

**Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596**

**PERFORMANCE ANALYSIS OF DISTRIBUTED PROCESSING
SYNCHRONIZATION ALGORITHMS**

Robert Edman Felderman

**June 1991
CSD-910019**

UNIVERSITY OF CALIFORNIA

Los Angeles

**Performance Analysis of
Distributed Processing
Synchronization Algorithms**

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

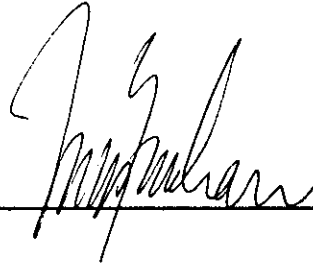
by

Robert Edman Felderman

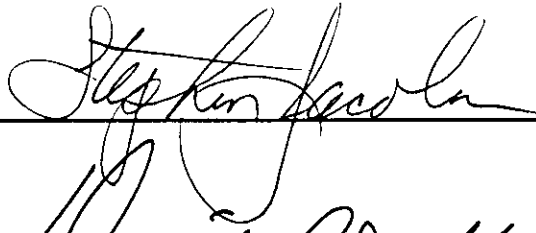
1991

© Copyright by
Robert Edman Felderman
1991

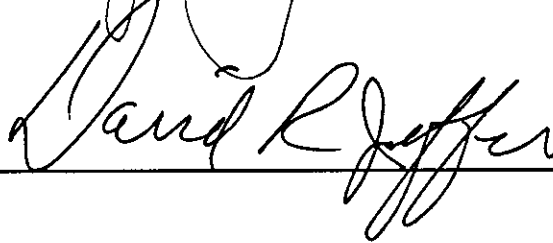
The dissertation of Robert Edman Felderman is approved.



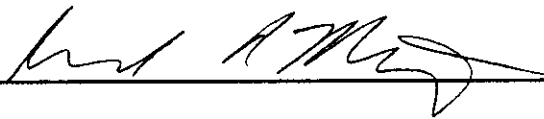
T. Chan



S. Jacobsen



D. Jefferson



R. Muntz



Leonard Kleinrock, Committee Chair

University of California, Los Angeles

1991

To my family,
for providing me with the love, support and freedom
to spend the past twenty-five years in school.

TABLE OF CONTENTS

1	Introduction	1
1.1	Discrete Event Simulation	1
1.2	Sequential Simulation	2
1.3	Parallel Discrete Event Simulation	3
1.3.1	Time-Stepped Simulation	5
1.3.2	Conservative Methods	6
1.3.3	Optimistic Strategies	9
1.4	Classes of Simulation Models	11
1.4.1	Message-Initiating Models	11
1.4.2	Self-Initiating Models	12
1.4.3	Hybrid Models	12
1.5	Previous Related Work	13
1.5.1	Empirical Studies	14
1.5.2	Analytical Work	15
1.6	Summary of Our Results	17
2	An Upper Bound on the Improvement of Asynchronous over Synchronous Distributed Processing	19
2.1	Introduction	19
2.2	The Models	19
2.2.1	Space of Synchronization Methods	22
2.3	Exponentially Distributed Task Times	22
2.3.1	Time-Stepped (Synchronous) Model	23
2.3.2	Time Warp (Asynchronous) Model	24
2.3.3	Relative Performance	28
2.4	Uniformly Distributed Task Times	32
2.5	Conclusions	34
3	Two Processor Time Warp Analysis: A Unifying Approach	35
3.1	Introduction	35
3.2	A Model for Time Warp on Two Processors	36
3.3	Discrete Time, Discrete State Analysis	39
3.3.1	Root Locus	47
3.4	Performance Measures	49
3.4.1	Speedup	50
3.5	Limiting Behavior	52
3.5.1	Continuous Time, Discrete State	53

3.5.2	Discrete Time, Continuous State	54
3.5.3	Continuous Time, Continuous State	55
3.6	Previous Work on 2-Processor Models	56
3.7	Results for a Restricted Model	60
3.7.1	Optimality Proofs	65
3.7.2	Adding a Cost for State Saving	67
3.8	Conclusions	70
4	Two Processor Message Queueing Model	71
4.1	Introduction	71
4.2	The Message Queueing Model	71
4.3	Analysis of the Message Queueing Model	75
4.4	Performance Measures	82
4.4.1	State Buffer Use	82
4.4.2	Message Queue Distribution	83
4.4.3	Normalized Rate of Progress	85
4.5	A Specific Example	88
4.5.1	State Probabilities and State Buffer Use	89
4.5.2	Message Queue Distribution and Buffer Use	91
4.6	Conclusions	92
5	Two Processor Model with Rollback and State Saving Costs	94
5.1	Introduction	94
5.2	The Rollback Cost Model	94
5.3	Analysis of the Cost Model	95
5.4	Performance Measures	101
5.4.1	State Buffer Use	101
5.4.2	Speedup	102
5.5	Conclusions	110
6	A Model for Conservative Simulation	111
6.1	Introduction	111
6.2	The Model	111
6.3	A System Without Null Messages	113
6.4	Lookahead	122
6.4.1	Types of Lookahead	122
6.5	The Lookahead Model	123
6.6	Comparison to Time Warp	125
6.7	Conclusions	129

7	Extensions of the Optimistic Model to Multiprocessors ($P > 2$)	133
7.1	Introduction	133
7.2	Definition of the Multiple Processor System	134
7.3	A Simple Upper Bound on Speedup	135
7.4	Tracking Global Virtual Time Advancement	137
7.5	A Simple Approximation Using Aggregation	138
7.6	Conclusions	139
8	Conclusions and Future Work	140
8.1	Future Work	141
8.1.1	Multiple Processes Per Processor	141
8.1.2	Communication Costs	142
8.1.3	Message-Initiating vs Self-Initiating	142
8.1.4	Optimistic Computation	143
8.1.5	Fault Tolerance of Time Warp	143
8.2	Final Remarks	144
A	Derivations of Summations for the Two Processor Model . .	145
A.1	$P(z)$ Sums Closed Form Derivation	145
A.2	Speedup Sums Closed Form Derivation	147
B	Cubic Equation Solution for the Message Queueing Model .	149
	References	151

LIST OF FIGURES

1.1	An example queueing network.	4
1.2	LP simulating a merge point.	7
1.3	A queueing network with potential deadlock.	8
2.1	A synchronous task graph.	21
2.2	An asynchronous task graph.	21
2.3	T_e versus P (log scale).	26
2.4	Regression slope and intercept values.	27
2.5	Comparison of approximation and simulation for $K, P \leq 10$	29
2.6	Comparison of approximation and simulation for $K, P \geq 100$	30
3.1	The states of two processors at times t_1 and t_2	37
3.2	State transition probability diagram for $\beta_1 = \beta_2 = 1$	41
3.3	Comparison of speedup results for a simplified case.	58
3.4	Previous work.	59
3.5	Speedup for the symmetric case $q_1 = q_2 = q$	62
3.6	Speedup for the balanced case $\lambda_1 = \lambda_2 = \lambda$	63
3.7	Speedup for the symmetric, balanced case $q_1 = q_2 = q$ and $\lambda_1 = \lambda_2 = \lambda$	64
3.8	The cost of state saving and its effect on performance.	69
4.1	Code executed by each processor.	72
4.2	State diagram for the message queueing model.	76
4.3	Normalized rate of progress (\hat{R}) versus f and q for the symmetric, balanced case.	87
4.4	\hat{R} versus q for the symmetric, balanced case.	88
4.5	State probabilities.	90
4.6	Distribution of the number of messages queued at each processor.	92
5.1	State diagram for the rollback cost model.	97
5.2	Speedup versus q and f for the symmetric, balanced case when $c = 1$	104
5.3	Region of $q - f$ space where speedup is possible.	105
5.4	State diagram when each processor stops at one step ahead.	106
5.5	Region of $q - f$ space where stopping at one step is better.	107

5.6	Achievable speedup for $c = 1$	108
6.1	State diagram for conservative synchronization with no null messages and a cost for breaking deadlocks.	115
6.2	Speedup versus a and q for various values of d	117
6.3	Derivative of speedup with respect to d (the cost of breaking a deadlock) versus q and d for $a = 1/2$	119
6.4	Speedup versus a and d for $q_1 = q_2 = 0$	120
6.5	Maximum conservative speedup (i.e. for the system with null messages).	121
6.6	State diagram for a system with K -step lookahead.	124
6.7	Speedup for a K -step lookahead conservative system.	126
6.8	Derivative of speedup with respect to K	127
6.9	Ratio of conservative speedup (no lookahead) to “free” Time Warp speedup.	128
6.10	Ratio of conservative speedup with K -step lookahead to “free” Time Warp with no lookahead.	130
6.11	Area of the $q - K$ plane where the conservative approach with lookahead wins out.	131
7.1	Normalized speedup versus K for 256 processors.	136
7.2	Percent difference between the upper bound and simulation.	137

ACKNOWLEDGMENTS

So many people have played a part in the completion of this dissertation.

First and foremost is Professor Leonard Kleinrock who saw something in me that I didn't even see in myself. His support on an intellectual and personal level has been truly outstanding.

Many thanks to David Jefferson who developed Time Warp, sparked my curiosity and took an interest in my research. Dick Muntz provided inspiration through his early work on Time Warp analysis and always could be counted on for a pointed question concerning aspects of my approach. To Tony Chan and Steve Jacobsen, I owe my thanks for serving on my committee and am only sorry I did not take advantage of their talents more fully. Gerald Estrin's comments and CS202 class greatly improved the clarity of my presentations.

The ATS/PSL research group has been a wonderful environment in which to work. Willard Korfhage, Jau Huang, Farid Mehovic, Eve Schooler, Rusti Baker and Joy Lin have gone on to "greener" pastures. They provided role models for me to follow. Special thanks goes to Willard for motivating the Benevolent Bandit Laboratory work. Current students Chris Ferguson, Shioupyn Shen, Simon Horng and Jon Lu have made the past several years very productive for me. They were always available when I needed some help with a tough problem. To Chris especially, I owe a great deal. Whether introducing me to a new video game, pulling out the playing cards to keep me sane during dry research spells, providing the solution to a problem (but never the proof!) or just putting up with the me as an office mate for six years, he was always available and supportive.

Without Lily Chien, this research group would degenerate into chaos. She does everything for us (and then some). I will miss her.

This research has been supported by the Advanced Research Projects Agency of the Department of Defense under contract MDA 903-82-C0064, Advanced Teleprocessing Systems, and contract MDA 903-87-C0663, Parallel Systems Laboratory. I thank the project managers for believing in what we've been doing.

Ultimaddicts (the various Ultimate Frisbee teams with which I have played) have helped keep me sane these past 11 years, while the intramural football, soccer, softball and (infamous) inner-tube water polo teams at UCLA provided welcome diversions from my studies.

Finally, I want to especially thank Eve Schooler for slaving away tirelessly with me on the Benevolent Bandit Laboratory Project and teaching me the wiles of C coding. On our joint papers and this dissertation, her comments and criticism have been invaluable. Lastly, she has made the past two years of my life very happy ones. May the rest of them be as wonderful.

VITA

- 1962 Born, Evanston, Illinois
- 1984 B.S.E. Magna Cum Laude, EECS Princeton University.
Elected to Tau Beta Pi and Sigma Xi
- 1984–1986 Member of Technical Staff, Hughes Aircraft Company, Buena
Park, California
- 1986 M.S., Computer Science, University of California, Los Angeles
- 1986–1991 Graduate Student Researcher for the Advanced Teleprocess-
ing Laboratory and Parallel Systems Laboratory contracts.
Computer Science Department, University of California, Los
Angeles
- 1991 Named one of three Outstanding Ph.D. students for 1990-1991
by the School of Engineering and Applied Science, University
of California, Los Angeles

PUBLICATIONS AND PRESENTATIONS

Robert E. Felderman. "Development of a Microcomputer based Controller for a Robotic System." Senior Thesis, Princeton University, June 1984.

Robert E. Felderman. "Flocks of Birds as a Paradigm for Distributed Systems." Master's thesis, University of California, Los Angeles, Computer Science Department UCLA Los Angeles, CA 90024-1596, June 1986.

R. E. Felderman. "Extension to the Rude-CSMA Analysis." *IEEE Transactions on Communications*, COM-35(8):848–849, August 1987. Correspondence.

Robert E. Felderman and D. Karen Beard. "Silicon Sigmund: An Expert System for Psychological Diagnosis." In *Proceedings of the 13th Western Educational*

Computing Conference, pp. 51–55. California Educational Computing Consortium, November 1989.

Robert E. Felderman and Leonard Kleinrock. “An Upper Bound on the Improvement of Asynchronous Versus Synchronous Distributed Processing.” In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pp. 131–136. Society for Computer Simulation, January 1990.

Robert E. Felderman and Leonard Kleinrock. “Two Processor Time Warp Analysis: Some Results on a Unifying Approach.” In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, volume 23,1, pp. 3–10. Society for Computer Simulation, January 1991.

Robert E. Felderman, Eve M. Schooler, and Leonard Kleinrock. “The Benevolent Bandit Laboratory: A Testbed for Distributed Algorithms.” *IEEE Journal on Selected Areas in Communications*, 7(2):303–311, February 1989.

Leonard Kleinrock and Robert E. Felderman. “Two Processor Time Warp Analysis: A Unifying Approach.” *International Journal of Computer Simulation*, To appear late 1991.

Eve M. Schooler and Robert E. Felderman. “The Benevolent Bandit Laboratory User’s Manual.” Technical Report 880017, UCLA Computer Science Department, March 1988.

Eve M. Schooler, Robert E. Felderman, and Leonard Kleinrock. “The Benevolent Bandit Laboratory: A Testbed for Distributed Algorithms Using PCs on an Ethernet.” Technical Report 880016, UCLA Computer Science Department, March 1988.

ABSTRACT OF THE DISSERTATION

**Performance Analysis of
Distributed Processing
Synchronization Algorithms**

by

Robert Edman Felderman

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1991

Professor Leonard Kleinrock, Chair

We analytically evaluate the performance of distributed simulation synchronization algorithms, focusing mainly on the optimistic protocol, Time Warp. We first provide an upper bound on the expected improvement of Time Warp over time-stepped simulation and show, for a particular system model, that Time Warp is only able to outperform time-stepped simulation by a factor of $(\ln P)$, where P is the number of processors used by each method. A model for two processor Time Warp operation is then developed. Closed-form expressions are derived for several interesting performance metrics including speedup, the distribution of virtual time separation between the processes and the average number of state buffers used. This model unifies previous work on two processor Time Warp analysis and provides further insight into the operation of systems synchronized by rollback. The model is generalized to include costs for message queueing, rollback and state saving while continuing to provide closed-form expressions for the performance measures. We then explore a simple model for conservative simulation on two processors and quantify the improvement in speedup by sending null messages and exploiting lookahead. We evaluate the

degradation due to costs for detecting and breaking deadlocks and also compare the conservative model with the optimistic models developed earlier. Finally, we address the issue of multiprocessor Time Warp by discussing techniques to evaluate the performance of the algorithm as the system scales to a large number of processors ($P > 2$).

CHAPTER 1

Introduction

1.1 Discrete Event Simulation

The systems that we are able to create become larger and more complex every day. In many cases, we have moved beyond a point where one is able to predict the performance of a large system, be it a high-speed computer network or a super-sonic airplane, by purely analytical means. It is now often necessary to **simulate** the operation of a proposed system in order to better understand its behavior. Additionally, simulation is a useful tool to examine complex existing systems such as global weather patterns or the world economy. As the size of these systems increases, the simulations demand more computing power. Naturally then, one would like to utilize the recent advances in parallel computing technology to speed up the execution of simulations. Unfortunately, it is a non-trivial task to efficiently implement a parallel simulation system, though several techniques have been developed to do so.

This dissertation examines the performance of several different algorithms used to synchronize distributed discrete event simulations. Our major focus is on the optimistic methods, though we consider conservative approaches as well. In the remainder of this chapter we introduce the various simulation algorithms that have been proposed and/or implemented, so that the reader will be better

able to understand the analysis that follows.

1.2 Sequential Simulation

In order to understand parallel simulation, we must first discuss sequential discrete event simulation techniques. Discrete event simulation (DES) allows the simulation time to advance in arbitrary increments as the system simulates events in increasing order (an example follows below). When the system completes the processing associated with an event, the clock is advanced to the occurrence time of the next event. There is no need to let the clock advance in smaller increments, since nothing will happen in the simulated system between the time of the previous event and the time of the next event. DES is not the only technique for simulation, some systems (e.g. those characterized by differential equations) are more naturally simulated by a continuous-time simulator [KW78]. However, we only concern ourselves with discrete event simulation algorithms in this work.

The basic method of sequential simulation centers around the “event list”. Events are scheduled by placing them in the event list. The simulator proceeds by taking the event with the smallest time off the the event list, incrementing the simulation clock to the time of this event, and executing it. This execution may generate new events in the future, and these new events are placed (in the proper time order) into the event list. A simple example is a single-server queuing system. Customers arrive to the system, are serviced, and leave the system. Typical events for this scenario are: the arrival of a customer and the departure of a customer. If the queue is empty, the arrival of a customer will generate two new events for the event list: the departure of that customer and

the arrival of the next customer. The simulated time of the departure event will be the arrival time plus the service time which is drawn from the distribution of interest. If the queue isn't empty then the execution of the arrival event merely schedules the next arrival event and adds this customer to the queue. A departure event removes a customer from the server and places the customer at the head of the queue into service by scheduling its departure event.

1.3 Parallel Discrete Event Simulation

The sequential nature of the event list precludes a direct parallel implementation. Parallel DES is generally accomplished by partitioning the simulation into logical processes (LP) which simulate some physical process in the system. Each process interacts with other processes by sending and receiving messages. Using our queueing example above, we could partition the system into three physical processes: the arrival generator, the queue/server and the departure "sink". The arrival process generates customer arrivals to the queue by sending messages stamped with the arrival time at the queue and the number of customers if we have bulk arrivals. The queue/server receives customer messages, queues the customers, and services them by sending a message to the departure sink stamped with the time that the customer left the server (and arrived at the sink). The departure "sink" is used to collect any necessary information as the job leaves the system. Each process operates autonomously by receiving messages, performing internal computation and sending messages. Each process terminates once its local clock, the time of receipt of the message currently being processed, has reached T_{max} , the total time of the simulation (a user specified duration).

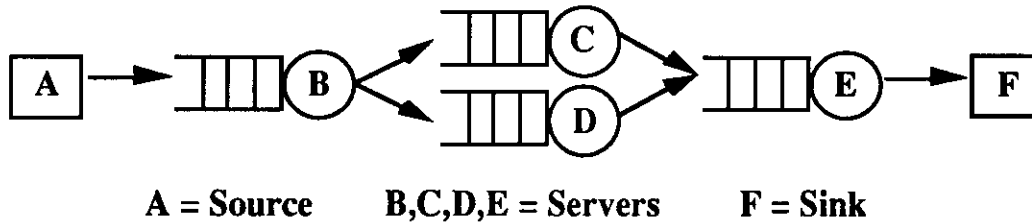


Figure 1.1: An example queueing network.

For clarification, we present a simple queueing network shown in Figure 1.1. The logical processes in this system are the customer arrival process (A), the queueing stations (B,C,D,E) and a final sink process (F) that collects departing customers. Logical process A (LP_A) connects to LP_B which is in turn connected to LP_C and LP_D etc. Every path which can be traversed by a customer in the physical system must correspond to a logical communication path in the simulation system. Messages passed between LPs in our queueing example are the actual customers flowing through the system.

Ideally, each logical process can be placed on its own processor, and we can gain speedup proportional to the number of processors used. Unfortunately, this is often not the case. Though not obvious from the simplistic description above, some controls are necessary to maintain causality between events. If an event A directly (or indirectly) affects the outcome of event B , then event A **must** be executed before event B . If these two events are located on different processors, then some communication/synchronization must take place so that the proper ordering is maintained. If blocking is used for synchronization, then care must be taken to prevent the simulation from deadlocking. There are basically three strategies used in the simulation community to combat this problem. The

first technique detects and breaks deadlocks. The other two prevent deadlocks from occurring through various methods. One of these methods is based on the knowledge of how deadlocks occur and prevents the deadlock from happening. The second prevention technique avoids blocking by using a rollback mechanism to repair causality violations. Algorithms which use the rollback mechanism are generally called “optimistic” strategies, while the more traditional methods of keeping logical process clocks in near synchronization are referred to as “conservative” strategies. Each of these techniques is described in more detail in the following sections.

1.3.1 Time-Stepped Simulation

Though more often used as a technique for simulating continuous-time systems, distributed time-stepped simulation [PWM79] can be used for discrete event simulation by keeping all the local clocks in strict synchronization (strictly speaking this is not DES since local clocks do not jump from event to event). At any point in real time each LP’s local clock has the same value as any other LP’s clock. As the simulation runs, the local clocks take on a sequence of discrete values (t_0, t_1, t_2, \dots) each differing by an amount Δ . The choice of an appropriate Δ is a non-trivial task. It must be chosen small enough such that causal events are executed in different time steps. All processors must complete execution of events up to t_i before any processor begins processing at t_{i+1} . Since each processor may have a different amount of work to do at each time step or some may operate at different speeds, many processors may have to wait for the slowest one to complete execution of the i^{th} step, thus degrading speedup. Also, if the LPs don’t have events to process at every t_i , then this

algorithm might produce little speedup since many processors might be idle during any given step. Time-stepped simulation is attractive due to its simplicity of implementation. By keeping all the LPs processing at the same simulation time, deadlocks cannot occur and no further effort needs to be expended to guarantee the correctness of the simulation.

1.3.2 Conservative Methods

Conservative methods of DES are based on the work of Chandy, Misra, Bryant and others [CM79] [CHM79] [Bry77]. The best survey of this area can be found in [Mis86]. As mentioned earlier, the system is partitioned into logical processes and a static communication network between the LPs is defined such that if the physical processes being simulated by the LPs need to communicate, the LPs have a communication link between them. A time stamp on each message indicates the customer's arrival at a particular process in the system. Each LP has a local clock, and a clock associated with each of its incoming and outgoing links. A process is only allowed to send messages on a link in strictly increasing order of timestamps. Therefore, when an LP receives a message with timestamp v on one of its incoming links, it knows that it can never receive a message on that link with a smaller timestamp than v . In the case where the LP only has one incoming link, it can immediately advance its clock to this time v , process the message (customer arrival) and possibly send messages on its outgoing links. The time associated with each link is the time of the most recently sent/received message. It indicates that no message can be sent/received on this link with a time lower than the clock of the link.

Logical processes advance their local clocks as far as the times on their

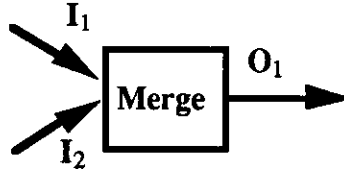


Figure 1.2: LP simulating a merge point.

input links allows them to. For example, consider an LP which is simulating a merge point (Figure 1.2) in our queueing network. A merge point simply forwards messages from its input links to its output link. Messages must go out in strictly increasing timestamp order. Assume it has two input links (I_1, I_2) and one output link (O_1). Initially it sets its local clock and all link clocks to zero. Further, let's assume that it receives a message M on I_1 which is a single customer arrival at time 10. It would like to forward this message on O_1 . Unfortunately, it cannot do so immediately. The problem is that the timestamp on I_2 , the other outgoing link is still zero. It is possible that a message with timestamp less than 10 might arrive on I_2 . Since messages on the outgoing link must be sent in strictly increasing timestamp order, this LP must wait until it knows it cannot receive a message with a timestamp smaller than 10 before it can forward message M .

This example shows the essential problem with the conservative approach. The problem can manifest itself in two ways. The first (as we saw) is that parallelism is limited. Another more problematic result is the possibility of deadlock. A simple example is shown in Figure 1.3. Suppose LP_A is waiting for some information from its top input link before it can proceed. This would be due to the fact that LP_B was unable to send it any messages because it

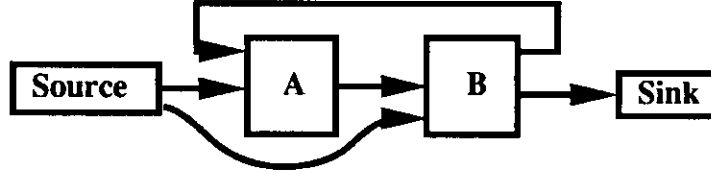


Figure 1.3: A queuing network with potential deadlock.

was waiting for input from either LP_A or the source LP. If it is waiting for LP_A we have a deadlock. A great deal of research has addressed this aspect of conservative simulation [PWM79] [CM81] [CHM79] [Mis86] and there are essentially two directions to pursue. One is deadlock avoidance, the other is detection and correction.

One method of avoiding deadlock is to introduce “null” messages. These are messages which only exist in the distributed simulation to avoid deadlocks. A null message is treated as a regular message with respect to the local LP clocks and the link clocks. A null message has a timestamp which is used to communicate the fact that no messages with an earlier timestamp will traverse a given link. The null message is an artifact of the distributed simulation and wouldn’t appear in a sequential simulation or in the real system. When sending a “real” message on a given link, an LP also sends null messages with the same timestamp on all other outgoing links. The formal proof that this algorithm avoids deadlocks can be found in [CM79]. Unfortunately, this scheme generates a great deal of additional overhead traffic due to the null messages [See79]. Deadlocks may be infrequent, so it may be extremely wasteful to send null messages constantly. Therefore, some researchers propose to detect deadlocks and correct them once detected [CM81]. Chandy and Misra describe a technique

based on Dijkstra and Scholten's work on termination detection [CM80]. Once the deadlock has been detected, each LP_i calculates the minimum timestamp of outgoing messages that it wants to send, under the assumption that it will not receive any further messages. Call this time v_i^{min} . Each LP can calculate its v_i^{min} independently. A distributed algorithm is then used to find the minimum over all the processors (i) of the v_i^{min} values. The node with the minimum v_i^{min} is allowed to send this message since no other LP will send it a message with a lower timestamp. By allowing this "min" LP to proceed, we break the deadlock. The authors discuss a modification to this algorithm which takes into account potential causality to allow more than just the "min" LP to resume after a deadlock [CM81].

An additional problem with the conservative strategy is that the interconnection between the LPs is static and fixed throughout the run. Ideally we'd like to dynamically create LPs and links throughout a simulation.

1.3.3 Optimistic Strategies

One of the more recent developments in the area of DES are the so-called optimistic strategies which are based on a protocol called Time Warp (TW) developed by Jefferson [Jef85]. The basic idea is that the restriction of strictly increasing timestamps on messages sent over a link is restrictive and leads to too many problems (deadlock and unnecessary waiting). The Time Warp mechanism allows LPs to process messages as they are received, though if more than one message is in the message queue, the one with the minimum timestamp is processed first. If a message arrives which has a lower timestamp than the value of the LP's clock, the LP is "rolled back" to the time of this message.

This is able to be accomplished because the system periodically saves the state of the LP. Any effects of having advanced too far (i.e. erroneous messages) are canceled through an elegant technique using “anti-messages”.

When a process is rolled back from virtual time v_1 to virtual time v_0 , all messages sent between v_0 and v_1 may be in error. They were produced without knowledge contained in a message (the one that caused the rollback) that could have had a causal effect on them. Therefore the effects of sending these (potentially incorrect) messages must be nullified. This is accomplished by sending an anti-message for each message sent between v_0 and v_1 . Anti-messages act in the following manner. If an anti-message is placed in a message queue with its positive partner, both messages are annihilated. If it arrives at the receiving LP and doesn't find its partner in the message queue, there are two possible scenarios. One possibility is that it beat its positive partner to the receiving LP. In this case the negative message will be in the message queue when the positive message arrives, thus deleting it and any potential effects. The other possibility is that the positive message has already been processed. In this case the receiving LP's clock will have a higher timestamp than the arriving anti-message. Therefore, the LP will be rolled back and may send its own set of anti-messages. It has been shown that this “cascading rollback” is bounded and that the system will make progress [Jef85].

The optimistic approach essentially gambles on a time/space tradeoff. By using extra space (memory storage) for saving state and messages, it hopes to reduce the total time it takes to complete a simulation. The conservative approaches use blocking, while Time Warp uses space (state saving) then rollback for synchronization.

One of the problems of the Time Warp mechanism is the overhead associated with state saving. In addition to states, every message received and every message sent must be saved by a process. Fortunately, we do not need to keep all state information back to the beginning of the run. A concept called Global Virtual Time (GVT) allows the system to periodically throw away obsolete information. GVT is defined as the minimum of all the local LP clocks and the timestamps of all messages in transit. Nothing in the system has a virtual time less than GVT, and the system can discard all state information with timestamps earlier than GVT. Obviously GVT is a very difficult quantity to obtain, since we cannot take a “global” snapshot of this distributed system [Lam78]. Algorithms have been developed to calculate a lower bound on GVT which can be used as an estimate to free up memory space [Bel90].

One of the benefits that Time Warp has over the conservative methods is its lack of insistence on a static interconnection network. Any LP can send a message to or receive a message from any other LP at any time during the simulation. This should prove to be very useful when attempting to perform load balancing or in a simulation where processes are created and destroyed.

1.4 Classes of Simulation Models

1.4.1 Message-Initiating Models

The queueing systems discussed above came from the general class of “message-initiating” models. A message-initiating model is one in which a logical process performs no work unless it receives a message from another logical process. The best examples are the aforementioned queueing networks where each

server/queue has nothing to do until the arrival of a message from another server. The messages in the simulation correspond to the customers in the queueing system. Messages carry the work in this class of systems. Generally the system starts with some messages “pre-placed” in some queues. The system progresses by processing these messages and generating new ones.

1.4.2 Self-Initiating Models

Another type of simulation is the “self-initiating” model [Nic91]. This system is one where each logical process performs work regardless of whether it has received any messages from other logical processes. Here, we find that messages do not carry work, rather they merely provide some sort of state information. The example used in [Nic91] is the Ising Spin model [Lub87]. In this system, each logical process models a particle which randomly and independently decides to modify its state (say at simulated time v). Its new state is a function of the states of neighboring particles at time v . Messages passed between LPs convey state information, they do not cause the processor to re-evaluate its state.

1.4.3 Hybrid Models

Some systems contain elements of both message-initiating and self-initiating models. A good example of this is a trace-driven multiprocessor cache simulation [Nic91] [LLB89]. Each logical process simulates operations (reads and writes) of one physical processor’s cache and each proceeds independently of the other LPs. A message is sent to other logical processes whenever a reference is made to global memory. The arrival of a message at a logical processor does not cause reads or writes, rather the state of the cache might have to be up-

dated (invalidation of entries). In this system, processors can proceed without the receipt of any messages, though when messages arrive the LP must perform some work.

This differentiation between models becomes important when discussing analytical work.

1.5 Previous Related Work

Our work focuses on the performance evaluation of the various synchronization algorithms described earlier, though our major focus is on the analysis of the optimistic protocol Time Warp. Most of the analysis of these simulation methods has been empirical rather than analytical. Unfortunately, the performance of either the conservative or optimistic strategies is highly data dependent. Some of the systems that are simulated have a great deal of built-in parallelism, while others do not. Therefore, one must be very careful when citing any performance measure to recognize that the performance (i.e. speedup or lack thereof) may not be due to the method of simulation, but rather to the system being simulated [WL89]. Another problem which may affect performance is the partitioning of the physical processes into logical processes and the assignment of these LPs to the parallel processors. The allocation of these tasks can have a large impact on the performance. A load imbalance between the processors can severely degrade performance. Also, LPs that communicate often, should be placed on the same processor, so that messages need not be sent over a communication network.

1.5.1 Empirical Studies

There have been a large number of papers published giving performance results for a specific application when using one particular synchronization algorithm, either optimistic or conservative [UJ88] [UF89] [Nic90b] [MNF91]. These papers generally show modest speedups over sequential simulation regardless of the underlying synchronization algorithm. Neither conservative nor optimistic strategies seem to be dominant over all application domains.

In contrast to studies that evaluate performance under a real application, Fujimoto has examined a variety synchronization strategies in [Fuj88a], [Fuj88b], [Fuj89a], [Fuj89b] through the use of an artificial application. Instead of using a real system to simulate, he controls various parameters of the LPs and examines the impact on performance. For the conservative approaches the following general principles were derived. Lookahead, the ability of a logical process to predict future behavior, is extremely important in gaining speedup. When lookahead is too small, parallelism is limited, and processes are forced to block more often. Message population, the number of messages or events circulating through the LP network, also regulates speedup. Not surprisingly, when lookahead is small, the message population must be large in order to gain speedup. Conversely, when lookahead is large, the message population may be small yet still provide good speedup. Finally, the deadlock avoidance approach (null messages) seems to be more robust in gaining speedup over a large class of problems, than the deadlock detection and recovery technique.

For Time Warp, lookahead is useful, but not necessary. This is an important point in favor of the optimistic approach. Conservative techniques generally **must** exploit lookahead, while Time Warp performs reasonably well without it.

Event computation granularity is important in its relation to the cost for state saving and rollback. If events are “cheap”, but rollback and state saving are expensive, then the overhead of Time Warp will keep speedup low. In order for TW to succeed in gaining high speedups, state saving/restoration and rollback costs must be kept to a minimum. Fujimoto suggests hardware support to reduce these costs to a negligible level [FTG88].

1.5.2 Analytical Work

Our major interest is in the area of performance *analysis*, not empirical studies. Very little work has appeared in the literature which discusses the average case behavior of TW. Lavenberg, Muntz and Samadi [LMS83] examined a model for two processor TW where messages are only used for synchronization (self-initiating). They developed an approximation for speedup which was valid if the interaction between the processors was small. Mitra and Mitrani [MM84] also examined a self-initiating two processor model and developed an exact formula for the distribution of separation in virtual time between the two processors and for the rate of progress in simulated time per unit real time of the two processor system. We address the relationship of these two studies to our work in more detail in Chapter 3. Madisetti [MWM90] [Mad89] provides bounds on the performance of a two processor self-initiating system where the processors must have different speeds of processing and move at constant (deterministic) rates. Madisetti extends his model to multiple processors, something we do not address until Chapter 7.

Jefferson and Witkowski [JW84] describe a Linear Poisson Process as a model of the arrival process of messages to a TW queue. In this work the

authors view a single LP in isolation by aggregating the effects of all the other LPs into two random processes, a real time message arrival process and the virtual timestamp of these messages. The authors use this technique to model the performance of the Time Warp database concurrency mechanism [JM84] (which is different from the simulation method) and to find a lower bound on the time it will take to process a message. They essentially find the time after which no message with a smaller timestamp will arrive to preempt a message with virtual time v .

Lin and Lazowska [LL90a] have examined Time Warp and conservative methods by using critical path analysis. They have also examined TW itself [LL89] [LL90b] [LL90c] to better understand the state saving overhead, roll-back mechanisms and processor scheduling when running the TW algorithm. Though their work provides important insights, it generates different types of results than ours. Our results are exact, detailed performance measures of two processor systems while Lin and Lazowska have opted to concentrate on higher-level views of more complicated systems.

Nicol [Nic91] has provided bounds on the performance of multiprocessor self-initiating models and introduces a new conservative synchronization algorithm. Nicol also examined the cost of conservative synchronization [Nic90a] and showed that, as the problem size scales, performance closes to within a constant factor of optimal for a particular conservative algorithm. Some excellent work solving for the performance of Time Warp with multiple, homogeneous, message-initiating processors has recently appeared [GAF91]. The authors use a Markov chain approach where the state variable is the number of executed events beyond GVT at a processor. An approximate solution is obtained which

In Chapter 3 we present a new model for two processors running the Time Warp protocol. The model unifies previous work in this area and provides more details about the performance tradeoffs involved in a Time Warp system. As in [Kie89], our technique involves solving for the separation in virtual time of the two processors. We find that the probability the processors are separated by a distance k (in virtual time) is geometrically distributed. These probabilities are then used to calculate exact formulas for various performance measures including speedup. Speedup is strongly affected by the balance of the load between the two processors and on the interaction probability.

This dissertation provides an understanding of some of the distributed simulation algorithms described earlier. In Chapter 2 we find an upper bound on the maximum performance improvement that an optimistic approach could have over a time-stepped technique under certain assumptions. We find that the optimistic approach is able, at most, to outperform the time-stepped approach by a factor of $(\ln P)$, where P is the number of processors used for each method. The above result is obtained when task execution times are exponentially distributed. When these times are uniformly distributed, the upper bound is independent of the number of processors.

1.6 Summary of Our Results

Finally, a limited version of our two processor work can be found in [Kie89]. Papers derived from this dissertation may be found in [FK90b] [FK91] [KF91] [FK90a].

appears to be quite accurate over a wide range of parameters.

Unsatisfied with the simplicity of this model, we extend it in Chapter 4 to include costs for message processing and queuing and in Chapter 5 to include costs for rollback. In both cases we find exact closed form expressions for the separation in virtual time between the processors and other performance parameters of interest.

Chapter 6 introduces a comparable model for a two processor conservative system. We examine the operation of the conservative protocol in isolation and also compare it to the Time Warp models developed in the previous chapters. We are able to show that lookahead is very useful in improving performance, but only if the processors are well balanced in processing capacity. The models allow us to quantitatively evaluate the improvement attributable to null messages and the degradation due to a cost for breaking deadlocks. Finally, we show that a conservative system with "free" null messages and a small amount of lookahead is able to outperform a Time Warp system with no cost for state saving or rollback, but no lookahead.

In Chapter 7 we examine some directions for future work in extending our analysis to the performance of Time Warp when operating on more than two processors. Chapter 8 provides a summary and conclusions with some further thoughts on future work in distributed simulation.

We have opted to use very simple models of the two approaches in order to assess the potential improvement of the asynchronous (Time Warp) over the synchronous (time-stepped) strategy. Our model of the time-stepped strategy will provide us with an accurate estimate of its time to finish executing a simulation. The model for the asynchronous strategy will provide us with an overly low

2.2 The Models

As mentioned previously, there are several algorithms used for Parallel Discrete Event Simulation (PDES), and this chapter demonstrates the potential improvement by using an asynchronous approach (e.g., Time Warp), over a synchronous technique, (e.g., time-stepped simulation). There is a fair amount of operational overhead in a Time Warp system and it is complicated to build such a simulator. Therefore, we would first like to estimate the potential payback for all of the effort expended to implement and operate a Time Warp system.

2.1 Introduction

An Upper Bound on the Improvement of Asynchronous over Synchronous Distributed Processing

CHAPTER 2

The asynchronous strategy has no such “staging” restriction, and moreover, under the best possible circumstances, no rollbacks will occur. We allow each processor to execute its tasks in order as fast as it can, without waiting for the other processors to finish. The total time to finish is simply the time that the slowest processor takes to complete its K tasks. To keep the model simple we are assuming no rollbacks; it is as if each processor never has to wait for any messages from other processors, and that all messages arrive in timestamp order. The asynchronous task graph is shown in Figure 2.2.

$$P = 7.$$

Our model of the synchronous approach is based on the idea that an LP must wait for all other LPs to complete a step before continuing. That is, each processor must wait until *every* processor has completed its i^{th} task prior to beginning execution of its $(i + 1)^{\text{st}}$ task. This is essentially a “staged” execution with K stages where each stage takes as long as the slowest processor. A task graph for this type of synchronization is shown in Figure 2.1 for $K = 4$ and

amount of time to complete execution on any processor.

To analyze the two techniques, we propose the following model: KP tasks are executed such that P processors each execute K of the tasks (events) sequentially. Each processor p must perform tasks $T_{p1} \dots T_{pk}$ in sequential order. K determines the “size” of the simulation. A task will take a random amount of time to complete execution on any processor.

method.

the potential improvement of the asynchronous strategy over the time-stepped estimate of its expected completion time. Thus, we establish an upper bound on

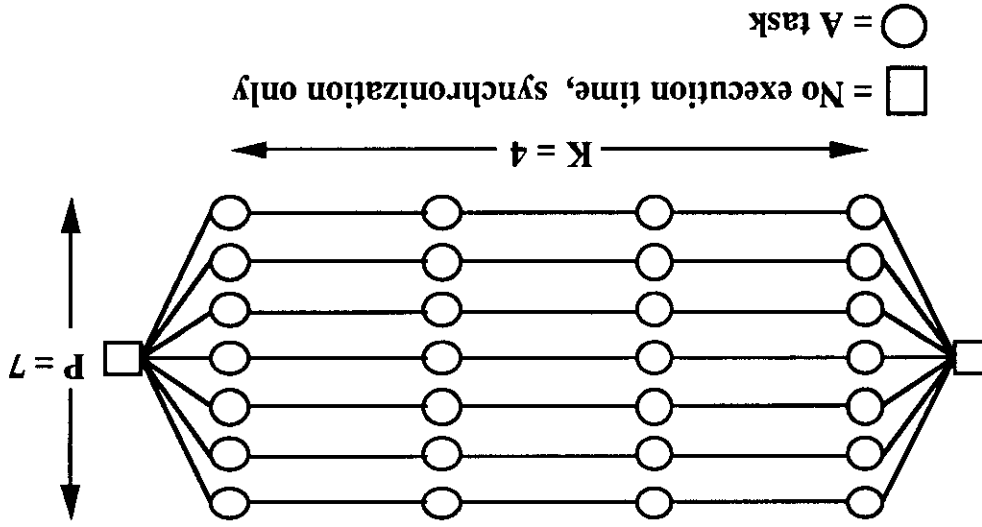


Figure 2.2: An asynchronous task graph.

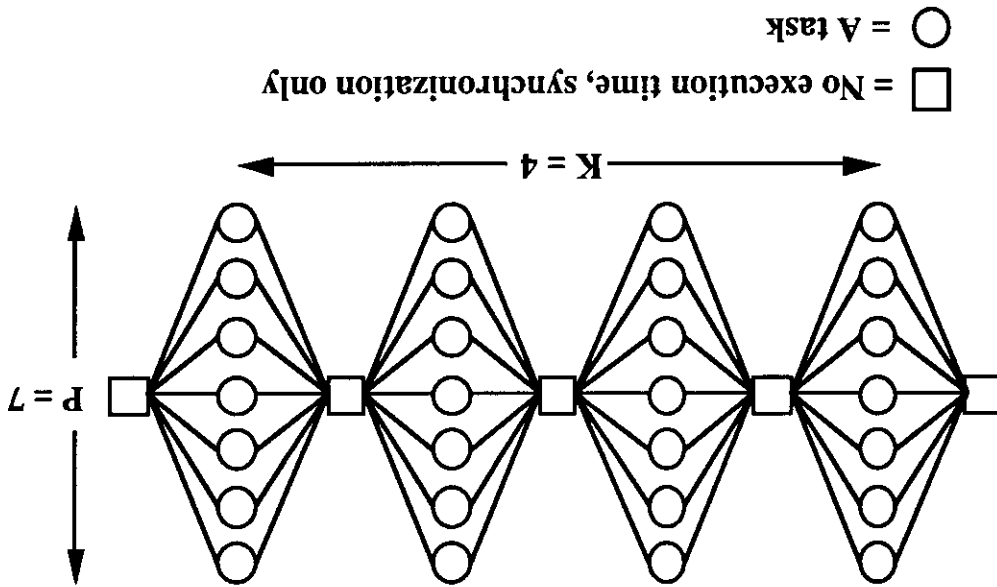


Figure 2.1: A synchronous task graph.

If we model task execution times with an exponential distribution and treat the processors as identical, then for the synchronous strategy, each stage will take time equal to the maximum of P exponentials. This is because each processor has to wait for all the other processors to complete. Therefore, the expected completion time for the synchronous strategy is K times the maximum of P exponentials. In contrast, the asynchronous strategy only has processors wait at the end of completing all K tasks. Therefore, the expected time for the asynchronous strategy is the maximum of P K -stage Erlangs [Kle75]. A K -stage Erlang random variable is the sum of K exponential random variables. We now proceed to calculate the ratio R_e (for exponentially distributed task times) of the expected completion times for the synchronous to the asynchronous case.

2.3 Exponentially Distributed Task Times

Though the models we are using are extremely simple, we believe they provide us with very important information. Our time-stepped model (Figure 2.1) requires the most synchronization, and therefore will take the longest time to complete execution of any system which exhibits full parallelism in each stage. Our asynchronous model (Figure 2.2) shows the least amount of internal synchronization, basically none, and should complete execution in less time than any other method. Therefore, we believe that these two models span the range of possibilities and give a good indication of the maximum performance improvement that could be gained by using the asynchronous strategy.

2.2.1 Space of Synchronization Methods

$$E[T_s] = \frac{h}{K} \sum_{i=1}^K \frac{1}{i} \quad (2.4)$$

K is the number of steps is

$KE[T]$. So the final equation for $E[T_s]$ where P is the number of processors and We now define T_s as the time for all K stages to complete. Clearly, $E[T_s] =$

$$E[T] = \frac{h}{1} \sum_{i=1}^K \frac{1}{i} \quad (2.3)$$

we find that

$$\sum_{i=1}^K \frac{1}{i} = \sum_{i=1}^K \left(\frac{1}{i} - \frac{1}{i+1} \right)$$

Since [GKP89]

$$\begin{aligned} E[T] &= \int_0^\infty (1 - F_T(x)) dx \\ &= \int_0^\infty \left[1 - \left(1 - e^{-hx} \right)^P \right] dx \\ &= \int_0^\infty \left[1 - \sum_{i=0}^{P-1} \binom{P-1}{i} (-1)^i e^{-hx} \right] dx \\ &= \sum_{i=0}^{P-1} \binom{P-1}{i} (-1)^i \int_0^\infty e^{-hx} dx \end{aligned}$$

Using the PDF we can calculate the expected value of T [Kle75].

$$f_T(x) = P(1 - e^{-hx})^{P-1} h e^{-hx} \quad (2.2)$$

with density function

$$F_T(x) = (1 - e^{-hx})^P \quad (2.1)$$

distribution (PDF) of T is

Let $T =$ the maximum of P exponentials, each with mean $\frac{1}{h}$. The cumulative

2.3.1 Time-Stepped (Synchronous) Model

$$E[T_a] = \int_0^\infty \left(1 - e^{-\mu x} \sum_{i=0}^{K-1} \frac{(\mu x)^i}{i!}\right) dx$$

Thus,

$$E[T_a] = \int_0^\infty [1 - F_{T_a}(x)] dx$$

Using $F_{T_a}(x)$ we can calculate the expectation of T_a .

$$F_{T_a}(x) = [F_i(x)]_P = \left(1 - e^{-\mu x} \sum_{i=0}^{K-1} \frac{(\mu x)^i}{i!}\right) \quad (2.8)$$

The cumulative distribution of the maximum of P K -stage Erlangs is

$$F_i(x) = \int_x^\infty f_i(x) dx = 1 - e^{-\mu x} \sum_{i=0}^{K-1} \frac{(\mu x)^i}{i!} \quad (2.7)$$

interval $[0 - t]$ from a Poisson process at rate μ . Therefore is simply one minus the probability that there are less than K arrivals in the equal to t is one minus the probability that it takes time greater than t , which by realizing that the probability that a K -stage Erlang takes time less than or The PDF can be found either by direct integration of the density function, or

$$f_i(x) = \frac{\mu^K (K-1)!}{\mu e^{-\mu x} (K-1)!} \quad (2.6)$$

$1/\mu$. The probability density function of a single K -stage Erlang is We define T_a as the maximum of P K -stage Erlangs where each stage has mean

2.3.2 Time Warp (Asynchronous) Model

where $E = \text{Euler's Constant} \approx 0.57722$.

$$E[T_a] \approx \frac{\mu}{K} \left(E + \ln P + \frac{1}{2P} - \frac{1}{12} \frac{1}{P(P+1)} \right) \quad (2.5)$$

An excellent approximation for this is [Jol61]

This approximation was developed by using least squares regression techniques three times. It was first noticed that for a fixed K , T_e seemed to be directly related to $\ln P$. This is clearly seen in Figure 2.3. For each value of $k \leq K$ we performed a linear regression for T_e such that $(T_e)^k = m_k(\ln P) + b_k$, thus generating K slopes and intercepts, one set for each value of k . Then, it appeared that the slopes for each k approximation were linearly related to $\ln^2 k$, while the intercepts seemed linearly related to k . This can be seen in Figure 2.4. Therefore, a second regression was performed on the slope values versus $\ln^2 k$ (which generates the values for the constants C and D), while a third regression was performed on the intercept values versus k (which generates the values

$$A = 0.02244 \approx 0.02 \quad B = 1.14571 \approx 1.15$$

$$C = 0.22278 \approx 0.22 \quad D = 0.55957 \approx 0.56.$$

Where

$$T_e \approx \frac{1}{P} \left((C \ln^2 K + D) \ln P + AK + B \right) \quad (2.9)$$

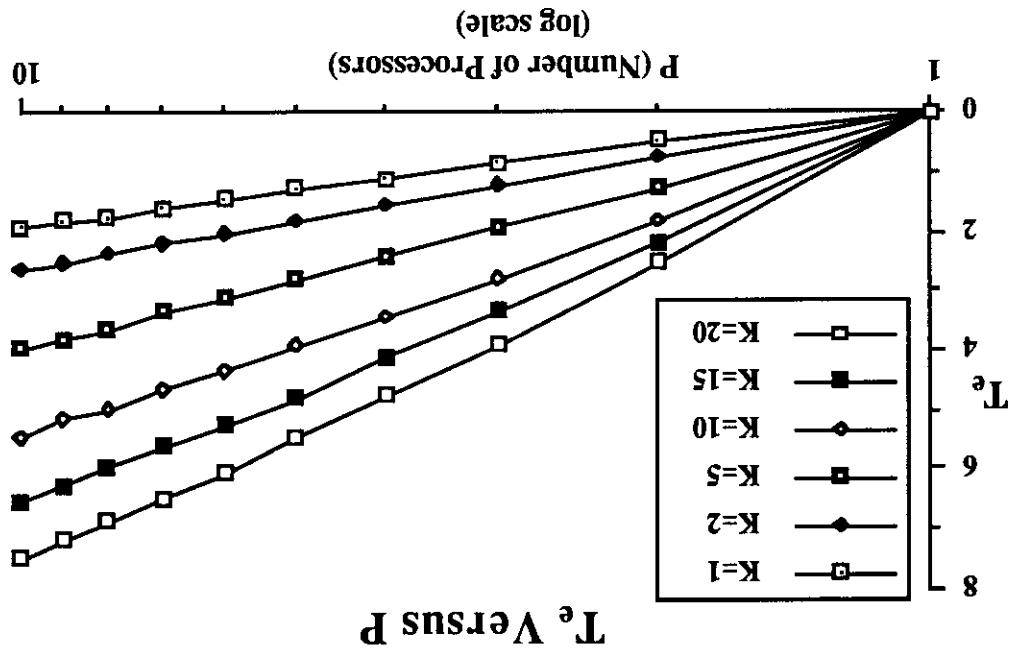
An excellent approximation for T_e when $K > 1$ and $P > 1$ is

$$E[T_e] = \text{Mean of } K\text{-stage Erlang} + T_e = \frac{K}{P} + T_e$$

$T_e \triangleq$ the difference between the mean and the expected value of the max of P K -stage Erlangs, we can approximate $E[T_e]$ as follows. Unfortunately, this equation has no closed form expression for the integral. By decomposing $E[T_e]$ into two components, the mean of a K -stage Erlang and

$$= \int_0^\infty \sum_{j=1}^P \binom{j}{P} (-1)^{j+1} \left(e^{-jx} \sum_{i=0}^{j-1} \frac{j!}{i!} (jx)^i \right) dx$$

Figure 2.3: T_e versus P (log scale).



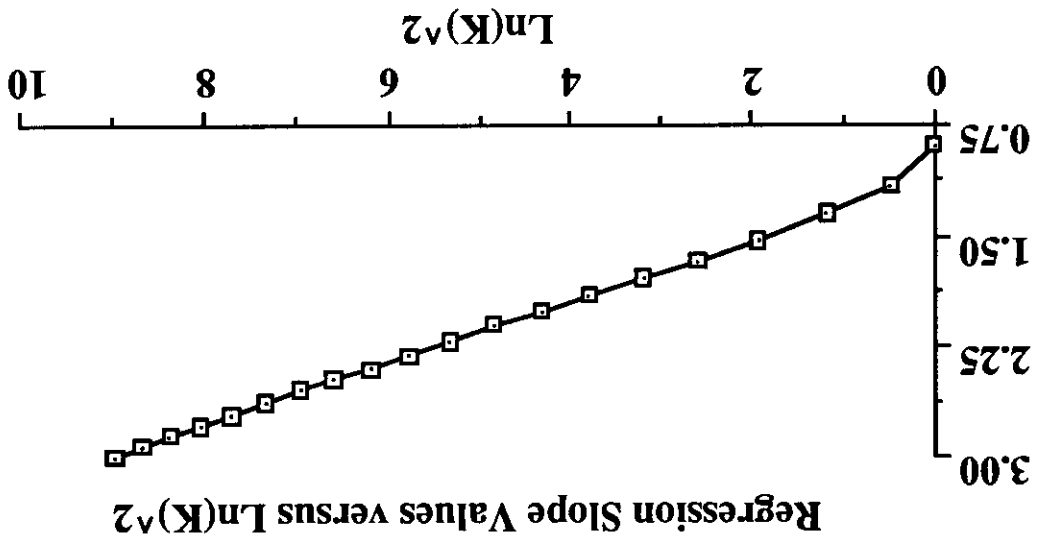
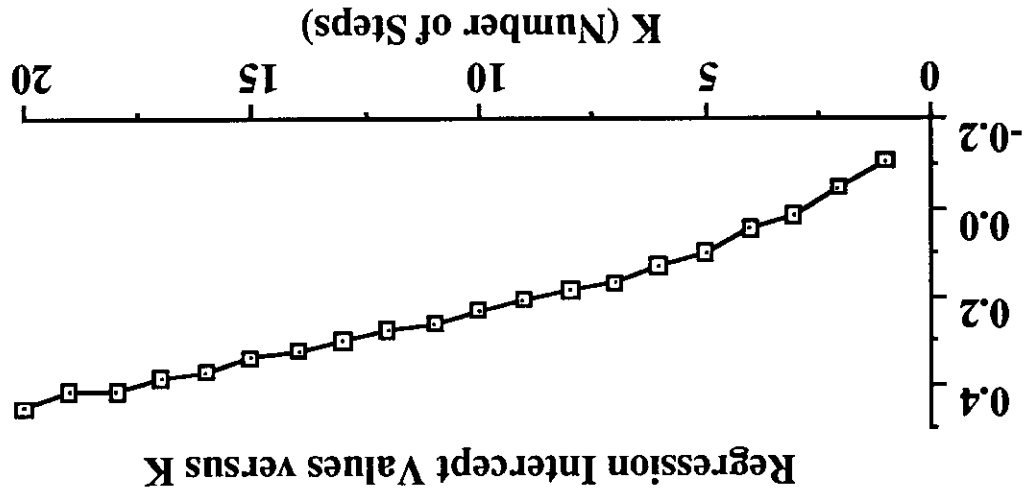


Figure 2.4: Regression slope and intercept values.

$$R_e = \frac{E[T_a]}{E[T_s]} = \frac{\frac{\mu}{K} \left((C \ln^2(K) + D) \ln P + (1 + A)K + B \right)}{\frac{\mu}{K} \left(E + \ln P + \frac{2D}{1} - \frac{1}{1/12} P^{(D+1)} \right)} = \frac{(C \ln^2(K) + D) \ln P + (1 + A)K + B}{E + \ln P + \frac{2D}{1} - \frac{1}{1/12} P^{(D+1)}} + \frac{K}{B}$$

$$E[T_a] \approx \frac{\mu}{K} + \frac{\mu}{1} \left((C \ln^2(K) + D) \ln P + AK + B \right)$$

$$E[T_s] \approx \frac{\mu}{K} \left(E + \ln P + \frac{2D}{1} - \frac{1}{1/12} P^{(D+1)} \right)$$

methods.

Let us look at the ratio, R_e , of the approximations of the two synchronization

2.3.3 Relative Performance

continue to use our own approximation.

Either approximation produces the same results; in the following sections, we

$$E[T_a] \approx \frac{\mu}{K} + \frac{\mu}{1} \sqrt{2K \ln P}$$

times

valid when $K \gg \ln P$ and is given below for exponentially distributed task

for a general (not just exponential) distribution. Their approximation is only

Kruskal and Weiss [KW85] have also developed an approximation for $E[T_a]$

the comparison for K and $P \geq 100$.

simulation for values of K and P between one and ten and Figure 2.6 shows

for the constants A and B). Figure 2.5 shows the approximation compared to

Figure 2.5: Comparison of approximation and simulation for $K, P \leq 10$.

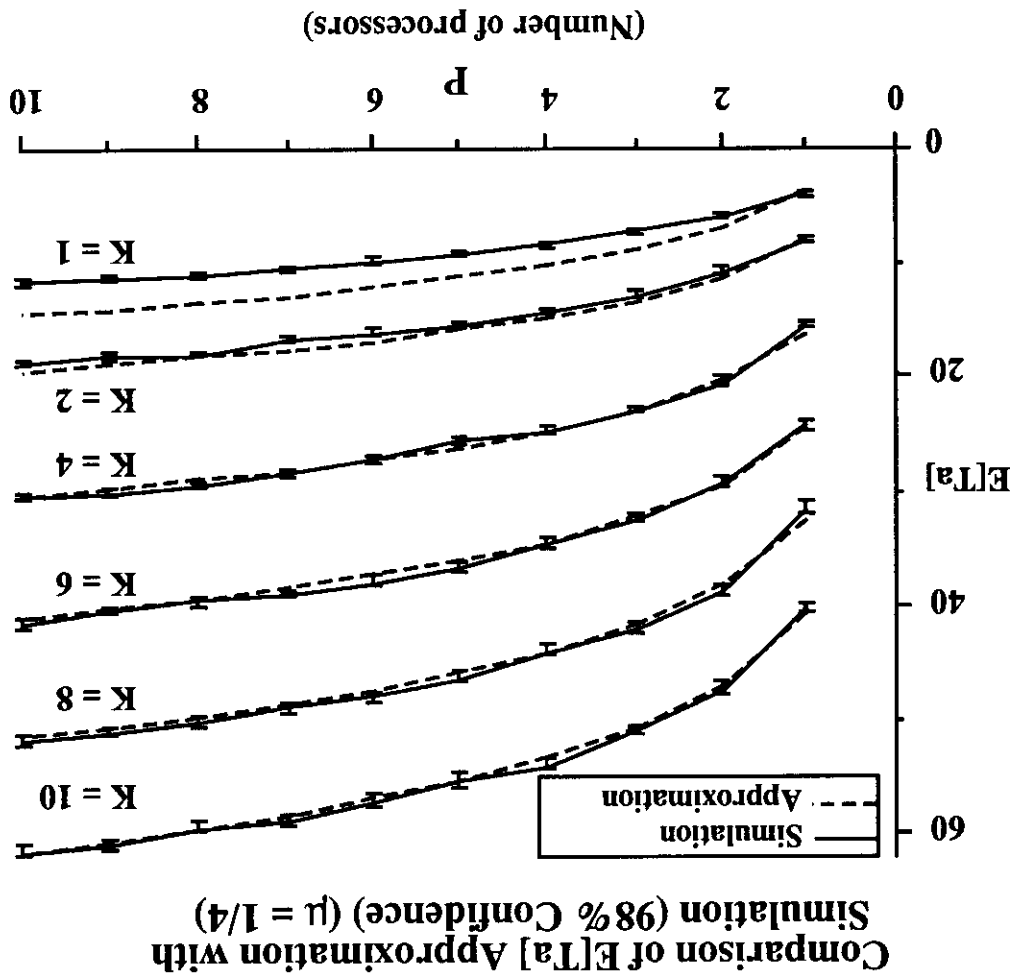
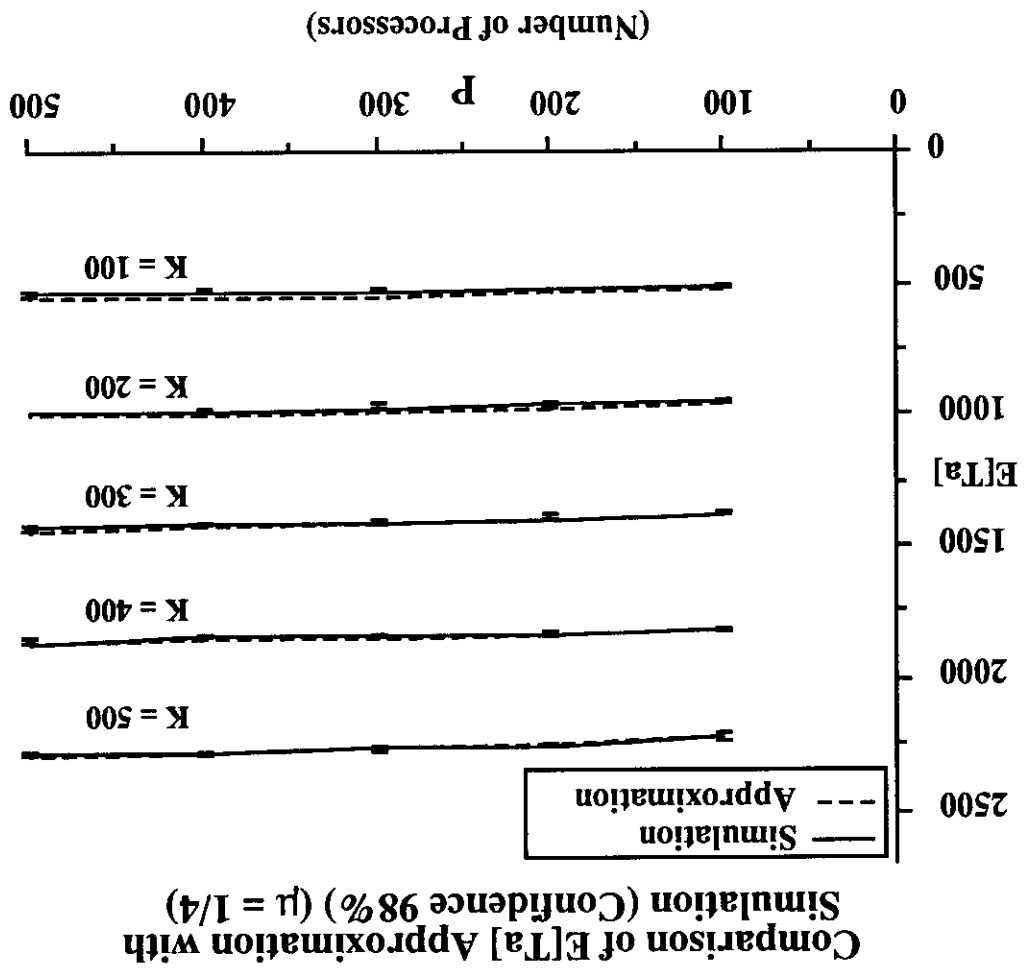


Figure 2.6: Comparison of approximation and simulation for $K, P \geq 100$.



Additionally, if we believe that no method could achieve a speedup greater than P for P processors over execution on a single (sequential) processor, then any time-stepped method is limited to a maximum speedup of $\frac{\ln P}{P}$. These results depend on the assumption of an exponential distribution for task times. The next section uses a uniform $[0, X]$ distribution for task execution times.

Thus, in the limit as the size of the simulation increases to infinity, the asynchronous approach could, at most, complete $(\ln P)$ times as fast as the time-stepped method on average. We can derive this result by appealing to intuition. We have exponential task times where each task takes, on average, $1/\mu$ seconds to complete. For synchronized execution, basic principles (Section 2.3.1) tell us that each stage will take time proportional to $\frac{1}{\mu} \ln P$ on the average. The asynchronous execution on average takes time equal to $\frac{1}{\mu}$ to "complete" a stage. Therefore, the ratio of the two times is $(\ln P)$. It should be noted that a trivial lower bound on T_a (thus an upper bound on R_e) is found by simply using the mean of P K -stage Erlangs since the mean is less than the max. Our approximation (Equation 2.9) confirms this result since A, B, C and D are all non-negative thus making T_e positive.

$$\lim_{K \rightarrow \infty, P \rightarrow \infty} R_e = \frac{\ln P (1 + A)}{\ln P} \approx \frac{1.02}{\ln P} \approx \ln P \quad (2.10)$$

Finally, for large P

$$\lim_{K \rightarrow \infty} R_e = \frac{(1 + A)}{E + \ln P + \frac{1}{2P} - \frac{1}{12P(P+1)}}$$

that $K \gg \ln P$ we get

Taking the limit as the size of the simulation increases ($K \rightarrow \infty$) and assuming

$$R_n = \frac{E[T_a]}{E[T_s]} = \frac{XK \frac{P}{P+1}}{X \left((C \ln^2(K) + D) \ln P + (A + 1/2)K + B \right)} = \frac{(C \ln^2(K) + D) \ln P + (A + 1/2)K + \frac{K}{B}}{\frac{P}{P+1}}$$

Finally, we look at the ratio of the expected completion times.

$$C = 0.053147 \approx 0.05 \quad D = 0.125102 \approx 0.13.$$

$$A = 0.012384 \approx 0.01 \quad B = 0.330691 \approx 0.33$$

Where

$$E[T_a] = \frac{KX}{2} + T_s \approx \frac{KX}{2} + X \left((C \ln^2(K) + D) \ln P + AK + B \right) \quad (2.12)$$

uniform case. Therefore

We can use the same regression technique used with the exponential distribution in Section 2.3.2 to develop an accurate approximation for $E[T_a]$ in the

$$E[T_s] = KX \frac{P}{P+1}. \quad (2.11)$$

If we make the assumption that the task times are uniformly distributed between 0 and X , we calculate a different limiting value for the ratio of completion times. It is easy to show that the mean of the maximum of P uniformly distributed random variables is $X \frac{P}{P+1}$. We immediately find that

2.4 Uniformly Distributed Task Times

As before, if we assume that no method can achieve speedup greater than P over a sequential execution, then the synchronous strategy could possibly have speedup proportional to P when the task times are uniformly distributed (specifically, $P/2$, when $a = 0$).

Again, we can appeal to intuition to find the speedup ratio. If task times are uniformly distributed in the range $[a, b]$ ($a \geq 0$), then, for large P , the synchronized execution will take b seconds per stage (the max) on average to complete a stage. The asynchronous system, on average, will take time equal to $\frac{a+b}{2}$. Therefore, the speedup ratio should be $\frac{2b}{a+b}$. When $a = 0$, the ratio is 2. At worst, when $a \rightarrow b$, the ratio approaches 1.

Which has its maximum value when $a = 0$. Thus, when using a uniform distribution, the asynchronous strategy is only able to complete in roughly half the time, regardless of the number of processors used (as compared to our previous case where the task execution times had exponential tails and the asynchronous strategy was able to gain its $(\ln P)$ performance improvement). This result should apply to any distribution with finite support since the maximum of many such random variables will invariably approach the upper limit.

$$\lim_{K \rightarrow \infty, P \rightarrow \infty} R_n = \frac{a+b}{2b} \quad (2.14)$$

For a more general uniform distribution between a and b ($a, b > 0$) we find that

$$\lim_{K \rightarrow \infty, P \rightarrow \infty} R_n = \frac{1}{1} \frac{(A+1/2)}{1} \approx \frac{1}{0.51} \approx 2 \quad (2.13)$$

Finally, for large P

$$\lim_{K \rightarrow \infty} R_n = \frac{(A+1/2)(P+1)}{P}$$

that $K \gg \ln P$ we get

Taking the limit as the size of the simulation increases ($K \rightarrow \infty$) and assuming

2.5 Conclusions

It has been shown that an asynchronous distributed simulation strategy can have at most an $(\ln P)$ performance improvement over a time-stepped method, in the case where task times are exponentials. We conjecture that this result is due to the infinite tail on the exponential distribution and may therefore be applicable to distributions with "long" tails. The maximum possible improvement when using a distribution with finite support (e.g. uniform) is reduced to a constant amount independent of P . This result is highly dependent on the fact that we allowed full parallelism in the models so the severe synchronization used by the time-stepped technique doesn't hurt too much. This type of parallelism might only be found in air or fluid dynamics simulations or possibly circuit simulations. Our result essentially says that Time Warp will not be able to gain very much in a small-grained (continuous time) simulation with high parallelism and that it is better suited to large-grained parallelism.

Since Time Warp is able to outperform the time-stepped technique even in a situation where the time-stepped approach is favored, a further investigation of the Time Warp algorithm is warranted. Accordingly, the next chapter describes a model of two processors running the Time Warp protocol.

CHAPTER 3

Two Processor Time Warp Analysis: A

Unifying Approach

3.1 Introduction

The results of the previous chapter indicated that Time Warp outperforms the time-stepped approach even when the assumptions favor the time-stepped approach. Therefore, in order to gain a better understanding of the costs of rollback synchronization, we develop a two processor model for Time Warp operation in this chapter.

The next section introduces our model for Time Warp. Section 3.3 provides its exact solution. In Section 3.4 we examine some performance measures including speedup. In Section 3.5 we take limits on various parameters, and we discuss the relationship of this work to that of Lavenberg et al. [LMS83] and Mitra and Mitrani [MM84] in Section 3.6. In Section 3.7 we examine a restricted version of the model in detail. Finally, in Section 3.8 we provide concluding remarks.

3.2 A Model for Time Warp on Two Processors

Assume we have a job that is partitioned into two processes, each of which is executed on a separate processor. As these processes are executed, we consider that they advance along the integers on the x-axis in discrete steps, each beginning at $x = 0$ at time $t = 0$. Each process independently makes jumps forward on the axis where the (integer) size of the jump is geometrically distributed with mean $1/\beta_i$ ($i = 1, 2$). The amount of real time jumps is a geometrically distributed number of time slots with parameter α_i ($i = 1, 2$). After process i makes an advance along the axis, it will send a message to the other process with probability q_i ($i = 1, 2$). Upon receiving a message from the other (sending) process, this (receiving) process will do one of the following:

1: If its position along the x-axis is equal to or behind the sending process, it ignores the message.

2: If it is ahead of the sending process, it will immediately move back (i.e., "rollback") along the x-axis to the current position of the sending process.

Let

$F(t)$ = the position (on the x-axis) of the first process (process one) at time t
and
 $S(t)$ = the position of the second process (process two) at time t .

Further, let

$$D(t) = F(t) - S(t)$$

This is a simple model of the Time Warp distributed simulation algorithm where two processors are both working on a simulation job in an effort to speed it up. They both proceed independently until such time as one (behind) process transmits a message in the "past" of the other (ahead) process. This causes the processors relative to the use of a single processor.

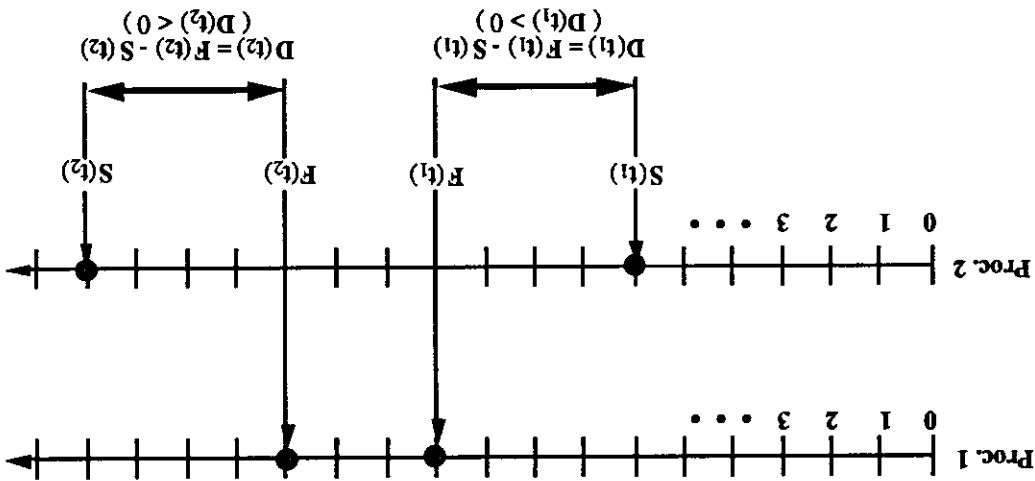
will find the speedup with which the computation proceeds when using two namely, the equilibrium probability for the Markov chain $D(t)$. Moreover, we

$$p_k = \lim_{t \rightarrow \infty} P[D(t) = k] \quad k = 0, 1, 2, \dots$$

$$n_k = \lim_{t \rightarrow \infty} P[D(t) = -k] \quad k = 1, 2, 3, \dots$$

it certainly can go negative, see Figure 3.1). We will solve for $S(0) = 0$, we have $D(0) = 0$. Clearly, $D(t)$ can take on any integer value (i.e., behavior we are interested in studying. From our assumptions that $F(0) = D(t) = 0$ whenever Case 2, a rollback, occurs. $D(t)$ is a Markov process whose

Figure 3.1: The states of two processors at times t_1 and t_2 .



Our model assumes that states are stored after every event, otherwise a rollback would not necessarily send the processor back to the time of the tardy message; rather it might have to rollback to an earlier time, namely that of the last saved state. We assume that communication between processors incurs no delay from transmission to reception. Finally, messages that arrive in the virtual-time future of a process are ignored. They are not queued, nor do they require work in the future. Messages (at any time) are only used for synchronization. We are mainly interested in the loss of forward progress due to the synchronization itself, not in the cost of communication. For a model that takes into account the queuing of messages see Chapter 4.

The interpretation of the model is that the position of a process on the axis is the value of the local clock (or virtual time) of that process. The amount of real time to execute a particular event is modelled by the geometric distribution of time slots between jumps. The geometrically distributed jumps along the axis indicate the increase in the virtual timestamp from one event to the next. Messages passed between processors (with probability q_i) have virtual time stamps equal to the virtual time of the sending process. The messages convey only synchronization information; they do not carry any work. We assume that each Logical Process (LP) is always able to perform work regardless of whether any messages have arrived. The messages are used to synchronize the processors and may be thought of as carrying state information only, not extra work. This is referred to as a self-initiating model.

faster process to rollback to the point where the slower process is located, after which they advance independently again until the next rollback.

In this section we provide the exact solution for the discrete time, discrete state model introduced in Section 3.2. Although, as we proceed, the equations may look formidable, the analysis is quite straightforward. First, we provide some definitions.

3.3 Discrete Time, Discrete State Analysis

It should be easy to see that this system will always make progress. Even if each processor always sends a message to the other ($q_1 = q_2 = 1$), the Global Virtual Time (GVT) of the system will always advance. For example, imagine that processor one (P_1) is at point x_1 on the axis (virtual time) and processor two (P_2) is at x_2 where $x_1 < x_2$ so that P_1 is behind P_2 . By the definition of our model, when a processor completes an event it moves by at least one step and possibly sends a message. Therefore, P_2 can only send a message to P_1 at a time that is greater than or equal to $x_2 + 1$. Therefore, P_2 's actions at the present time can not roll back P_1 or GVT. P_1 , on the other hand, precisely determines the value of GVT and the rate at which it advances at the present moment. Since P_1 will advance at least one step when it advances, GVT will make progress and the system will make progress. It should be noted here that the two processor system is attractive since it will not suffer from "cascading rollbacks" [LSW89] where a processor that gets rolled back causes another to roll back and so on.

Since the transitions in our system are quite complex (there are an infinite number of transitions into and out of each state) we choose to show the state diagram only for a simplified version of our system where $\beta_1 = \beta_2 = 1$ in Figure 3.2. This is the case where the processors only make jumps of a single step (from k to $k + 1$).

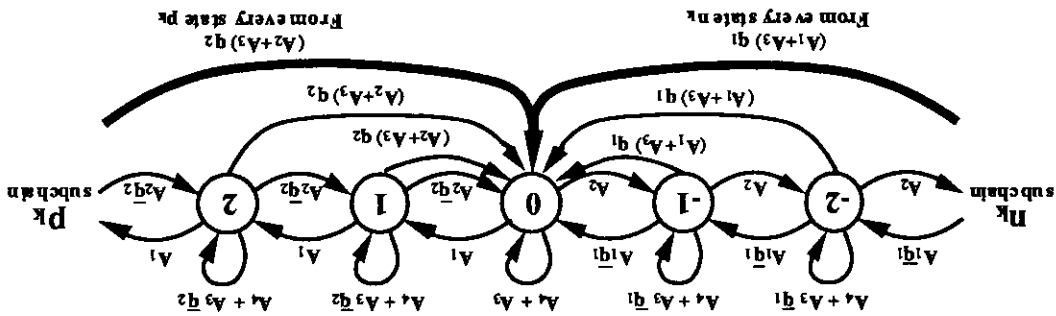
$$\begin{aligned}
 \alpha_i &= P^{[i^{\text{th}}]} \text{ processor advances in a time slot} \quad (i = 1, 2) \\
 \underline{\alpha}_i &= 1 - \alpha_i \\
 A_1 &= \alpha_1 \underline{\alpha}_2 \text{ (Only processor 1 advances in a time slot)} \\
 A_2 &= \alpha_2 \underline{\alpha}_1 \text{ (Only processor 2 advances in a time slot)} \\
 A_3 &= \alpha_1 \alpha_2 \text{ (Both processors advance in a time slot)} \\
 A_4 &= \underline{\alpha}_1 \underline{\alpha}_2 \text{ (Neither processor advances in a time slot)} \\
 f_j &= P[\text{Processor 1 advances } j \text{ units} \mid \text{it advances}] \\
 &= \beta_1 \underline{\beta}_1^{j-1} (j > 0) \quad (\underline{\beta}_1 = 1 - \beta_1) \\
 g_j &= P[\text{Processor 2 advances } j \text{ units} \mid \text{it advances}] \\
 &= \beta_2 \underline{\beta}_2^{j-1} (j > 0) \quad (\underline{\beta}_2 = 1 - \beta_2) \\
 \pi &= P[\text{Procs. 1 and 2 advance the same dist.} \mid \text{both advance}] \\
 &= \frac{\beta_1 \beta_2}{1 - \beta_1 \beta_2} \\
 q_i &= P^{[i^{\text{th}}]} \text{ processor sends a message after advancing} \quad (i = 1, 2) \\
 \underline{q}_i &= 1 - q_i
 \end{aligned}$$

$$\begin{aligned}
 [A_1 + A_2 + A_3 (1 - \pi + q_2\pi)] p_k = & A_1 \left(\sum_{i=0}^{k-1} p_i f_{i-k-i} + \sum_{i=1}^{\infty} n_i f_{k+i} \right) \\
 & + A_2 \bar{q}_2 \sum_{i=k+1}^{\infty} p_i g_{i-k} \\
 & + A_3 \bar{q}_2 \sum_{i=\infty}^{k+1} p_i \sum_{j=1}^{\infty} f_j g_{j+i-k} \\
 & + A_3 \bar{q}_2 \sum_{i=k-1}^{\infty} p_i \sum_{j=1}^{\infty} f_j g_{i-k-j} \\
 & + A_3 \bar{q}_2 \sum_{i=\infty}^{\infty} n_i \sum_{j=1}^{\infty} f_j g_{i+k-j} \quad (3.1)
 \end{aligned}$$

restrictions on β_i) are given below.

The steady-state balance equations for our completely general system (no

Figure 3.2: State transition probability diagram for $\beta_1 = \beta_2 = 1$.



Using Equation 3.1 we solve for $P(z)$ by multiplying each term by z^k and summing from $k = 1$ to ∞ . Since this equation has so many summations, we list the solution for each term separately and provide the derivations in Appendix A.

$$P(z) = \sum_{k=1}^{\infty} p_k z^k \quad Q(z) = \sum_{k=1}^{\infty} n_k z^k$$

$$F(z) = \sum_{k=1}^{\infty} f_k z^k = \frac{1}{g_1 z} (1 - \beta_1 z)$$

We define the following z-transforms.

$$D(t).$$

The above equations will have a steady-state solution only if $q_1, q_2 > 0$ and $\alpha_1, \alpha_2 > 0$. These are reasonable limitations. The alphas must be greater than zero or a processor will take an infinite amount of time to process an event. Each of the q s must be greater than zero so that each processor has some positive probability of being rolled back when it gets ahead, thus keeping a limit on $D(t)$.

$$p_0 = 1 - \sum_{i=1}^{\infty} p_i - \sum_{i=1}^{\infty} n_i$$

$$[A_1 + A_2 + A_3(1 - \pi + q_1\pi)]n_k = A_2 \left(\sum_{i=k-1}^{\infty} n_i g_{k-i} + \sum_{i=1}^{\infty} p_i g_{k+i} \right) \\ + A_1 q_1 \sum_{i=k+1}^{\infty} n_i f_{i-k} \\ + A_3 q_1 \sum_{i=\infty}^{\infty} n_i \sum_{j=1}^{i-k+1} g_j f_{j+i-k} \\ + A_3 q_1 \sum_{i=k-1}^{\infty} n_i \sum_{j=1}^{i-k} g_{j+k-i} f_j \\ + A_3 q_1 \sum_{i=\infty}^{\infty} p_i \sum_{j=1}^{i-k+1} g_{j+k+i} f_j \quad k \geq 1 \quad (3.2)$$

$$\begin{aligned}
 & (A_1 + A_2 + A_3 - A_3 q_2 \pi) P(z) = A_1 F(z) (P(z) + p_0 + Q(\beta_1)) \\
 & + \frac{A_2 q_2 \beta_2}{z P(\beta_2)} \left(P(z) - \frac{\beta_2}{z} \right) + \frac{A_3 q_2 \beta_1 \beta_2 \beta_2}{z P(\beta_2)} \left(P(z) - \frac{\beta_2}{z} \right) \\
 & + \frac{A_3 q_2 \beta_1 \beta_2 F(z)}{z P(\beta_2)} \left(P(z) - \frac{\beta_2}{z} \right) + \frac{A_3 q_2 \beta_1 \beta_2 F(z)}{z P(\beta_2)} \left(P(z) + p_0 + Q(\beta_1) \right)
 \end{aligned}$$

$P(z)$.

Finally, by combining terms, we arrive at the following equation that defines

$$\begin{aligned}
 \sum_{k=1}^{\infty} (A_1 + A_2 + A_3 - A_3 q_2 \pi) p_k z^k &= \left(A_1 + A_2 + A_3 - A_3 q_2 \frac{1 - \beta_1 \beta_2}{\beta_1 \beta_2} \right) P(z) \\
 A_1 \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i f_{k-i} &= A_1 F(z) (P(z) + p_0) \\
 A_1 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{k-1} n_i f_{k+i} &= A_1 F(z) Q(\beta_1) \\
 A_2 q_2 \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i g_{i-k} &= \frac{A_2 q_2 \beta_2}{z P(\beta_2)} \left(P(z) - \frac{\beta_2}{z} \right) \\
 A_3 q_2 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} p_i \sum_{j=i+k}^{\infty} f_j g_{j+i-k} &= \frac{A_3 q_2 \beta_1 \beta_2 \beta_2}{z P(\beta_2)} \left(P(z) - \frac{\beta_2}{z} \right) \\
 A_3 q_2 \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \sum_{j=k+i}^{\infty} f_j g_{j+i} &= \frac{A_3 q_2 \beta_1 \beta_2 F(z)}{z P(\beta_2)} (P(z) + p_0) \\
 A_3 q_2 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} n_i \sum_{j=i+k}^{\infty} g_j f_{j+k+i} &= \frac{A_3 q_2 \beta_1 \beta_2 F(z) Q(\beta_1)}{z P(\beta_2)} \frac{1 - \beta_1 \beta_2}{\beta_1 \beta_2}
 \end{aligned}$$

so we only need to solve explicitly for $P(z)$.

Also, $P(z)$ and $Q(z)$ are symmetric with respect to $(\alpha_1, \alpha_2), (\beta_1, \beta_2)$ and (q_1, q_2) ,

$$(s_1, s_2) = \frac{2a_n}{-b_n \pm \sqrt{b_n^2 - 4a_n c_n}}$$

denominator of $Q(z)$ to get

Since $Q(z)$ is symmetric to $P(z)$, we proceed to find the roots, (s_1, s_2) of the

$$\begin{aligned} a_p &= A_1 + \underline{\beta}_1(A_2 + A_3) \\ b_p &= -((A_1 + A_2 + A_3)(1 + \underline{\beta}_1 \underline{\beta}_2) + A_1 \underline{\beta}_1 \underline{\beta}_2 + \underline{\beta}_2 \underline{q}_2(A_2 \underline{\beta}_1 - A_3 \underline{\beta}_1)) \\ c_p &= \underline{\beta}_2(A_1 + A_2 + A_3) + A_2 \underline{\beta}_2 \underline{q}_2 \end{aligned}$$

where

$$(r_1, r_2) = \frac{2a_p}{-b_p \pm \sqrt{b_p^2 - 4a_p c_p}}$$

Using the quadratic equation we solve for the roots (r_1, r_2) as given below.

where r_1 and r_2 are the roots of the quadratic expression in $D(z)$.

$$D(z) = \underline{\beta}_2 \left(1 - \underline{\beta}_1 \underline{\beta}_2 \right) a_p (z - r_1)(z - r_2) \quad (3.5)$$

We simplify $D(z)$ to

$$\begin{aligned} D(z) &= \underline{\beta}_2 \left(1 - \underline{\beta}_1 \underline{\beta}_2 \right) \left[(A_1 + A_2 + A_3) \underline{\beta}_1 z^2 \right. \\ &\quad - (A_1 \underline{\beta}_1 \underline{\beta}_2 + (A_1 + A_2 + A_3)(1 + \underline{\beta}_1 \underline{\beta}_2) + \underline{\beta}_2 \underline{q}_2(A_2 \underline{\beta}_1 - A_3 \underline{\beta}_1)) z \\ &\quad \left. + \underline{\beta}_2(A_1 + A_2 + A_3) + A_2 \underline{\beta}_2 \underline{q}_2 \right] \end{aligned} \quad (3.4)$$

$$\begin{aligned} N(z) &= z \left(\underline{\beta}_2 (A_3 \underline{\beta}_1 \underline{\beta}_2 + A_2 (1 - \underline{\beta}_1 \underline{\beta}_2)) \underline{q}_2 (1 - \underline{\beta}_1 z) P(\underline{\beta}_2) \right. \\ &\quad \left. - \underline{\beta}_1 \underline{\beta}_2 (A_1 (1 - \underline{\beta}_1 \underline{\beta}_2) + A_3 \underline{\beta}_1 \underline{\beta}_2 \underline{q}_2) (z - \underline{\beta}_2) (p_0 + Q(\underline{\beta}_1)) \right) \end{aligned} \quad (3.3)$$

polynomials, explicitly.

We simplify this equation by solving for $P(z) = N(z)/D(z)$, a ratio of two

$$D_n = A_2(1 - \beta_1 \bar{\beta}_2) + A_3 \beta_1 \bar{\beta}_2 \bar{q}_1 \quad (3.9)$$

$$D_p = A_1(1 - \beta_1 \bar{\beta}_2) + A_3 \beta_1 \bar{\beta}_2 \bar{q}_2 \quad (3.8)$$

where

$$Q(z) = \frac{a_n (1 - \beta_2 s_2) (s_1 - z)}{\beta_2 D_n z (p_0 + P(\beta_2))} \quad (3.7)$$

and by symmetry

$$P(z) = \frac{a_p (1 - \beta_1 r_2) (r_1 - z)}{\beta_1 D_p z (p_0 + Q(\beta_1))} \quad (3.6)$$

Therefore,

$$N(z) = \frac{1 - \beta_1 r_2}{\beta_1 \bar{\beta}_2 (1 - \beta_1 \bar{\beta}_2) D_p z (r_2 - z) (p_0 + Q(\beta_1))}$$

Equation 3.3 resulting in the following equation.

D_p and D_n are constants given below. We then substitute $P(\beta_2)$ back into

$$P(\beta_2) = \frac{\beta_2 \bar{q}_2 (A_3 \beta_1 \bar{\beta}_2 + A_2 (1 - \beta_1 \bar{\beta}_2)) (1 - \beta_1 r_2)}{\beta_1 \bar{\beta}_2 D_p (r_2 - \beta_2) (p_0 + Q(\beta_1))}$$

zero we solve for $P(\beta_2)$.

In Section 3.3.1 we show that r_1 and r_2 are real, $r_1 \geq 1$ and $0 \leq r_2 \leq 1$. Since $P(z) \triangleq \sum_{i=1}^{\infty} p_i z^i$ is the transform of a probability distribution, we know that $P(z)$ must be analytic whenever $|z| \leq 1$. Therefore, the numerator of $P(z)$ must equal zero when $z = r_2$ (since $r_2 \leq 1$) so as to cancel the $(z - r_2)$ term in the denominator. Plugging r_2 into $N(z)$ (Equation 3.3) and setting it equal to

$$\begin{aligned} c_n &= \beta_1 (A_1 + A_2 + A_3) + A_1 \beta_1 \bar{q}_1 \\ b_n &= -((A_1 + A_2 + A_3)(1 + \beta_1 \bar{\beta}_2) + A_2 \beta_2 \bar{\beta}_1 + \beta_1 \bar{q}_1 (A_1 \bar{\beta}_2 - A_3 \beta_2)) \\ a_n &= A_2 + \beta_2 (A_1 + A_3) \end{aligned}$$

where

We solve for the unknown constants $P(\bar{\beta}_2)$ and $Q(\bar{\beta}_1)$ by solving Equations 3.6 and 3.7 simultaneously with z replaced by $\bar{\beta}_1$ and $\bar{\beta}_2$ respectively.

$$\begin{aligned} P(\bar{\beta}_2) &= K_p (p_0 + Q(\bar{\beta}_1)) \\ Q(\bar{\beta}_1) &= K_n (p_0 + P(\bar{\beta}_2)) \end{aligned}$$

Solving them simultaneously yields.

$$\begin{aligned} P(\bar{\beta}_2) &= \frac{p_0 K_p (1 + K_n)}{1 - K_p K_n} \\ Q(\bar{\beta}_1) &= \frac{p_0 K_n (1 + K_p)}{1 - K_p K_n} \end{aligned}$$

where

$$K_p = \frac{\beta_1 \bar{\beta}_2 D_p}{a_p (r_1 - \bar{\beta}_2) (1 - \bar{\beta}_1 r_2)} \quad (3.10)$$

$$K_n = \frac{\bar{\beta}_1 \beta_2 D_n}{a_n (s_1 - \bar{\beta}_1) (1 - \bar{\beta}_2 s_2)} \quad (3.11)$$

Substituting these values into Equations 3.6 and 3.7 and simplifying we find

$$P(z) = \frac{z p_0 C_p}{r_1 - z} \quad (3.12)$$

$$Q(z) = \frac{z p_0 C_n}{s_1 - z} \quad (3.13)$$

where

$$C_p = \frac{\beta_1 D_p (1 + K_n)}{(1 - K_n K_p) a_p (1 - \bar{\beta}_1 r_2)} \quad (3.14)$$

$$C_n = \frac{\beta_2 D_n (1 + K_p)}{(1 - K_n K_p) a_n (1 - \bar{\beta}_2 s_2)} \quad (3.15)$$

By conservation of probability we know that $P(1) + Q(1) + p_0 = 1$. Therefore,

$$p_0 = \frac{1}{1 + \left(\frac{C_p}{r_1 - 1}\right) + \left(\frac{C_n}{s_1 - 1}\right)} \quad (3.16)$$

Finally, we invert the z -transforms to get our final answer.

$$p_k = C_p p_0 \left(\frac{1}{r_1}\right)^k \quad k \geq 1 \quad (3.17)$$

$$n_k = C_n p_0 \left(\frac{1}{s_1}\right)^k \quad k \geq 1 \quad (3.18)$$

3.3.1 Root Locus

In this section we show that r_1 and r_2 are real, $r_1 \geq 1$ and $0 \leq r_2 \leq 1$. To show that the roots r_1 and r_2 are real, we must show that the quantity under the square root is greater than or equal to zero, or that

$$b_p^2 - 4a_p c_p \geq 0$$

Substituting in the values for a_p , b_p , c_p and simplifying, we find that the roots will be real if the following inequality is satisfied.

$$\begin{aligned} & (\beta_2 \alpha_1 - \beta_1 \alpha_2)^2 + \beta_2^2 \alpha_2^2 q_2^2 (\bar{\beta}_1 - \alpha_1)^2 \\ & + 2\beta_2 \alpha_2 q_2 (\beta_2 \alpha_1 \bar{\alpha}_1 + \beta_1 (\alpha_1 (2 - \beta_2 - \alpha_2) + \bar{\beta}_1 \alpha_2)) \geq 0 \end{aligned} \quad (3.19)$$

Since all the factors on the left-hand side of Equation 3.19 are non-negative, the inequality must hold. Therefore, both r_1 and r_2 are real roots.

We now show that $r_1 \geq 1$. Assuming it is true, we require

$$\begin{aligned} r_1 = \frac{-b_p + \sqrt{b_p^2 - 4a_p c_p}}{2a_p} & \geq 1 \\ -b_p + \sqrt{b_p^2 - 4a_p c_p} & \geq 2a_p \\ \sqrt{b_p^2 - 4a_p c_p} & \geq 2a_p + b_p \\ b_p^2 - 4a_p c_p & \geq 4a_p^2 + 4a_p b_p + b_p^2 \\ 0 & \geq a_p + b_p + c_p \end{aligned}$$

Substituting in the values for a_p , b_p and c_p we arrive at the condition

$$0 \geq -\beta_1\beta_2\alpha_2q_2 \quad (3.20)$$

Since all the terms on the right-hand side of Equation 3.20 are non-negative, the inequality holds.

To show that $r_2 \leq 1$ we need to prove the following

$$\begin{aligned} r_2 &= \frac{-b_p - \sqrt{b_p^2 - 4a_p c_p}}{2a_p} \leq 1 \\ -b_p - \sqrt{b_p^2 - 4a_p c_p} &\leq 2a_p \\ -b_p - 2a_p &\leq \sqrt{b_p^2 - 4a_p c_p} \\ b_p^2 + 4a_p b_p + 4a_p^2 &\leq b_p^2 - 4a_p c_p \\ a_p + b_p + c_p &\leq 0 \end{aligned}$$

which was shown above to be true.

Finally, we show that $r_2 \geq 0$. We require

$$\begin{aligned} r_2 &= \frac{-b_p - \sqrt{b_p^2 - 4a_p c_p}}{2a_p} \geq 0 \\ -b_p - \sqrt{b_p^2 - 4a_p c_p} &\geq 0 \\ b_p + \sqrt{b_p^2 - 4a_p c_p} &\leq 0 \text{ (multiply by } -1) \\ \sqrt{b_p^2 - 4a_p c_p} &\leq -b_p \\ -4a_p c_p &\leq 0 \end{aligned} \quad (3.21)$$

Since a_p and c_p are non-negative, the inequality in Equation 3.21 holds, thus proving that $r_2 \geq 0$. Similar proofs follow for (s_1, s_2) , the roots of $Q(z)$.

3.4 Performance Measures

With the complete solution to the Markov chain in hand (Equations 3.16, 3.17 and 3.18), we calculate several interesting performance measures. The first is \bar{K}_1 which is defined as the average distance processor one is ahead of processor two, given that processor one is ahead. This measure is useful in determining the number of states that will need to be saved on average as we will see below.

$$\begin{aligned}\bar{K}_1 &= \frac{\sum_{k=1}^{\infty} k p_k}{\sum_{k=1}^{\infty} p_k} \\ &= \frac{r_1}{r_1 - 1}\end{aligned}$$

We find a symmetric value for processor two.

$$\begin{aligned}\bar{K}_2 &= \frac{\sum_{k=1}^{\infty} k n_k}{\sum_{k=1}^{\infty} n_k} \\ &= \frac{s_1}{s_1 - 1}\end{aligned}$$

Since we know the expected size of a state jump at processor one is $1/\beta_1$, we find that the expected number of buffers needed for state saving when processor one is ahead is

$$\begin{aligned}\text{E[Buffers needed when Proc. 1 is ahead]} &= \frac{\bar{K}_1}{\frac{1}{\beta_1}} \\ &= \frac{r_1 \beta_1}{r_1 - 1}\end{aligned}\tag{3.22}$$

and for processor two

$$\begin{aligned}\text{E[Buffers needed when Proc. 2 is ahead]} &= \frac{\bar{K}_2}{\frac{1}{\beta_2}} \\ &= \frac{s_1 \beta_2}{s_1 - 1}\end{aligned}\tag{3.23}$$

Another useful measure, $\Theta_{1,b}$, is the probability that processor one needs more than b buffers for storing state.

$$\begin{aligned}
\Theta_{1,b} &= \text{P}[\text{Proc. 1 needs } > b \text{ buffers}] \\
&= \sum_{i=b+1}^{\infty} \text{P}[\text{Proc. 1 is using } i \text{ buffers}] \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \text{P}[\text{Proc. 1 is using } i \text{ buffers} \mid \text{Proc. 1 is } k \text{ units ahead}] p_k \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \text{P}[\text{Sum of } i \text{ geometric random variables} = k] p_k \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \binom{k-1}{i-1} \beta_1^i \bar{\beta}_1^{k-i} C_p p_0 \left(\frac{1}{r_1}\right)^k \\
&= C_p p_0 \left(\frac{\beta_1}{r_1 - 1}\right) \left(\frac{\beta_1}{r_1 - \beta_1}\right)^b \tag{3.24}
\end{aligned}$$

If $\beta_1 = 1$ (single step state jumps), then $\Theta_{1,b}$ reduces to the following expression.

$$\Theta_{1,b} = \sum_{k=b+1}^{\infty} p_k = \frac{C_p p_0}{r_1^b (r_1 - 1)} \tag{3.25}$$

A similar (symmetric) value, $\Theta_{2,b}$, can be found for processor two. The quantity of most interest though is speedup, and we calculate its value in the next section.

3.4.1 Speedup

Using the formulae for p_k and n_k we calculate the speedup S when using two processors versus using only one. S is the rate of progress when using two processors (R_2) divided by the rate of progress when using only one processor (R_1). The rate of forward progress for one processor (obviously not running Time Warp) is defined as the average rate of virtual time progress per real time step of the two processes defined earlier by the real time and virtual time geometric distributions (Section 3.2). Since process one completes events (on average) every $1/\alpha_1$ seconds and process two does so every $1/\alpha_2$ seconds and

they make jumps in virtual time of distance $1/\beta_1$ and $1/\beta_2$ respectively, then the average rate of virtual time progress would be

$$R_1 \triangleq \frac{\frac{\alpha_1}{\beta_1} + \frac{\alpha_2}{\beta_2}}{2} = \frac{\alpha_1\beta_2 + \alpha_2\beta_1}{2\beta_1\beta_2}$$

The average rate of forward progress for two processors is the expected “unfettered” rate of progress (without rollbacks) per time step minus the (rollback-distance-weighted) expected rollback rate per time step for the two processors. The first three terms (positive terms) in the following expression give the forward rate while the negative terms give the rollback rate. The negative terms are derived by noting that when a process advances f units and causes the other to rollback r units, the net progress is given by the difference $(f - r)$.

$$\begin{aligned} R_2 = & \frac{A_1}{\beta_1} + \frac{A_2}{\beta_2} + A_3 \left(\frac{1}{\beta_1} + \frac{1}{\beta_2} \right) \\ & - A_3 q_2 p_0 \sum_{i=2}^{\infty} f_i \sum_{j=1}^{i-1} g_j (i-j) - A_3 q_1 p_0 \sum_{i=2}^{\infty} g_i \sum_{j=1}^{i-1} f_j (i-j) \\ & - A_2 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{k-1} i g_{k-i} - A_1 q_1 \sum_{k=1}^{\infty} n_k \sum_{i=1}^{k-1} i f_{k-i} \\ & - A_3 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{k+i-1} j g_{k+i-j} - A_3 q_1 \sum_{k=1}^{\infty} n_k \sum_{i=1}^{\infty} g_i \sum_{j=1}^{k+i-1} j f_{k+i-j} \\ & - A_3 q_1 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{\infty} j g_{k+i+j} - A_3 q_2 \sum_{k=1}^{\infty} n_k \sum_{i=1}^{\infty} g_i \sum_{j=1}^{\infty} j f_{k+i+j} \end{aligned}$$

As with the $P(z)$ calculation we list the solution for each term separately, but since each pair of terms above is symmetric with respect to f and g we only need to derive the closed-form solution for one of the two. The derivations can

be found in Appendix A.

$$\begin{aligned}
A_3 q_2 p_0 \sum_{i=2}^{\infty} f_i \sum_{j=1}^{i-1} g_j (i-j) &= \frac{A_3 \bar{\beta}_1 \beta_2 q_2 p_0}{\beta_1 (1 - \bar{\beta}_1 \beta_2)} \\
A_2 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{k-1} i g_{k-i} &= \frac{A_2 q_2 \beta_2 C_p p_0 r_1}{(r_1 - \bar{\beta}_2)(r_1 - 1)^2} \\
A_3 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{k+i-1} j g_{k+i-j} &= \frac{A_3 q_2 \beta_2 C_p p_0}{(1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - 1)^2} \left(\frac{(r_1 - \bar{\beta}_1)}{\beta_1} + \frac{\beta_1 \bar{\beta}_2 r_1}{(r_1 - \bar{\beta}_2)} \right) \\
A_3 q_1 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{\infty} j g_{k+i+j} &= \frac{A_3 q_1 \beta_1 \bar{\beta}_2^2 C_p p_0}{\beta_2 (1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - \bar{\beta}_2)}
\end{aligned}$$

Finally, combining all the terms together we find the formula for speedup.

$$\begin{aligned}
S = & \left(\frac{2\beta_1\beta_2}{\alpha_1\beta_2 + \alpha_2\beta_1} \right) \left[\frac{A_1}{\beta_1} + \frac{A_2}{\beta_2} + A_3 \left(\frac{1}{\beta_1} + \frac{1}{\beta_2} \right) \right. \\
& - \frac{A_3 \bar{\beta}_1 \beta_2 q_2 p_0}{\beta_1 (1 - \bar{\beta}_1 \bar{\beta}_2)} - \frac{A_3 \bar{\beta}_2 \beta_1 q_1 p_0}{\beta_2 (1 - \bar{\beta}_1 \bar{\beta}_2)} \\
& - \frac{A_2 q_2 \beta_2 C_p p_0 r_1}{(r_1 - \bar{\beta}_2)(r_1 - 1)^2} - \frac{A_1 q_1 \beta_1 C_n p_0 s_1}{(s_1 - \bar{\beta}_1)(s_1 - 1)^2} \\
& - \frac{A_3 q_2 \beta_2 C_p p_0}{(1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - 1)^2} \left(\frac{(r_1 - \bar{\beta}_1)}{\beta_1} + \frac{\beta_1 \bar{\beta}_2 r_1}{(r_1 - \bar{\beta}_2)} \right) \\
& - \frac{A_3 q_1 \beta_1 C_n p_0}{(1 - \bar{\beta}_1 \bar{\beta}_2)(s_1 - 1)^2} \left(\frac{(s_1 - \bar{\beta}_2)}{\beta_2} + \frac{\beta_2 \bar{\beta}_1 s_1}{(s_1 - \bar{\beta}_1)} \right) \\
& \left. - \frac{A_3 q_1 \beta_1 \bar{\beta}_2^2 C_p p_0}{\beta_2 (1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - \bar{\beta}_2)} - \frac{A_3 q_2 \beta_2 \bar{\beta}_1^2 C_n p_0}{\beta_1 (1 - \bar{\beta}_1 \bar{\beta}_2)(s_1 - \bar{\beta}_1)} \right] \quad (3.26)
\end{aligned}$$

3.5 Limiting Behavior

Before examining the results of the previous section, we explore what sorts of models arise when taking limits on the α and β parameters. The next three subsections will present the results for $(\alpha_1, \alpha_2) \rightarrow 0$ and $(\beta_1, \beta_2) \rightarrow 0$. By taking these limits we transform the geometric distributions into exponential distributions, thus moving from discrete time and state to continuous time and state.

3.5.1 Continuous Time, Discrete State

We transform our model into a continuous time, discrete state (CD) model by taking the limit as α_1 and $\alpha_2 \rightarrow 0$ while keeping the ratio $\frac{\alpha_1}{\alpha_2}$ constant and defining $\frac{A_1}{A_1+A_2} = a$. We can take the limit either on the p_k equations or on our formula for speedup. The final result for speedup is given below.

$$S = 2 \left(1 - \frac{p_0}{\frac{a}{\beta_1} + \frac{\bar{a}}{\beta_2}} \left(\frac{C_p \bar{a} q_2 \beta_2 r_1}{(r_1 - \bar{\beta}_2)(r_1 - 1)^2} + \frac{C_n a q_1 \beta_1 s_1}{(s_1 - \bar{\beta}_1)(s_1 - 1)^2} \right) \right) \quad (3.27)$$

where

$$p_0 = \frac{1}{1 + \left(\frac{C_p}{r_1 - 1}\right) + \left(\frac{C_n}{s_1 - 1}\right)}$$

$$C_p = \frac{a \beta_1 (1 - \bar{\beta}_1 \bar{\beta}_2) (1 + K_n)}{(1 - K_p K_n) (1 - \bar{\beta}_1 r_2) (1 - \beta_1 \bar{a})}$$

$$C_n = \frac{\bar{a} \beta_2 (1 - \bar{\beta}_1 \bar{\beta}_2) (1 + K_p)}{(1 - K_p K_n) (1 - \bar{\beta}_2 s_2) (1 - \beta_2 a)}$$

$$K_p = \frac{a \beta_1 \bar{\beta}_2 (1 - \bar{\beta}_1 \bar{\beta}_2)}{(1 - \bar{\beta}_1 r_2) (1 - \beta_1 \bar{a}) (r_1 - \bar{\beta}_2)}$$

$$K_n = \frac{\bar{a} \beta_2 \bar{\beta}_1 (1 - \bar{\beta}_1 \bar{\beta}_2)}{(1 - \bar{\beta}_2 s_2) (1 - \beta_2 a) (s_1 - \bar{\beta}_1)}$$

$$(r_1, r_2) = \frac{(1 + \bar{\beta}_2 (1 - \beta_1 \bar{a}) + \bar{\beta}_1 \beta_2 \bar{a} \bar{q}_2)}{2(1 - \beta_1 \bar{a})} \pm \frac{\sqrt{(1 + \bar{\beta}_2 (1 - \beta_1 \bar{a}) + \bar{\beta}_1 \beta_2 \bar{a} \bar{q}_2)^2 - 4(1 - \beta_1 \bar{a})(\bar{\beta}_2 + \beta_2 \bar{a} \bar{q}_2)}}{2(1 - \beta_1 \bar{a})}$$

$$(s_1, s_2) = \frac{(1 + \bar{\beta}_1 (1 - \beta_2 a) + \bar{\beta}_2 \beta_1 a \bar{q}_1)}{2(1 - \beta_2 a)} \pm \frac{\sqrt{(1 + \bar{\beta}_1 (1 - \beta_2 a) + \bar{\beta}_2 \beta_1 a \bar{q}_1)^2 - 4(1 - \beta_2 a)(\bar{\beta}_1 + \beta_1 a \bar{q}_1)}}{2(1 - \beta_2 a)}$$

3.5.2 Discrete Time, Continuous State

We create a discrete time, continuous state (DC) model by taking the limit as β_1 and $\beta_2 \rightarrow 0$ while keeping $\frac{\beta_1}{\beta_2} = b$. We find the value for speedup by taking limits on the speedup formula calculated for the discrete time, discrete state model. The resulting formula is given below. We first substitute $\beta_2 = \beta_1/b$, then take the limit as $\beta_1 \rightarrow 0$. When taking the limit, we were often confronted with functions of the form $0/0$ and were forced to use l'Hospital's rule repeatedly. Any terms of the form F' are a shorthand notation for $\partial F/\partial\beta_1$. We find

$$S = \frac{2(A_1 + A_3 + A_2b + A_3b)}{\alpha_1 + \alpha_2b} - \frac{2A_3p_0(b^2q_1 + q_2)}{(1+b)(\alpha_1 + \alpha_2b)} - \frac{2p_0C'_p(A_2(1+b)q_2 + A_3((1+b)q_2(1+r'_1) + b(b^2q_1 + q_2)r_1'^2))}{(1+b)(\alpha_1 + \alpha_2b)r_1'^2(1+br'_1)} - \frac{2p_0C'_q(A_1(1+b)q_1 + A_3((1+b)q_1(1+bs'_1) + (b^2q_1 + q_2)s_1'^2))}{(1+b)(\alpha_1 + \alpha_2b)s_1'^2(1+s'_1)} \quad (3.28)$$

where

$$p_0 = \frac{r'_1s'_1}{C'_nr'_1 + C'_ps'_1 + r'_1s'_1}$$

$$C'_p = \frac{(1 + K_n) D'_p}{(1 - K_p K_n) (1 - r'_2) (A_1 + A_2 + A_3)}$$

$$C'_n = \frac{(1 + K_p) D'_n}{(1 - K_p K_n) (1 - bs'_2) (A_1 + A_2 + A_3)}$$

$$K_p = \frac{D'_p}{(1 - r'_2) \left(\frac{1}{b} + r'_1\right) (A_1 + A_2 + A_3)}$$

$$K_n = \frac{D'_n}{(1 - bs'_2) (1 + s'_1) (A_1 + A_2 + A_3)}$$

$$D'_p = \frac{A_1 + A_1 b + A_3 \bar{q}_2}{b}$$

$$D'_n = \frac{A_2 + A_2 b + A_3 b \bar{q}_1}{b}$$

$$(r'_1, r'_2) = \left(\frac{1}{2b(A_1 + A_2 + A_3)} \right) \left\{ -A_1 - A_3 + b(A_2 + A_3) - A_2 q_2 \right. \\ \left. \pm \left[(A_1 + A_3 - b(A_2 + A_3))(1 - 2q_2) + A_2 q_2 \right]^2 \right. \\ \left. + 4bq_2 \bar{q}_2 (A_2 + A_3) (A_2 + b(A_2 + A_3)) \right\}^{\frac{1}{2}}$$

$$(s'_1, s'_2) = \left(\frac{1}{2b(A_1 + A_2 + A_3)} \right) \left\{ A_1 + A_3 - b(A_2 + A_3) - A_1 b q_1 \right. \\ \left. \pm \left[(A_1 + A_3 - b(A_2 + A_3))^2 \right. \right. \\ \left. \left. + 2bq_1 (A_1^2 + 4A_1 A_2 + 6A_1 A_3 + 4A_2 A_3 + 4A_3^2 \right. \right. \\ \left. \left. + 2A_1 A_2 b + 2A_1 A_3 b + A_1^2 b q_1) \right] \right\}^{\frac{1}{2}}$$

3.5.3 Continuous Time, Continuous State

Finally, we solve a continuous time, continuous state (CC) model by taking limits on both α_i and β_i . This can be done either by going first to the CD (α_i) or DC (β_i) model from DD, and then finishing by taking limits on the other variable. The final equation for speedup is given below.

$$S = 2 \left(1 - \frac{\bar{a}q_2 p_0 C'_p}{(1 + br_1')(a + \bar{a}b)r_1'^2} - \frac{aq_1 p_0 C'_n}{(1 + s_1')(a + \bar{a}b)s_1'^2} \right) \quad (3.29)$$

where

$$p_0 = \frac{r_1' s_1'}{C'_n r_1' + C'_p s_1' + r_1' s_1'}$$

$$C'_p = \frac{a(1+b)(1+K_n)}{b(1-K_pK_n)(1-r_2')}$$

$$C'_n = \frac{\bar{a}(1+b)(1+K_p)}{b(1-K_pK_n)(1-bs_2')}$$

$$K_p = \frac{a(1+b)}{(1+br_1')(1-r_2')}$$

$$K_n = \frac{\bar{a}(1+b)}{b(1+s_1')(1-bs_2')}$$

$$(r'_1, r'_2) = \frac{-1 + \bar{a}b + \bar{a}q_2 \pm \sqrt{1 + 2\bar{a}b + \bar{a}^2b^2 - 2\bar{a}q_2 - 4\bar{a}bq_2 + 2\bar{a}^2bq_2 + \bar{a}^2q_2^2}}{2b}$$

$$(s'_1, s'_2) = \frac{a - b + ab\bar{q}_1 \pm \sqrt{a^2 + 2ab + b^2 - 4ab\bar{q}_1 + 2a^2b\bar{q}_1 - 2ab^2\bar{q}_1 + a^2b^2\bar{q}_1^2}}{2b}$$

3.6 Previous Work on 2-Processor Models

There has been some similar work on two processor Time Warp models. Lavenberg, Muntz and Samadi [LMS83] used a continuous time, continuous state model to solve for the speedup (S_{ims}) of two processors over one processor. Their work resulted in an approximation for speedup that was valid only for $0 \leq q_i \leq 0.05$, where q_i is the probability that processor i will send a message to the other processor. Our result for this CC case is exact, has no restrictions on any of the parameters and therefore subsumes their work. In fact, we can compare our results directly for a simplified case where $a = 1/2$ (same processing rate for both processors), $b = 1$ (same average jump in virtual time

for both) and $q_1 = q_2 = q$ (same probability of sending a message), which we refer to as the symmetric, balanced case. Lavenberg et al. derive the following approximation for speedup in this case.

$$S_{lms} \approx 2 - \sqrt{2q}$$

Our equation for speedup in this restricted case is exactly

$$S = \frac{2(\sqrt{8+q} - \sqrt{q})}{\sqrt{8+q} + \sqrt{q}}$$

If we expand this formula using a power series about the point ($q = 0$) and list only the first few terms, we see the essential difference between our result and Lavenberg et al.

$$S \approx 2 - \sqrt{2q} + \frac{q}{2} - \frac{\sqrt{2}q^{\frac{3}{2}}}{16} + O(q^{\frac{5}{2}})$$

This clearly shows that the Lavenberg et al. result matches ours in the first two terms. Figure 3.3 shows the Lavenberg et al. result and our result compared to simulation with 99% confidence intervals.

Mitra and Mitrani [MM84] also solve a two processor model but use a discrete time, continuous state approach. They solve for the distribution of the separation between the two processors and the rate of progress of the two. In the definition of their model, a processor sends a message (with probability q_i) **before** advancing. Our model has a processor send a message **after** advancing. This difference between the two models disappears in the calculation of the average rate of progress. Their solution allows a general continuous distribution for the state jumps (virtual time), but requires (deterministic) single steps for the discrete time. In each time slot both processors always advance forward in virtual time some arbitrary distance. In our model this is equivalent to setting $\alpha_1 = \alpha_2 = 1$. Since our analysis only supports an exponential distribution for

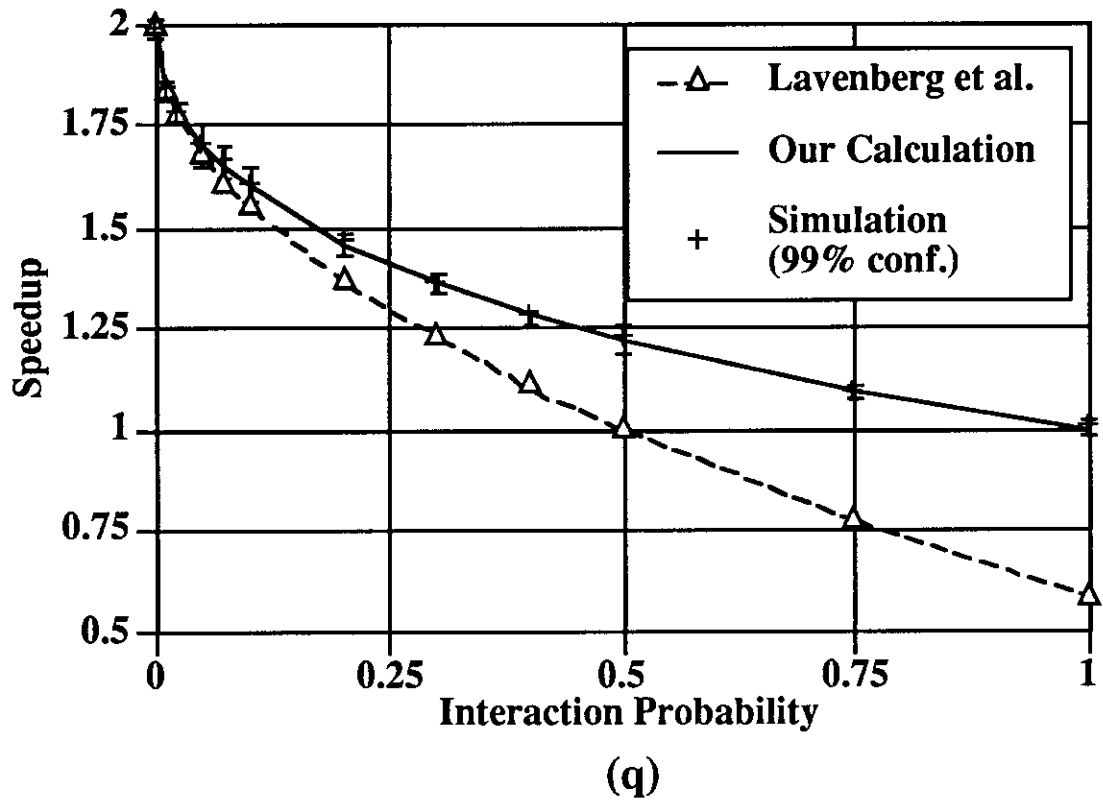


Figure 3.3: Comparison of speedup results for a simplified case.

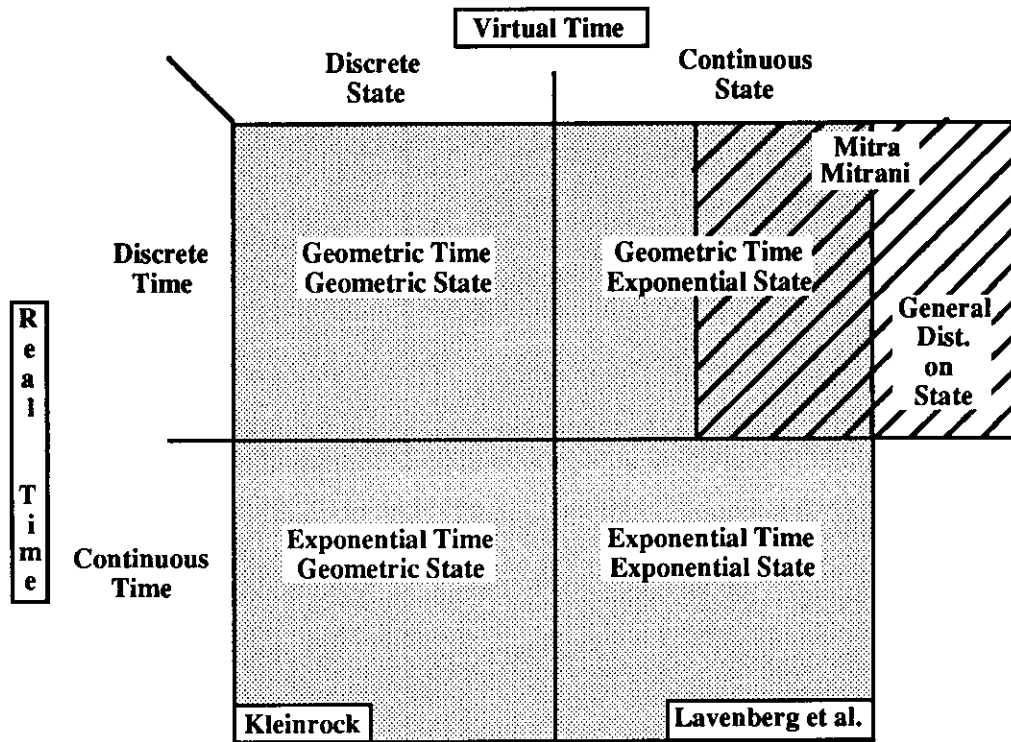


Figure 3.4: Previous work.

state changes, but their analysis doesn't have a distribution on time, neither model subsumes the other.

Finally, the DD and CD models have not appeared in the literature, although an early version of this work has been published by Kleinrock [Kle89]. It is a simplified version of the CD model where $\beta_1 = \beta_2 = 1$ (which is single step state jumps). Figure 3.4 shows how all of this work fits together. The work discussed in this paper covers the shaded region.

3.7 Results for a Restricted Model

In order to better understand our results, we examine a restricted version of the CD model (i.e. the model analyzed in [Kle89]). In this less general model we eliminate two variables by forcing each processor to advance **exactly** one virtual time unit each time it advances ($\beta_1 = \beta_2 = 1$). Again, we define q_i as the interaction parameter; the probability that processor i sends a message to the other processor. We also define a as the ratio $\frac{\lambda_i}{\lambda_1 + \lambda_2}$ where λ_i is the rate for the continuous time distribution for processor i (rate at which messages are processed). The parameter a can be thought of as a measure of “load balancing”. When $a = 1/2$ the load is balanced.

The solution for this simplified system is given below.

$$S = 2 \left(1 - p_0 \left(\frac{\bar{a}q_2}{(r_1 - 1)^2} + \frac{aq_1}{(s_1 - 1)^2} \right) \right)$$

$$p_0 = \frac{1}{1 + \left(\frac{1}{r_1 - 1}\right) + \left(\frac{1}{s_1 - 1}\right)}$$

$$r_1 = \frac{1 + \sqrt{1 - 4a\bar{a}q_2}}{2a}$$

$$s_1 = \frac{1 + \sqrt{1 - 4a\bar{a}q_1}}{2\bar{a}}$$

The equations above indicate that speedup reaches a maximum value of two when $q_1 = q_2 = 0$ (no interaction). Since neither processor hinders the other, we can exploit the full potential of each processor. In general one might assume that the speedup would simply be twice the speed of the slowest processor. In fact, the system does a little bit worse. Even though one processor might be

faster than the other, it is possible (stochastically) that the slower processor gets ahead of the faster one. At this point it is possible that the faster processor could cause the slower one to rollback. Overall therefore, the speedup is less than twice the speed of the slower processor on average.

Figure 3.5 shows the speedup for the symmetric case where $q_1 = q_2 = q$, though it does not show the discontinuity in the function S at $q = 0$. For $q = 0$, $S = 2$ for all a and so S is discontinuous for all $a \neq 1/2$. This is not shown in the figure. For $q = 0$ no messages are sent, therefore no rollbacks will occur, and it is clear that $S = 2$. For $q > 0$ as $a \rightarrow 0$ or $a \rightarrow 1$ ($\lambda_1 \rightarrow 0$ or $\lambda_2 \rightarrow 0$), then the speedup goes to zero as shown in the figure. This occurs because one process moves extremely slowly (compared to the equivalent single process) and it will eventually drag the faster process back to its lagging position. The TW system moves at less than twice the speed of the slowest processor, while the equivalent single processor moves at the average rate $\frac{\lambda_1 + \lambda_2}{2}$. It is clear that load balancing is extremely important since good speedup only occurs near $a = 1/2$. Notice that the interaction parameter is important when a is near $1/2$.

Figure 3.6 shows the speedup for the balanced case where $\lambda_1 = \lambda_2$. Note that the speedup is 2 for $q_1 = q_2 = 0$ and goes to $4/3$ for $q_1 = q_2 = 1$. Specifically, it never goes below one. We always get speedup with two processors as long as $a = 1/2$.

Figure 3.7 shows the speedup for the extremely simplified symmetric, balanced case where $q_1 = q_2 = q$ and $\lambda_1 = \lambda_2 = \lambda$. For this special case the formula for speedup is

$$S = \frac{4}{2 + \sqrt{q}}$$

Note for $q = 0$ that $S = 2$ and for $q = 1$ we have $S = 4/3$. We can see this last

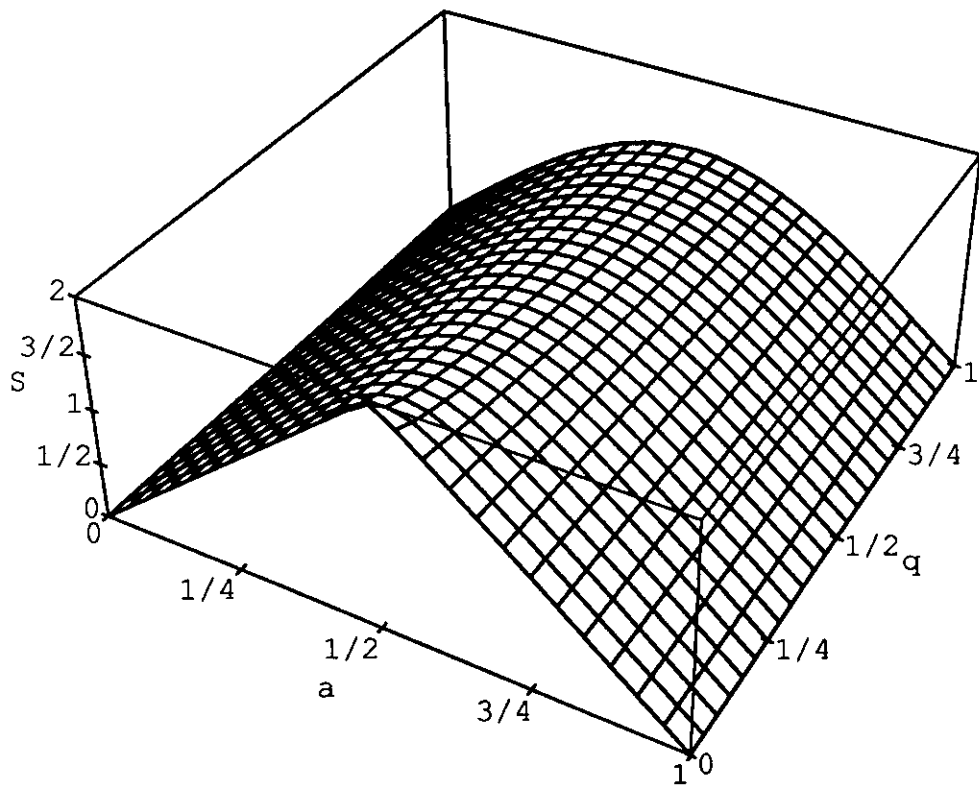


Figure 3.5: Speedup for the symmetric case $q_1 = q_2 = q$

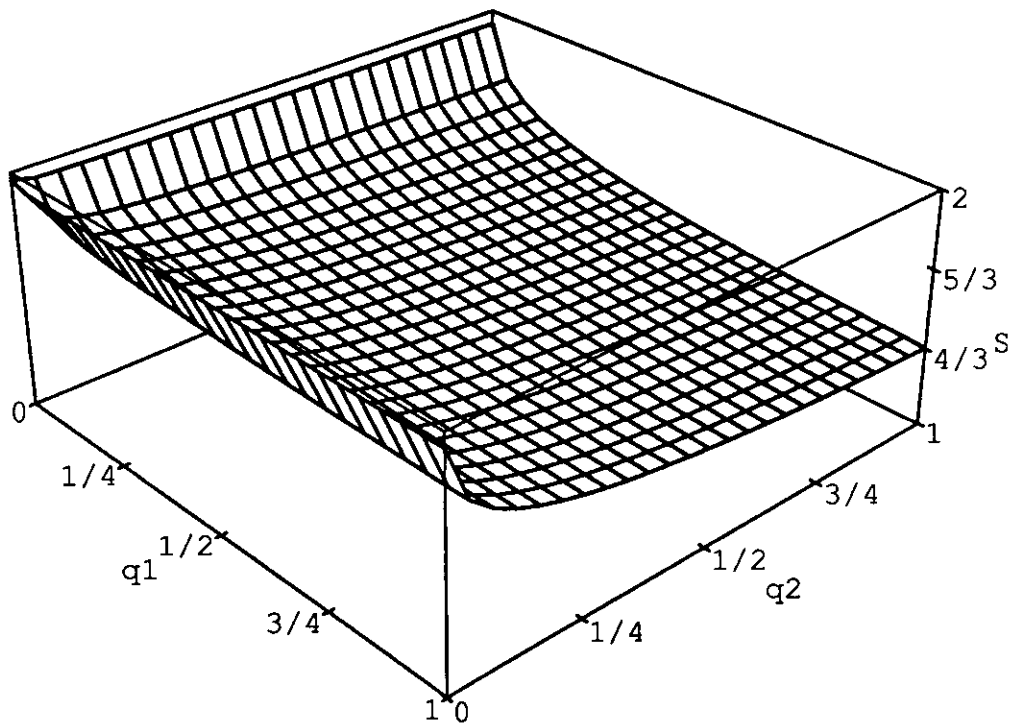


Figure 3.6: Speedup for the balanced case $\lambda_1 = \lambda_2 = \lambda$.

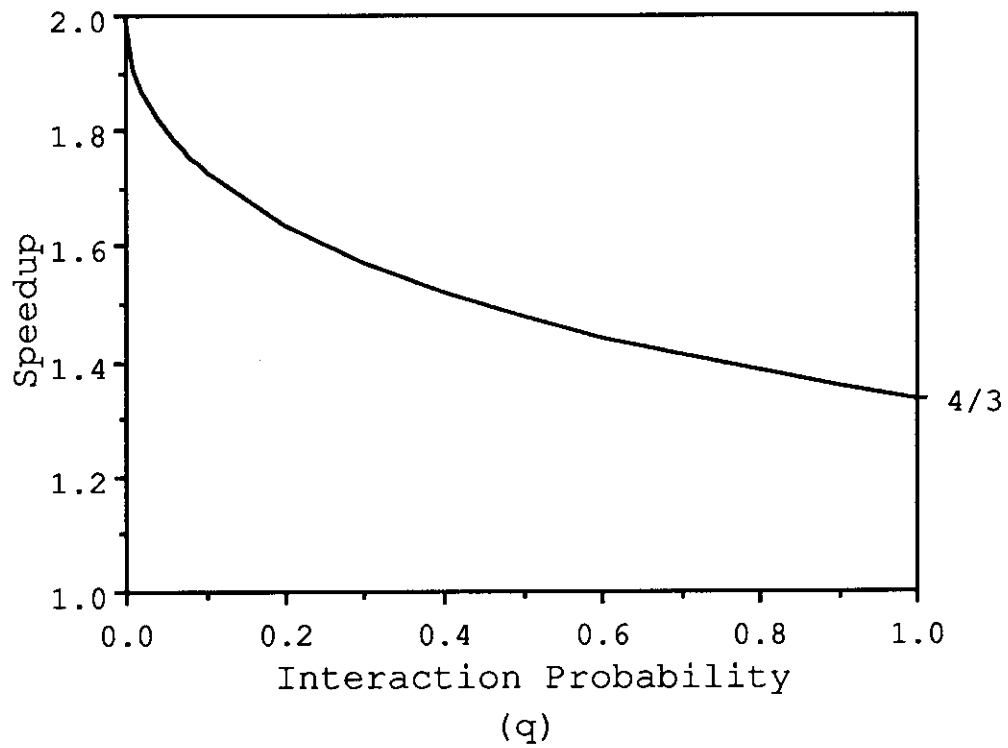


Figure 3.7: Speedup for the symmetric, balanced case $q_1 = q_2 = q$ and $\lambda_1 = \lambda_2 = \lambda$.

result intuitively. Since each process always sends a message to the other after it advances, then the time for both processes to advance one unit is equal to the maximum of two exponential delays at rate λ which is $3/2$ times $1/\lambda$. Thus, the rate of progress for each process is simply $\frac{2\lambda}{3}$. Since both are moving at this rate, the sum equals $\frac{4\lambda}{3}$ while $R_1 = \lambda$ which yields $S = 4/3$ for $q = 1$. The curve shown in Figure 3.7 is the “spine” of the surface plotted in Figure 3.5 and is the “45 degree” line ($q_1 = q_2$) of the surface plotted in Figure 3.6.

3.7.1 Optimality Proofs

Using the simple model described above, we prove several results about optimality with respect to the parameters of the system. We first show that the speedup is monotonically decreasing in both q_1 and q_2 , the interaction parameters (i.e. q_1 and q_2 should be as small as possible). We do this by showing that $\frac{\partial S}{\partial q_i}$ is negative. If we differentiate S with respect to q_1 we arrive at the following formula

$$\frac{\partial S}{\partial q_1} = \Phi(q_1, q_2, a) \left(-(-1 + 2a)^2 - 2a\bar{a}q_1 + (1 - 2a)\sqrt{1 - 4a\bar{a}q_1} \right)$$

where $\Phi(q_1, q_2, a)$ is a non-negative function of q_1, q_2, a and is given below

$$\Phi(q_1, q_2, a) = \frac{128a^3\bar{a}^3q_2}{(-1 + 2\bar{a} - f(\bar{q}_1))^2 f(\bar{q}_1) \left(-1 + 4a\bar{a} - f(\bar{q}_1) - \sqrt{f(\bar{q}_2)} - f(\bar{q}_1)\sqrt{f(\bar{q}_2)} \right)^2}$$

where

$$f(x) = \sqrt{1 - 4a\bar{a}x}$$

In order to show that $\frac{\partial S}{\partial q_1}$ is negative, we must show that

$$-(-1 + 2a)^2 - 2a\bar{a}q_1 + (1 - 2a)\sqrt{1 - 4a\bar{a}q_1} \leq 0 \quad (3.30)$$

When $a \geq \frac{1}{2}$ Equation 3.30 is trivially satisfied. Our concern is in the range $0 \leq a < \frac{1}{2}$, in which case our condition becomes

$$\begin{aligned} -(-1 + 2a)^2 - 2a\bar{a}q_1 &\leq -(1 - 2a)\sqrt{1 - 4a\bar{a}q_1} < 0 \\ \left(-(-1 + 2a)^2 - 2a\bar{a}q_1\right)^2 &\geq \left(-(1 - 2a)\sqrt{1 - 4a\bar{a}q_1}\right)^2 \\ 4a^2q_1^2 - 8a^3q_1^2 + 4a^4q_1^2 &\geq 0 \\ 4a^2\bar{a}^2q_1^2 &\geq 0 \end{aligned}$$

which is trivially true. A similar (symmetric) proof for q_2 is omitted here.

Optimization with respect to a is a little more difficult. When we differentiate S with respect to a we get such a complicated formula that it is prohibitive to solve for the optimum value of a . Fortunately, by plotting S versus a , q_1 and q_2 (Figures 3.5 and 3.6) we see that S is unimodal and that the optimum value of a is $1/2$ ($\lambda_1 = \lambda_2$). When we plug this value ($a = 1/2$) into $\frac{\partial S}{\partial a}$ we see that the result is 0.

$$\frac{\partial S}{\partial a}|_{a=\frac{1}{2}} = \frac{2(-((1 - \bar{q}_1)q_2) + q_1(1 - \bar{q}_2))}{(1 - \bar{q}_1)(1 - \bar{q}_2)} = 0$$

To show that this is a maximum we must show that the second derivative is negative at $a = 1/2$.

$$\frac{\partial^2 S}{\partial q_1^2}|_{a=\frac{1}{2}} = \frac{-8(\sqrt{q_1} + \sqrt{q_2})(q_1 + \sqrt{q_1}\sqrt{q_2} + 2q_1\sqrt{q_2} + q_2 + 2\sqrt{q_1}q_2 + q_1q_2)}{\sqrt{q_1}(\sqrt{q_1} + \sqrt{q_2} + \sqrt{q_1}\sqrt{q_2})^2\sqrt{q_2}} \quad (3.31)$$

Equation 3.31 is clearly negative since the numerator is negative and the denominator is positive. For the more general case, where the processors are not restricted to single step advances, we find from analyzing plots of speedup that the above result ($a = 1/2$ ($\lambda_1 = \lambda_2$) for optimal performance) generalizes to

$$\frac{\lambda_1}{\beta_1} = \frac{\lambda_2}{\beta_2} \quad \text{or} \quad b = \frac{a}{1 - a} \quad (3.32)$$

meaning that the average “unfettered” rate of progress in virtual time for each processor should be the same. For a fixed value of a the best performance can be found when Equation 3.32 is true, and overall best performance is found at $a = 1/2$ with Equation 3.32 holding true.

We have not seen this result before in the literature since the two processor models haven’t been general enough. It says that for optimum performance we would like to place tasks on processors such that the average “independent” rate of progress in virtual time is the same for both processors. Ideally we want this to be true while also having each processor execute events at the same rate ($a = 1/2$). This result is generally applicable to systems consisting of more than two processors. The intuition is that if every processor tends to move forward in virtual time at the same rate as the others, then the processors will remain nearly synchronized without suffering a large penalty for rollbacks.

3.7.2 Adding a Cost for State Saving

One simple way of examining how state saving overhead affects the performance of the system is to modify the value of R_1 , the rate of progress on a single processor. We introduce a parameter c ($c \geq 1$) that indicates how much faster events are executed without state saving. If $c = 2$ an event completes twice as fast on average without state saving. Since our model requires that each processor save its state after every event, we can think of each event as taking longer to complete in the TW system than in the single processor system where no state saving is required. We note here that this is actually an upper bound on the cost for state saving in this two processor system. Lin and Lazowska [LL90c] have shown that to achieve minimal state saving costs, TW should save

state less often than after every event. This result depends on certain system parameters, most notably the cost for state saving. We make no attempt to optimize the frequency of state saving, nonetheless this simple model provides some interesting results as shown below.

By examining the CD model with the single step restriction (as above) we arrive at the following value for R_1

$$R_1 = \frac{c(\lambda_1 + \lambda_2)}{2}$$

For this model we find that the new formula for speedup is simply $1/c$ times the old value. Let us examine a very simple case in detail. If we look at the symmetric, balanced case, the updated formula for speedup is

$$S = \frac{4}{c(2 + \sqrt{q})}$$

It is easy to see that as $c \rightarrow \infty$ speedup will go to zero. For $c \geq 2$ Time Warp with two processors is always worse than running on one processor without TW. Conversely, for $c \leq 4/3$ TW always wins out. The interesting range is $4/3 \leq c \leq 2$. In this range, certain values of q will yield speedup, while others won't. We are most concerned with the boundary where $S = 1$ which is the transition from areas where TW on two processors helps to where it hurts. Setting $S = 1$ and solving for q we find the necessary condition for two processors running TW to be faster than the single (non-TW) processor.

$$q \leq \frac{4(2 - c)^2}{c^2}$$

Figure 3.8 shows the regions in the $c - q$ plane where TW on two processors is effective and where it is not. Thus, if we know the values of both c and q for our symmetric, balanced system we can immediately tell whether the application will run faster under Time Warp on two processors.

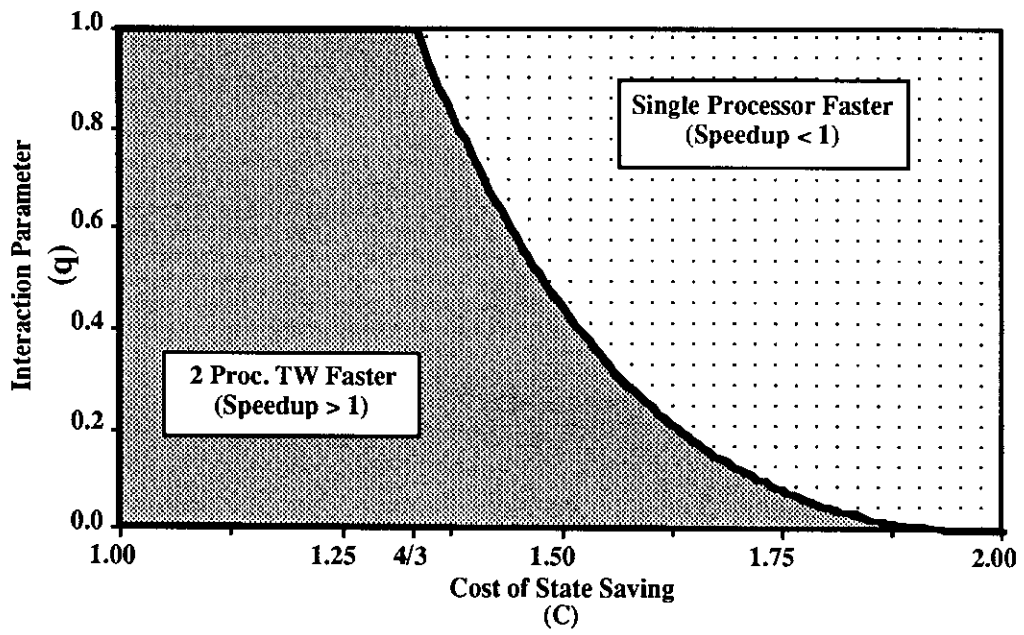


Figure 3.8: The cost of state saving and its effect on performance.

3.8 Conclusions

In this chapter we have created a model for two processor Time Warp execution and provided the results of its exact solution. The model is general enough to subsume the work of Lavenberg, Muntz and Samadi [LMS83] and to partially subsume the work of Mitra and Mitrani [MM84]. Further, we examined a simplified version of our model and showed for optimal performance that the processors should send as few messages as possible. Further, q (the interaction parameter) has a large effect on speedup for when the load is balanced and speedup changes rapidly when q is near zero. Tasks should be placed on processors such that the average “independent” rate of progress in virtual time is the same for both processors to achieve good speedup. Ideally we want this to be true while also having each processor process events at the same rate ($\lambda_1 = \lambda_2$). Finally, we addressed the cost of state saving by using a very simple extension to the model, and examined its effect on performance. Small state saving costs or infrequent message interactions indicate that TW is effective in gaining speedup.

CHAPTER 4

Two Processor Message Queueing Model

4.1 Introduction

The model introduced in the previous chapter ignored any messages that arrive in the virtual time future of the receiver. There are many simulation models where the messages actually carry the work. These messages must be queued before processing and the memory costs of queueing these messages is an important performance measure. In this chapter we create and analyze a model that accounts for the queueing and processing of all messages passed between processors.

4.2 The Message Queueing Model

Assume we have a job that is partitioned into two processes, each of which is executed on a separate processor. A process (say process i) at virtual time v operates by first executing any message in its input queue with timestamp v and then executing its locally scheduled work. After completing its local work at virtual time v , the process advances its clock one unit and will then send a message to the other process with probability q_i . Note that we only allow single step state advances which is less general than the model presented in Chapter 3. The process places its current virtual time on any message it sends. We will

```

1  Set local clock ( $v$ ) to 0.
2  Execute local events for  $v=0$ .
3  With probability  $q(i)$ , send message stamped with 1.

REPEAT*
4  Advance local clock to  $v=v+1$ .
5  Process message in queue with timestamp =  $v$  (if it exists).
6  Execute local event for time  $v$ .
7  With probability  $q(i)$ , send message stamped with  $v+1$ .
UNTIL ( $v \geq \text{MAXTIME}$ )

* If a message arrives at any time with a timestamp ( $tm \leq v$ ):
- set local clock to  $tm$ 
- goto line 5 and continue from there

```

Figure 4.1: Code executed by each processor.

restrict the virtual times in our system to have integer values (i.e., 0, 1, 2, ...). A process will schedule an event for itself at every point in virtual time. This means that a process will have its own work to do at every point in virtual time, and occasionally will have work sent to it from the other process. If a message arrives with a timestamp v equal to or smaller than the local clock time of the receiving processor, that processor is forced to rollback (discarding any work performed at a virtual time greater than or equal to v), executes the arriving message, then proceeds forward again from virtual time v . We show the execution sequence for each LP in Figure 4.1. Note that there can be at most one queued message for a process at any (integer) virtual time v .

More formally, we define two processes each executing on a separate processor. As these processes are executed, we consider that they visit the integers on

the x-axis each beginning at $x = 0$ at time $t = 0$. To process a queued message, each processor takes an exponentially distributed amount of time with mean $1/\mu_i$ ($i = 1, 2$). Executing locally generated work takes an exponentially distributed amount of time with mean $1/\lambda_i$ ($i = 1, 2$). We assume that $\mu_i = f\lambda_i$ where $0 < f \leq \infty$. If $f = \infty$ then messages take zero time to process and are only used for synchronization as in the previous model. As $f \rightarrow 0$ messages become extremely expensive to process relative to the local work that must be performed. The parameter f essentially allows the modelling of a spectrum of systems from self-initiating ($f \rightarrow \infty$) to message-initiating ($f \rightarrow 0$).

After process i makes an advance along the axis, it will send a message to the other process with probability q_i ($i = 1, 2$). This message carries a timestamp that is the time of the sender after making the advance. Upon receiving a message from the sending process, this receiving process will do the following:

- 1: If its position along the x-axis is behind the sending process, it queues the message.
- 2: If its position is equal to or ahead of the sending process, it will immediately move back (i.e., “rollback”) along the x-axis to the current position of the sending process and begin to process that message. All work completed at a virtual time greater than or equal to its current position is discarded and must be re-executed.

Let $F(t)$ = the position of the First process (process one) at time t and let $S(t)$ = the position of the Second process (process two) at time t . Further, let

$$D(t) = F(t) - S(t)$$

$D(t) = 0$ whenever Case 2 occurs (i.e., a rollback). As before, we are interested in studying the Markov process $D(t)$ whose state diagram is shown in Figure 4.2.

We will solve for

$$P[\text{Processors separated by } k \text{ units of virtual time}] = \lim_{t \rightarrow \infty} P[D(t) = k] \quad -\infty < k < \infty$$

namely, the equilibrium probability for the Markov chain $D(t)$. In order to find the solution, we split the chain into six regions.

$$P_k = \lim_{t \rightarrow \infty} P[D(t) = k \text{ and Processor 2 is **not** processing a msg}] \quad k \geq 1$$

$$Q_k = \lim_{t \rightarrow \infty} P[D(t) = -k \text{ and Processor 1 is **not** processing a msg}] \quad k \geq 1$$

$$S_k = \lim_{t \rightarrow \infty} P[D(t) = k \text{ and Processor 2 is processing a msg}] \quad k \geq 0$$

$$R_k = \lim_{t \rightarrow \infty} P[D(t) = -k \text{ and Processor 1 is processing a msg}] \quad k \geq 0$$

$$N_0 = \lim_{t \rightarrow \infty} P[D(t) = 0 \text{ and neither is processing a msg}]$$

$$B_0 = \lim_{t \rightarrow \infty} P[D(t) = 0 \text{ and both are processing a msg}]$$

Using our solution, we will go on to solve for some interesting performance measures including the average rate of progress of the two processor system.

There are some implicit assumptions in our description. Our model assumes that states are stored after **every** event, otherwise a rollback would not necessarily send the processor back to the time of the tardy message; rather it might have to rollback to a much earlier time, namely, that of the last saved state. When process i causes the other process to rollback, process i immediately discards any messages it has queued in its future. This is as if the rolled back processor is able to transmit anti-messages instantaneously. This is not an unrealistic assumption in a shared-memory environment [Fuj89b]. Another implicit assumption is that each process always schedules events for itself. We also assume that communication between processors incurs no delay from transmission

to reception. Finally, the interaction between the processes is probabilistic.

4.3 Analysis of the Message Queueing Model

In this section we provide the exact solution for the continuous time, discrete state model introduced in Section 4.2. First, some definitions are in order.

$$\lambda_i = \text{Rate at which Processor } i \text{ processes local events}$$

$$\mu_i = f\lambda_i = \text{Rate at which Processor } i \text{ processes messages}$$

$$a = \frac{\lambda_1}{\lambda_1 + \lambda_2}$$

$$\bar{a} = \frac{\lambda_2}{\lambda_1 + \lambda_2} = 1 - a$$

$$A = a + \bar{a}f$$

$$B = \bar{a} + af$$

$$q_i = \text{P}[} i^{\text{th}} \text{ processor sends a message after advancing]}$$

$$\bar{q}_i = 1 - q_i$$

Referring to the state diagram for this system shown in Figure 4.2, we find the following balance equations.

The balance equations for our system are:

$$P_k = aP_{k-1} + \bar{a}\bar{q}_2\bar{q}_1P_{k+1} + \bar{a}fS_k \quad k \geq 2 \quad (4.1)$$

$$P_1 = aN_0 + \bar{a}\bar{q}_2\bar{q}_1P_2 + \bar{a}fS_1 \quad (4.2)$$

$$Q_k = \bar{a}Q_{k-1} + a\bar{q}_1\bar{q}_2Q_{k+1} + afR_k \quad k \geq 2 \quad (4.3)$$

$$Q_1 = \bar{a}N_0 + a\bar{q}_1\bar{q}_2Q_2 + afR_1 \quad (4.4)$$

$$N_0 = a\bar{q}_1\bar{q}_2Q_1 + \bar{a}\bar{q}_2\bar{q}_1P_1 + afR_0 + \bar{a}fS_0 \quad (4.5)$$

$$fB_0 = \bar{a}q_1q_2 \sum_{i=1}^{\infty} P_i + aq_1q_2 \sum_{i=1}^{\infty} Q_i \quad (4.6)$$

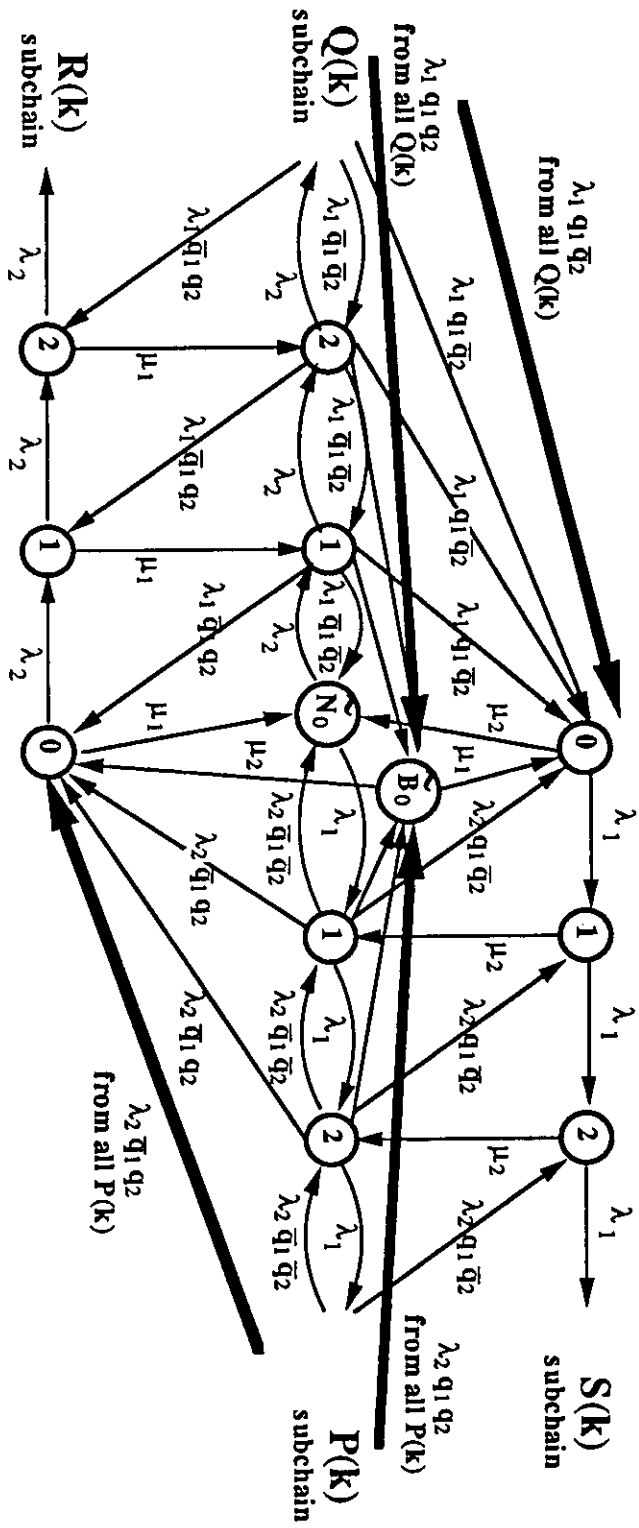


Figure 4.2: State diagram for the message queuing model.

$$AS_k = aS_{k-1} + \bar{a}\bar{q}_2q_1P_{k+1} \quad k > 0 \quad (4.7)$$

$$AS_0 = \bar{a}\bar{q}_2q_1P_1 + aq_1\bar{q}_2 \sum_{i=1}^{\infty} Q_i + afB_0 \quad (4.8)$$

$$BR_k = \bar{a}R_{k-1} + a\bar{q}_1q_2Q_{k+1} \quad k > 0 \quad (4.9)$$

$$BR_0 = a\bar{q}_1q_2Q_1 + \bar{a}q_2\bar{q}_1 \sum_{i=1}^{\infty} P_i + \bar{a}fB_0 \quad (4.10)$$

$$1 = \sum_{i=1}^{\infty} P_i + \sum_{i=1}^{\infty} Q_i + \sum_{i=0}^{\infty} S_i + \sum_{i=0}^{\infty} R_i + N_0 + B_0 \quad (4.11)$$

This system will have a steady-state solution if $\lambda_i > 0$, $q_i > 0$ and $f > 0$. These are fairly straightforward restrictions. The λ_i must be greater than zero or the system makes no progress at all. The q_i must be greater than zero so that there is some probability that a processor will be rolled back once it gets ahead. This maintains a finite expectation for $D(t)$. Finally, f must be greater than zero so that when a message is being processed the system will eventually complete the operation.

We define the following z -transforms (note the different ranges on k).

$$\begin{aligned} P(z) &= \sum_{k=1}^{\infty} P_k z^k & Q(z) &= \sum_{k=1}^{\infty} Q_k z^k \\ S(z) &= \sum_{k=0}^{\infty} S_k z^k & R(z) &= \sum_{k=0}^{\infty} R_k z^k \end{aligned}$$

Using the above equations we can solve for $P(z)$, $Q(z)$, $S(z)$ and $R(z)$ by multiplying the appropriate equation by z^k and summing over the applicable range of k . To simplify the expressions we define the following constants.

$$F_S = \bar{a}q_2P(1) + (1 - \bar{a}q_2)Q(1)$$

$$F_R = aq_1Q(1) + (1 - aq_1)P(1)$$

Solving for $P(z)$ in terms of $S(z)$ we get

$$P(z) = \frac{z(-AS(z)\bar{a}f) + F_S a\bar{a}f q_1 + P_1 \bar{a}(A - aq_1)\bar{q}_2 - AN_0 a z}{A(\bar{a}\bar{q}_1\bar{q}_2 - z + az^2)} \quad (4.12)$$

and for $S(z)$ in terms of $P(z)$

$$S(z) = \frac{q_1 (P(z)\bar{a}\bar{q}_2 + F_S a z)}{z (A - a z)} \quad (4.13)$$

Solving them simultaneously we arrive at

$$P(z) = -\frac{z (- (F_S a^2 \bar{a} f q_1 z) + P_1 \bar{a} (A - a q_1) \bar{q}_2 (A - a z) - A N_0 a z (A - a z))}{A (- (\bar{a} (A - a q_1) \bar{q}_2) + (A + a \bar{a} \bar{q}_1 \bar{q}_2) z - (1 + A) a z^2 + a^2 z^3)}$$

and

$$S(z) = -\frac{P_1 \bar{a}^2 q_1 (A - a q_1) \bar{q}_2^2 - A N_0 a \bar{a} q_1 \bar{q}_2 z + F_S a q_1 (\bar{a} (A - a q_1) \bar{q}_2 - A z + A a z^2)}{A (- (\bar{a} (A - a q_1) \bar{q}_2) + (A + a \bar{a} \bar{q}_1 \bar{q}_2) z - (1 + A) a z^2 + a^2 z^3)}$$

The denominator polynomial, $D(z)$, for $P(z)$ may be factored as follows.

$$D(z) = A a^2 (z - r_1) (z - r_2) (z - r_3)$$

where r_1 , r_2 and r_3 are the roots of the cubic polynomial in $D(z)$.

$$\begin{aligned} r_1 &= \frac{1 + A - 2\sqrt{1 - A + A^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{2\pi + \theta_r}{3})}{3a} \\ r_2 &= \frac{1 + A - 2\sqrt{1 - A + A^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{\theta_r}{3})}{3a} \\ r_3 &= \frac{1 + A - 2\sqrt{1 - A + A^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{4\pi + \theta_r}{3})}{3a} \end{aligned}$$

Similar roots (s_1 , s_2 , s_3) for the denominator of $Q(z)$ can be written down directly

$$\begin{aligned} s_1 &= \frac{1 + B - 2\sqrt{1 - B + B^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{2\pi + \theta_s}{3})}{3\bar{a}} \\ s_2 &= \frac{1 + B - 2\sqrt{1 - B + B^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{\theta_s}{3})}{3\bar{a}} \\ s_3 &= \frac{1 + B - 2\sqrt{1 - B + B^2 - 3a\bar{a}\bar{q}_1\bar{q}_2} \cos(\frac{4\pi + \theta_s}{3})}{3\bar{a}} \end{aligned}$$

where

$$\theta_r = \arccos \left(\frac{-((A-2)(1+A)(2A-1)) + 9a\bar{a}\bar{q}_2(-3A+3aq_1+(1+A)\bar{q}_1)}{2(1-A+A^2-3a\bar{a}\bar{q}_1\bar{q}_2)^{\frac{3}{2}}} \right)$$

$$\theta_s = \arccos \left(\frac{-((B-2)(1+B)(2B-1)) + 9a\bar{a}\bar{q}_1(-3B+3\bar{a}q_2+(1+B)\bar{q}_2)}{2(1-B+B^2-3a\bar{a}\bar{q}_1\bar{q}_2)^{\frac{3}{2}}} \right)$$

See Appendix B for a derivation of the roots. We find that r_1 , r_2 and r_3 are real and that $|r_2| \leq 1$ while $|r_1|, |r_3| > 1$. Since $P(z)$ is the z -transform of a probability distribution, it must be analytic in the range $|z| \leq 1$, and we know that the numerator of $P(z)$, namely $N(z)$, must go to zero at $z = r_2$ since $D(z)$ goes to zero at $z = r_2$. We can use this fact to solve for P_1 , yielding

$$P_1 = \frac{ar_2(F_S a \bar{a} f q_1 + AN_0(A - ar_2))}{\bar{a}(A - aq_1)\bar{q}_2(A - ar_2)}$$

Substituting this value back into $N(z)$ we may write

$$N(z) = \frac{Aaz(z - r_2)(F_S a \bar{a} f q_1 + N_0(A - ar_2)(A - az))}{A - ar_2}$$

and thus

$$P(z) = \frac{z(F_S a \bar{a} f q_1 + N_0(A - ar_2)(A - az))}{a(A - ar_2)(r_1 - z)(r_3 - z)} \quad (4.14)$$

A similar procedure can be carried out on $S(z)$, resulting in

$$S(z) = \frac{q_1(N_0 \bar{a} \bar{q}_2 + F_S(1 - ar_2 - az))}{a(r_1 - z)(r_3 - z)} \quad (4.15)$$

Moreover, $Q(z)$ and $R(z)$ are symmetric in (λ_1, λ_2) , (μ_1, μ_2) and (q_1, q_2) to $P(z)$ and $S(z)$ so we can write them down directly.

$$Q(z) = \frac{z(F_R a \bar{a} f q_2 + N_0(B - \bar{a}s_2)(B - \bar{a}z))}{\bar{a}(B - \bar{a}s_2)(s_1 - z)(s_3 - z)} \quad (4.16)$$

$$R(z) = \frac{q_2(N_0 a \bar{q}_1 + F_R(1 - \bar{a}s_2 - \bar{a}z))}{\bar{a}(s_1 - z)(s_3 - z)} \quad (4.17)$$

Recalling that F_S and F_R are functions of both $P(1)$ and $Q(1)$, we see that $P(z)$ and $Q(z)$ are functions of $P(1)$, $Q(1)$ and N_0 . We solve for $P(1)$ and $Q(1)$

by solving Equations 4.14 and 4.16 simultaneously with $z = 1$.

$$P(1) = C_{pn_0}N_0 + C_{pp}P(1) + C_{pq}Q(1)$$

$$Q(1) = C_{qn_0}N_0 + C_{qp}P(1) + C_{qq}Q(1)$$

Therefore,

$$P(1) = C_P N_0 \quad Q(1) = C_Q N_0$$

where

$$C_P = \frac{C_{pn_0} + C_{pq}C_{qn_0} - C_{pn_0}C_{qq}}{1 - C_{pp} - C_{pq}C_{qp} - C_{qq} + C_{pp}C_{qq}}$$

$$C_Q = \frac{C_{qn_0} - C_{pp}C_{qn_0} + C_{pn_0}C_{qp}}{1 - C_{pp} - C_{pq}C_{qp} - C_{qq} + C_{pp}C_{qq}}$$

and

$$C_{pn_0} = \frac{\bar{a}f}{a(r_1 - 1)(r_3 - 1)}$$

$$C_{pp} = \frac{\bar{a}^2 f q_1 q_2}{(r_1 - 1)(A - ar_2)(r_3 - 1)}$$

$$C_{pq} = \frac{\bar{a}f q_1 (1 - \bar{a}q_2)}{(r_1 - 1)(A - ar_2)(r_3 - 1)}$$

$$C_{qn_0} = \frac{af}{\bar{a}(s_1 - 1)(s_3 - 1)}$$

$$C_{qp} = \frac{af(1 - aq_1)q_2}{(s_1 - 1)(B - \bar{a}s_2)(s_3 - 1)}$$

$$C_{qq} = \frac{a^2 f q_1 q_2}{(s_1 - 1)(B - \bar{a}s_2)(s_3 - 1)}$$

Noting that $P(1) + Q(1) + S(1) + R(1) + N_0 + B_0 = 1$ we solve for N_0 .

$$N_0 = \left[1 + \frac{(C_Q a + C_P \bar{a}) q_1 q_2}{f} + \frac{C_{F_S} a \bar{a} f q_1 + \bar{a} f (A - ar_2)}{a(r_1 - 1)(A - ar_2)(r_3 - 1)} + \frac{q_1 (\bar{a}q_2 + C_{F_S} (\bar{a} - ar_2))}{a(r_1 - 1)(r_3 - 1)} + \frac{C_{F_R} a \bar{a} f q_2 + af(B - \bar{a}s_2)}{\bar{a}(s_1 - 1)(B - \bar{a}s_2)(s_3 - 1)} + \frac{q_2 (a\bar{q}_1 + C_{F_R} (a - \bar{a}s_2))}{\bar{a}(s_1 - 1)(s_3 - 1)} \right]^{-1} \quad (4.18)$$

Finally, by inverting the transforms we find the probability of being in any state (other than \tilde{N}_0).

$$P_k = K_1 \left(\frac{1}{r_1}\right)^k + K_2 \left(\frac{1}{r_3}\right)^k \quad k \geq 1 \quad (4.19)$$

$$Q_k = K_3 \left(\frac{1}{s_1}\right)^k + K_4 \left(\frac{1}{s_3}\right)^k \quad k \geq 1 \quad (4.20)$$

$$S_k = K_5 \left(\frac{1}{r_1}\right)^k + K_6 \left(\frac{1}{r_3}\right)^k \quad k \geq 0 \quad (4.21)$$

$$R_k = K_7 \left(\frac{1}{s_1}\right)^k + K_8 \left(\frac{1}{s_3}\right)^k \quad k \geq 0 \quad (4.22)$$

$$B_0 = \frac{N_0 (C_Q a + C_P \bar{a}) q_1 q_2}{f} \quad (4.23)$$

where

$$K_1 = \frac{N_0 (C_{F_S} a \bar{a} f q_1 + (A - ar_1) (A - ar_2))}{a (A - ar_2) (r_3 - r_1)}$$

$$K_2 = \frac{N_0 (C_{F_S} a \bar{a} f q_1 + (A - ar_2) (A - ar_3))}{a (A - ar_2) (r_1 - r_3)}$$

$$K_3 = \frac{N_0 (C_{F_R} a \bar{a} f q_2 + (B - \bar{a} s_1) (B - \bar{a} s_2))}{\bar{a} (B - \bar{a} s_2) (s_3 - s_1)}$$

$$K_4 = \frac{N_0 (C_{F_R} a \bar{a} f q_2 + (B - \bar{a} s_2) (B - \bar{a} s_3))}{\bar{a} (B - \bar{a} s_2) (s_1 - s_3)}$$

$$K_5 = \frac{N_0 q_1 (\bar{a} \bar{q}_2 + C_{F_S} (1 - ar_1 - ar_2))}{ar_1 (r_3 - r_1)}$$

$$K_6 = \frac{N_0 q_1 (\bar{a} \bar{q}_2 + C_{F_S} (1 - ar_2 - ar_3))}{a (r_1 - r_3) r_3}$$

$$K_7 = \frac{N_0 q_2 (a\bar{q}_1 + C_{FR} (1 - \bar{a}s_1 - \bar{a}s_2))}{\bar{a}s_1 (s_3 - s_1)}$$

$$K_8 = \frac{N_0 q_2 (a\bar{q}_1 + C_{FR} (1 - \bar{a}s_2 - \bar{a}s_3))}{\bar{a} (s_1 - s_3) s_3}$$

$$C_{FS} = C_P \bar{a} q_2 + C_Q (1 - \bar{a} q_2)$$

$$C_{FR} = C_Q a q_1 + C_P (1 - a q_1)$$

This completes the calculation of the explicit expressions for the equilibrium state probabilities of the Markov chain for the message queuing model.

4.4 Performance Measures

Using the solution to the Markov chain that was calculated above, we may solve for almost any performance measure of interest. In the following sections a few important ones are examined.

4.4.1 State Buffer Use

When a processor completes its local processing it advances its virtual time clock by one time unit. Therefore, if a processor is ahead by k units of virtual time (k units of distance on the axis), then it will need to have saved k states. The expected number of buffers (\bar{B}_i) needed to save state at each processor can be found from

$$\begin{aligned} \bar{B}_1 &= \sum_{i=1}^{\infty} i(P_i + S_i) \\ &= \frac{(K_1 + K_5)r_1}{(r_1 - 1)^2} + \frac{(K_2 + K_6)r_3}{(r_3 - 1)^2} \end{aligned} \quad (4.24)$$

$$\begin{aligned}
\bar{B}_2 &= \sum_{i=1}^{\infty} i(Q_i + R_i) \\
&= \frac{(K_3 + K_7)s_1}{(s_1 - 1)^2} + \frac{(K_4 + K_8)s_3}{(s_3 - 1)^2}
\end{aligned} \tag{4.25}$$

More interestingly, the probability that a fixed size buffer of size $b \geq 1$ overflows at processor i ($\Theta_{i,b}$) is

$$\begin{aligned}
\Theta_{1,b} &= \sum_{i=b+1}^{\infty} (P_i + S_i) \\
&= \sum_{i=0}^{\infty} (P_i + S_i) - \sum_{i=0}^b (P_i + S_i) \\
&= \frac{(K_1 + K_5)}{r_1^b(r_1 - 1)} + \frac{(K_2 + K_6)}{r_3^b(r_3 - 1)}
\end{aligned} \tag{4.26}$$

$$\begin{aligned}
\Theta_{2,b} &= \sum_{i=b+1}^{\infty} (Q_i + R_i) \\
&= \sum_{i=0}^{\infty} (Q_i + R_i) - \sum_{i=0}^b (Q_i + R_i) \\
&= \frac{(K_3 + K_7)}{s_1^b(s_1 - 1)} + \frac{(K_4 + K_8)}{s_3^b(s_3 - 1)}
\end{aligned} \tag{4.27}$$

4.4.2 Message Queue Distribution

Messages that arrive in the virtual time future are queued until the processor completes all work with a virtual time less than the arriving message. The size of the message queue is defined as the number of messages queued in the virtual time future of the processor, plus any message that is currently being processed. The distribution of message queue length at each processor is found by conditioning on the probability that a process is ahead by a certain distance k summing over the appropriate ranges of the state probabilities.

$$\begin{aligned}
m_{1,k} &= \text{P}[k \text{ msgs queued at Processor 1}] \\
&= \sum_{i=k}^{\infty} \text{P}[k \text{ msgs queued} \mid \text{Processor 2 is ahead by } i \text{ steps}] Q_i \\
m_{1,k} &= \sum_{i=k}^{\infty} Q_i \binom{i}{k} q_2^k \bar{q}_2^{i-k} + \sum_{i=k-1}^{\infty} R_i \binom{i}{k-1} q_2^{k-1} \bar{q}_2^{i-k+1} & k \geq 2 \\
&= \frac{K_3 q_2^k s_1}{(s_1 - \bar{q}_2)^{k+1}} + \frac{K_7 q_2^{k-1} s_1}{(s_1 - \bar{q}_2)^k} + \frac{K_4 q_2^k s_3}{(s_3 - \bar{q}_2)^{k+1}} + \frac{K_8 q_2^{k-1} s_3}{(s_3 - \bar{q}_2)^k} & (4.28)
\end{aligned}$$

$$\begin{aligned}
m_{1,1} &= \sum_{i=1}^{\infty} Q_i i q_2 \bar{q}_2^{i-1} + \sum_{i=0}^{\infty} R_i \bar{q}_2^i + B_0 \\
&= \frac{K_3 q_2 s_1}{(s_1 - \bar{q}_2)^2} + \frac{K_7 s_1}{s_1 - \bar{q}_2} + \frac{K_4 q_2 s_3}{(s_3 - \bar{q}_2)^2} + \frac{K_8 s_3}{s_3 - \bar{q}_2} + B_0 & (4.29)
\end{aligned}$$

$$\begin{aligned}
m_{1,0} &= P(1) + S(1) + N_0 + \sum_{i=1}^{\infty} Q_i \bar{q}_2^i \\
&= P(1) + S(1) + N_0 + \frac{K_3 \bar{q}_2}{s_1 - \bar{q}_2} + \frac{K_4 \bar{q}_2}{s_3 - \bar{q}_2} & (4.30)
\end{aligned}$$

$$\begin{aligned}
m_{2,k} &= \text{P}[k \text{ msgs queued at Processor 2}] \\
m_{2,k} &= \sum_{i=k}^{\infty} P_i \binom{i}{k} q_1^k \bar{q}_1^{i-k} + \sum_{i=k-1}^{\infty} S_i \binom{i}{k-1} q_1^{k-1} \bar{q}_1^{i-k+1} & k \geq 2 \\
&= \frac{K_1 q_1^k r_1}{(r_1 - \bar{q}_1)^{k+1}} + \frac{K_5 q_1^{k-1} r_1}{(r_1 - \bar{q}_1)^k} + \frac{K_2 q_1^k r_3}{(r_3 - \bar{q}_1)^{k+1}} + \frac{K_6 q_1^{k-1} r_3}{(r_3 - \bar{q}_1)^k} & (4.31)
\end{aligned}$$

$$\begin{aligned}
m_{2,1} &= \sum_{i=1}^{\infty} P_i i q_1 \bar{q}_1^{i-1} + \sum_{i=0}^{\infty} S_i \bar{q}_1^i + B_0 \\
&= \frac{K_1 q_1 r_1}{(r_1 - \bar{q}_1)^2} + \frac{K_5 r_1}{r_1 - \bar{q}_1} + \frac{K_2 q_1 r_3}{(r_3 - \bar{q}_1)^2} + \frac{K_6 r_3}{r_3 - \bar{q}_1} + B_0 & (4.32)
\end{aligned}$$

$$\begin{aligned}
m_{2,0} &= Q(1) + R(1) + N_0 + \sum_{i=1}^{\infty} P_i \bar{q}_1^i \\
&= Q(1) + R(1) + N_0 + \frac{K_1 \bar{q}_1}{r_1 - \bar{q}_1} + \frac{K_2 \bar{q}_1}{r_3 - \bar{q}_1} & (4.33)
\end{aligned}$$

The mean number of message buffers needed at each processor is

$$\begin{aligned}\bar{m}_1 &= \sum_{i=0}^{\infty} i m_{1,i} \\ &= \frac{s_1 (K_3 q_2 + K_7 (s_1 - \bar{q}_2))}{(s_1 - 1)^2} + \frac{s_3 (K_4 q_2 + K_8 (s_3 - \bar{q}_2))}{(s_3 - 1)^2} + B_0\end{aligned}\quad (4.34)$$

$$\begin{aligned}\bar{m}_2 &= \sum_{i=0}^{\infty} i m_{2,i} \\ &= \frac{r_1 (K_1 q_1 + K_5 (r_1 - \bar{q}_1))}{(r_1 - 1)^2} + \frac{r_3 (K_2 q_1 + K_6 (r_3 - \bar{q}_1))}{(r_3 - 1)^2} + B_0\end{aligned}\quad (4.35)$$

4.4.3 Normalized Rate of Progress

From the complete solution of the Markov chain, the average rate of progress of the two processor system may be calculated. We define δ_2 as the average rate of progress in virtual time of the two processor system. This value is simply the average “unfettered” rate of progress of the two processors minus the average rollback rate.

$$\begin{aligned}\delta_2 &= (\lambda_1 + \lambda_2) \left(\sum_{k=1}^{\infty} Q_k + N_0 + \sum_{k=1}^{\infty} P_k \right) \\ &\quad + \lambda_1 \sum_{k=0}^{\infty} S_k + \lambda_2 \sum_{k=0}^{\infty} R_k - \lambda_2 q_2 \sum_{k=1}^{\infty} P_k (k-1) - \lambda_1 q_1 \sum_{k=1}^{\infty} Q_k (k-1) \\ &= (\lambda_1 + \lambda_2) \left(\frac{K_1}{r_1 - 1} + \frac{K_2}{r_3 - 1} + N_0 + \frac{K_3}{s_1 - 1} + \frac{K_4}{s_3 - 1} \right) \\ &\quad + \lambda_1 \left(\frac{K_5 r_1}{r_1 - 1} + \frac{K_6 r_3}{r_3 - 1} \right) + \lambda_2 \left(\frac{K_7 s_1}{s_1 - 1} + \frac{K_8 s_3}{s_3 - 1} \right) \\ &\quad - \lambda_1 q_1 \left(\frac{K_3}{(s_1 - 1)^2} + \frac{K_4}{(s_3 - 1)^2} \right) \\ &\quad - \lambda_2 q_2 \left(\frac{K_1}{(r_1 - 1)^2} + \frac{K_2}{(r_3 - 1)^2} \right)\end{aligned}\quad (4.36)$$

We calculate a “normalized” rate of progress (\hat{R}) by dividing the above equation by $(\lambda_1 + \lambda_2)$. We arrive at

$$\begin{aligned} \hat{R} = & \left(\frac{K_1}{r_1 - 1} + \frac{K_2}{r_3 - 1} + N_0 + \frac{K_3}{s_1 - 1} + \frac{K_4}{s_3 - 1} \right) \\ & + a \left(\frac{K_5 r_1}{r_1 - 1} + \frac{K_6 r_3}{r_3 - 1} \right) + \bar{a} \left(\frac{K_7 s_1}{s_1 - 1} + \frac{K_8 s_3}{s_3 - 1} \right) \\ & - \bar{a} q_2 \left(\frac{K_1}{(r_1 - 1)^2} + \frac{K_2}{(r_3 - 1)^2} \right) - a q_1 \left(\frac{K_3}{(s_1 - 1)^2} + \frac{K_4}{(s_3 - 1)^2} \right) \end{aligned} \quad (4.37)$$

Normalized rate of progress is used instead of the usual measure of speedup because it is now difficult to describe what the “equivalent” single processor rate is. The message queuing model for Time Warp generates work for a processor each time a message is sent. In single processor operation, no messages are sent so this work will be unaccounted for. There is no simple way of adding the extra work to a single processor model, since not every message that is generated will actually be processed due to rollbacks. To avoid this problem, we use the \hat{R} measure. This measure is equally useful since it does show us how well the Time Warp system performs.

In Figure 4.3 we show the value for \hat{R} when $a = 1/2$ and $q_1 = q_2 = q$ (the symmetric, balanced case). The figure shows \hat{R} versus q for various values of f . We see that for the best performance we want q to be small and $f \rightarrow \infty$. This is the case where there is little interaction between the processors and it takes zero time to process a message from the other processor. By setting $f = 1$ we can examine \hat{R} versus q only. This plot is shown in Figure 4.4 compared to the average rate of progress for the same system where messages are only used for synchronization ($f = \infty$). We see that the system where messages carry work performs more poorly than where they are only used for synchronization. Yet, this system is not twice as bad as the synchronization-only system even

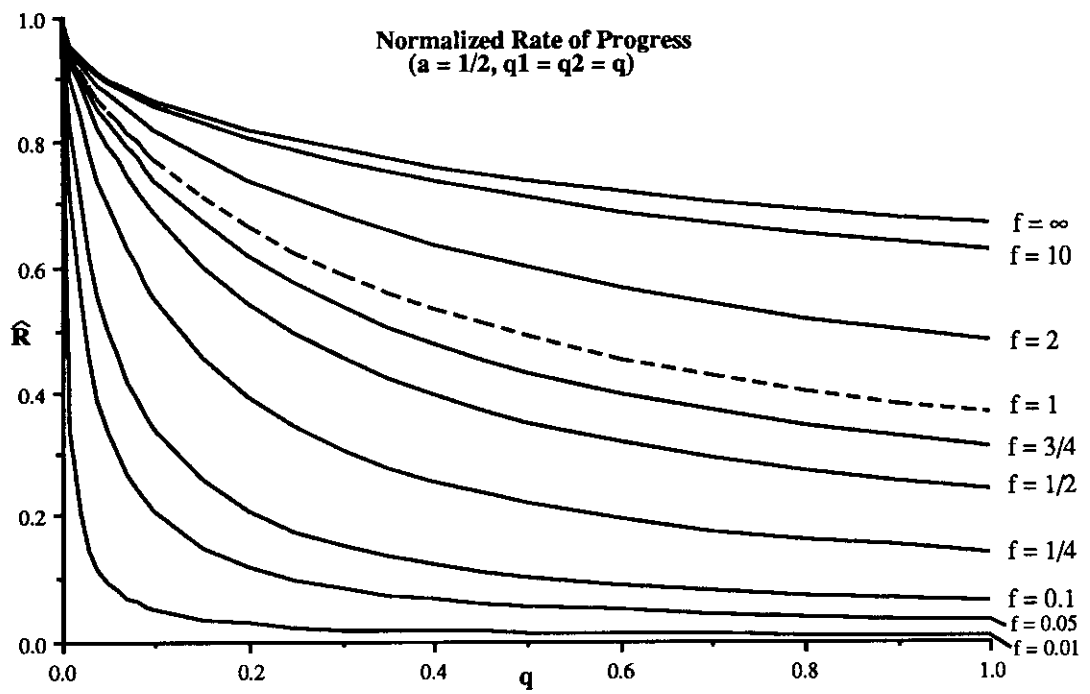


Figure 4.3: Normalized rate of progress (\hat{R}) versus f and q for the symmetric, balanced case.

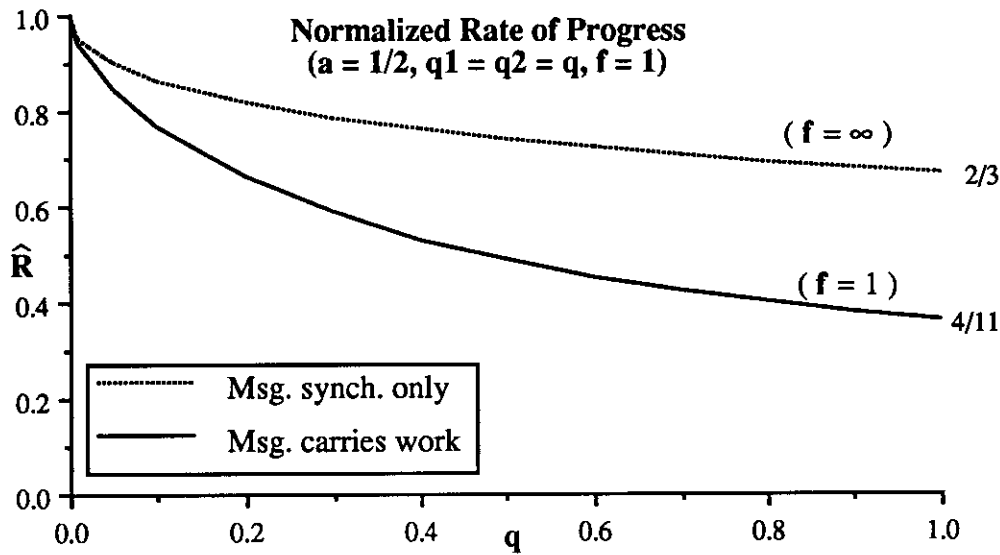


Figure 4.4: \hat{R} versus q for the symmetric, balanced case.

at $q = 1$. In fact, at $q = 1$ we can verify the \hat{R} result for $f = 1$ by realizing that each processor will always have a message to process. Therefore, the rate of progress at each step is governed by the maximum time it takes for the two processors to each finish a message and local work. This is simply the expected value of the maximum of two 2-stage Erlangs at rate λ which is equal to $\frac{11}{4\lambda}$. Taking the reciprocal and dividing by λ to find the normalized rate, we get $\hat{R} = 4/11$ which is the value plotted in Figure 4.4.

4.5 A Specific Example

To better understand the above results we explicitly calculate values of the performance measures for a specific instance of the parameters of the system.

The values chosen are given below.

$$\begin{aligned} \lambda_1 &= 11 & \lambda_2 &= 9 \\ a &= \frac{11}{20} & \bar{a} &= \frac{9}{20} \\ f &= 1 \\ q_1 &= \frac{1}{2} & q_2 &= \frac{1}{3} \end{aligned}$$

Note that processor one will move slightly faster than processor two while the cost of processing a message is the same as processing a locally generated event. Finally, processor one will send a message with probability 1/2 while processor two will send a message with probability 1/3 after advancing.

4.5.1 State Probabilities and State Buffer Use

The resulting equations for the probability of being in any state are

$$\begin{aligned} N_0 &\approx 0.0781 \\ B_0 &\approx 0.0423 \\ P_k &\approx \frac{0.114}{1.281^k} - \frac{0.0359}{2.086^k} \quad k \geq 1 \\ Q_k &\approx \frac{0.1385}{1.702^k} - \frac{0.0605}{2.468^k} \quad k \geq 1 \\ S_k &\approx \frac{0.0452}{1.281^k} + \frac{0.0175}{2.086^k} \quad k \geq 0 \\ R_k &\approx \frac{0.0319}{1.702^k} + \frac{0.0203}{2.468^k} \quad k \geq 0 \end{aligned}$$

These probabilities are plotted in Figure 4.5. As you would expect, $P_k > Q_k$ and $S_k > R_k$ since processor one is moving at a faster rate than processor two.

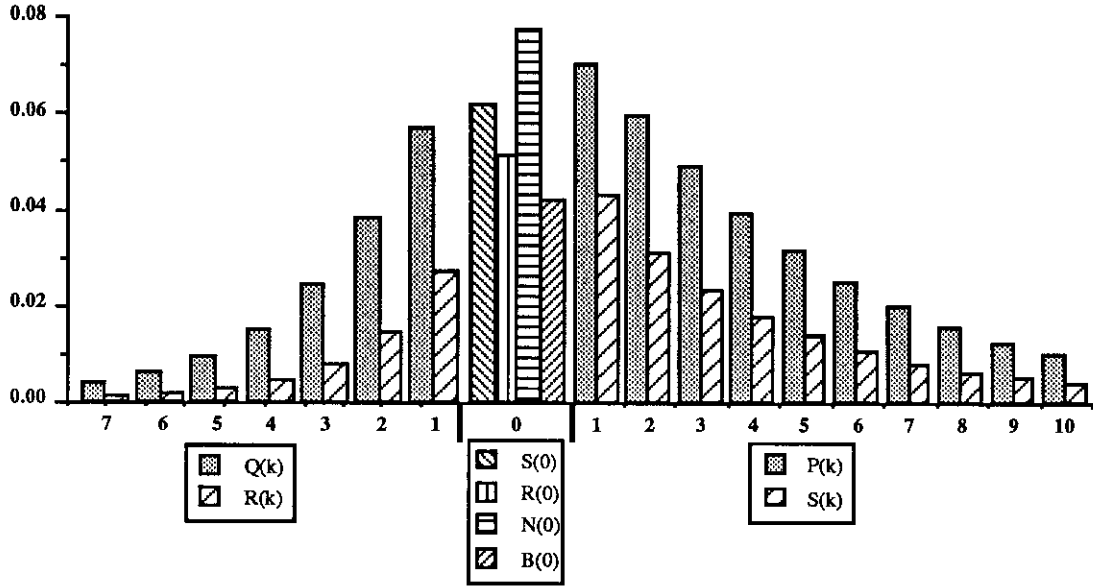


Figure 4.4: State probabilities.

The expected number of buffers needed to save state at each processor (\bar{B}_i) is given by

$$\bar{B}_1 = \sum_{i=1}^{\infty} i(P_i + S_i) \approx 2.5489$$

$$\bar{B}_2 = \sum_{i=1}^{\infty} i(Q_i + R_i) \approx 0.5429$$

From the values for $\Theta_{1,b}$ and $\Theta_{2,b}$

$$\Theta_{1,b} = \frac{0.5663}{1.281^b} - \frac{0.0169}{2.086^b}$$

$$\Theta_{2,b} = \frac{0.2428}{1.702^b} - \frac{0.0273}{2.468^b}$$

we find that with probability > 0.99 processor one will not need more than seventeen buffers. A similar value can be found for processor two.

$$P[\text{Processor 1 needs } > 17 \text{ state buffers}] \approx 0.00841 < 0.01$$

$$P[\text{Processor 2 needs } > 6 \text{ state buffers}] \approx 0.00988 < 0.01$$

4.5.2 Message Queue Distribution and Buffer Use

The distribution of messages at each processor is given below.

$$m_{1,0} \approx 0.7569$$

$$m_{1,1} \approx 0.1805$$

$$m_{1,k} \approx \frac{0.3904\left(\frac{1}{3}\right)^k}{1.035^k} + \frac{0.0676\left(\frac{1}{3}\right)^k}{1.801^k} \quad k \geq 2$$

$$m_{2,0} \approx 0.4074$$

$$m_{2,1} \approx 0.2441$$

$$m_{2,k} \approx \frac{0.3026\left(\frac{1}{2}\right)^k}{0.781^k} + \frac{0.0258\left(\frac{1}{2}\right)^k}{1.586^k} \quad k \geq 2$$

The values of these functions are plotted in Figure 4.6. The mean number of message buffers needed at each processor is

$$\bar{m}_1 \approx 0.3346$$

$$\bar{m}_2 \approx 1.5562$$

As with the state buffers we can find the number of message buffers needed to store messages such that the buffers will overflow with probability < 0.01 .

$$P[\text{Processor 1 needs } > 3 \text{ message buffers}] \approx 0.0063 < 0.01$$

$$P[\text{Processor 2 needs } > 9 \text{ message buffers}] \approx 0.0097 < 0.01$$

Finally, the value for the normalized rate of progress is $\hat{R} \approx 0.5071$.

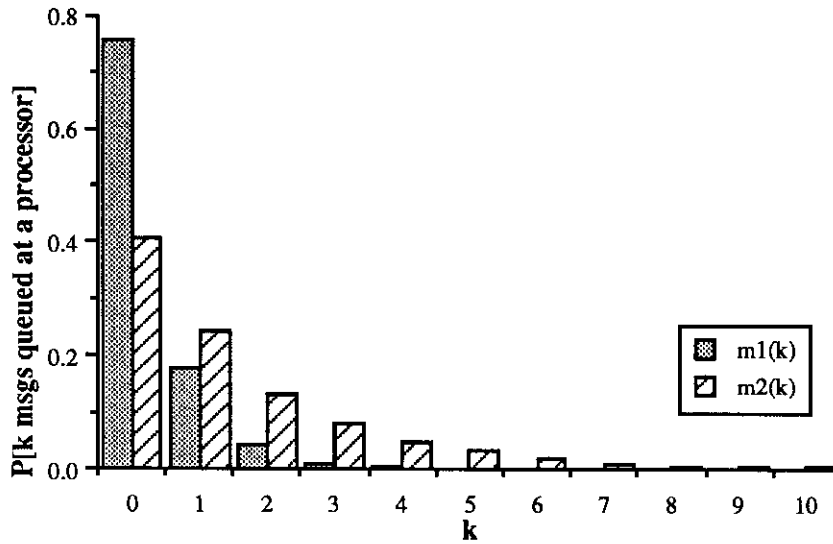


Figure 4.5: Distribution of the number of messages queued at each processor.

4.6 Conclusions

We introduced and solved exactly a new model for two processor Time Warp operation. The importance of our new model is that it explicitly accounts for the work that must be performed by each processor in response to the receipt of a message. Messages that arrive in the past cause rollbacks, while messages that arrive in the future are queued until the LP moves forward in simulation time. In all cases the messages create work for the LP.

With the complete Markov chain solution we calculated the normalized rate of progress of the two processors, and the distribution of the number of messages queued at each processor. Further, we found the expected number of buffers needed to save state and/or messages at each processor. Since we have the exact solution to the complete Markov chain, we can calculate nearly any parameter

that may be of interest.

CHAPTER 5

Two Processor Model with Rollback and State Saving Costs

5.1 Introduction

As in the previous chapter, we will examine an extension to the original model presented in Chapter 3. Here, we will add a cost for rollback and continue to include the cost for state saving that we introduced in Section 3.7.2.

5.2 The Rollback Cost Model

If the costs for rollback and/or state saving are high, TW may perform poorly. The following sections examine the two processor system when we account for rollback and state saving costs. We use a model similar to the one introduced in Section 4.2, namely, a continuous time, discrete state model where each processor makes only single step state advances whenever it advances. Immediately after a processor is forced to rollback, it pays a cost for restoring state by making the expected rate of forward progress smaller than normal for one event. When processing the “rollback event” each processor moves at a rate $\gamma_i = f\lambda_i$ where $0 < f \leq 1$. Once this event is completed, the processor moves again at its normal rate of λ_i . Note that when $f = 1$ there is no additional cost for

rollback and this model reduces to the one in [Kle89]. When $f \rightarrow 0$ then a rollback becomes very expensive. The range $f > 1$ means that an event after a rollback actually costs less real time to process on average than a normal event. This seems not to make sense, but upon further thought it might be a technique to account for work that has already been completed that need not be recomputed. A cost for state saving is added in Section 5.4.2.

To solve the system, we separate the Markov chain into five different regions.

$$\begin{aligned}
P_k &= \lim_{t \rightarrow \infty} P[D(t) = k \text{ and Processor 2 is **not** in a rollback state}] & k \geq 1 \\
Q_k &= \lim_{t \rightarrow \infty} P[D(t) = -k \text{ and Processor 1 is **not** in a rollback state}] & k \geq 1 \\
S_k &= \lim_{t \rightarrow \infty} P[D(t) = k \text{ and Processor 2 is in a rollback state}] & k \geq 0 \\
R_k &= \lim_{t \rightarrow \infty} P[D(t) = -k \text{ and Processor 1 is in a rollback state}] & k \geq 0 \\
P_0 &= \lim_{t \rightarrow \infty} P[D(t) = 0 \text{ and neither is in a rollback state}]
\end{aligned}$$

5.3 Analysis of the Cost Model

In this section we find the exact solution for the model that addresses rollback and state saving costs. The parameters of this system are

$$\begin{aligned}
\lambda_i &= \text{Rate at which processor } i \text{ executes events} \\
\gamma_i &= f\lambda_i = \text{Rate at which processor } i \text{ executes after a rollback} \\
a &= \frac{\lambda_1}{\lambda_1 + \lambda_2} \\
\bar{a} &= \frac{\lambda_2}{\lambda_1 + \lambda_2} = 1 - a \\
A &= a + \bar{a}f \\
B &= \bar{a} + af \\
q_i &= P[i^{\text{th}} \text{ processor sends a message after advancing}]
\end{aligned}$$

$$\bar{q}_i = 1 - q_i$$

The state diagram is shown in Figure 5.1. Note that the S_0 and R_0 states were duplicated to keep the figure from being too cluttered with transition arcs. As with the previous model, this system will have an equilibrium solution when $\lambda_i > 0$, $q_i > 0$ and $f > 0$.

The balance equations for this new system are

$$(\lambda_1 + \gamma_2)S_k = \lambda_1 S_{k-1} \quad k \geq 1 \quad (5.1)$$

$$(\lambda_1 + \gamma_2)S_0 = \lambda_1 q_1 \sum_{i=1}^{\infty} Q_i + \gamma_1 q_1 \sum_{i=1}^{\infty} R_i \quad (5.2)$$

$$(\lambda_2 + \gamma_1)R_k = \lambda_2 R_{k-1} \quad k \geq 1 \quad (5.3)$$

$$(\lambda_2 + \gamma_1)R_0 = \lambda_2 q_2 \sum_{i=1}^{\infty} P_i + \gamma_2 q_2 \sum_{i=1}^{\infty} S_i \quad (5.4)$$

$$(\lambda_1 + \lambda_2)P_k = \lambda_1 P_{k-1} + \lambda_2 \bar{q}_2 P_{k+1} + \gamma_2 \bar{q}_2 S_{k+1} \quad k \geq 2 \quad (5.5)$$

$$(\lambda_1 + \lambda_2)P_1 = \lambda_1 P_0 + \lambda_2 \bar{q}_2 P_2 + \gamma_2 \bar{q}_2 S_2 + \gamma_1 R_0 \quad (5.6)$$

$$(\lambda_1 + \lambda_2)P_0 = \lambda_1 \bar{q}_1 Q_1 + \lambda_2 \bar{q}_2 P_1 + \gamma_1 \bar{q}_1 R_1 + \gamma_2 \bar{q}_2 S_1 \quad (5.7)$$

$$(\lambda_2 + \lambda_1)Q_k = \lambda_2 Q_{k-1} + \lambda_1 \bar{q}_1 Q_{k+1} + \gamma_1 \bar{q}_1 R_{k+1} \quad k \geq 2 \quad (5.8)$$

$$(\lambda_2 + \lambda_1)Q_1 = \lambda_2 P_0 + \lambda_1 \bar{q}_1 Q_2 + \gamma_1 \bar{q}_1 R_2 + \gamma_2 S_0 \quad (5.9)$$

$$1 = P_0 + \sum_{i=1}^{\infty} P_i + \sum_{i=1}^{\infty} Q_i + \sum_{i=1}^{\infty} S_i + \sum_{i=1}^{\infty} R_i. \quad (5.10)$$

We define the following z-transforms (note, $S(z)$ and $R(z)$ are defined from $k = 1$ not $k = 0$ as in the previous model).

$$\begin{aligned} P(z) &= \sum_{k=1}^{\infty} P_k z^k & Q(z) &= \sum_{k=1}^{\infty} Q_k z^k \\ S(z) &= \sum_{k=1}^{\infty} S_k z^k & R(z) &= \sum_{k=1}^{\infty} R_k z^k \end{aligned}$$

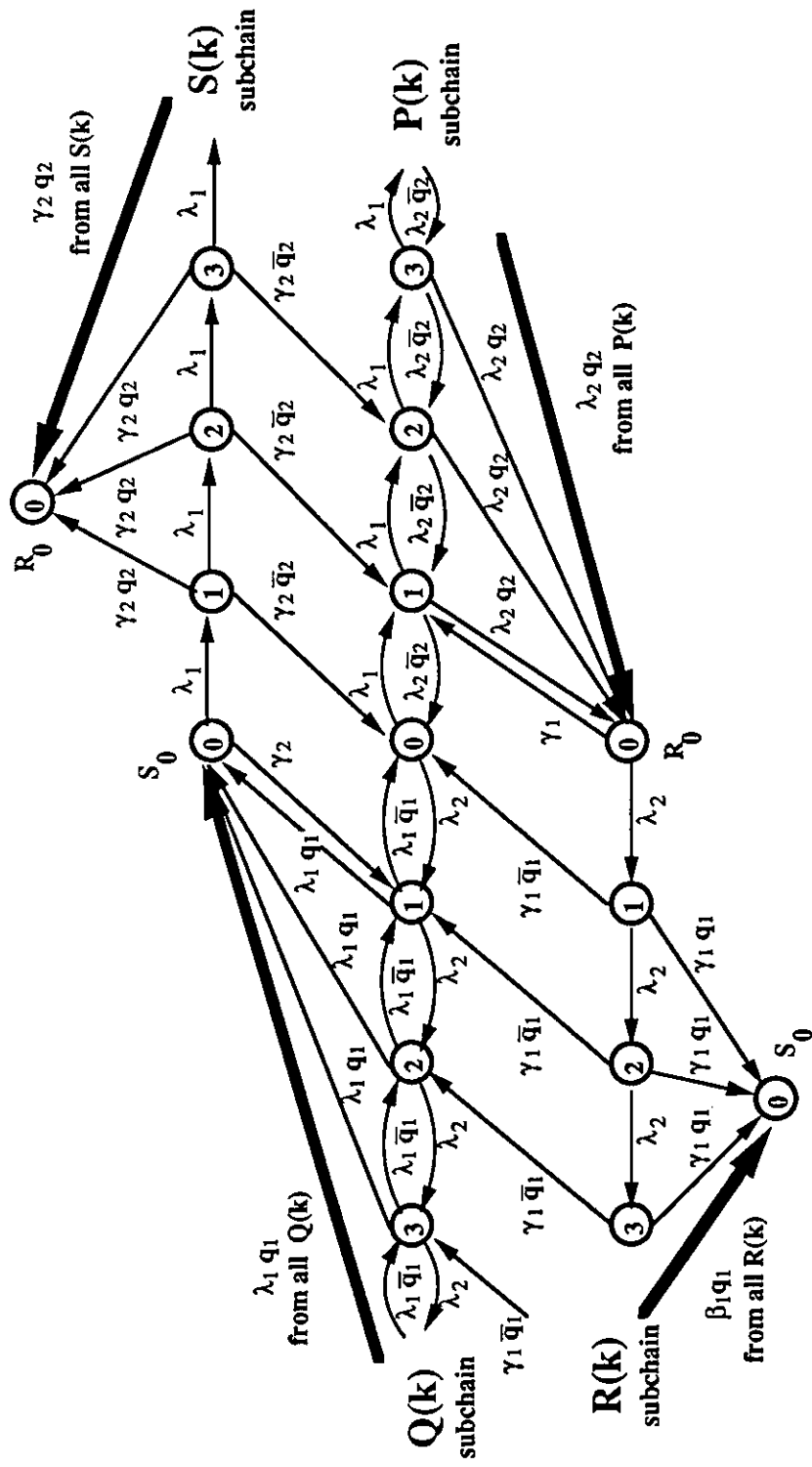


Figure 5.1: State diagram for the rollback cost model.

We proceed to find $P(z)$, $Q(z)$, $S(z)$ and $R(z)$ by multiplying the appropriate equation above by z^k and summing over the valid range of k . This leads to

$$P(z) = \frac{-(Aa(P_0 + R_0f)z^2) - \bar{a}\bar{q}_2(AS(z)f - AP_1z - S_0afz)}{A(\bar{a}\bar{q}_2 - z + az^2)}$$

$$Q(z) = \frac{-(B\bar{a}(P_0 + S_0f)z^2) - a\bar{q}_1(BR(z)f - Bq_1z - R_0\bar{a}fz)}{B(a\bar{q}_1 - z + \bar{a}z^2)}$$

$$S(z) = \frac{S_0az}{A - az}$$

$$R(z) = \frac{R_0\bar{a}z}{B - \bar{a}z}$$

Substituting the value for $S(z)$ into the equation for $P(z)$ we arrive at the following equation that defines $P(z)$.

$$P(z) = \frac{z(-(S_0a^2\bar{a}f\bar{q}_2z) + AP_1\bar{a}\bar{q}_2(A - az) - Aa(P_0 + R_0f)z(A - az))}{A(A - az)(\bar{a}\bar{q}_2 - z + az^2)} \quad (5.11)$$

The denominator of $P(z)$ can be factored into $A(A - az)(z - r_1)(z - r_2)$ and the denominator of $Q(z)$ into $B(B - \bar{a}z)(z - s_1)(z - s_2)$ where

$$(r_1, r_2) = \frac{1 \pm \sqrt{1 - 4a\bar{a}\bar{q}_2}}{2a}$$

$$(s_1, s_2) = \frac{1 \pm \sqrt{1 - 4a\bar{a}\bar{q}_1}}{2\bar{a}}$$

In Section 3.3.1 we showed that r_1 and r_2 are real and that $r_1 \geq 1$ while $0 \leq r_2 \leq 1$. Since $P(z)$ must be analytic in the region $|z| \leq 1$ the numerator of $P(z)$ must go to zero when $z = r_2$. Using this information we solve for P_1 .

$$P_1 = \frac{ar_2(S_0a\bar{a}f\bar{q}_2 + A(P_0 + R_0f)(A - ar_2))}{A\bar{a}\bar{q}_2(A - ar_2)}$$

We substitute this value back into the equation for $P(z)$ and arrive at

$$P(z) = \frac{z(S_0 a \bar{a} f \bar{q}_2 + (P_0 + R_0 f)(A - ar_2)(A - az)}{(A - ar_2)(r_1 - z)(A - az)} \quad (5.12)$$

Similarly for $Q(z)$ we find

$$Q(z) = \frac{z(R_0 a \bar{a} f \bar{q}_1 + (P_0 + S_0 f)(B - \bar{a}s_2)(B - \bar{a}z)}{(B - \bar{a}s_2)(s_1 - z)(B - \bar{a}z)} \quad (5.13)$$

Our task now is to find the values for the unknown constants P_0 , S_0 and R_0 .

We can solve the equations for S_0 (5.2) and R_0 (5.4) simultaneously to find

$$\begin{aligned} S_0 &= \frac{q_1(\bar{a}^2 q_2 P(1) + BaQ(1))}{AB - a\bar{a}q_1q_2} \\ R_0 &= \frac{q_2(A\bar{a}P(1) + a^2q_1Q(1))}{AB - a\bar{a}q_1q_2}. \end{aligned}$$

The above values are substituted into the equations for $P(z)$ and $Q(z)$ and we find $P(1)$ and $Q(1)$ by solving Equations 5.12 and 5.13 simultaneously with $z = 1$.

$$\begin{aligned} P(1) &= C_{pp_0}P_0 + C_{pp}P(1) + C_{pq}Q(1) \\ Q(1) &= C_{qp_0}P_0 + C_{qp}P(1) + C_{qq}Q(1) \end{aligned}$$

Therefore,

$$P(1) = C_P P_0 \quad Q(1) = C_Q P_0$$

where

$$\begin{aligned} C_P &= \frac{C_{pp_0} + C_{pq}C_{qp_0} - C_{pp_0}C_{qq}}{1 - C_{pp} - C_{pq}C_{qp} - C_{qq} + C_{pp}C_{qq}} \\ C_Q &= \frac{C_{pp_0}C_{qp} + C_{qp_0} - C_{pp}C_{qp_0}}{1 - C_{pp} - C_{pq}C_{qp} - C_{qq} + C_{pp}C_{qq}} \end{aligned}$$

and

$$C_{pp_0} = \frac{1}{(r_1 - 1)}$$

$$\begin{aligned}
C_{pp} &= \frac{\bar{a}q_2 (AfA + a\bar{a}q_1\bar{q}_2 - afAr_2)}{(AB - a\bar{a}q_1q_2)(r_1 - 1)(A - ar_2)} \\
C_{pq} &= \frac{a^2q_1 (Afq_2 + B\bar{q}_2 - afq_2r_2)}{(AB - a\bar{a}q_1q_2)(r_1 - 1)(A - ar_2)} \\
C_{qpo} &= \frac{1}{(s_1 - 1)} \\
C_{qp} &= \frac{\bar{a}^2q_2 (Bfq_1 + A\bar{q}_1 - \bar{a}fq_1s_2)}{(AB - a\bar{a}q_1q_2)(s_1 - 1)(B - \bar{a}s_2)} \\
C_{qq} &= \frac{aq_1 (BfB + a\bar{a}q_1q_2 - \bar{a}fBs_2)}{(AB - a\bar{a}q_1q_2)(s_1 - 1)(B - \bar{a}s_2)}
\end{aligned}$$

P_0 is derived from the fact that the probabilities must sum to 1. Note that we are using the same constant names (e.g. C_{pp}) as we did in the previous chapter.

Finally, the equations for $P(z)$, $Q(z)$, $S(z)$ and $R(z)$ can be inverted to find the complete solution to the Markov chain.

$$\begin{aligned}
P_k &= (P_0 + R_0f) \left(\frac{1}{r_1}\right)^k \\
&\quad + \frac{S_0a\bar{a}f\bar{q}_2}{(A - ar_1)(A - ar_2)} \left(\left(\frac{1}{r_1}\right)^k - \left(\frac{a}{A}\right)^k \right) \quad k \geq 1 \quad (5.14)
\end{aligned}$$

$$S_k = S_0 \left(\frac{a}{A}\right)^k \quad k \geq 0 \quad (5.15)$$

$$\begin{aligned}
Q_k &= (P_0 + S_0f) \left(\frac{1}{s_1}\right)^k \\
&\quad + \frac{R_0a\bar{a}f\bar{q}_1}{(B - \bar{a}s_1)(B - \bar{a}s_2)} \left(\left(\frac{1}{s_1}\right)^k - \left(\frac{\bar{a}}{B}\right)^k \right) \quad k \geq 1 \quad (5.16)
\end{aligned}$$

$$R_k = R_0 \left(\frac{\bar{a}}{B}\right)^k \quad k \geq 0 \quad (5.17)$$

$$S_0 = C_{S_0}P_0$$

$$R_0 = C_{R_0}P_0$$

$$C_{S_0} = \frac{q_1 (C_Q a B + C_P \bar{a}^2 q_2)}{AB - a\bar{a}q_1q_2}$$

$$C_{R_0} = \frac{(C_P \bar{a} A + C_Q a^2 q_1) q_2}{AB - a\bar{a}q_1q_2}$$

$$P_0 = \left(1 + C_{R_0} + C_{S_0} + \frac{C_{S_0} a}{\bar{a} f} + \frac{C_{R_0} \bar{a}}{a f} + \frac{1 + C_{S_0} f}{(s_1 - 1)} + \frac{1 + C_{R_0} f}{(r_1 - 1)} + \frac{C_{S_0} a \bar{q}_2}{(r_1 - 1)(A - ar_2)} + \frac{C_{R_0} \bar{a} q_1}{(s_1 - 1)(B - \bar{a} s_2)} \right)^{-1} \quad (5.18)$$

5.4 Performance Measures

5.4.1 State Buffer Use

Using the state probabilities we find the average state buffer occupancy at processors one and two.

$$\begin{aligned} \bar{B}_1 &= \sum_{i=1}^{\infty} i (P_i + S_i) \\ &= \frac{AS_0 a}{\bar{a}^2 f^2} + \frac{(P_0 + R_0 f) r_1}{(r_1 - 1)^2} \\ &\quad + \frac{S_0 a \bar{a} f \bar{q}_2}{(A - ar_1)(A - ar_2)} \left(\frac{r_1}{(r_1 - 1)^2} - \frac{Aa}{\bar{a}^2 f^2} \right) \end{aligned} \quad (5.19)$$

$$\begin{aligned} \bar{B}_2 &= \sum_{i=1}^{\infty} i (Q_i + R_i) \\ &= \frac{BR_0 \bar{a}}{a^2 f^2} + \frac{(P_0 + S_0 f) s_1}{(s_1 - 1)^2} \\ &\quad + \frac{R_0 a \bar{a} f q_1}{(B - \bar{a} s_1)(B - \bar{a} s_2)} \left(\frac{s_1}{(s_1 - 1)^2} - \frac{B\bar{a}}{a^2 f^2} \right) \end{aligned} \quad (5.20)$$

As with the previous models, we also find $\Theta_{i,b}$, the probability that a fixed sized buffer of size $b \geq 1$ overflows.

$$\Theta_{1,b} = \sum_{i=b+1}^{\infty} (P_i + S_i)$$

$$\begin{aligned}
&= \frac{S_0 a}{\bar{a} f} \left(\frac{a}{A}\right)^b + \frac{(P_0 + R_0 f)}{(r_1 - 1)r_1^b} \\
&\quad + \frac{S_0 a \bar{a} f \bar{q}_2}{(A - ar_1)(A - ar_2)} \left(\frac{1}{(r_1 - 1)r_1^b} - \frac{a}{\bar{a} f} \left(\frac{a}{A}\right)^b \right) \quad (5.21)
\end{aligned}$$

$$\begin{aligned}
\Theta_{2,b} &= \sum_{i=b+1}^{\infty} (Q_i + R_i) \\
&= \frac{R_0 \bar{a}}{a f} \left(\frac{\bar{a}}{B}\right)^b + \frac{(P_0 + S_0 f)}{(s_1 - 1)s_1^b} \\
&\quad + \frac{R_0 a \bar{a} f \bar{q}_1}{(B - \bar{a}s_1)(B - \bar{a}s_2)} \left(\frac{1}{(s_1 - 1)s_1^b} - \frac{\bar{a}}{a f} \left(\frac{\bar{a}}{B}\right)^b \right) \quad (5.22)
\end{aligned}$$

5.4.2 Speedup

From the complete solution of the Markov chain we calculate the speedup S of the two processor TW system over an equivalent single processor. The speedup is simply the rate of the two processor system δ_2 divided by the rate of progress for a single processor system δ_1 . The rate of forward progress for one processor is defined (as earlier) simply as the average rate of progress of the two processes

$$\delta_1 = \frac{\lambda_1 + \lambda_2}{2}.$$

At this point we add in the additional cost for state saving by allowing a single processor to move at a rate that is c times faster than the TW processors. Thus, state saving increases the average execution time of an event from $1/\lambda_i$ to c/λ_i when running TW. The revised rate of progress for a single processor is

$$\delta_1 = \frac{c(\lambda_1 + \lambda_2)}{2},$$

while the rate of progress for the two processor TW system is found from the following equation.

$$\begin{aligned}
\delta_2 = & (\lambda_1 + \lambda_2)P_0 + (\lambda_1 + \gamma_2)S_0 + (\lambda_2 + \gamma_1)R_0 \\
& + (\lambda_1 + \lambda_2)P(1) + (\lambda_2 + \lambda_1)Q(1) + (\lambda_1 + \gamma_2)S(1) + (\lambda_2 + \gamma_1)R(1) \\
& - \lambda_2 q_2 \sum_{k=1}^{\infty} P_k(k-1) - \lambda_1 q_1 \sum_{k=1}^{\infty} Q_k(k-1) \\
& - \gamma_2 q_2 \sum_{k=1}^{\infty} S_k(k-1) - \gamma_1 q_1 \sum_{k=1}^{\infty} R_k(k-1)
\end{aligned}$$

Taking the ratio $S = \delta_2/\delta_1$ (i.e., the speedup) we arrive at

$$\begin{aligned}
S = & \frac{2}{c} \left(P_0 + P(1) + Q(1) + \frac{B^2 R_0}{af} + \frac{A^2 S_0}{\bar{a}f} - \frac{R_0 \bar{a}^2 q_1}{af} - \frac{S_0 a^2 q_2}{\bar{a}f} \right. \\
& - \bar{a} q_2 \left(\frac{P_0 + R_0 f}{(r_1 - 1)^2} + \frac{S_0 a \bar{a} f \bar{q}_2}{(A - ar_1)(A - ar_2)} \left(\frac{1}{(r_1 - 1)^2} - \frac{a^2}{\bar{a}^2 f^2} \right) \right) \\
& \left. - a q_1 \left(\frac{P_0 + S_0 f}{(s_1 - 1)^2} + \frac{R_0 a \bar{a} f \bar{q}_1}{(B - \bar{a} s_1)(B - \bar{a} s_2)} \left(\frac{1}{(s_1 - 1)^2} - \frac{\bar{a}^2}{a^2 f^2} \right) \right) \right) \quad (5.23)
\end{aligned}$$

Note that we have returned to a speedup measure (as opposed to \hat{R}) since, for the rollback cost system, the rate of progress on a single processor is well defined.

For the symmetric, balanced case where $\lambda_1 = \lambda_2 = \lambda$ and $q_1 = q_2 = q$ we get the following equation for speedup.

$$S = \frac{4f(f + \sqrt{q})}{c(2f^2 + f(2 + f)\sqrt{q} + (2 - f)fq + 2(1 - f)q^{\frac{3}{2}})} \quad (5.24)$$

A plot of this function is shown in Figure 5.2 for $c = 1$. (Note that for $f = 1$, $S = 4/(2 + \sqrt{q})$ as in Equation 3.7.)

Using this simple formula for speedup we find the values of f , q , and c that allow two processors running TW to progress faster than a single processor without TW. This is the region where $S \geq 1$. We solve Equation 5.24 for c when $S \geq 1$ resulting in the inequality

$$c \leq \frac{4f(f + \sqrt{q})}{(2f^2 + f(2 + f)\sqrt{q} + (2 - f)fq + 2(1 - f)q^{\frac{3}{2}})} \quad (5.25)$$

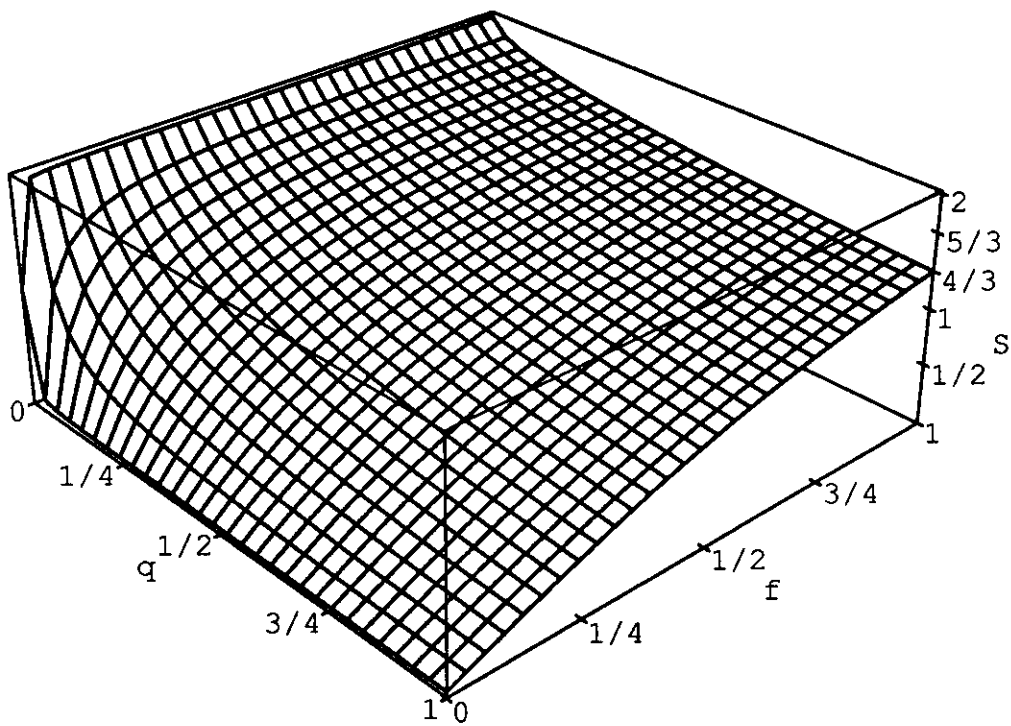


Figure 5.2: Speedup versus q and f for the symmetric, balanced case when $c = 1$.

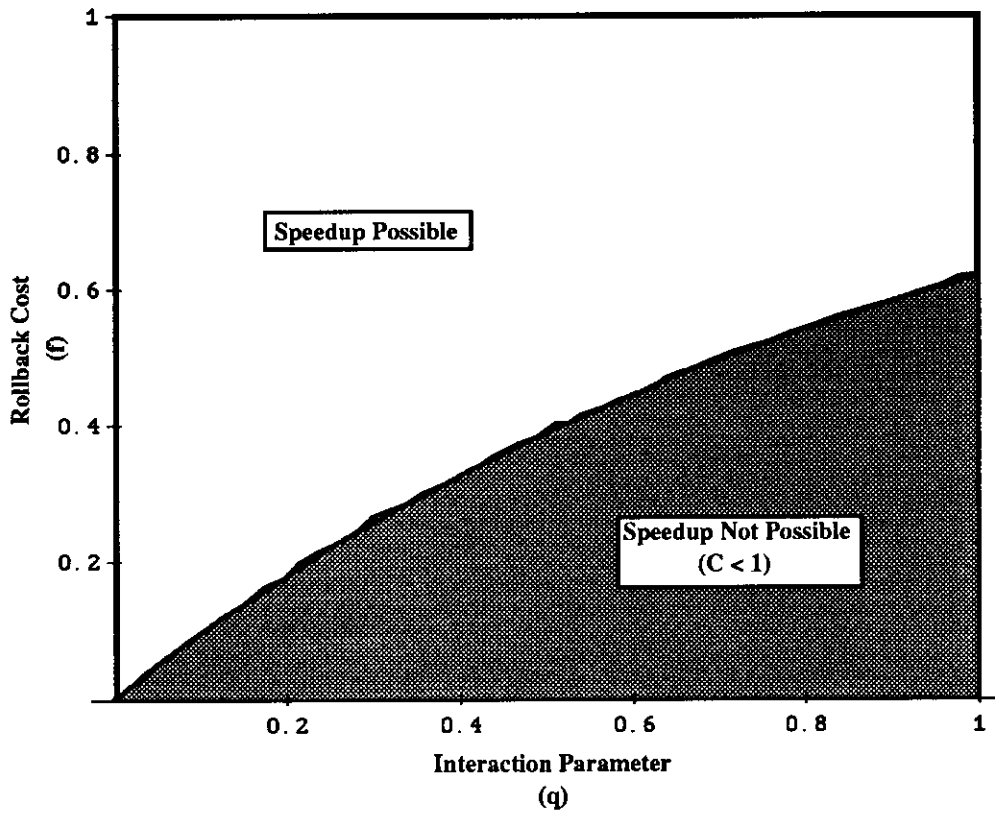


Figure 5.3: Region of $q - f$ space where speedup is possible.

Therefore, we find that c must lie below the surface plotted in Figure 5.2 for $S > 1$. It is clear for $c > 2$ that TW on two processors is always slower than using a single processor without TW. Further, since c must be greater than or equal to one (cost of state saving is ≥ 0), there is a region in the $q - f$ space where speedup is not possible. That is the shaded region shown in Figure 5.3.

Since rollbacks can be costly ($c > 1$), there may be an advantage to slowing down or stopping the faster processor when it gets ahead so as to avoid rollbacks.

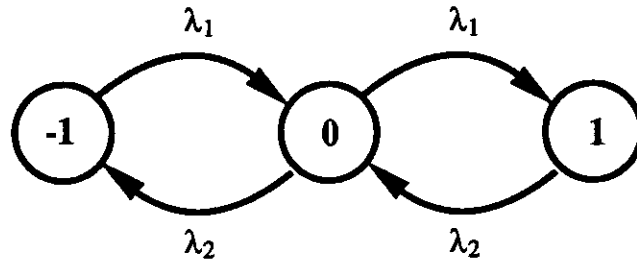


Figure 5.4: State diagram when each processor stops at one step ahead.

Mitra and Mitrani [MM84] (see Section 3.6), using an optimization function $J = D - (cost)R$, where D is the average forward rate of progress and R is the average rollback rate, find regions of the parameter space where the maximum of the function is found at the boundary where the processors have zero processing capacity (i.e., don't perform the task at all). Essentially, they found that Time Warp could perform poorly if the cost for rollback was high. Unfortunately, their method of adding a cost for rollback was to create a function external to the model, rather than having the system actually pay a real-time cost for rollback during operation. We find our model somewhat more satisfying.

Our effort is to improve TW by slowing down or stopping the processor that gets too far ahead, and we find that it sometimes pays to stop the leading processor when it gets *exactly* one step ahead. The state diagram for such a system is shown in Figure 5.4. Each processor will stop when it gets exactly one step ahead of the other processor. There will be no rollbacks and therefore no need for state saving. When $\lambda_1 = \lambda_2 = \lambda$ we find that $P_1 = Q_1 = P_0 = 1/3$, and that speedup over the equivalent single processor system is $4/3$. Therefore, we can always get a speedup of $4/3$ regardless of the values of f , q and c . For

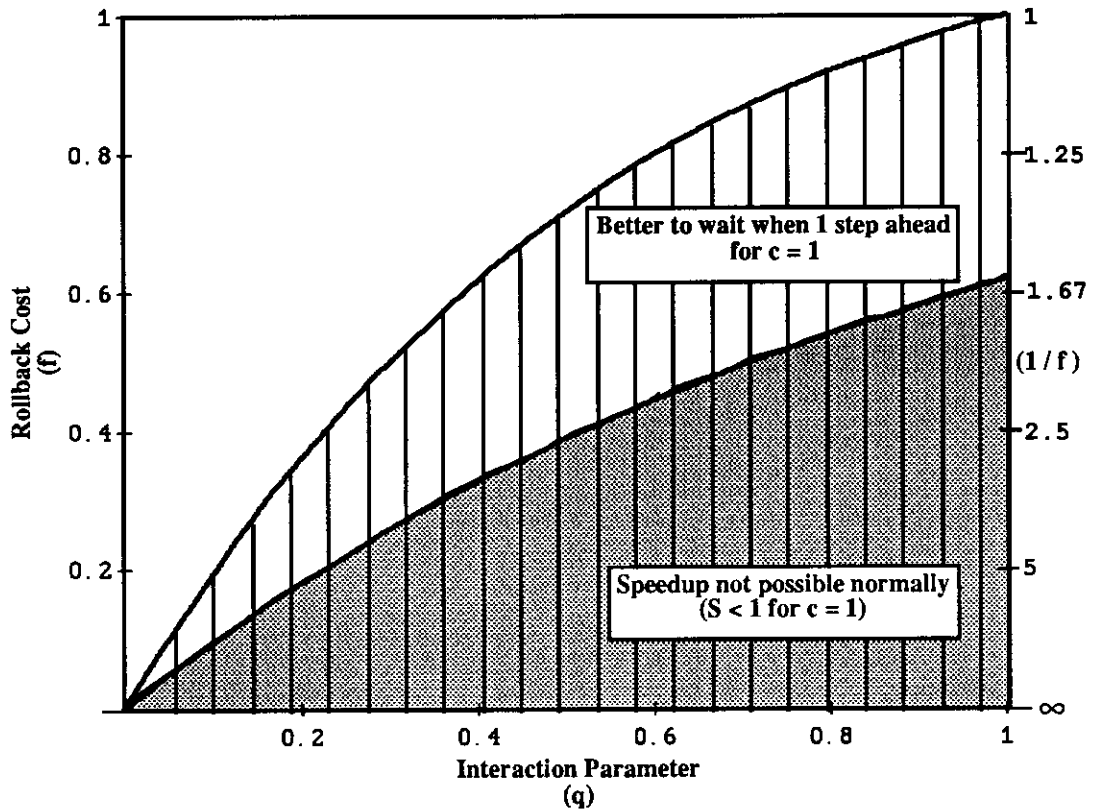


Figure 5.5: Region of $q - f$ space where stopping at one step is better.

general values of λ_1 and λ_2 the speedup is

$$S = \frac{4(1-a)a}{1-a+a^2}$$

which has its maximum of $4/3$ at $a = 1/2$. For the symmetric, balanced case where $\lambda_1 = \lambda_2 = \lambda$ and $q_1 = q_2 = q$, we show in Figure 5.5 the area of the $q - f$ plane where waiting at one step is better than rushing ahead when $c = 1$. Note that this region includes all the shaded $q - f$ area where we were not able to get speedup with two processors using Time Warp. Finally, in Figure 5.6 we show

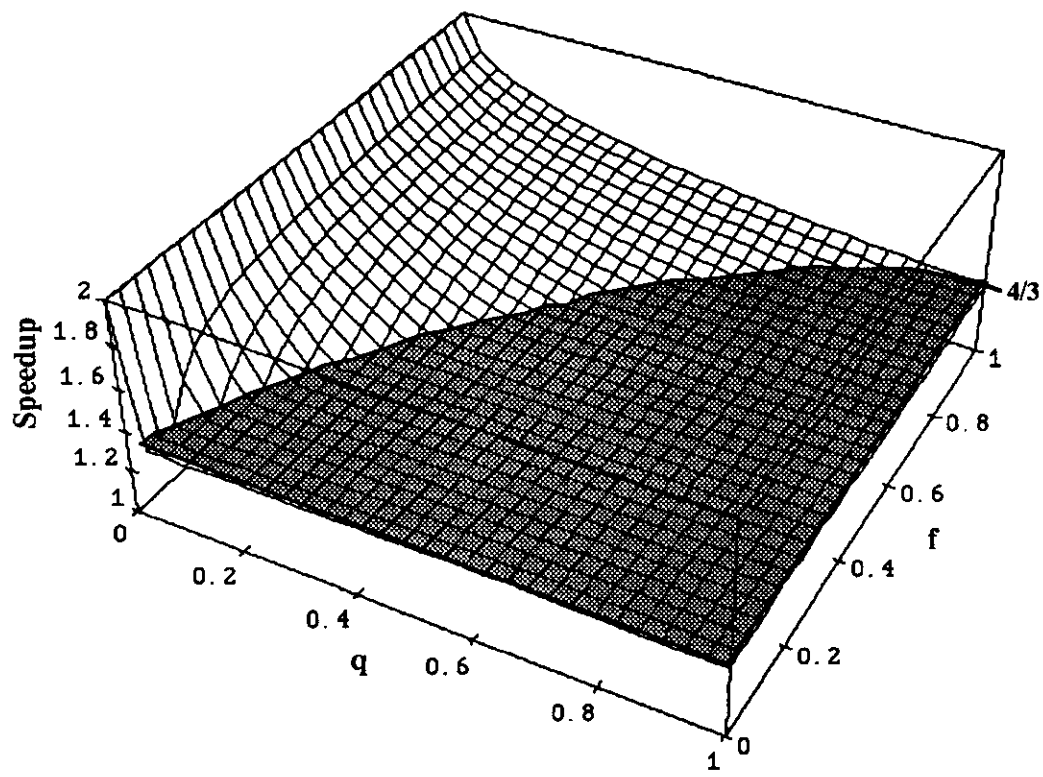


Figure 5.6: Achievable speedup for $c = 1$.

the achievable speedup when $c = 1$. The shaded region is where a processor waits when it gets one step ahead of the other. In the unshaded region, if c is less than the value plotted in the figure we are able to gain at least some speedup over the equivalent single processor not running Time Warp.

Since it sometimes pays to stop a processor when it gets one step ahead, one might surmise that there are ranges of the parameters where stopping a processor when it gets k ($k > 1$) steps ahead improves performance. For our model, this turns out not to be the case. By examining the Markov chain for $k = 2$, we find that the speedup is never greater than the speedup gained by the standard algorithm. Therefore, it is never practical to stop a processor once it gets more than one step ahead. The Markov chain in Figure 5.4 is unique in the respect that at no point in time will a processor incur a cost for state saving or rollback. Once the processors are allowed to get more than one step out of synchronization, state must be saved since rollbacks are possible. Intuitively, the fact that we might only stop at one step ahead makes sense since a process at virtual time v can only send a message to the other at time $v + 1$. By getting two or more steps ahead, a rollback is already possible and we will incur a cost for rollback if a message is sent regardless of whether we wait further down the line. Waiting now only causes the system to have a smaller speedup. In a more general system where a processor may send a message arbitrarily far into the future, we may find that there are regions of the parameter space where it pays to stop a processor when it gets further than one step ahead. We are currently extending the rollback cost model so that the processors are able to make arbitrary sized jumps when advancing (i.e., not restricted to single-steps) like in the original model in Chapter 3. This more general model will give us a

better opportunity to examine the improvements we might gain by stopping or slowing down the lead processor when it gets more than one step ahead.

5.5 Conclusions

We developed a model that incorporated costs for rollback and state saving. In addition to calculating the complete solution to the Markov chain and the speedup over a single processor, we were able to find regions of the parameter space where it was better to stop either processor when it was only one step ahead. Stopping the lead processor when it was two or more steps ahead led to no performance gain. As with our previous models, since we have the exact solution to the Markov chain, we are able to calculate nearly any performance measure of interest.

CHAPTER 6

A Model for Conservative Simulation

6.1 Introduction

In previous chapters the performance of an optimistic method of distributed simulation (Time Warp) was examined. In this chapter we create models for the two processor system using a conservative synchronization algorithm rather than Time Warp. The emphasis is to create a model for a conservative algorithm that we may directly compare to our previous model for Time Warp.

Conservative methods of Discrete Event Simulation are based on the work of Chandy, Misra, Bryant and others [CM79] [CHM79] [Bry77], and we discussed the algorithms in Section 1.3.2. Where TW proceeds ahead as fast as it can, only rolling back when a mistake is found, conservative methods allow an LP to proceed forward only when it is sure that it is performing correct computation. That is, conservative methods use blocking for synchronization, while optimistic techniques use state saving and rollback.

6.2 The Model

We now describe our model for the conservative method of synchronization. Our goal is to create a model that can easily be compared to the previous

models created for TW. We again use a continuous time, discrete state model, assuming that each process/processor advances along its own virtual time axis visiting only the integers. Each process takes an exponential amount of time to process an event and advances one step forward in virtual time (along its axis) after finishing the event. After advancing, each processor will send a synchronization message to the other processor with a given probability. Since the synchronization is conservative, no process can perform work at virtual time v until it is sure that the other processor will not send it a message time stamped with a virtual time less than or equal to v . We again exploit the Markov process defined as the difference in virtual time (position on the axes) of the two processes, and find the probability that one processor is ahead of the other by a distance k . Note that $|k| \leq 1$ for unimproved conservative systems.

Here are the parameters of the model (the same as the TW model).

$$\begin{aligned} \lambda_i &= \text{rate that processor } i \text{ executes events} \\ a &= \frac{\lambda_1}{\lambda_1 + \lambda_2} \\ \bar{a} &= \frac{\lambda_2}{\lambda_1 + \lambda_2} = 1 - a \\ q_i &= \text{P[} i^{\text{th}} \text{ processor sends a message after advancing]} \\ \bar{q}_i &= 1 - q_i \end{aligned}$$

If the processors start out at the same virtual time v , eventually, one (say P_1) advances to $v + 1$. Since a conservative synchronization mechanism is being employed, this processor must wait to see if the other processor will send it a message with virtual time $v + 1$. Its only choice is to wait until the lagging processor (P_2) advances, at which point that processor will “flip a coin” to decide whether to send a message. If a message is sent, P_1 receives it and is able

to continue processing again. If a message isn't sent, P_1 thinks it is still ahead of the other processor and will not continue processing. If P_2 were to advance again and not send a message, it would think (correctly) it was now ahead of P_1 and stop processing. At this point we have a deadlock that must be broken. Deadlock detection and recovery algorithms were discussed in Section 1.3.2. Essentially, we break the deadlock by letting each processor know where it is relative to the other processor. In this example, P_1 would learn it was behind and begin processing, thus breaking the deadlock. If, on the other hand, each processor is able to notify the other that it has advanced its local clock, then the lagging processor is able to advance whether or not a "data" message is sent. This latter type of notification is referred to as the "null message" technique that is used to speed up conservative models. When used, we assume that this information (null messages) is propagated without cost. We now examine several models for two processor conservative simulation.

6.3 A System Without Null Messages

We first solve a model where the processors do not send null messages. We assume that when a deadlock occurs it is detected and corrected after an exponential delay with mean $d/(\lambda_1 + \lambda_2)$. If $d = 0$ then deadlocks are broken instantaneously, while $d \rightarrow \infty$ means that deadlock detection/correction takes an infinite amount of time. This system can be described by a Markov chain with the following state description.

$$(D, t_1, t_2)$$

D = Actual virtual time difference between P_1 and P_2

t_1 = P_1 's belief about the virtual time difference

t_2 = P_2 's belief about the virtual time difference

Discrepancies arise between D , t_1 and t_2 when the processors don't inform each other about state changes. This happens more often when the processors are less likely to send messages (small q_i). When a processor thinks it is ahead, it does not try to advance further. When both processors believe they are leading, we have a deadlock. The state diagram for this system is shown in Figure 6.1. Each state is labeled with its state description (D, t_1, t_2) and an alphanumeric label for calculation of the steady-state probabilities. The balance equations for this system are

$$\lambda_1 p_A = \lambda_2 \bar{q}_2 p_0$$

$$\lambda_2 p_B = \lambda_1 \bar{q}_1 p_0$$

$$\lambda_1 p_C = \lambda_2 q_2 p_0 + \lambda_2 q_2 p_F + \frac{\lambda_1 + \lambda_2}{d} p_G$$

$$\lambda_2 p_D = \lambda_1 q_1 p_0 + \lambda_1 q_1 p_E + \frac{\lambda_1 + \lambda_2}{d} p_H$$

$$\lambda_1 p_E = \lambda_2 q_2 p_B + \lambda_1 \bar{q}_1 p_C$$

$$\lambda_2 p_F = \lambda_1 q_1 p_A + \lambda_2 \bar{q}_2 p_D$$

$$\frac{\lambda_1 + \lambda_2}{d} p_G = \lambda_2 \bar{q}_2 p_F$$

$$\frac{\lambda_1 + \lambda_2}{d} p_H = \lambda_1 \bar{q}_1 p_E$$

$$\frac{\lambda_1 + \lambda_2}{d} p_I = \lambda_1 \bar{q}_1 p_A + \lambda_2 \bar{q}_2 p_B$$

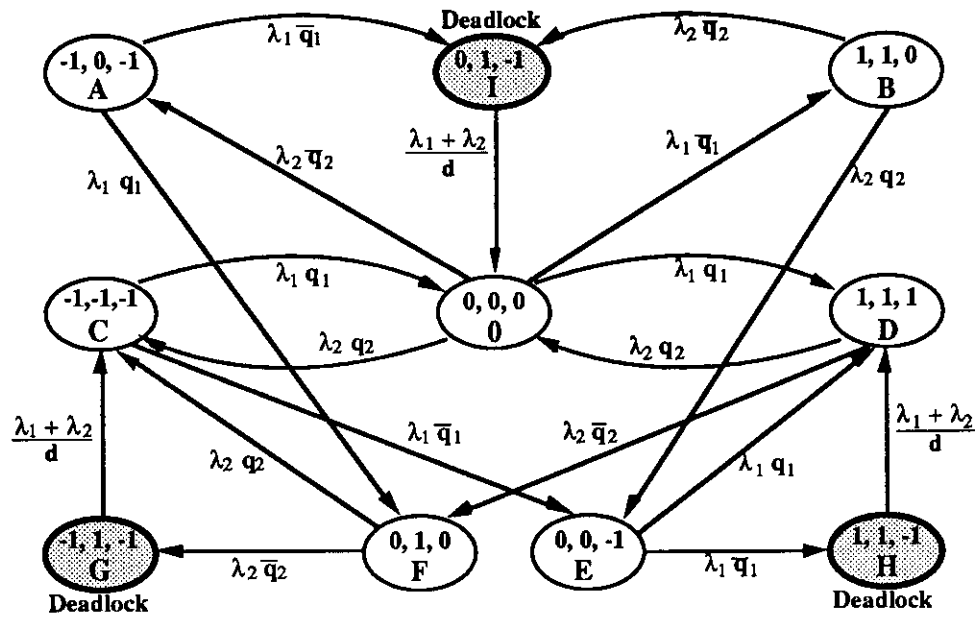


Figure 6.1: State diagram for conservative synchronization with no null messages and a cost for breaking deadlocks.

$$1 = p_0 + p_A + p_B + p_C + p_D + p_E + p_F + p_G + p_H + p_I$$

We first solve explicitly for $\{p_0, p_A, p_B, p_C, p_D, p_E, p_F, p_G, p_H, p_I\}$, then find the rate at which the two processors move forward in virtual time as

$$R_2 = (\lambda_1 + \lambda_2)p_0 + \lambda_1(p_A + p_C + p_E) + \lambda_2(p_B + p_D + p_F)$$

We compare this rate to our equivalent single processor rate again (see Section 3.4.1)

$$R_1 = \frac{\lambda_1 + \lambda_2}{2}$$

to find speedup

$$S = \frac{R_2}{R_1} = 2(p_0 + a(p_A + p_C + p_E) + \bar{a}(p_B + p_D + p_F)) \quad (6.1)$$

For the simple case where $q_1 = q_2 = q$ the formula for speedup is

$$S = \frac{4a\bar{a}(3 - 2q)}{3 - a + a^2 + 3a\bar{a}d(1 - q)^2 - 2q} \quad (6.2)$$

and if the cost of breaking a deadlock is zero ($d = 0$) then the formula reduces to

$$S = \frac{4a\bar{a}(3 - 2q)}{3 - a + a^2 - 2q} \quad (6.3)$$

and if $a = 1/2$, then

$$S = \frac{3 - 2q}{\frac{11}{4} - 2q} \quad (6.4)$$

We show Equation 6.2 plotted versus a and q for various values of d in Figure 6.2. We note here that the conservative system performs better as q , the interaction parameter, increases. This is in contrast to the Time Warp system where speedup decreased as q increased. In the conservative system we are better off sending a large number of messages because the messages keep each process informed as to the virtual time progress of the other thus allowing potential

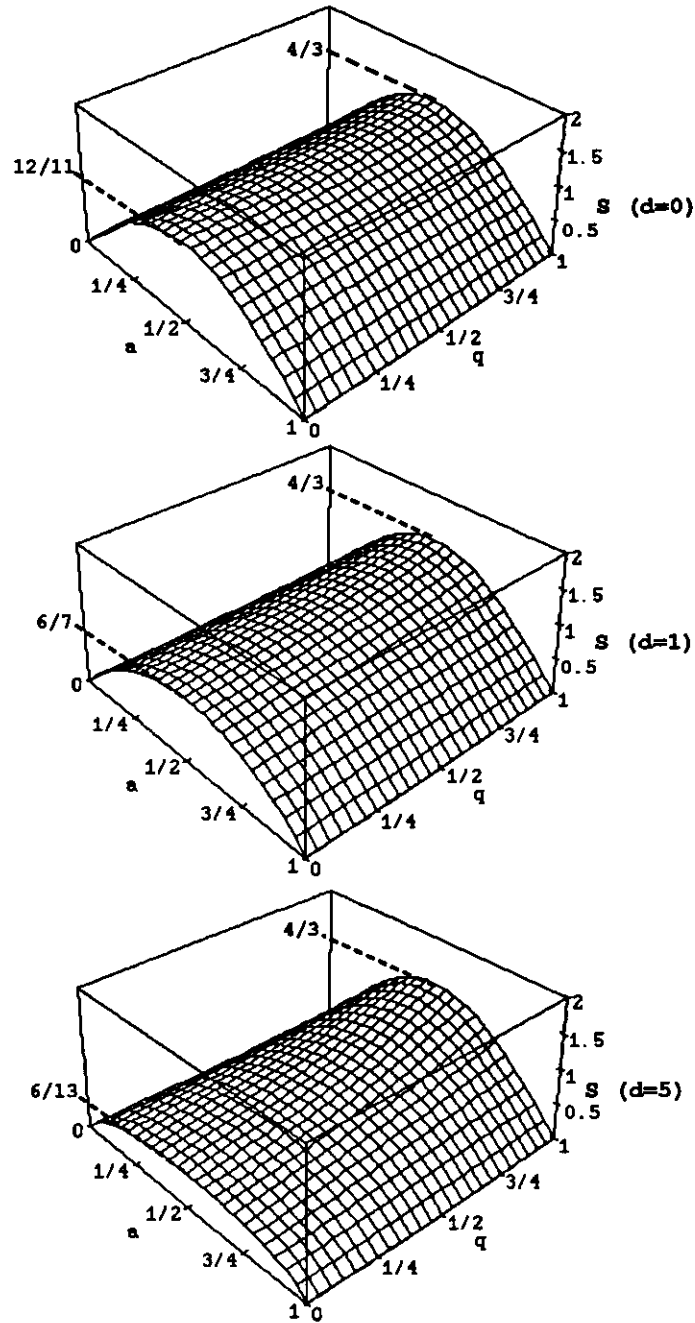


Figure 6.2: Speedup versus a and q for various values of d .

parallelism to be exploited. When more messages are sent, the processors are less likely to be waiting due to lack of information and less likely to become deadlocked.

It is also clear from the figures that the cost of deadlock has a large impact on the performance if the probability of interaction is small. This is to be expected, since the probability of deadlock is higher when the processes exchange information infrequently. We can take the derivative of speedup with respect to d (the cost of breaking deadlock) to quantify the effect of d on performance.

$$\frac{\partial S}{\partial d} = \frac{-12a^2\bar{a}^2(3-2q)(1-q)^2}{(3-a\bar{a}+3a\bar{a}d(1-q)^2-2q)^2}$$

We plot this function versus d and q for $a = 1/2$ in Figure 6.3 and see that changes in d have a large effect on S when q is small.

Returning again to speedup, we note that Equation 6.2 is only valid if $q_1 > 0$ and $q_2 > 0$. If both of these values are equal to zero (i.e., we never send messages), then speedup reduces to

$$S = \frac{4a\bar{a}}{\bar{a}(1+ad) + a^2} \quad (6.5)$$

and if $d = 0$ in this case we get

$$S = \frac{4a\bar{a}}{\bar{a} + a^2} \quad (6.6)$$

Coincidentally, this is also the formula we get if $q_1 = q_2 = 1$ or if $q_1, q_2 < 1$ and we always send null messages. For the $q_1 = q_2 = d = 0$ case, the system travels between states (A,0,B). In the null message case, the system travels between the states (C,0,D). Both systems produce the same probabilities and speedup. These systems produce the optimum speedup that can be gained from the conservative model. Equation 6.5 is plotted in Figure 6.4 and Equation 6.6 is plotted in Figure 6.5.

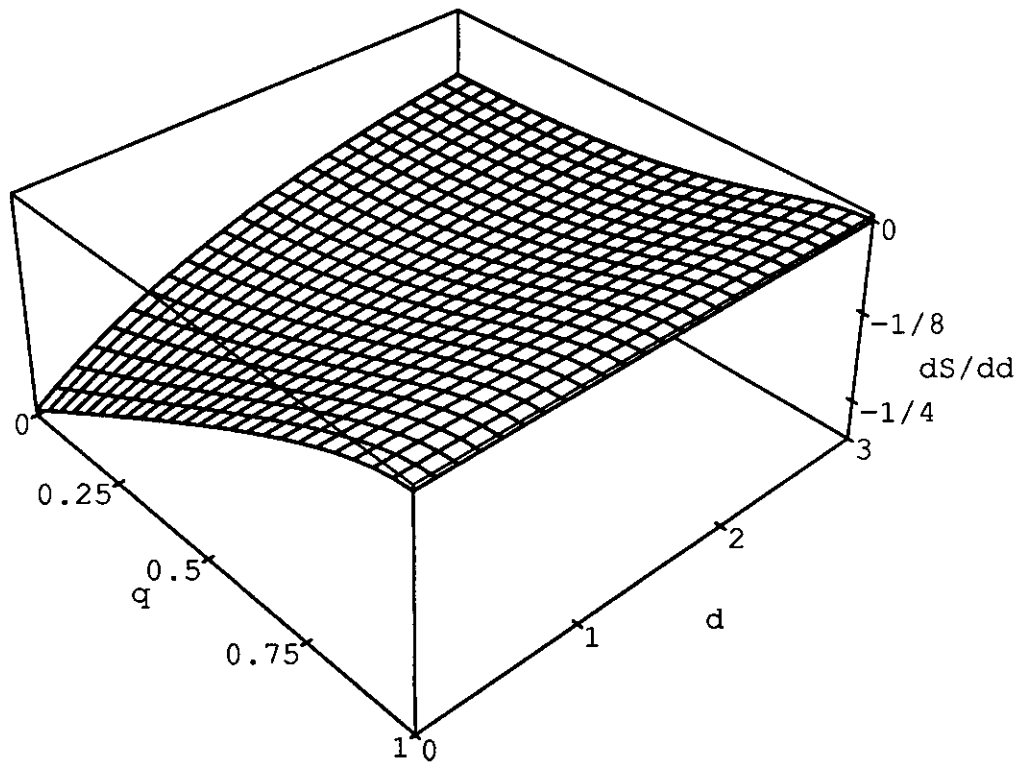


Figure 6.3: Derivative of speedup with respect to d (the cost of breaking a deadlock) versus q and d for $a = 1/2$.

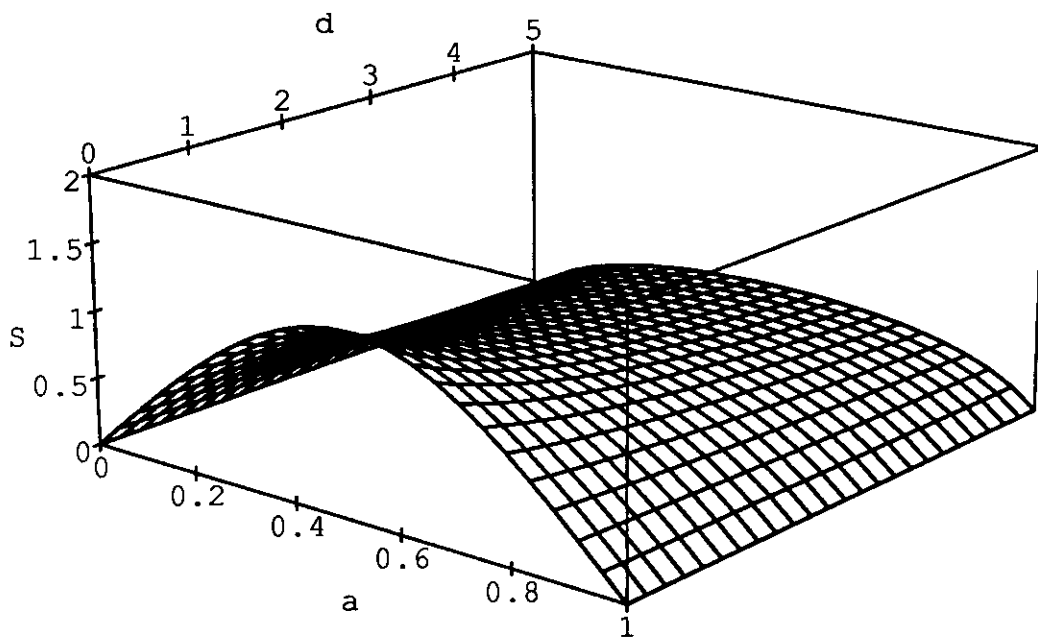


Figure 6.4: Speedup versus a and d for $q_1 = q_2 = 0$.

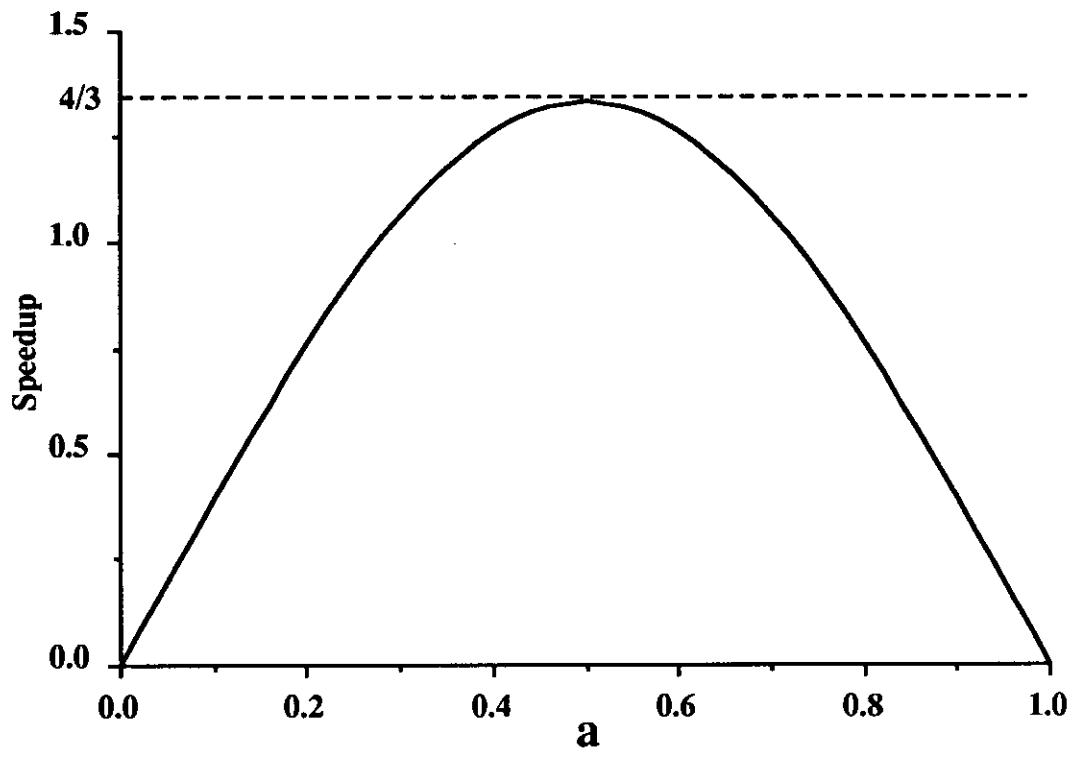


Figure 6.5: Maximum conservative speedup (i.e. for the system with null messages).

6.4 Lookahead

It has been noted by several researchers that exploiting lookahead is necessary to make conservative simulation a viable alternative to the optimistic approach [Fuj88a] [Nic88]. Lookahead is the ability of a logical process to predict its future behavior and especially its future output. In conservative simulation, when a process gives any downstream neighbor processes information about the arrival (or lack thereof) of future messages, the downstream processes are able to continue processing, thus enabling more parallelism in the system. The typical example of lookahead occurs in a FIFO queueing process. If jobs have a deterministic service time of S seconds (of simulated time), then if a server is empty at real time t and virtual time v , it can notify any downstream neighbor that no customer will arrive to this downstream queue with a virtual time stamp less than $v + S$. Therefore recipient processes are able to execute any events they may have scheduled for virtual time less than $v + S$ (assuming no other input links).

6.4.1 Types of Lookahead

In order to formulate a model for a system using lookahead, we need to be very precise about what sort of future prediction is available. One example of this future prediction is that a process might always be able to inform the other processes of the virtual time of the next message it is going to send, but not the contents. With this sort of information, the receivers in a conservative system would be able to process all messages that had virtual times less than the time of the “scheduled” virtual time of the next message. In a two processor system each processor would execute messages with timestamps less than the virtual

time of the “future” message, then wait for the arrival of that message. This system has the same performance as a TW system with no cost for state saving and rollback. TW is really forced to “wait” for the arrival of the message, but it is actually just performing useless work instead of waiting. Both systems return to processing useful work at the instant that the “straggler” message arrives.

Another type of lookahead is information that bounds the virtual time of future messages. The typical example (a FIFO queue) was given in the previous section. If we know something about the process that is being simulated, we may be able to provide information to downstream processes.

The type of lookahead that we use in our model was introduced by Nicol [Nic91]. We can think of lookahead as the ability to transmit messages in our future to other processors. The farther into the future we are allowed to “pre-compute”, the more lookahead we have. Nicol points out that there are two pieces of information contained in a lookahead message. The first is the virtual time of the pending message, the other is the actual contents of the message. Our previous example conveys only virtual time information while, in general, we could transmit both virtual time and data information. Nicol calls the lookahead with time and data information “full lookahead” while the time only message is “time lookahead”. We use the idea of full lookahead in the next model due to its analytical tractability.

6.5 The Lookahead Model

Our definition of lookahead is based on a model which only allows processors to advance a single step in virtual time when advancing. By assuming that the processes have K -step full lookahead, each of the two processes is able to be

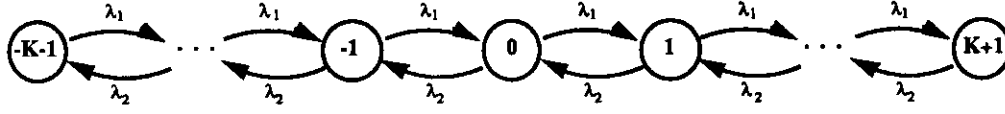


Figure 6.6: State diagram for a system with K -step lookahead.

at most $K + 1$ units of virtual time (events) ahead of the other (as opposed to K messages ahead). Essentially we believe that a process is able to give the other process the content of any messages up to K virtual time units in the future. By assuming that null messages are used, each processor always knows its position relative to the other. Note that if $K = 0$, this model reverts to the simple no-lookahead model where a processor must wait when it gets ahead at all. The state diagram for this system is very simple and is shown in Figure 6.6.

The balance equations for this system are:

$$\lambda_1 p_k = \lambda_2 p_{k+1} \quad k = -K - 1, \dots, 0, \dots, K$$

$$p_0 = 1 - \sum_{i=1}^{K+1} (p_i + p_{-i})$$

The solution is

$$p_k = \left(\frac{a}{\bar{a}}\right)^k p_0 \quad k = -K - 1, \dots, 0, \dots, K + 1$$

$$p_0 = \frac{a^{K+1} (a - \bar{a}) \bar{a}^{K+1}}{a^{3+2K} - \bar{a}^{3+2K}}$$

Speedup relative to the equivalent single processor implementation is

$$S = \frac{4a\bar{a} (a^{2+2K} - \bar{a}^{2+2K})}{a^{3+2K} - \bar{a}^{3+2K}} \quad a \neq \frac{1}{2} \quad (6.7)$$

$$S = \frac{4(1+K)}{3+2K} \quad a = \frac{1}{2} \quad (6.8)$$

Equation 6.7 is plotted versus a and K in Figure 6.7. We can see from this figure that lookahead is extremely useful when the processors are nearly balanced in processing speed ($a = 1/2$). In the imbalanced situation, the faster processor quickly runs out to its limit of K steps, then waits for the other processor to move forward before it can continue again. By taking the derivative of speedup with respect to K , we see this result more clearly. In Figure 6.8 we show $\partial S/\partial K$. When K is small and a is near $1/2$, any change in K has a major effect on speedup, though once we move away from $a = 1/2$ or $K > 5$, the impact is significantly reduced. The moral of this story is to make sure the processes progress at nearly the same rate in virtual time or lookahead will be useless.

6.6 Comparison to Time Warp

We now make a direct comparison between the speedup results obtained from our Time Warp models and conservative models derived in the previous sections. To clearly display the tradeoffs, we compare simplified versions of each. Figure 6.9 shows the ratio of speedup for the conservative model using null messages but no lookahead to Time Warp with no cost for state saving and rollback. It is clear that “free” Time Warp is always a winner since the ratio never exceeds one. The optimistic approach with no costs for its aggressive computation is always better.

Let us now compare free TW to the conservative model with lookahead when both systems are operating at $a = 1/2$ and when the conservative system has

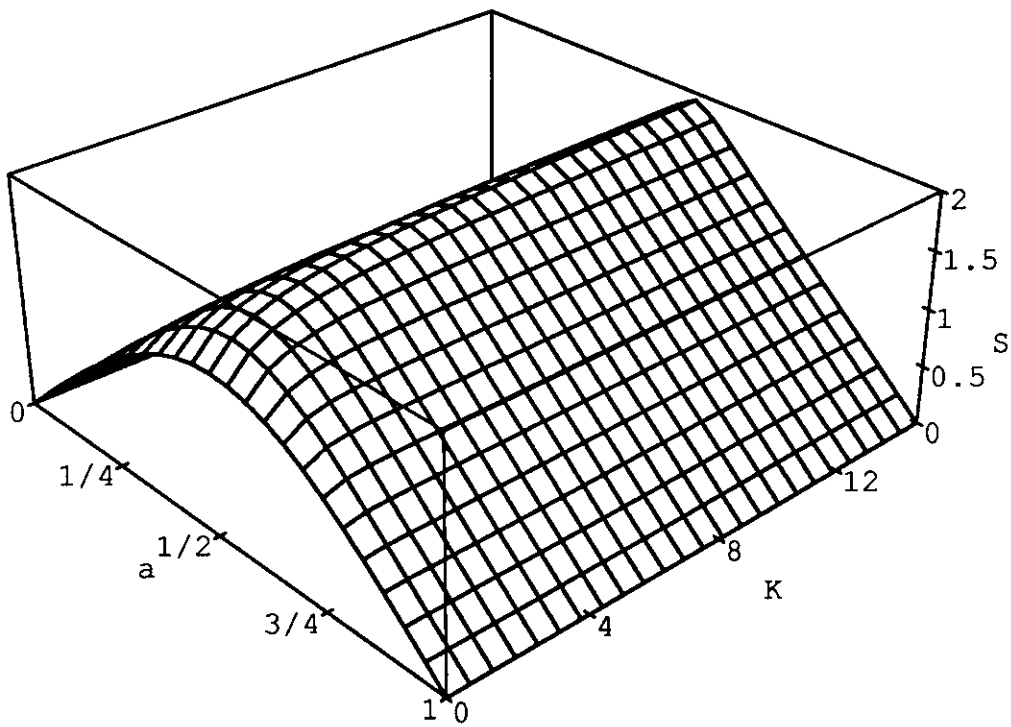


Figure 6.7: Speedup for a K -step lookahead conservative system.

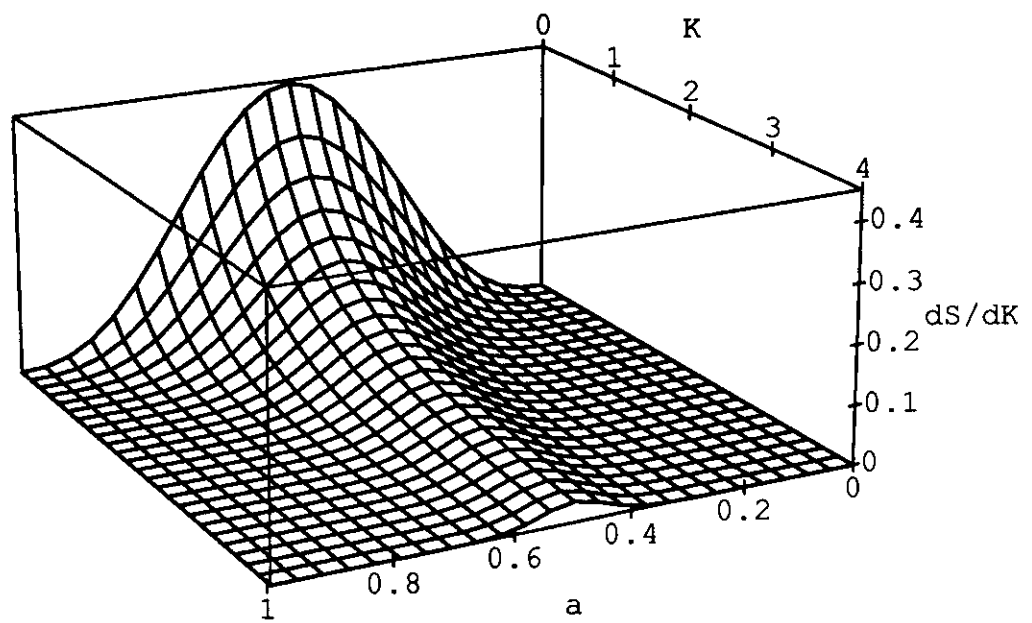


Figure 6.8: Derivative of speedup with respect to K .

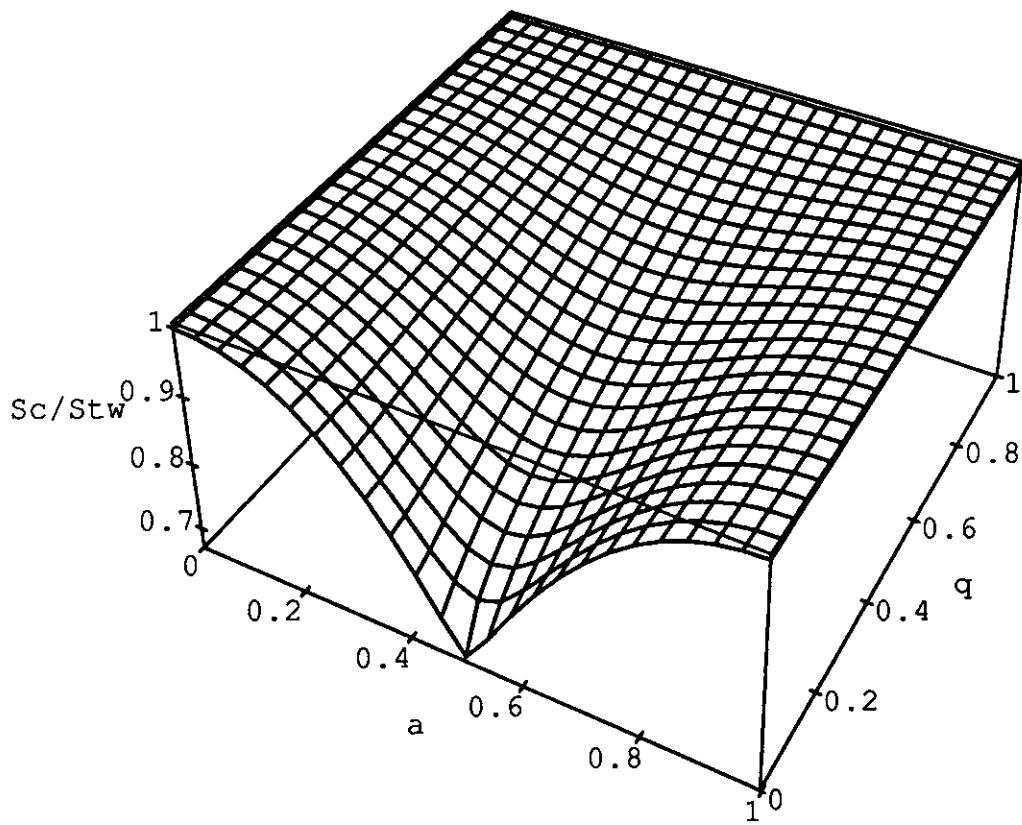


Figure 6.9: Ratio of conservative speedup (no lookahead) to “free” Time Warp speedup.

K -step lookahead. Proponents of the optimistic approach point out that their systems work well regardless of whether lookahead is exploited. Our comparison is an attempt to see how well the conservative approach exploiting lookahead fares with respect to a Time Warp system that uses no lookahead. This ratio is plotted in Figure 6.10 and suggests that a little lookahead combined with null messages goes a long way. For almost any value of K greater than one, we see that the conservative model outperforms “free” Time Warp (ratio > 1). We find the threshold where the conservative approach beats TW by solving the following inequality for K .

$$\frac{S_{\text{cons}}}{S_{\text{TW}}} = \frac{(1+K)(2+\sqrt{q})}{3+2K} \geq 1 \quad (6.9)$$

The condition for the conservative approach to beat Time Warp is

$$K \geq \frac{1-\sqrt{q}}{\sqrt{q}} \quad (6.10)$$

For q (the interaction parameter) very small we need a large lookahead, but for $q > 0.1$, K only needs to be 1 or 2. Figure 6.11 shows the areas of the $q - K$ plane where the conservative approach beats “free” Time Warp. Note that if an optimistic system with no rollback and state saving costs is afforded the same lookahead as a conservative system with no cost for null message transmissions, the optimistic approach will always perform better since it is able to aggressively compute along the critical path for free.

6.7 Conclusions

This chapter examined some simple two processor models for the conservative synchronization method. It showed that lookahead is very useful in gaining performance, but only if the processors are well balanced in processing capacity.

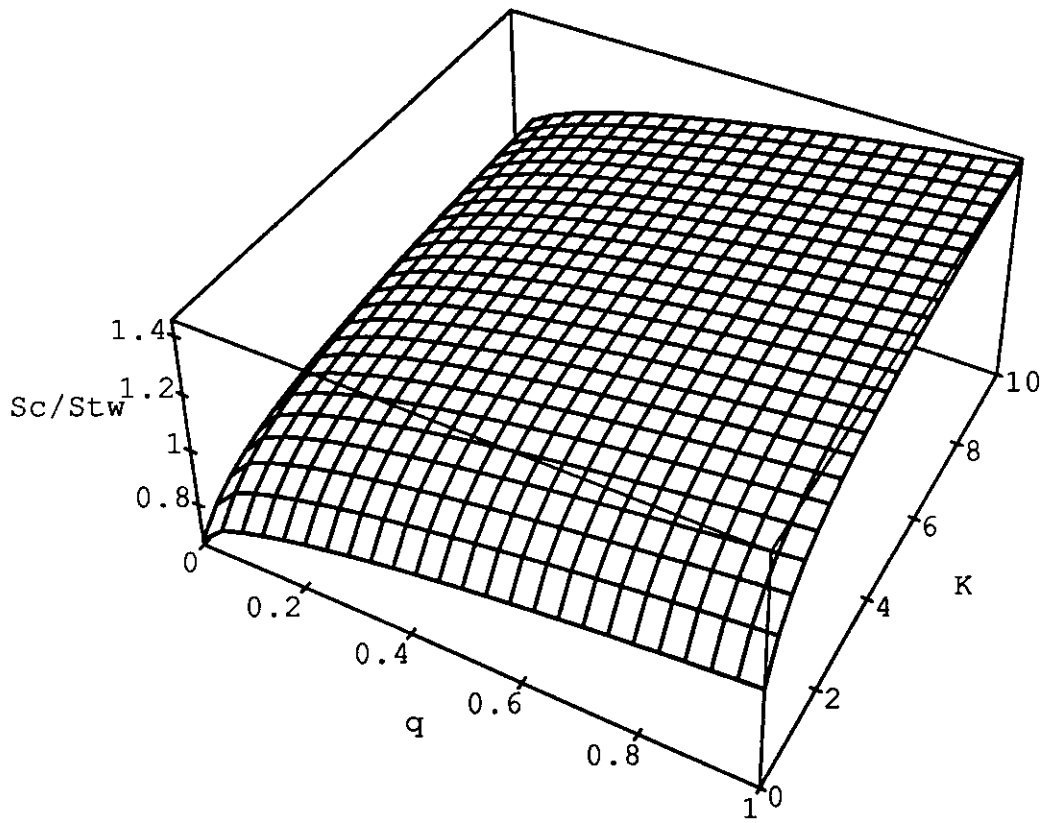


Figure 6.10: Ratio of conservative speedup with K -step lookahead to “free” Time Warp with no lookahead.

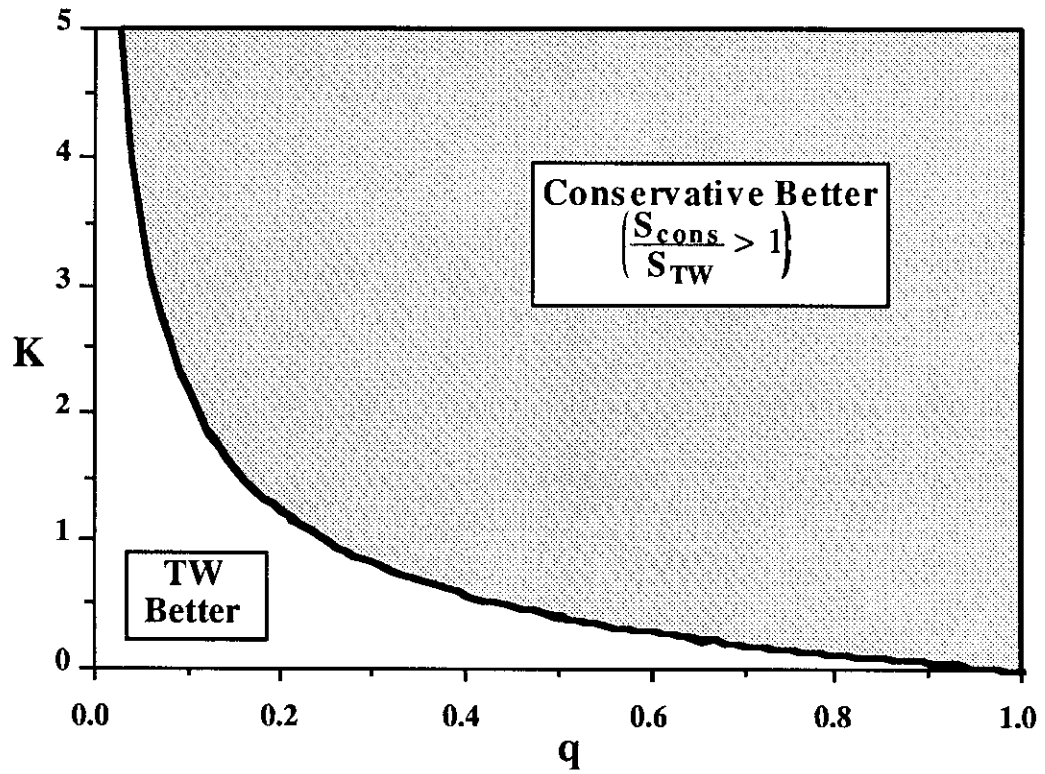


Figure 6.11: Area of the $q - K$ plane where the conservative approach with lookahead wins out.

The models allowed quantitative evaluation of the improvement attributed to null messages, as well as the degradation due to a cost for breaking deadlocks. Finally, a conservative system with “free” null messages and a small amount of lookahead was shown to outperform a Time Warp system with no cost for state saving or rollback. However, if they both incorporate lookahead, then TW is the winner. Unfortunately for the conservative approach, lookahead is not often easy to come by [Nic88] [Fuj88a]. The simple FIFO queueing system provides great lookahead, but add in preemptive-priority queueing and all the lookahead disappears. It may be unwise to utilize a synchronization mechanism which needs lookahead to perform well.

CHAPTER 7

Extensions of the Optimistic Model to Multiprocessors ($P > 2$)

7.1 Introduction

With the exception of Chapters 2 and 6 the previous chapters have introduced and evaluated models for two processor optimistic systems. This chapter explores methods for attacking the important problem of Time Warp running on multiple processors. Most distributed simulation systems do not run on two processors; numbers on the order of tens or hundreds of processors is more likely. How then might we extend our analysis of the optimistic approach to more than two processors? Our first attempt might be to utilize our basic technique with P processors. Unfortunately, the Markov chain approach, with the difference in virtual times between $(P - 1)$ pairs of processors as a state variable, quickly becomes intractable. Even with only three processors, the state space and transitions become fairly complex. Therefore, our technique is to look to the two processor model for hints on calculating bounds or approximations for the multiprocessor case. The next section introduces the model for Time Warp running on many processors.

7.2 Definition of the Multiple Processor System

The multiple (P) processor model for Time Warp is a straightforward extension to the model developed in Chapter 3 and is similar to the model introduced by Nicol [Nic91]. Each processor advances independently along its own virtual time axis processing events. The assumption again is that processors only make single steps forward in virtual time at each advance and that virtual times are restricted to the integers. After a processor advances, it sends a message to exactly K randomly chosen processors out of the other $P - 1$ processors. The messages are only be used for synchronization and do not carry work (as was the case in Chapter 4). Messages that arrive in the future are ignored. Messages that arrive in the past cause a rollback. With more than two processors comes the possibility for “cascading” rollbacks. Each processor must maintain a queue of messages that it has sent to other processors. If processor P_1 is forced to rollback to virtual time v , then it must “cancel” any messages it sent to other processors with virtual time greater than v . These cancellation messages may potentially cause a rollback at the receiving processor, forcing it to send its own cancellation messages, etc. Receivers only rollback to the time of the erroneous message; they do not necessarily rollback all the way back to virtual time $v + 1$.

The performance measure of interest is speedup and is measured as the rate of virtual time progress of the P processor system divided by the rate of an equivalent single processor. We assume that all processors take an exponential amount of time with mean $1/\lambda$ to complete an event. The equivalent single processor moves at rate λ .

7.3 A Simple Upper Bound on Speedup

A simple upper bound on the speedup achievable by this P processor system was motivated by our two processor analysis. As noted in Section 3.7, if each processor always sends a message to the other after advancing, then the whole system takes time equal to the maximum of two exponential delays to move forward one step in virtual time. We can use this idea to create a bound on speedup for the P processor case.

Assume that each processor sends to K other processors after advancing. Group the processors into clusters of size $K+1$ and instead of randomly selecting K processors to send a message to, each processor always sends to all the other K processors in its cluster after advancing. The clusters do not overlap (i.e., no communication between clusters), each, on average, takes time equal to the average of the maximum of $K+1$ exponential delays to move forward one unit of virtual time. Each cluster moves independently. This model provides an upper bound on speedup since each cluster moves forward without synchronizing with any of the other clusters. In the true system each processor is able to receive messages from all the others and is forced to stay more closely synchronized.

Defining r as the effective rate that each processor moves in the P processor system, speedup S is

$$S = \frac{rP}{\lambda}$$

The variable r is found by the exponential delay argument described above.

$$\begin{aligned} t &= \text{E}[\text{Time for processors in a cluster to move 1 step}] \\ &= \frac{1}{\lambda} \sum_{i=1}^{K+1} \frac{1}{i} \end{aligned}$$

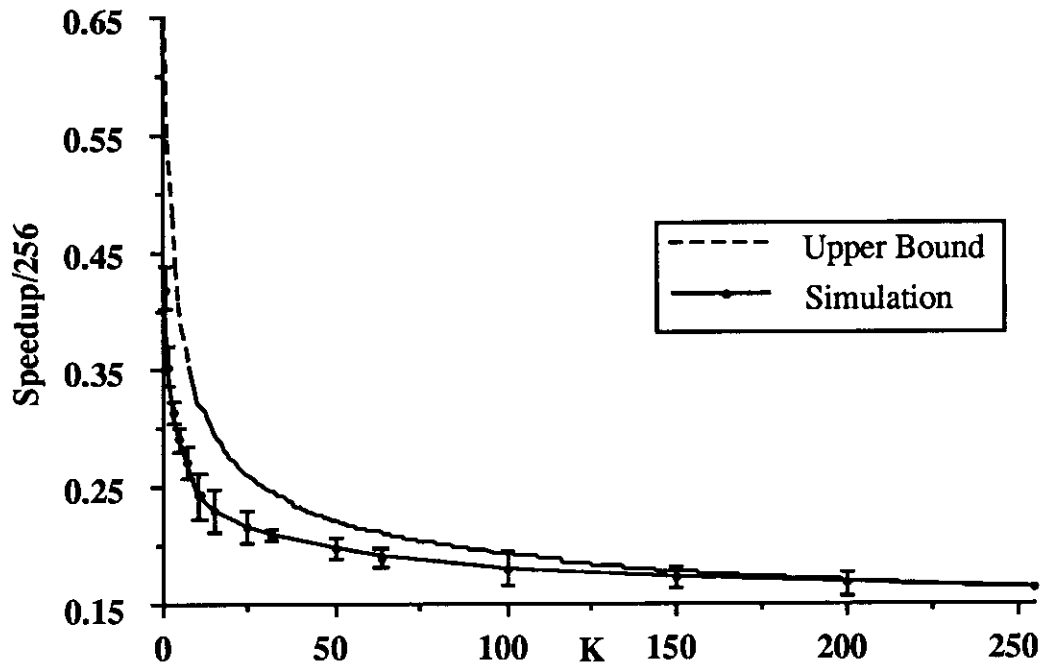


Figure 7.1: Normalized speedup versus K for 256 processors.

$$r \triangleq \frac{1}{t} = \lambda / \left(\sum_{i=1}^{K+1} \frac{1}{i} \right)$$

Therefore the upper bound on speedup is

$$S_u = \frac{P}{\sum_{i=1}^{K+1} \frac{1}{i}} \quad (7.1)$$

In Figure 7.1 we show Speedup/ P versus K for 256 processors. As one might expect, the upper bound is not very tight when K is small, but is exact for $K = P - 1$ and quite tight for $\frac{K}{P} > \frac{1}{2}$. Figure 7.2 shows the percentage error between the upper bound and simulation.

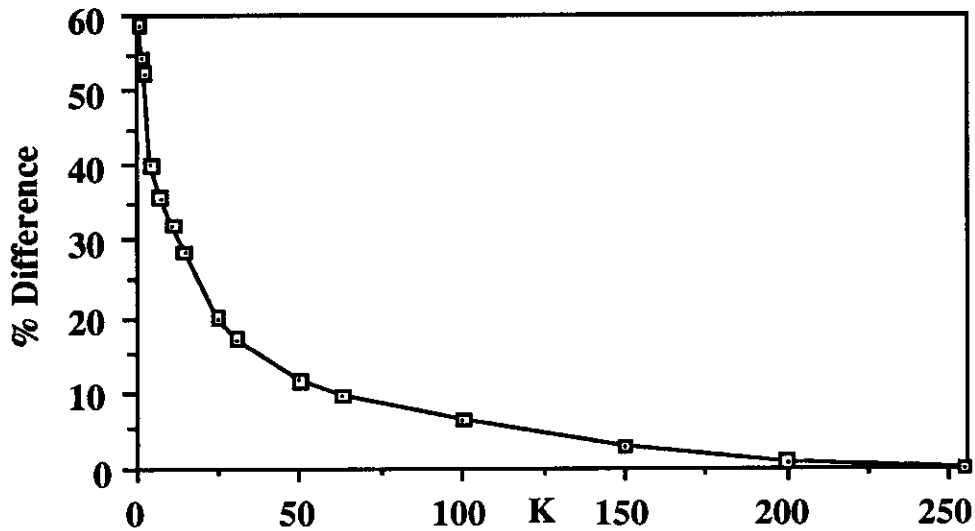


Figure 7.2: Percent difference between the upper bound and simulation.

7.4 Tracking Global Virtual Time Advancement

Another technique that might be employed to better understand the behavior of Time Warp on multiple processors is to directly calculate or estimate the rate of Global Virtual Time (GVT) advancement. This idea is motivated by our two processor work where we had the notion of separation in virtual time between the processors. With both processors at the same virtual time, we know exactly how long before GVT advances one step (both processors have to move). When the event processing time is exponentially distributed at rate λ , the time to advance GVT is simply the time it takes for the maximum of two exponentials to complete. This is $\frac{3}{2\lambda}$ by the arguments given in the previous section. In the P processor case, if we were able to calculate the distribution of the number of processors sitting at GVT after GVT advances, we could derive

the average rate of progress of the P processor system and its speedup over a single processor.

Certainly, the minimum number of processors at GVT, after GVT advances, will be $K + 1$ since when the last processor moves forward to allow GVT to advance, it will send messages to K processors and pull them back to the new GVT. This is exactly the upper bound calculated in the previous section. In general there will be more than $K + 1$ processors at GVT after GVT advances so our speedup will be less. It is believed is that a simple probabilistic argument should be able to give us either the expected value we seek or a more useful bound on this number than $K + 1$.

7.5 A Simple Approximation Using Aggregation

Finally, “bootstrapping” our two processor model to create an approximation for P processors might be possible. The most straightforward method would be to use a series of hierarchical aggregation steps. Assume we have a system with four processors numbered one to four. In Chapter 3 the exact solution for two processors with arbitrary rates of processing and arbitrary interaction probabilities was developed. We might use this model to isolate processors one and two and calculate the effective rate of progress of these two processors in the absence of the other two. The same could be done for processors three and four. After these two operations, we are left with two “effective” single processors moving at a rates determined by our original two processor model. Finally, these two “effective” processors may be combined using the two processor model solution again so as to obtain the approximate rate of progress of the four processor system. This approach would work in general for P a power of two.

One of the challenges of this approximation technique is to provide a value for the interaction parameter “ q_i ” in the two processor solution. Somehow, a mapping from the message transmission probabilities in the P processor system needs to be created so as to generate the interaction parameters (q_i). The first attempts at a solution using this method gave us a very low approximation for speedup. It is believed that the amount of rollback caused by each processor was overestimated.

7.6 Conclusions

This chapter provided some ideas on possible approaches to tackling the multiple processor problem. Our first approach provided a crude upper bound on speedup. The GVT advancement technique provided the same upper bound but holds promise for a better bound and/or approximation. Finally, the aggregation technique has the most direct link to our previous work, but the actual details of assigning variables to the interaction parameters needs to be clearly thought out.

CHAPTER 8

Conclusions and Future Work

This dissertation addressed the performance analysis of distributed simulation techniques. Chapter 2 created a simple bound on the potential improvement of asynchronous versus synchronous processing. Our examples were Time Warp and time-stepped simulation. The optimistic Time Warp approach was shown to only gain an $(\ln P)$ improvement where P is the number of processors used by each algorithm.

In Chapters 3, 4 and 5 several models of two processors running the Time Warp algorithm were created. Exact solutions were found for all the performance parameters of interest including speedup, distribution of state buffers needed, distribution of message buffers needed, etc. It was shown for optimum performance that the processors should send as few messages as possible and that processes should be placed on processors such that the average “independent” rate of progress in virtual time is the same for both processors. Finally, the impact of state saving and rollback costs on performance was shown to be significant and that there are times when “throttling” a Time Warp process when it gets too far ahead in virtual time is better than letting it run freely.

A simple two processor model for a conservative synchronization mechanism was developed in Chapter 6 and lookahead was observed to be very useful in gaining performance, but only if the processors are well balanced in processing

capacity. The models allowed us to quantitatively evaluate the improvement due to sending null messages and to establish the degradation in speedup due to a cost for breaking deadlocks. Finally it was shown that a conservative system with “free” null messages and a small amount of lookahead is able to outperform a Time Warp system with no cost for state saving or rollback.

8.1 Future Work

While distributed simulation on multiple processors ($P > 2$) seems the most obvious area for future work, there are several other possible areas as well. They include:

- multiple processes per processor
- communication costs
- an understanding of message-initiating versus self-initiating models
- fault tolerance of Time Warp
- optimistic computation.

The next few sections describe these possible research areas.

8.1.1 Multiple Processes Per Processor

Since most large simulations have more logical processes than physical computational processors, one area of interest is the need to address multiple processes per processor. Our two processor models for Time Warp were predicated on the assumption that each processor only had a single process running on it. The

results do not generalize to multiple processes per processor. Our model has the entire processor rollback whenever a rollback occurs. This would only be accurate if all the processes were rolled back when any were. Another complication is that the processes would each have different virtual times, and therefore could send messages with different timestamps. Multiple processes per processor begins to look very much like the multiple processor case and therefore inherits many of the same difficulties associated with finding a solution.

8.1.2 Communication Costs

Communication costs have not been adequately addressed in any work in this area. In our conservative model, null messages are allowed to propagate without cost, thus encouraging the processors to send as many messages as possible in order to avoid deadlock. This is unrealistic. One can see that communication cost/delay could also have a major impact in the Time Warp system. Consider the following example. If one processor is ahead of the other in virtual time when it sends a message, in our current models, the message arrives instantaneously at the other processor. This means it arrives in the virtual time future of the receiver. If there were communication delay, it might be possible for the trailing processor (receiver) to move forward in virtual time sufficiently so that the delayed message arrives in the receiver's past. This would cause a rollback and decrease performance.

8.1.3 Message-Initiating vs Self-Initiating

Another area that needs exploration is to understand the performance differences between the "self-initiating" and "message-initiating" models. This dis-

inction was discussed by Nicol [Nic91], though his analysis focused on self-initiating models. Our simple two processor model used self-initiating processors, meaning that each processor always had work to do regardless of whether it had received any messages from other processors. In a message-initiating model [GAF91], the processors only perform work in response to the receipt of a message. Our message queueing model in Chapter 4 is somewhat of a hybrid of both. Each processor always performs local work, but also will process messages and send messages that create work for the recipients. At the moment there are no clear results that state whether or not there are significant performance differences between the pure self-initiating and pure message-initiating models.

8.1.4 Optimistic Computation

Our performance analysis of optimistic simulation can actually be applied to a broader class of algorithms than just simulation synchronization. It really provides an analysis for a type of optimistic computation. We need to examine in more detail what differentiates optimistic computation from optimistic simulation. Our results from modelling Time Warp should provide insights into creating and analyzing optimistic computing systems.

8.1.5 Fault Tolerance of Time Warp

The optimistic approach requires the storage of input messages, output messages and a history of states. There seems to be a fair amount of redundancy in the system simply to allow the algorithm to operate. One might believe that some simple additions to the basic algorithm might render it fault tolerant in

the face of processor failures. This would be especially useful as the running times of simulations increases and also if the Time Warp system were to be used for more general areas of computation than distributed simulation.

8.2 Final Remarks

The performance analysis of distributed simulation is still in its infancy. We have provided a strong foundation for the understanding of both the optimistic and conservative synchronization algorithms, but there remains much work to be done.

APPENDIX A

Derivations of Summations for the Two Processor Model

A.1 $P(z)$ Sums Closed Form Derivation

This section provides the derivation of the closed form expression for $P(z)$ found in Section 3.3.

$$\sum_{k=1}^{\infty} (A_1 + A_2 + A_3 \bar{q}_2 \pi) p_k z^k = \left(A_1 + A_2 + A_3 - A_3 \bar{q}_2 \frac{\beta_1 \beta_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \right) P(z) \quad (\text{A.1})$$

$$\begin{aligned} A_1 \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i f_{k-i} &= \frac{A_1 \beta_1}{\bar{\beta}_1} \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \bar{\beta}_1^{k-i} \\ &= \frac{A_1 \beta_1}{\bar{\beta}_1} \sum_{i=0}^{\infty} p_i z^i \sum_{k=1}^{\infty} (\bar{\beta}_1 z)^k \\ &= \frac{A_1 \beta_1}{\bar{\beta}_1} (P(z) + p_0) \left(\frac{\bar{\beta}_1 z}{1 - \bar{\beta}_1 z} \right) \\ &= A_1 F(z) (P(z) + p_0) \end{aligned} \quad (\text{A.2})$$

$$\begin{aligned}
A_1 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} n_i f_{k+i} &= \frac{A_1 \beta_1}{\bar{\beta}_1} \sum_{k=1}^{\infty} z^k \bar{\beta}_1^k \sum_{i=1}^{\infty} n_i \bar{\beta}_1^i \\
&= \frac{A_1 \beta_1}{\bar{\beta}_1} \left(\frac{\bar{\beta}_1 z}{1 - \bar{\beta}_1 z} \right) Q(\bar{\beta}_1) \\
&= A_1 F(z) Q(\bar{\beta}_1)
\end{aligned} \tag{A.3}$$

$$\begin{aligned}
A_2 \bar{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i g_{i-k} &= \frac{A_2 \bar{q}_2 \beta_2}{\bar{\beta}_2} \sum_{i=2}^{\infty} p_i \bar{\beta}_2^i \sum_{k=1}^{i-1} \left(\frac{z}{\bar{\beta}_2} \right)^k \\
&= \frac{A_2 \bar{q}_2 \beta_2}{\bar{\beta}_2} \sum_{i=2}^{\infty} p_i \bar{\beta}_2^i \left(\frac{\frac{z}{\bar{\beta}_2} - \left(\frac{z}{\bar{\beta}_2} \right)^i}{1 - \frac{z}{\bar{\beta}_2}} \right) \\
&= \frac{A_2 \bar{q}_2 \beta_2}{\bar{\beta}_2 - z} \left(\frac{z}{\bar{\beta}_2} \sum_{i=2}^{\infty} p_i \bar{\beta}_2^i - \sum_{i=2}^{\infty} p_i z^i \right) \\
&= \frac{A_2 \bar{q}_2 \beta_2}{\bar{\beta}_2 - z} \left(\frac{z}{\bar{\beta}_2} P(\bar{\beta}_2) - P(z) \right) \\
&= \frac{A_2 \bar{q}_2 \beta_2}{z - \bar{\beta}_2} \left(P(z) - \frac{z P(\bar{\beta}_2)}{\bar{\beta}_2} \right)
\end{aligned} \tag{A.4}$$

$$\begin{aligned}
A_3 \bar{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i \sum_{j=1}^{\infty} f_j g_{j+i-k} &= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i \bar{\beta}_2^{i-k} \sum_{j=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^j \\
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \sum_{i=2}^{\infty} p_i \bar{\beta}_2^i \sum_{k=1}^{i-1} \left(\frac{z}{\bar{\beta}_2} \right)^k \\
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2 \bar{\beta}_2}{(1 - \bar{\beta}_1 \bar{\beta}_2)(z - \bar{\beta}_2)} \left(P(z) - \frac{z P(\bar{\beta}_2)}{\bar{\beta}_2} \right)
\end{aligned} \tag{A.5}$$

$$\begin{aligned}
A_3 \bar{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \sum_{j=1}^{\infty} f_{j+k-i} g_j &= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \bar{\beta}_1^{k-i} \sum_{j=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^j \\
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \bar{\beta}_1^{k-i}
\end{aligned}$$

$$\begin{aligned}
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \sum_{i=0}^{\infty} p_i z^i \sum_{k=i+1}^{\infty} (z \bar{\beta}_1)^{k-i} \\
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{1 - \bar{\beta}_1 \bar{\beta}_2} (P(z) + p_0) \left(\frac{\bar{\beta}_1 z}{1 - \bar{\beta}_1 z} \right) \\
&= \frac{A_3 \bar{q}_2 \bar{\beta}_1 \beta_2 F(z)}{1 - \bar{\beta}_1 \bar{\beta}_2} (P(z) + p_0) \tag{A.6}
\end{aligned}$$

$$\begin{aligned}
A_3 \bar{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} n_i \sum_{j=1}^{\infty} g_j f_{j+k+i} &= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \sum_{k=1}^{\infty} z^k \bar{\beta}_1^k \sum_{i=1}^{\infty} n_i \bar{\beta}_1^i \sum_{j=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^j \\
&= \frac{A_3 \bar{q}_2 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \left(\frac{\bar{\beta}_1 z}{1 - \bar{\beta}_1 z} \right) Q(\bar{\beta}_1) \left(\frac{\bar{\beta}_1 \bar{\beta}_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \right) \\
&= \frac{A_3 \bar{q}_2 \bar{\beta}_1 \beta_2 F(z) Q(\bar{\beta}_1)}{1 - \bar{\beta}_1 \bar{\beta}_2} \tag{A.7}
\end{aligned}$$

A.2 Speedup Sums Closed Form Derivation

In this section we derive the closed form expression for Speedup found in Section 3.4.1.

$$\begin{aligned}
A_3 q_2 p_0 \sum_{i=2}^{\infty} f_i \sum_{j=1}^{i-1} g_j (i-j) &= \frac{A_3 q_2 p_0 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \sum_{i=2}^{\infty} \bar{\beta}_1^i \sum_{j=1}^{i-1} \bar{\beta}_2^j (i-j) \\
&= \frac{A_3 q_2 p_0 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \sum_{j=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^j \sum_{i=j+1}^{\infty} \bar{\beta}_1^{i-j} (i-j) \\
&= \frac{A_3 q_2 p_0 \beta_1 \beta_2}{\bar{\beta}_1 \bar{\beta}_2} \left(\frac{\bar{\beta}_1 \bar{\beta}_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \right) \left(\frac{\bar{\beta}_1}{\beta_1^2} \right) \\
&= \frac{A_3 q_2 p_0 \bar{\beta}_1 \beta_2}{\beta_1 (1 - \bar{\beta}_1 \bar{\beta}_2)} \tag{A.8}
\end{aligned}$$

$$A_2 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{k-1} i g_{k-i} = \frac{A_2 q_2 \beta_2 C_p p_0}{\bar{\beta}_2} \sum_{i=1}^{\infty} i \left(\frac{1}{r_1} \right)^i \sum_{k=i+1}^{\infty} \left(\frac{\bar{\beta}_2}{r_1} \right)^{k-i}$$

$$\begin{aligned}
&= \frac{A_2 q_2 \beta_2 C_p p_0}{\beta_2} \left(\frac{r_1}{(r_1 - 1)^2} \right) \left(\frac{\bar{\beta}_2}{r_1 - \bar{\beta}_2} \right) \\
&= \frac{A_2 q_2 \beta_2 C_p p_0 r_1}{(r_1 - \bar{\beta}_2)(r_1 - 1)^2} \tag{A.9}
\end{aligned}$$

$$\begin{aligned}
&A_3 q_2 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{k+i-1} j g_{k+i-j} \\
&= \frac{A_3 q_2 \beta_1 \beta_2 C_p p_0}{\beta_1 \bar{\beta}_2} \sum_{k=1}^{\infty} \left(\frac{1}{r_1} \right)^k \sum_{i=1}^{\infty} \bar{\beta}_1^i \sum_{j=1}^{k+i-1} j \bar{\beta}_2^{k+i-j} \\
&= (\dots) \sum_{k=1}^{\infty} \left(\frac{1}{r_1} \right)^k \left[\sum_{j=1}^{k-1} j \bar{\beta}_2^{k-j} \sum_{i=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^i + \sum_{j=k}^{\infty} j \bar{\beta}_1^{j-k} \sum_{i=j-k+1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^{k+i-j} \right] \\
&= \frac{A_3 q_2 \beta_1 \beta_2 C_p p_0}{1 - \bar{\beta}_1 \bar{\beta}_2} \sum_{k=1}^{\infty} \left(\frac{1}{r_1} \right)^k \left[\sum_{j=1}^{k-1} j \bar{\beta}_2^{k-j} + \sum_{j=k}^{\infty} j \bar{\beta}_1^{j-k} \right] \\
&= \frac{A_3 q_2 \beta_1 \beta_2 C_p p_0}{1 - \bar{\beta}_1 \bar{\beta}_2} \left[\sum_{j=1}^{\infty} j \left(\frac{1}{r_1} \right)^j \sum_{k=j+1}^{\infty} \left(\frac{\bar{\beta}_2}{r_1} \right)^{k-j} + \sum_{j=1}^{\infty} j \bar{\beta}_1^j \sum_{k=1}^j \left(\frac{1}{\bar{\beta}_1 r_1} \right)^k \right] \\
&= (\dots) \left[\left(\frac{r_1}{(r_1 - 1)^2} \right) \left(\frac{\bar{\beta}_2}{r_1 - \bar{\beta}_2} \right) + \left(\frac{1}{\bar{\beta}_1 r_1 - 1} \right) \left(\sum_{j=1}^{\infty} j \bar{\beta}_1^j - \sum_{j=1}^{\infty} j \left(\frac{1}{r_1} \right)^j \right) \right] \\
&= \frac{A_3 q_2 \beta_1 \beta_2 C_p p_0}{1 - \bar{\beta}_1 \bar{\beta}_2} \left[\left(\frac{r_1}{(r_1 - 1)^2} \right) \left(\frac{\bar{\beta}_2}{r_1 - \bar{\beta}_2} \right) + \left(\frac{1}{\bar{\beta}_1 r_1 - 1} \right) \left(\frac{\bar{\beta}_1}{\beta_1^2} - \frac{r_1}{(r_1 - 1)^2} \right) \right] \\
&= \frac{A_3 q_2 \beta_2 C_p p_0}{(1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - 1)^2} \left(\frac{(r_1 - \bar{\beta}_1)}{\beta_1} + \frac{\beta_1 \bar{\beta}_2 r_1}{(r_1 - \bar{\beta}_2)} \right) \tag{A.10}
\end{aligned}$$

$$\begin{aligned}
A_3 q_1 \sum_{k=1}^{\infty} p_k \sum_{i=1}^{\infty} f_i \sum_{j=1}^{\infty} j g_{k+i+j} &= \frac{A_3 q_1 \beta_1 \beta_2 C_p p_0}{\beta_1 \bar{\beta}_2} \sum_{k=1}^{\infty} \left(\frac{\bar{\beta}_2}{r_1} \right)^k \sum_{i=1}^{\infty} (\bar{\beta}_1 \bar{\beta}_2)^i \sum_{j=1}^{\infty} j \bar{\beta}_2^j \\
&= \frac{A_3 q_1 \beta_1 \beta_2 C_p p_0}{\beta_1 \bar{\beta}_2} \left(\frac{\bar{\beta}_2}{r_1 - \bar{\beta}_2} \right) \left(\frac{\bar{\beta}_1 \bar{\beta}_2}{1 - \bar{\beta}_1 \bar{\beta}_2} \right) \left(\frac{\bar{\beta}_2}{\beta_2^2} \right) \\
&= \frac{A_3 q_1 \beta_1 \bar{\beta}_2^2 C_p p_0}{\beta_2 (1 - \bar{\beta}_1 \bar{\beta}_2)(r_1 - \bar{\beta}_2)} \tag{A.11}
\end{aligned}$$

APPENDIX B

Cubic Equation Solution for the Message Queueing Model

In this appendix the roots of a cubic equation to be used in Section 4.3 are derived. This material is taken directly from the *CRC Handbook of Mathematical Sciences* [Bey87].

A cubic equation, $y^3 + p_y y^2 + q_y y + r_y = 0$ may be reduced to the form,

$$x^3 + a_x x + b_x = 0$$

by substituting for y the value $x - p_y/3$. Here

$$a_x = \frac{1}{3}(3q_y - p_y^2) \text{ and } b_x = \frac{1}{27}(2p_y^3 - 9p_y q_y + 27r_y)$$

The form $x^3 + ax + b = 0$ with $ab \neq 0$ can always be solved by transforming it to the trigonometric identity

$$4 \cos^3(\theta) - 3 \cos(\theta) - \cos(3\theta) \equiv 0$$

as follows:

let $x = m \cos(\theta)$, then

$$\begin{aligned} x^3 + ax + b &\equiv 0 \\ &\equiv m^3 \cos^3(\theta) + am \cos(\theta) + b \end{aligned}$$

$$\begin{aligned} &\equiv 4 \cos^3(\theta) - 3 \cos(\theta) - \cos(3\theta) \\ &\equiv 0. \end{aligned}$$

Hence

$$\frac{4}{m^3} = -\frac{3}{am} = \frac{-\cos(3\theta)}{b},$$

from which it follows that

$$m = 2\sqrt{-\frac{a}{3}}, \quad \cos(3\theta) = \frac{3b}{am}.$$

Any solution θ_1 which satisfies $\cos(3\theta) = \frac{3b}{am}$, will also have the solutions

$$\theta_1 + \frac{2\pi}{3} \quad \text{and} \quad \theta_1 + \frac{4\pi}{3}$$

The roots of the cubic $x^3 + ax + b = 0$ are therefore

$$\begin{aligned} x_1 &= m \cos\left(\theta_1 + \frac{2\pi}{3}\right) \\ x_2 &= m \cos(\theta_1) \\ x_3 &= m \cos\left(\theta_1 + \frac{4\pi}{3}\right) \end{aligned}$$

For the denominator of $P(z)$ we have

$$\begin{aligned} r_y &= -\frac{\bar{a}(A - aq_1)\bar{q}_2}{a^2} \\ q_y &= \frac{A + a\bar{a}q_1\bar{q}_2}{a^2} \\ p_y &= -\frac{1 + A}{a}. \end{aligned}$$

These values can then be substituted into the solutions given above to find r_1 , r_2 , and r_3 . The values for s_i are symmetric in (a, \bar{a}) and (q_1, q_2) to the r_i values.

REFERENCES

- [Bel90] Steven Bellenot. "Global Virtual Time Algorithms." In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pp. 122–127. Society for Computer Simulation, January 1990.
- [Bey87] William H. Beyer, editor. *CRC handbook of mathematical sciences*. CRC Press, 6th edition, 1987.
- [Bry77] R.E. Bryant. "Simulation of packet communication architecture computer systems." Technical Report MIT,LCS,TR-188, Massachusetts Institute of Technology, Cambridge, Mass., 1977.
- [CHM79] K.M. Chandy, Victor Holmes, and J. Misra. "Distributed Simulation of Networks." *Computer Networks*, **3**(2), 1979.
- [CM79] K. Mani Chandy and Jayadev Misra. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *IEEE Transactions on Software Engineering*, **SE-5**(5), 1979.
- [CM80] K.M. Chandy and J. Misra. "Termination detection of diffusing computations in communicating sequential processes." Technical Report TR-144, Computer Science Dept. University of Texas at Austin, Austin, TX, 1980.
- [CM81] K.M. Chandy and J. Misra. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations." *Communications of the ACM*, **24**(11), 1981.
- [FK90a] Robert E. Felderman and Leonard Kleinrock. "Two Processor Time Warp Analysis: Capturing the Effects of Message Queueing and Rollback/State Saving Costs." Submitted to *ACM Transactions on Modelling and Computer Simulation*, November 1990.
- [FK90b] Robert E. Felderman and Leonard Kleinrock. "An Upper Bound on the Improvement of Asynchronous Versus Synchronous Distributed Processing." In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pp. 131–136. Society for Computer Simulation, January 1990.
- [FK91] Robert E. Felderman and Leonard Kleinrock. "Two Processor Time Warp Analysis: Some Results on a Unifying Approach." In *Proceed-*

ings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, volume 23,1, pp. 3–10. Society for Computer Simulation, January 1991.

- [FTG88] Richard M. Fujimoto, Jya-Jang Tsai, and Ganesh Gopalakrishnan. “Design and Performance of Special Purpose Hardware for Time Warp.” In *Proceedings of the 15th International Symposium on Computer Architecture*, June 1988.
- [Fuj88a] Richard M. Fujimoto. “Lookahead in Parallel Discrete Event Simulation.” In *International Conference on Parallel Processing*, 1988.
- [Fuj88b] Richard M. Fujimoto. “Performance Measurements of Distributed Simulation Strategies.” In *Proceedings of the SCS Multiconference on Distributed Simulation*. The Society for Computer Simulation, July 1988.
- [Fuj89a] Richard M. Fujimoto. “Performance Measurements of Distributed Simulation Strategies.” *TRANSACTIONS of The Society for Computer Simulation*, 6(2), 1989.
- [Fuj89b] Richard M. Fujimoto. “Time Warp on a Shared Memory Multiprocessor.” Technical Report UUCS-88-021a, Computer Science Department, University of Utah, Salt Lake City, UT 84112, January 1989.
- [GAF91] A. Gupta, I. F. Akyildiz, and R. M. Fujimoto. “Performance Analysis of “Time Warp” With Homogeneous Processors and Exponential Task Times.” In *Proceedings of the 1991 SIGMETRICS Conference*, pp. 101–110. Association for Computing Machinery, May 1991.
- [GKP89] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley Publishing Co., 1989.
- [Jef85] David R. Jefferson. “Virtual Time.” *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.
- [JM84] D. Jefferson and A. Motro. “The Time Warp Concurrency Control Mechanism For Distributed Databases.” Technical Report TR-84-302, Computer Science Department, University of Southern California, January 1984.
- [Jol61] L.B.W. Jolley. *Summation of Series*. Dover Publications, Inc., second revised edition, 1961.

- [JW84] David Jefferson and Andrew Witkowski. "An Approach to Performance Analysis of Timestamp-driven Synchronization Mechanisms." In *Proceedings of the 3rd annual ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C. Canada, August 27-29 1984.
- [KF91] Leonard Kleinrock and Robert E. Felderman. "Two Processor Time Warp Analysis: A Unifying Approach." *International Journal of Computer Simulation*, late 1991.
- [Kle75] Leonard Kleinrock. *Queueing Systems: Volume 1: Theory*. John Wiley and Sons, Inc., 1975.
- [Kle89] Leonard Kleinrock. "On Distributed Systems Performance." In *Proceedings of the 7th ITC Specialist Seminar, Adelaide, Australia*. ITC, September 1989. (Also published in "Computer Networks and ISDN Systems" vol. 20, no.1-5, pp. 206-215, December 1990.).
- [KW78] Granino A. Korn and John V. Wait. *Digital Continuous-System Simulation*. Prentice-Hall, Englewood Cliffs, N.J., 1978.
- [KW85] Clyde P. Kruskal and Alan Weiss. "Allocating Independent Subtasks on Parallel Processors." *IEEE Transactions on Software Engineering*, SE-11(10), October 1985.
- [Lam78] L. Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System." *Communications of the ACM*, 21(7):558-564, July 1978.
- [LL89] Yi-Bing Lin and Edward D. Lazowska. "A Study of Time Warp Rollback Mechanisms." Technical Report 89-09-07, Department of Computer Science and Engineering, University of Washington, November 1989.
- [LL90a] Yi-Bing Lin and Edward D. Lazowska. "Optimality Considerations for "Time Warp" Parallel Simulation." In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pp. 29-34. Society for Computer Simulation, January 1990.
- [LL90b] Yi-Bing Lin and Edward D. Lazowska. "Processor Scheduling for Time Warp Parallel Simulation." Technical Report 90-03-03, Department of Computer Science and Engineering, University of Washington, February 1990.

- [LL90c] Yi-Bing Lin and Edward D. Lazowska. "Reducing the State Saving Overhead For Time Warp Parallel Simulation." Technical Report 90-02-03, Department of Computer Science and Engineering, University of Washington, February 1990.
- [LLB89] Yi-Bing Lin, Edward D. Lazowska, and Jean-Loup Baer. "Parallel trace-Driven Simulation of Multiprocessor Cache Performance: Algorithms and Analysis." Technical Report 89-07-06, Department of Computer Science and Engineering, University of Washington, July 1989.
- [LMS83] Steven Lavenberg, Richard Muntz, and Behrokh Samadi. "Performance Analysis of a Rollback Method for Distributed Simulation." In *Performance '83*, pp. 117–132. North-Holland, 1983.
- [LSW89] B. Lubachevsky, A. Shwartz, and A. Weiss. "Rollback Sometimes Works... If Filtered." In *Proceedings of the 1989 Winter Simulation Conference*, pp. 630–639, December 1989.
- [Lub87] B.D. Lubachevsky. "Efficient Parallel Simulations of Asynchronous Cellular Arrays." *Complex Systems*, 1:1099–1123, 1987.
- [Mad89] Vijay Krishna Madiseti. "Self Synchronizing Concurrent Computing Systems." Technical Report UCB/ERL M89/122, Electronics Research Laboratory, College of Engineering University of California Berkeley, CA 94720, October 1989.
- [Mis86] Jayadev Misra. "Distributed Discrete-Event Simulation." *Computing Surveys*, 18(1):39–65, March 1986.
- [MM84] Debasis Mitra and I. Mitrani. "Analysis and Optimum Performance of Two Message-Passing Parallel Processors Synchronized by Rollback." In *Performance '84*, pp. 35–50. North-Holland, 1984.
- [MNF91] Vijay Madiseti, David Nicol, and Richard Fujimoto, editors. *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, volume 23,1. Society for Computer Simulation, January 1991.
- [MWM90] Vijay Madiseti, Jean Walrand, and David Messerschmitt. "Synchronization in Message-Passing Computers: Models, Algorithms and Analysis." In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pp. 35–48. Society for Computer Simulation, January 1990.

- [Nic88] D. M. Nicol. "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks." *SIGPLAN Not.*, **23**(9):124–137, September 1988.
- [Nic90a] D. M. Nicol. "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations." Technical Report 90-20, Institute for Computer Applications in Science and Engineering(ICASE), May 1990.
- [Nic90b] David Nicol, editor. *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1. Society for Computer Simulation, January 1990.
- [Nic91] David M. Nicol. "Parallel Self-initiating Discrete-Event Simulations." *Transactions on Modelling and Computer Simulation*, **1**(1):24–50, January 1991.
- [PWM79] J. Kent Peacock, J.W. Wong, and Eric G. Manning. "Distributed Simulation Using a Network of Processors." *Computer Networks*, **3**(1):44–56, 1979.
- [See79] M. Seethalakshmi. "A study and analysis of performance of distributed simulation." Master's thesis, Computer Science Dept. University of Texas at Austin, Austin, TX, 1979.
- [UF89] Brian Unger and Richard Fujimoto, editors. *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21,2. Society for Computer Simulation, March 1989.
- [UJ88] Brian Unger and David Jefferson. *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 19,3. Society for Computer Simulation, July 1988.
- [WL89] D.B. Wagner and E.D. Lazowska. "Parallel Simulation of Queueing Networks: Limitations and Potentials." In *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE '89*, volume 17,1, pp. 146–155, May 1989.