations Research*, Vol. 22, p. 417-421, 1974

[W1] Walrand, J., "A Probabilistic Look at Networks of Quasi-Reversible Queues", *IEEE Transactions on Information Theory*, Vol. 29, No. 6, November 1983

[W2] Whitt, Ward, "Open and Closed Models for Networks of Queues", *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 9, Nov. 1984

[W3] Wiley, R.P., "Performance Analysis of Stochastic Timed Petri Nets", Ph.D. dissertation, Dept. of Elec. Eng. and Comp. Science, MIT, 1985

OFFICIAL DISTRIBUTION LIST

Director Information Processing Techniques Office Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	2 Copies
Office of Naval Research 800 North Quincy Street Arlington, VA 22217 Attn: Dr. R. Grafton, Code 433	2 Copies
Director, Code 2627 Naval Research Laboratory Washington, DC 20375	6 Copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12 Copies
National Science Foundation Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 Copies
Dr. E.B. Royce, Code 38 Head, Research Department Naval Weapons Center China Lake, CA 93555	1 Copy
Dr. G. Hopper, USNR NAVDAC-OOH Department of the Navy Washington, DC 20374	1 Copy

END

9-87

DTIC