Efficient Parallel Simulations and Their Application to Communication Networks

by

Jain-Chung J. Wang

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

APPROVED:

Marc Abrams

Marc Abrams, Chairman

[illegible]

Dennis G. Kafura

[illegible]

Scott F. Midkiff

[illegible]

Richard E. Nance

Calvin J. Ribbens

Calvin J. Ribbens

December, 1994

Blacksburg, Virginia

Efficient Parallel Simulations and Their Application to Communication Networks

by

Jain-Chung J. Wang

Committee Chairman: Marc Abrams

Computer Science

(ABSTRACT)

Simulation is one of the most important tools for system performance evaluation in communication networks as well as many other areas. However, simulation is computationally intensive. A traditional sequential simulation of a complex model or a rare-event system may require days or even weeks of computer execution time. Therefore, simulation often becomes a bottleneck of a performance study. With the growing availability of multi-processor computing systems (e.g., tightly-coupled parallel computers or distributed networks of workstations), parallel simulation, which parallelizes a simulation program for execution on multiple processors, becomes an attractive means to reduce simulation execution time.

With few exceptions, existing parallel simulation algorithms can be broadly classified into four methods: *multiple replication, time-parallel, parallel regenerative, and space-parallel.* Each method is associated with some advantages and limitations. We study these methods and propose a number of parallel simulation algorithms for a class of communication network systems modeled by queueing systems.

In multiple replication simulation, each processor simulates a replication of the target simulation model independently. Due to the lack of *a priori* knowledge about the steady-state conditions, an arbitrarily selected initial state is often used for each simulation run. This can result in significant bias in the simulation outcome. To reduce this initial transient bias, we propose a polling initialization technique, in which a pilot simulation is used to find 'good' initial states that are representative of the steady-state conditions.

Time-parallel simulation obtains parallelism by partitioning the time domain of the simulation model into a number of batches. Each batch is computed by a processor independently. Time-parallel simulation has not been fully explored by the research community partly because finding the exact initial states for the batches is often challenging and problem dependent. We develop two approximation time-parallel simulation algorithms for acyclic networks of loss G/G/1/K and G/D/1/K queues. These algorithms exploit unbounded parallelism and can achieve near-linear speedup when the number of arrivals is large. Two other time-parallel approaches are also proposed for Markov chains. For more general simulation models, an approximation approach that uses a substate matching technique is presented.

Parallel regenerative simulation exploits parallelism by partitioning the simulation trajectory into a number of regeneration cycles. The amount of parallelism relies on the regeneration frequency of the model. In practice a regeneration state that has a short expected regeneration cycle length often does not exist in the target simulation model. As a result, a sufficient number of observations can not be obtained in a finite simulation interval. To overcome this constraint, we propose a partial regeneration algorithm that uses a substate matching technique to increase the number of observations.

When the memory requirement of the target simulation models exceeds the storage capacity of a single processor, space-parallel simulation is an appropriate method. In space-parallel simulation, the target simulation model is decomposed into a number of components such that each component contains a disjoint subset of the model state variables. Each component is mapped into a logical process which is responsible for computing the trajectory corresponding to the component over the simulation time interval. An important class of space-parallel simulation is the conservative simulation, in which each logical process can proceed processing an event only if the process ensures that no event will arrive later with a smaller timestamp.

A number of previous experimental studies have suggested that *lookahead*, a capability that allows a simulation to look into the simulation time future, plays an important role in

the performance of the conservative simulation. Although the performance of conservative simulation has been the interest in many previous studies, there has been a lack of formal arguments to quantify the impact of lookahead to conservative simulation performance. To address this question, we develop stochastic models to study the relationship between the amount of lookahead and the simulation performance with respect to different model topologies. We show that for closed simulation models, the simulation execution time is proportional to the amount of lookahead. For open models, on the other hand, lookahead is effectively useless when the simulation length is long.

ACKNOWLEDGEMENTS

I am grateful to a large number of people for their support during my graduate years at Virginia Tech. I am greatly indebted to my advisor Professor Marc Abrams for his advise and continuous encouragement. His guidance and patience was indispensable to the completion of this work.

I thank the members of my dissertation committee, Professors Dennis Kafura, Scott Midkiff, Richard E. Nance, and Calvin Ribbens for their valuable comments and suggestions.

I would like to express my gratitude to my friends and colleagues RongSheng Liu and Tom Yu for discussions which directly contribute to some of the proofs in this dissertation. Thanks should also be expressed to my good friends Siva Challa, Qizhong Chen, Arindam Datta, Yi Lin, Anup Mathur, Yanchun Su, and Devendra Vazirani for their consistent friendship and sharing in the past years. I can never forget the fun times that I had with them.

Special thanks are due to my brother Jian-Ren for his friendship and 'consulting services'. Without his inspiration I would not have gone to graduate school. Similar sentiments apply to my sisters Wendy and Fung-Mei.

Finally, I can not thank my wife Grace enough for her care, patience, and moral support throughout these years. Without her, I could not have completed my Ph.D. program.

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

Simulation has been one of the most important tools to study system performance in many areas, such as meteorology, economics, and computer networking, in which analytical solutions to estimate the system performance are generally unavailable. However, because simulation is computationally intensive, executing a simulation model often requires hours or even days of CPU time. The *real* (i.e., wall clock) time required by executing the simulation model is referred to as the *simulation execution time*. This research studies *parallel simulation* which refers to the parallel execution of a simulation model on multiple processors for a purpose of reducing the simulation execution time.

1.1 Motivation

Since the first commercial digital network was installed in the U.S. in the early 1960s, computer networks have become crucial to information communication and computing. New digital carrier technology has increased communication bandwidth by several orders of magnitude and the link speed of current internets has a range as large as 1000:1. With such a rapid growth of communication networks, many questions are often asked by researchers and engineers, such as: What changes must be made to an existing communication network to allow addition of higher speed connections? What additions or changes must be made to an existing network to accommodate a wider range (e.g., voice, video, images) of communication services? What type of performance can we expect for a network using new technologies? Therefore, tools that can help study the performance of existing or planned communication networks are highly desirable.

1.1.1 Why Simulation?

When studying the performance of a communication network, experimentation can yield important insights about protocols and their implementations. However, experimentation is limited by practical consideration to a relatively small number of hosts and network architectures. Communication networks have also been studied using analytical approaches for years [8, 17, 51, 79]. However, analytical techniques are often limited in the number of network connections that can be analyzed [9, 52, 80] and usually require assumptions that are violated by real networks, such as stationary Poisson arrival processes, independent service time distributions, and Markovian routing [33, 42, 46]. In addition, analytical techniques often require validation using simulation. This leaves simulation as the only tool to study communication networks with arbitrary configurations, traffic, and protocols.

1.1.2 Why Parallel Simulation?

A major drawback of simulation is its requirement of intensive computation. Simulating a 'big' system may require days, weeks, or even months of computer time. A system can be big in the following cases:

1. The state vector of the simulation model consists of a large number of variables.
2. The event of interest occurs rarely so that a long simulation length is required.
3. The system behavior is highly variable so that a large number of simulation runs is required to obtain a reliable estimate for the 'average' system behavior.

For the first case, consider an Internet model which contains thousands of nodes. Because of the large system state vector, simulating such a system needs an enormous amount of computer resources, both storage and CPU cycles.

An example for the second case is an ATM (asynchronous transfer mode) switch, in which the *cell* (i.e., the data unit used in an ATM network) loss rate could be on the order of 10^{-9} or lower [28, 85]. To obtain a reliable loss rate estimate, a simulation will require a number of cell arrivals that is many times larger than 10^9. Although some variance reduction techniques using importance sampling [40, and references therein], which could largely reduce the simulation length (while maintaining an accurate estimate), have been proposed, these variance reduction techniques require sophisticated statistical skills to find an appropriate importance sampling distribution. Also, these techniques reveal only the limiting distribution of the system and hence can not be used to investigate the dynamic behavior of the system [50]. Moreover, systems of interest may include a number of interconnected switches. In such cases how to apply the importance sampling technique remains an open question [40].

To exemplify the third case, consider a connectionless datagram network. Suppose we are interested in the frequency that a datagram is routed through a particular node for a pair of sending and receiving nodes. If many possible paths are available for a datagram to reach the receiving node, then a large number of simulation runs is required for an accurate estimate.

To deal with such a dilemma, parallel processing provides a promising solution to speed up simulations. In particular, with the advent of massively parallel computers, where hundreds to thousands of processors are connected together in a single computer to provide up to teraflop computing power, parallel simulation provides great potential in reducing the simulation execution time by several orders of magnitude.

1.2 Problem Statement and Objectives

Parallel simulation partitions a simulation for a parallel execution on multiple processors. That is, the computation required by simulation is distributed among multiple processors so that events are executed by different processors concurrently. Parallel simulation has been recognized as a challenging problem [27]. The difficulty is threefold: *parallelism, efficiency*, and *accuracy*. More specifically, a simulation algorithm has to exploit sufficient parallelism to support parallel execution while maintaining low overhead and highly reliable estimates. It is usually not straightforward to achieve this requirement. Consider simulation of a queueing network. To obtain parallelism, we usually divide the set of queues into a number of subsets, each computed by a processor [11, 44]. Then the amount of parallelism is determined by the number of subsets. To ensure that events are executed in a correct order, synchronization is required among the participating processors. However, the synchronization cost grows (e.g., quadratically for a fully connected network) as the number of subsets increases.

Consider another example where a packet multiplexer modeled by a single server queue is simulated. For such a small model, we may simply let each processor simulate the entire model independently and then average the results from all runs together in the end to obtain an estimate for the measure of interest [37]. To justify the use of multiple processors, the simulation time of each processor has to decrease as the number of processors used increases. However, this will in turn increase the initial transient bias and hence result in less accurate simulation results.

The objectives of this research work are to:

1. Explore a variety of methods to obtain a better understanding of parallel simulation.
2. Develop a suite of useful parallel simulation algorithms suitable for communication network models.
3. Evaluate the performance of the proposed algorithms analytically or experimentally, or both.
4. Identify the limitations of the proposed algorithms with respect to model characteristics such as topology and workload.

1.3 Organization

This dissertation is organized as follows: In Chapter 2 we review four major parallel simulation methods: *multiple replication*, *time-parallel*, *parallel regenerative* and *space-parallel.*

The advantages and limitations of each method are identified. Chapter 3 describes the simulation models considered in this research. In Chapter 4, use of parallel processors through multiple replication simulation is studied. The problem of initial transient bias is addressed. A polling initialization technique that deals with this problem is proposed. Chapter 5 develops a set of time-parallel simulation algorithms for a variety of simulation models. In Chapter 6, we consider parallel regenerative simulation. Theories of regenerative simulation are reviewed. Limitations that arise when applying this approach to practical models are addressed. A partial regenerative algorithm that enhances the applicability of parallel regenerative simulation is proposed. In Chapter 7, space-parallel simulation is investigated. We examine the impact of lookahead to the conservative space-parallel simulation. We formally derive the relationship between lookahead and the simulation performance for both open and closed simulation models. Chapter 8 summarizes this research and suggests some avenues for future research.

Chapter 2

PARALLEL SIMULATION PRIMER

The time evolution of a simulation model can be viewed as a realization of a random process $\mathbf{X} = \{X(t), t \geq 0\}$. Therefore, *the execution of a simulation model is a process of computing a sample path of the corresponding random process.* A number of parallel simulation algorithms have been proposed in the past [13, 27, 38, 47, and references therein]. With few exceptions, each of these algorithms falls into one of the following four methods: *multiple replication*, *space-parallel*, *time-parallel*, and *parallel regenerative*. Each of these methods will also be referred to as a *framework* of parallel simulation. These methods are reviewed individually in the following sections.

2.1 Multiple Replication Simulation

Consider the case when $X(t)$ converges in distribution to some random variable X as $t \rightarrow \infty$. If the goal of simulation is to estimate the steady-state mean $\mu = E(X) = (1/\tau) \int_0^\tau X(t)dt$, where $\tau \rightarrow \infty$, then an intuitive parallel simulation approach is to make each processor simulate a replication (representing the entire simulation model) of the stochastic process independently beginning with a fixed state for a pre-determined simulation run length $T > 0$. When all runs are completed, the results from the runs are then averaged together to obtain an estimate for the measure of interest. This approach is referred to as the multiple replication method [37, 32]. We discuss how to construct a point estimate and a confidence interval using this approach next.

Let P denote the number of processors. Also, let M_i and Y_{ij}, respectively, denote the number of *observations* and the j^{th} observation generated by processor $i, 1 \leq i \leq P$, during the simulation such that

$$Y_{ij} = \int_{t_{ij-1}}^{t_{ij}} X(t)dt,$$

for some $0 \leq t_{ij-1} < t_{ij} \leq T$. Then each processor i obtains an estimate for μ, denoted $\bar{Y}_i$, where

$$\bar{Y}_i = \frac{\sum_{j=1}^{M_i} Y_{ij}}{T}.$$

Then an obvious estimator for μ, denoted $\hat{\mu}_1(P,T)$, based on $\bar{Y}_i's$ can be obtained by:

$$\hat{\mu}_1(P,T) = \frac{\sum_{i=1}^{P} \bar{Y}_i}{P}.$$

By the central limit theorem, we can obtain an approximate $100(1-\alpha)$ percent confidence interval for μ by:

$$\hat{\mu}_1(P,T) \pm z_{1-\alpha/2}\sqrt{\frac{S^2(P)}{P}},$$

where $S^2(P) = \sum_{i=1}^{P} \frac{[\bar{Y}_i - \hat{\mu}_1(P,T)]^2}{P-1}$ is the sample variance, and $z_{1-\alpha/2}$ is the upper $1-\alpha/2$ critical point for the standard normal distribution (i.e., N(0,1)).

Multiple replication simulation is attractive due to its simplicity (each processor in fact performs a sequential simulation) and low overhead (no communication is required among processors during the simulation). However, because a simulation generally does not start from the 'steady-state condition' (which is unknown before the simulation) of the system, the transient period before the system reaches the steady-state will thus cause a bias in the simulation outcome. Also, in a simulation where rare events are of interest (e.g., cell losses for an ATM switch), a long simulation run (as opposed to multiple short runs in a multiple replication simulation) is required to obtain a reliable estimate. Moreover, Heidelberger [37] has shown that when the number of processors is large, multiple replication simulation is not statistically efficient. Generally speaking, multiple replication simulation is appropriate only if all of the following conditions are satisfied:

1. The memory requirement of each replication does not exceed the memory of a single processor.

2. The initial transient period is short.

3. The number of processors is relatively small.

4. Rare instances are not the main interest of the simulation.

5. For a trace-driven simulation the number of processors usable is bounded by the number of traces available.

2.2 Time-Parallel Simulation

Chandy and Sherman [13] proposed a *space-time* model which describes simulation as a process of *filling-in* a space-time rectangle. The space-time model suggests that parallelism can also be obtained through a time-domain partitioning. Time-parallel simulation that uses time-domain partitioning can be generally described as follows. Suppose that a simulation model has M state variables, $v_1, v_2, \ldots, v_M$. Let $v_k(t)$ denote the value of state variable v_k at t. Then the system state at time t is represented by an M-tuple: $X(t) = (v_1(t), v_2(t), \ldots v_M(t))$. Let $t_1, t_2, \ldots, t_n$, where $0 \leq t_1 < t_2 \ldots < t_n \leq \tau$, be all the time points when the system changes its state. The *trajectory* of a simulation model represented by a random process $X(t)$ in the simulation time interval $[0, \tau]$ can be represented by the sequence $X(0), X(t_1), \ldots, X(t_n)$, where $X(0)$ is the initial state of the system. In a time-parallel simulation, the simulation time interval is partitioned into P intervals: $[b_0, b_1], (b_1, b_2], \ldots, (b_{P-1}, b_P]$, where $0 = b_0 < b_1 \ldots < b_P = \tau$, and $\{b_0, \ldots, b_P\} \cap \{t_1, \ldots, t_n\} = \emptyset$. The subtrajectory in interval $[b_{k-1}, b_k]$ for $k = 1$, and $(b_{k-1}, b_k]$ for $1 < k \leq P$, is referred to as *batch* k. Then $X(b_{k-1})$ and $X(b_k)$ are the initial state and final state, respectively, of batch k. In time-parallel simulation, each batch is computed concurrently by a processor.

Because of its time-domain partitioning, time-parallel simulation can potentially yield massive parallelism even when the simulation model is small (e.g., [34]). However, to achieve efficient time-parallel simulation, the batch initial states have to be determined efficiently.

That is, a prediction of states that occur in the simulation time future is required. This limits the applicability of time-parallel simulation. Most existing time-parallel simulation algorithms are designed for specific models, for instance, queueing systems [34, 59, 67, 88, 89, 90], Markov chains [4], and cache memory systems [39].

2.3 Parallel Regeneration Simulation

A random process is said to be *regenerative* if there exists an increasing *regeneration time* sequence $t_0, t_1, \ldots, t_n$ (where $t_0 = 0$), such that processes defined in the time intervals $[t_0, t_1), [t_1, t_2), \ldots$ are independent and identically distributed (IID) replications. The time interval between two consecutive regeneration times is called a *regeneration cycle*. The system state at the regeneration time points $t_0, t_1, \ldots$ is referred to as the *regeneration state*. For example, in a GI/G/1 queue, the system reaches a regeneration state whenever a job arrives at an empty queue as illustrated in Figure 2.1.

Because the process in each regeneration cycle is IID, this property provides potential parallelism for parallel simulation. For example, in Figure 2.1, each processor can simulate the process for one regeneration cycle independently. That is, the model trajectory is partitioned at the regeneration time points (i.e., $t_0, t_1, \ldots$). Therefore, parallel regeneration simulation can be viewed as a special case of time-parallel simulation. Because regeneration simulation represents a very important class of simulation, we consider this approach individually. Most previous research in parallel regeneration simulation is due to Glynn and Heidelberger [31, 38].

The regenerative structure of a random process allows us to construct an accurate point estimate and a confidence interval for the measure of interest. Further details of parallel regenerative simulation will be given in Chapter 6. The regenerative structure also allows us to partition the execution of a simulation easily. However, a (parallel) regeneration simulation requires that the target process regenerates frequently so that sufficient IID observations can be obtained in the desired simulation time interval. This requirement

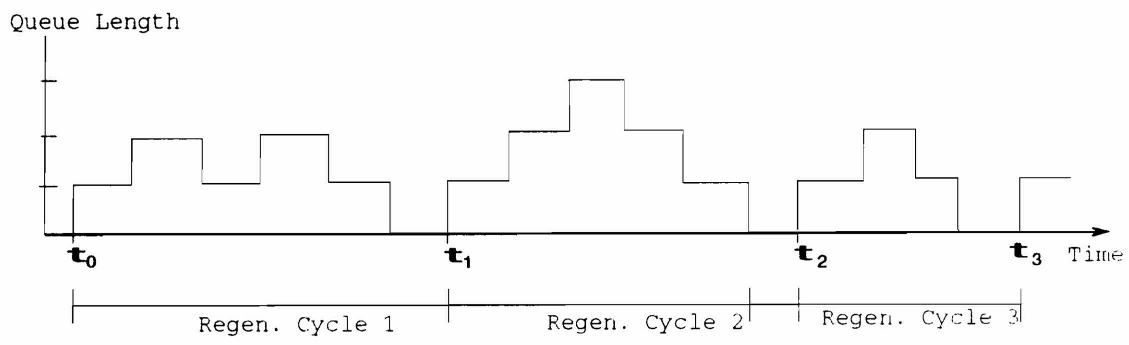


Figure 2.1: A trajectory of a GI/G/1 queue. The system regenerates at t_0, t_1, t_2, and t_3.

largely limits the applicability of regeneration simulation.

2.4 Space-Parallel Simulation

Space-parallel simulations decompose a simulation model into a number of components from the space domain of the model, such that each component contains a disjoint subset of the model state variables. Each component is mapped into a *logical process* (LP) which is responsible for computing the values of the state variables of the corresponding component over the simulation time interval. Each LP maintains a *local clock* whose value is equal to the timestamp of the last event processed by the LP. To avoid causality errors (i.e., events executed out of order), synchronization among LPs is required. LPs communicate with each other by passing time-stamped *messages*.

Two important classes of space-parallel simulations are the *conservative* and the *optimistic* approaches. The major algorithmic difference between these two is that conservative approaches strictly avoid causality errors, while optimistic approaches allow causality errors and provide mechanisms to correct them. The first algorithms that used the conservative approach were developed by Chandy and Misra [11], Bryant [10], and Peacock et. al [73]. Jefferson and Sowizral's [43] Time Warp mechanism developed in 1982 is the first work using the optimistic approach.

2.4.1 Conservative Approaches

In conservative simulation, the communication channel between any pair of LPs is modeled by a *link* which is specified statically. That is, all possible communication between two LPs has to be identified before the simulation run. For each LP, an event, denoted e, can be executed if and only if (1) e has the smallest timestamp among all events that have arrived, and (2) it is guaranteed that all events that arrive later will not have a timestamp smaller than that of e. This policy ensures a correct execution (i.e., by the order of timestamps) of events throughout the simulation. However, it can also cause an LP to block unnecessarily. More details of the conservative approach will be discussed in section 7.1.

2.4.2 Optimistic Approaches

In contrast to conservative approaches, optimistic approaches do not prohibit causality errors from occurring. The following discussion of the optimistic simulation is based on Jefferson's Time Warp mechanism [44], which is the first optimistic simulation algorithm. In an optimistic simulation, each LP maintains one input message queue, which receives all messages sent to the process. The messages in the queue are sorted by their timestamps. The LP always executes the next smallest time-stamped message in the queue. When an LP receives a message with a timestamp, say t, smaller than the timestamp of a previously processed message, a causality error occurs and the process has to *roll back* to a simulation time point earlier then t. If some messages with timestamps greater than t have already been send to other LPs before the rollback, these messages have to be canceled or *unsent*. To unsend a message, the process sends an *anti-message* to *annihilate* the message to be canceled. If the message to be canceled has already been processed by the destination process when the corresponding anti-message arrives, a rollback in the destination process is again required.

To accomplish rollbacks, each LP has to save the system states periodically. To bound the amount of storage required by such state saving and to determine when the simulation may terminate, an optimistic simulation periodically calculates the *global virtual time* (GVT), which is a lower bound on the earliest simulation time to which any LP can roll back. In other words, all computation before the GVT is guaranteed to be correct, and hence all saved system states whose simulation times are smaller than the GVT can be discarded.

Space-parallel simulation is appropriate when the target simulation model is large such that the entire model can not fit into a single processor. However, process synchronization for a space-parallel simulation is relatively difficult to implement and usually costly. The performance (in terms of the execution time) of the simulator is highly model dependent and hence is often unpredictable. Also, the speedup of a space-parallel simulation is bounded by the number of logical processes. For models with few components such as a single queue, space-parallel simulation is not viable because the limited number of components limits spatial parallelism. A number of variations on the Time Warp mechanism have been proposed in the literature. They include *lazy cancellation* [29], *lazy reevaluation* [93], *moving time windows* [81], *wolf calls* [62], *direct cancellation* [26], and *limited optimism* [78].

2.5 Summary

In this chapter, we review the four major parallel simulation methods. Advantages and limitations of these methods are identified. A summary is given in Table 2.1, where MR, TP, PR, and SP represent multiple replication, time-parallel, parallel regenerative, and space-parallel, respectively. It is clear that for a given problem, a method has to be carefully chosen in order to achieve a 'good' result, where parallelism, efficiency, and accuracy are properly addressed. Table 2.1 and Figure 2.2 give a general guideline for selecting a method. However, the performance and applicability of a particular algorithm are often affected by subtle changes in the simulation model (e.g., the value of a numerical or a topological parameter). Thus, it is impossible to give rules that derive absolute answers as to what method to use without considering the problem specifically.

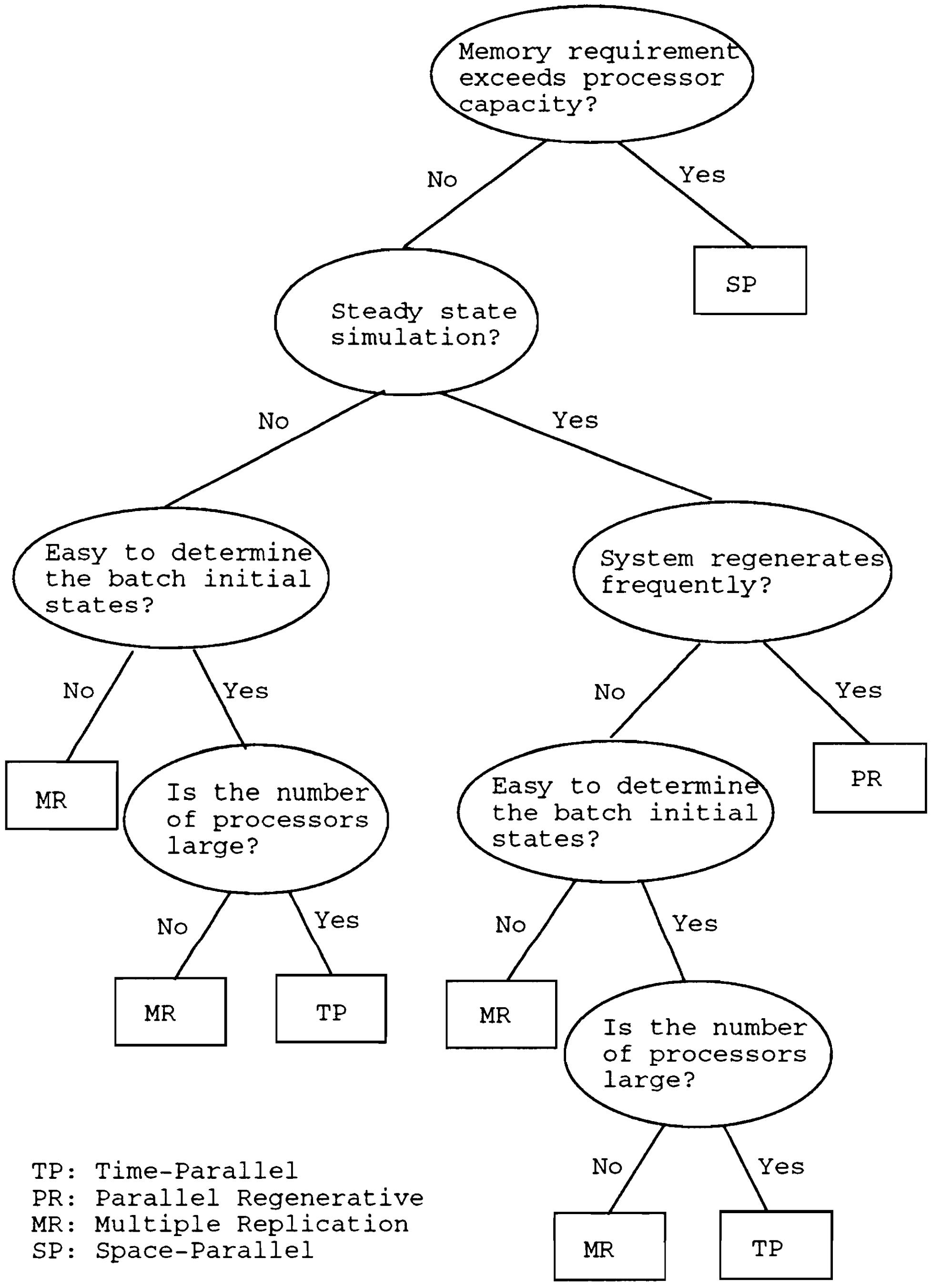

Memory requirement exceeds processor capacity?
No
Yes
SP
Steady state simulation?
No
Yes
Easy to determine the batch initial states?
System regenerates frequently?
No
Yes
No
Yes
MR
Is the number of processors large?
Easy to determine the batch initial states?
PR
No
Yes
No
Yes
MR
TP
MR
Is the number of processors large?
No
Yes
MR
TP
TP: Time-Parallel
PR: Parallel Regenerative
MR: Multiple Replication
SP: Space-Parallel

Figure 2.2: A general guideline for method selection.

Table 2.1: A summary of the advantages and limitations of parallel simulation methods, where MR, TP, PR, and SP represent multiple replication, time-parallel, parallel regenerative, and space-parallel simulation, respectively.

	Advantages	*Limitations*
MR	simple to implement, efficient execution, generate IID observations.	small models only, results subject to initial transient bias, need multiple traces for trace-driven simulations, not suitable for rare event simulation.
TP	potential massive parallelism.	implicit parallelism difficult to discover, model dependent algorithms.
PR	generate IID observations, simple to implement.	regenerative systems only, steady-state simulation only.
SP	provide explicit parallelism, allow large simulation models.	limited parallelism for small models, difficult to implement, difficult to predict performance, high execution overhead.

Chapter 3
SIMULATION MODELS

Four fundamental parallel simulation methods have been discussed in the last chapter. In the subsequent chapters we develop a number of parallel simulation algorithms based on these parallel simulation methods. Each of these algorithms is evaluated empirically with at least one simulation model. This chapter describes the simulation models that are considered for the evaluation of the proposed algorithms.

3.1 Multiplexer and Switch

A communication network consists of a set of computers, or *nodes*, which are interconnected by physical transmission channels. A *host* is a node which transmits and receives data onto and from the network. A *switch* is a node which routes data by diverting the traffic that enters the node from its input channel(s) to its output channel(s). A *multiplexer* is a device which receives traffic from several sources and merges them into a single output channel to share the bandwidth of the channel. In general, there are three multiplexing schemes: *statistical* multiplexing, *time-division* multiplexing (TDM), and *frequency-division* multiplexing (FDM) [8, pp. 52,150-151]. In statistical multiplexing, all traffic streams are merged into a single queue and data are transmitted in a first-come first-serve basis. In time-division multiplexing, multiple source bitstreams are multiplexed into successive frames of one bitstream. Each frame contains multiple fix-sized slots, one for each source bitstream. In frequency-division multiplexing, a physical channel is shared by multiple streams of signals, each constrained to a different portion (in terms of signal frequency) of the available bandwidth. Statistical multiplexing is the most commonly used scheme for computer com-

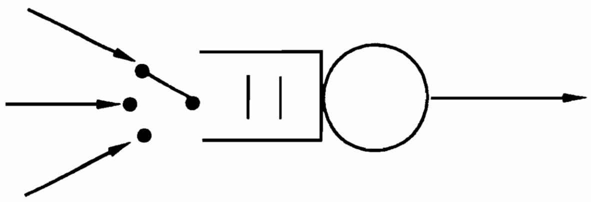


Figure 3.1: A multiplexer.

munication networks. Thus, we consider only statistical multiplexers.

A multiplexer can be modeled by a loss G/G/1/K queue (Figure 3.1) which receives packets from several sources and merges them into a single output stream. Any packet that arrives at a full buffer will be *dropped*, representing a packet loss due to a buffer overflow. A switch can thus be modeled by a set of loss G/G/1/K queues, which routes arriving packets from multiple sources to selected destinations.

3.2 Virtual Circuit

Based on the switching technology, communication networks can be divided into two types: *circuit-switched* and *packet-switched*. In a circuit-switched communication network, before a transmission between any pair of nodes starts, a *circuit*, or the path through which the bitstream will be transmitted to the destination, has to be established. A circuit once established will be dedicated to this connection during the transmission. The classic example of a circuit-switched network is the public telephone system.

In packet-switched networks, bitstreams are grouped into smaller pieces called *packets*. Each packet carries additional information which identifies the destination of the packet such that multiple communications among computers can proceed concurrently. There are two data transmission modes in packet-switched networks, namely *connection-oriented* and *connectionless*

The connection-oriented mode is similar to circuit switching; before a transmission between a pair of computers begins, an end-to-end path called *virtual circuit* (VC) has to be

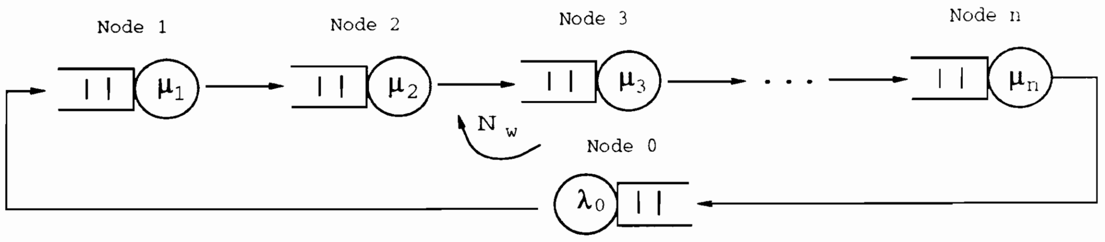


Figure 3.2: A virtual circuit with a sliding-window control. Acknowledgments are transmitted back to the source node via the feedback connection.

set up. Unlike a circuit, a VC (or a portion of it) can be shared by multiple connections. To control the amount of traffic admitted to a VC and to ensure correct data transmission over the VC, an end-to-end control mechanism is required. We consider the sliding-window mechanism [79, section 5.2]) which is commonly used in computer communication networks.

A VC with a sliding-window control is often modeled by a queueing network [79, section 5.2.2] as depicted in Figure 3.2. A sliding-window control mechanism allows packets to be transmitted onto the circuit only if there are less than N_W (or the *window size*) outstanding packets of the connection in the VC. When the destination node receives a packet from the source node, the destination node acknowledges the source node the reception of the packet. Acknowledgments are assumed to be transmitted at a higher priority than regular packets, so the transmission of the acknowledgments is modeled by a direct connection from the destination node to the source node as shown in Figure 3.2. Node 0 is a *virtual* node used to model the arrival process of the source node. When there are N_W outstanding packets from the source node in the VC, node 0 is empty so that no more packets will be transmitted onto the circuit. Otherwise, packets enter the VC at an arrival rate of λ_0.

3.3 Datagram Network

In contrast to connection-oriented transmission, in connectionless transmission no dedicated connection is established. Instead, each individual packet (sometimes referred to as a *datagram*) is routed independently from one end to another. Datagrams may thus arrive

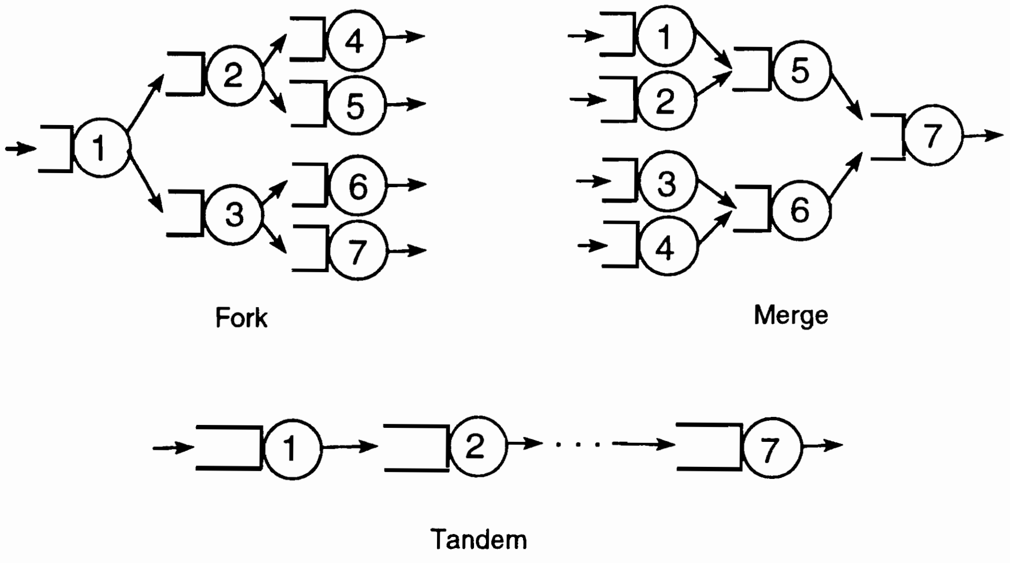


Figure 3.3: Feed-forward queueing models. Each model contains 7 queues.

at the destination node out of order. Also, the datagram may be lost, duplicated, or delayed, but the sender will not be informed for such conditions [16, Chapter 7]. Therefore, connectionless transmission provides *unreliable* service. Due to the lack of 'hand-shaking' between the sender and the receiver, a network which provides connectionless transmission (referred to as a datagram network) can be modeled by an acyclic feed-forward queueing network (assuming that a datagram will not visit a node for more than once).

There are three basic components for acyclic feed-forward networks: fork, merge, and tandem (e.g., Figure 3.3). All feed-forward queueing networks are compositions of these three basic structures.

3.4 Central Server System

Many empirical studies of parallel simulation (e.g., [25, 45, 77, 86]) have used the central server system as shown in Figure 3.4 as a benchmark model for performance evaluation. The system consists of a CPU, two disks, and an arbitrary number of jobs which circulate

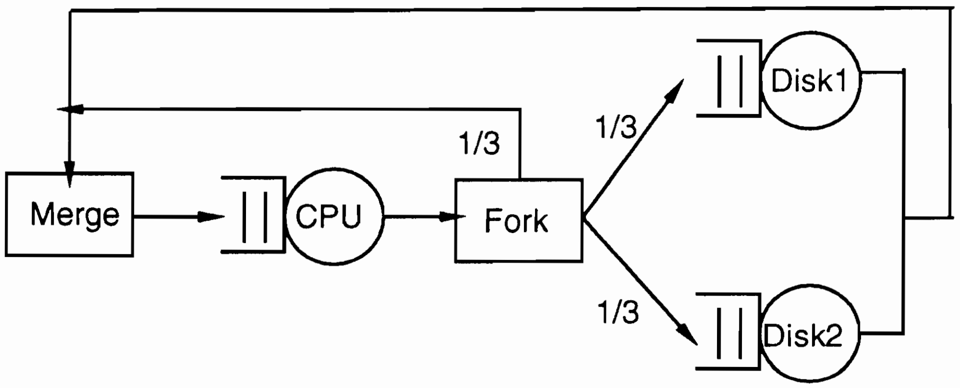


Figure 3.4: A Central Server System. The number associated with each arc represents the probability of the transition corresponding to the arc.

around the system. Each job, after being served by the CPU, is routed randomly to one of the two disks or back to the CPU for service based on a probability distribution.

3.5 Summary

This chapter describes the simulation models considered in this research. The models focus on computer systems, particularly communication networks, that can be modeled by queueing systems.

Chapter 4

MULTIPLE REPLICATION SIMULATION

This chapter discusses multiple replication simulation which is suitable for small models with a short initial transient period (Table 2.1 and Figure 2.2). One major difference between multiple replication simulation and other methods (i.e., time-parallel, parallel regenerative, and space-parallel) reviewed in the last chapter is that in multiple replication simulation, multiple short trajectories are computed, while in other methods, one long trajectory is computed.

Let $\{X(t), t \geq 0\}$ be a random process representing the target simulation model. Suppose $X(t) \rightarrow X$, where $t \rightarrow \infty$ and $\mu = E(X)$. Recall the discussion in Chapter 2 that in multiple replication simulation, each processor estimates μ by computing:

$$\bar{Y}_i = \frac{1}{T} \int_0^T X(t)dt,$$

where T is the simulation run length for each processor. Then an estimator for μ can be given by:

$$\hat{\mu}_1(P,T) = \frac{\sum_{i=1}^{P} \bar{Y}_i}{P}.$$

Heidelberger [37] showed that the expected initialization bias of $\hat{\mu}(T)$ can be generally assumed to be $|\mu - E(\hat{\mu}(\tau))| = \eta/\tau$ for some constant $\eta > 0$. Then based on the Strong Law of Large Numbers, we can see that the estimator $\hat{\mu}_1(P,T)$ converges to a wrong value (i.e., $E(\hat{\mu}(\tau))$ with probability one as the number of processors P increases.

To improve the accuracy, we may run the simulation for a long time to alleviate the initial transient bias. However, this imposes a large lower bound on the simulation time regardless of the number of processors available. In this chapter, we first review two bias-control techniques in sections 4.1 and 4.2. In section 4.3, we propose a multiple replication simulation algorithm that allows short simulation runs while maintaining high accuracy. Some experimental results of the proposed algorithm are given in section 4.4.

4.1 Multiple Replication Simulation with Deletion

A commonly suggested solution to control the initial transient bias is to discard the observations computed in an estimated initial transient (or *warm-up*) period $[0, t_w)$, where $t_w > 0$ is the estimated entry time point to the steady-state. That is, for each processor i, an estimate is computed by:

$$\bar{Y}_i(t_w) = \frac{1}{T - t_w} \int_{t_w}^{T} X(t)dt.$$

The multiple replication approach with such warm-up deletion will be referred to as the *MR-D approach* throughout our discussion.

For parallel simulation, the use of multiple processors is justified by a shorter simulation completion time or using fewer overall CPU cycles while achieving the same level of accuracy that would have been achieved by a sequential simulation. Therefore, for the MR-D approach, when the number of processors P increases, the simulation length of each processor must decrease accordingly. When the simulation length of each processor becomes too short, the MR-D approach will fail to remove the initial transient bias. Therefore, it is important that the multiple replication simulation starts from a state that is relatively 'close' to the steady-state condition in order to achieve a small initial transient. For queueing systems, Kelton [49] has shown that, an initial state that is as congested as the steady-state (in terms of the number of jobs that exist in the system) can induce comparatively short transients. However, due to a lack of *a priori* knowledge of the steady-state, an initial state is usually selected arbitrarily and thus significant bias may result.

4.2 Multiple Replication Simulation with Random Initialization

To deal with the initial transient problem, Kelton [49] proposed a random initialization approach, which assumes that the set of all possible initial states of the simulation model can be represented by a finite integer set $I_r = \{1, 2 \ldots, r\}$. The random initialization approach first conducts a short pilot simulation run and records the 'maximum' state $m \leq r$ that is observed during the pilot simulation run. Then the initial states of the actual production runs are selected randomly from $\{1, 2 \ldots, m\}$. This approach, however, still faces the problem of initializing the pilot run. Kelton has suggested deterministic initialization for the pilot run.

Kelton has shown that this random initialization approach usually results in a smaller bias than a deterministic initialization. However, this approach is not appropriate for massively parallel simulation. This is because when P is large, the pilot simulation run length needs to be short (in order to achieve a short completion time) and thus the resultant upper bound m will be relatively close to the initial state of the pilot simulation. Hence, the resultant maximum state will be inaccurate. In the next section, we present a stochastic initialization approach suitable for parallel simulation.

4.3 Multiple Replication Simulation with Polling Initialization

Let $\{X_s(t), t \geq 0\}$ represent the random process representing the target simulation model with an initial state $s \in I_r$ (i.e., $X_s(0) = s$). Let $\hat{\mu}(s,T) = \frac{1}{T}\int_0^T X_s(t)dt$ and $b(s,T) = |\hat{\mu}(s,T) - \mu|$ be the initialization bias for $\hat{\mu}(s,T)$. We assume that $E(b(s,T))$ decreases monotonically as T increases (alternatively $\hat{\mu}(s,T)$ converges monotonically toward μ). This assumption has been shown to hold for a wide range of models [48, 37].

Suppose there exists a state $s_{opt} \in I_r$ that is sufficiently representative of the steady-state condition such that $b(s_{opt},T) \leq b(s,T)$ for all $T > 0$ and $s \neq s_{opt}$. Because s_{opt} is generally unknown, our approach uses a pilot simulation that applies a polling initialization (PI) technique to determine a small interval $[a,b]$ (where $a \in I_r, b \in I_r$, and $a \leq b$) which is likely to contain s_{opt}. Then the initial states of the actual production runs are selected randomly from the range $[a,b]$. Assuming that P is a multiple of r (the cases where this assumption is not true will be discussed later), the PI pilot simulation is described by the following steps:

PI Pilot Simulation

1. For all $s \in I_r$, perform the following steps concurrently:

   (1) Start P/r independent runs concurrently, one processor for each run, to simulate the target system starting from state s for a length T_P. For each run, if the resultant estimate $\hat{\mu}(s, T_P) \geq \hat{\mu}(s, T'_P)$ for some pre-determined intermediate time point T'_P, $0 \leq T'_P < T_P$, we say state s 'wins' an *h-vote*, otherwise (i.e., $\hat{\mu}(s, T_P) < \hat{\mu}(s, T'_P)$) s wins an *l-vote.*

   (2) Calculate the total numbers of h-votes and l-votes won by state s. If the number of h-votes is greater than l-votes, label s with an 'h', otherwise label s with an 'l'.

2. Compute an interval $[a, b]$ based on the following rules:

   (1) If all states are labeled with $l's$, let $a = b = 1$.

   (2) If all states are labeled with $h's$, let $a = b = r$.

   (3) Otherwise, let $a = \max\{k, 1\}$ such that state $k + 1$ is the only state labeled l in $\{1, 2, \ldots k + 1\}$. Let $b = \min\{k, r\}$ such that state $k - 1$ is the only state labeled h in $\{k - 1, k, \ldots, r\}$.

3. For each processor $i, 1 \leq i \leq P$, randomly select an initial state from $[a, b]$.

In the actual production runs, the processors concurrently simulate the system for a fixed length $T > 0$ using the initial states obtained from the PI pilot runs and then compute an estimate for μ using estimator $\hat{\mu}_1(P, T)$. We refer to this approach as MR-PI.

In the PI pilot simulation, step 1 performs a *test* for each state s, in which P/r independent pilot runs are performed concurrently. For each run, if $\hat{\mu}(s, T_P) \geq \hat{\mu}(s, T'_P)$, it gives a strong indication that $\hat{\mu}(s, T)$ is converging toward μ from below. Hence, we 'guess' that

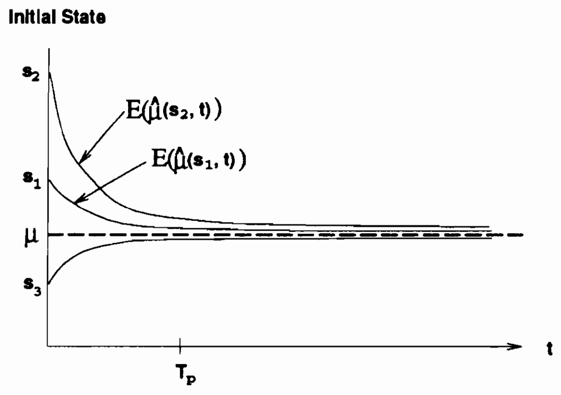


Figure 4.1: Examples of convergence curves.

$s \leq s_{opt}$ by issuing an h-vote. Similarly, if $\hat{\mu}(s, T_P) < \hat{\mu}(s, T'_P)$, we guess $s > s_{opt}$ by issuing an l-vote. The outcome of the test (i.e., $s_{opt} \geq s$ or $s_{opt} < s$) is determined by the majority of the votes (we break a tie by favoring l-vote). Based on these tests, step 2 obtains an interval $[a, b]$ that is likely to contain s_{opt}. The lower bound a is obtained in a way that a is the largest number satisfying that all states in $I_a = \{1, 2, \ldots, a\}$ are labeled with an h (so that we are 'confident' in predicting that $s_{opt} \geq a$). If $I_a = \emptyset$ (i.e., state 1 is labeled with l), we let $a = 1$. The upper bound b is determined in the same manner.

Let $p(s, T_P)$ denote the probability of each guess (as a result of a pilot run) being correct for state s. Apparently, $p(s, T_P)$ increases as T_P increases. Also, $p(s, T_P)$ is a function of the initial state s because the convergence rate typically depends on the initial state. This is illustrated in Figure 4.1, where the top curve (corresponding to s_2) shows a stronger convergence tendency in the initial transient period. Thus, we expect that $p(s_2, T_P) > p(s_1, T_P)$.

Let $p_h(s, T_P)$ and $p_l(s, T_P)$ represent the probabilities that state s is labeled h and l, respectively. Then

$$p_h(s,T_P) = \begin{cases} \sum_{i=\lfloor \frac{P}{2r} \rfloor + 1}^{P/r} \begin{pmatrix} \frac{P}{r} \\ i \end{pmatrix} (p(s,T_P))^i (1-p(s,T_P))^{\frac{P}{r}-i} & \text{if } 1 \leq s \leq s_{opt}, \\ \sum_{i=\lfloor \frac{P}{2r} \rfloor + 1}^{P/r} \begin{pmatrix} \frac{P}{r} \\ i \end{pmatrix} (1-p(s,T_P))^i (p(s,T_P))^{\frac{P}{r}-i} & \text{if } r \geq s > s_{opt}, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$p_l(s,T_P) = \begin{cases} \sum_{i=\lceil \frac{P}{2r} \rceil}^{P/r} \begin{pmatrix} \frac{P}{r} \\ i \end{pmatrix} (p(s,T_P))^i (1-p(s,T_P))^{\frac{P}{r}-i} & \text{if } r \geq s > s_{opt}, \\ \sum_{i=\lceil \frac{P}{2r} \rceil}^{P/r} \begin{pmatrix} \frac{P}{r} \\ i \end{pmatrix} (1-p(s,T_P))^i (p(s,T_P))^{\frac{P}{r}-i} & \text{if } 1 \leq s \leq s_{opt}, \\ 0 & \text{otherwise.} \end{cases}$$

Then the probability that the PI pilot simulation finds an interval containing s_{opt} is given by:

$$\begin{aligned} prob\{a \leq s_{opt} \leq b\} &= 1 - (prob\{a > s_{opt}\} + prob\{b < s_{opt}\}) \\ &= 1 - \left(\Pi_{s=1}^{s_{opt}+1} p_h(s,T_P) + \Pi_{s=s_{opt}-1}^{r} p_l(s,T_P) \right). \end{aligned}$$

Assume that $p(s,T_P) > 0.5$ for any s and T_P. If we assign the outcome of each vote a numerical value (e.g., 1 for h-vote, 0 for l-vote), then based on the Law of Large Numbers we can see that $prob\{a \leq s_{opt} \leq b\} \to 1$ as $P \to \infty$. That is, the region $[a,b]$ will almost always contain μ when $P \to \infty$. Also, the interval of $[a,b]$ is expected to decrease as P increases.

Let $w_{a,b} = b - a + 1$ denote the 'width' of the interval $[a,b]$. It can be derived that

$$E(w_{a,b}) = r - \sum_{s=1}^{r} (s-1)[p_l'(s+1,T_P)\Pi_{i=1}^{s} p_h(s,T_P)] - \sum_{s=1}^{r} (r-s)[p_h'(s-1,T_P)\Pi_{i=s}^{r} p_l(s,T_P)].$$

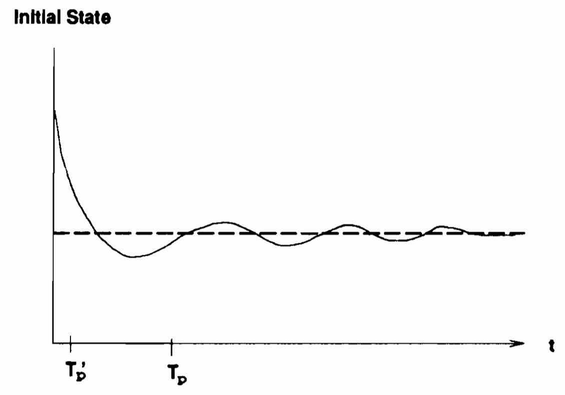


Figure 4.2: An example of a convergence curve.

where

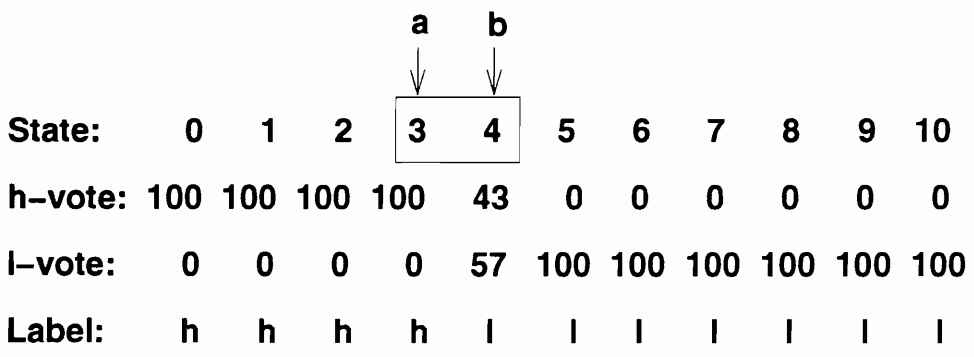
$$p_l'(s+1,T_P) = \begin{cases} p_l(s+1,T_P) & \text{if } s+1 \le r, \\ 1 & \text{otherwise,} \end{cases}$$

and

$$p_h'(s-1,T_P) = \begin{cases} p_h(s-1,T_P) & \text{if } s-1 \ge 1, \\ 1 & \text{otherwise.} \end{cases}$$

We have assumed that $\hat{\mu}(s,T)$ converges monotonically toward μ. It should be noted that this is not a necessary condition to apply the PI approach because the purpose of the PI pilot simulation is simply to obtain an initial state that is relatively close to μ. For example, for a convergence curve as shown in Figure 4.2 (where the convergence is not monotonic), the PI technique is viable when T_P' and T_P are properly chosen.

An application of PI pilot simulation is shown in Figure 4.3 for an M/M/1/10 queue. The state of the queue is represented by the queue length. Let λ and μ represent the average arrival rate and service rate, respectively of the queue. For $\lambda = 90, \mu = 100, P = 1000$, $T_p = 20$ and $T_p' = 0$, the PI pilot simulation obtains an interval [3,4] for the initial queue length (the analytical result for the average queue length is 3.97). Note that, for the cases when $P < r$, we may simply group I_r into P subsets and apply the PI pilot simulation by treating each aggregated state as a single state. Then an initial state may be randomly



				a	b						
State:	0	1	2	3	4	5	6	7	8	9	10
h-vote:	100	100	100	100	43	0	0	0	0	0	0
l-vote:	0	0	0	0	57	100	100	100	100	100	100
Label:	h	h	h	h	l	l	l	l	l	l	l

Figure 4.3: An example of the PI pilot simulation. The queue length process of an M/M/1/10 queue is considered, where $P = 1000$, $T_P = 20$, $T'_P = 0$, $\lambda = 90$, and $\mu = 100$. The pilot simulation obtains an interval [3,4] for s_{opt}.

chosen from the original states corresponding to the resultant aggregated state.

4.4 Experimental Results

This section empirically evaluates the performance of the MR-PI approach against the MR-D approach and for some queueing models.

4.4.1 Multiplexers

We first consider a loss G/G/1/K queue which is often used in modeling multiplexers (Figure 3.1) or other components in communication networks. The loss G/G/1/K model can be represented by a regenerative process $\{NAT(t), QL(t), ST_j(t), t \geq 0\}$, where NAT, QL, and $ST_j, 1 \leq j \leq K$ are the time interval before the next arrival time, queue length, and the remaining service time of the j^{th} job in the buffer ($ST_j = 0$ for $j > QL$), respectively. Supposed we are interested in the average queue length. For the MR-PI runs, the initial value for NAT is drawn from the interarrival time distribution, while the initial values of ST_i, $1 \leq j \leq QL$, are drawn independently from the service time distribution. For the MR-D runs, the queue are idle initially.

The M/M/1/100 queue and $\Gamma(2)$/D/1/100 queue are subjected to experimentation, where $\Gamma(2)$ denotes the gamma distribution with a shape parameter 2. Note that, the M/M/1/K model is considered because an analytical solution is readily available. In our experiments, we let $\mu = 100$ and $\lambda = 50, 70, 90, 100$ and 110, respectively to represent low to overloaded traffic conditions. Under each traffic condition, $P = 10^2, 10^3$, and 10^4, respectively. For the MR-D simulation, the length of each run is approximately $10^4/P$ simulation time units. A variety of warm-up period lengths ranged from 0% to 90% of the total simulation length are tested. For the MR-PI simulations, we arbitrarily let $T_P = 2000/P$ (i.e., 20% of the MR-D run length) and $T'_P = 0$. For a fair comparison, the MR-D and the MR-PI runs use the same overall simulation time units. That is, the lengths of the MR-PI production runs are approximately equal to $8000/P$ simulation time units.

Experimental results are given in Tables 4.1 and 4.2, where MR-D$_x$ represents MR-D simulation with the first x% of the simulation time interval deleted. We define the *statistical efficiency* of a simulation to be the ratio of the simulation error to the total simulation time spent. Then the normalized simulation errors shown in the square brackets in the tables indicate the statistical efficiencies of the simulation runs.

By comparing the MR-PI and the MR-D$_0$ runs (because both of them does not perform transient period deletion), it shows that the MR-PI approach almost always produces better results. In many cases, the MR-PI approach is more accurate than the MR-D approach for all deletion lengths (Note that, for the MR-PI simulation, warm-up deletion can also be applied to further reduce the bias). This is due to the PI pilot simulation which enables the MR-PI to start with a state that is more representative of the steady-state. The MR-D$_0$ runs are competitive in accuracy only when the traffic load is low (e.g., $\lambda/\mu = 0.5$). This is because $QL = 0$ is a 'good' initial state for a light traffic load. As P or λ/μ increases, the initial transient increasingly impairs the accuracy of the MR-D approach. When $P > 1000$

Table 4.1: The average queue lengths with approximate 90% confidence intervals of the M/M/1/100 model. Values in row 'EXPECT' are computed analytically. The values in the square brackets show the normalized errors against the expected values. Absence of a value in square brackets denotes an error no more than 1.0%.

Average Queue Length (M/M/1/100)

P	Approach	λ/μ =0.5	λ/μ =0.7	λ/μ =0.9	λ/μ =1.0	λ/μ =1.1
	EXPECT	1.0	2.333	8.997	50.0	90.007
	MR-PI	1.00(±.01)	2.34(±.03)	8.90(±.32)	49.49(±1.53) [1.0%]	90.12(±.31)
	MR-D$_0$	1.01(±.01)	2.35(±.03)	9.20(±.33) [2.2%]	47.13 (±1.67) [5.7%]	86.54(±.40) [3.8%]
100	MR-D$_{20}$	1.00(±.01)	2.34(±.03)	9.15(±.30) [1.7%]	51.85(±2.12) [3.7%]	90.01(±.33)
	MR-D$_{50}$	1.00(±.01)	2.32(±.03)	8.94(±.39)	50.30(±2.23)	90.06(±.38)
	MR-D$_{90}$	1.00(±.02)	2.31(±.07)	9.98(±1.1) [10.9%]	48.56(±3.91) [2.9%]	89.51(±.84)
	MR-PI	1.00(±.01)	2.33(±.03)	8.54(±.22) [5.1%]	46.73(±1.06) [6.5%]	90.64(±.25)
	MR-D$_0$	1.00(±.01)	2.31(±.02)	8.33(±.21) [7.4%]	38.16(±.84) [23.7%]	60.69(±.73) [32.6%]
1000	MR-D$_{20}$	1.00(±.01)	2.35(±.03)	8.56(±.23) [4.9%]	42.19(±.99) [15.6%]	68.44(±.82) [43.5%]
	MR-D$_{50}$	1.00(±.01)	2.34(±.03)	9.31(±.32) [3.4%]	46.31(±1.16) [7.4%]	81.53(±.78) [9.4%]
	MR-D$_{90}$	1.03(±.01) [3.0%]	2.37(±.06) [1.7%]	9.12(±.38) [1.3%]	48.08(±1.42) [3.6%]	87.04(±.74) [3.3%]
	MR-PI	.95(±.01) [4.0%]	2.43(±.02) [4.3%]	6.38(±.06) [29.1%]	55 .18(±.4) [10.4%]	93.96(±.06) [4.4%]
	MR-D$_0$	.96(±.01) [4.0%]	2.08(±.02) [10.7%]	5.62(±.06) [37.5%]	27.40(±.24) [45.2%]	14.14(±.12) [84.3%]
10000	MR-D$_{20}$	1.01(±.01)	2.27(±.02) [2.6%]	6.11(±.07) [32.1%]	30.37(±.28) [39.3%]	16.76(±.15) [81.4%]
	MR-D$_{50}$	.99(±.01)	2.30(±.03) [1.3%]	6.86±.09) [23.7%]	35.3 4(±.36) [28.9%]	20.10(±.24) [77.5%]
	MR-D$_{90}$	1.00(±.01)	2.46(±.04) [5.6%]	7.44(±.11) [17.3%]	38.96(±.43) [22.1%]	24.71(±.24) [72.5%]

Table 4.2: The average queue lengths of the $\Gamma(2)/D/1/K$ model with approximate 90% confidence intervals. Each entry in row 'EXPECT' is estimated by a simulation of 10 independent replications for a total of 10^8 arrivals. The values in the square brackets are the normalized errors against the 'EXPECT' values. Absence of a value in square brackets denotes an error less than 1.0%.

Average Queue Length ($\Gamma(2)/D/1/100$)

P	Approach	λ/μ =0.5	λ/μ =0.7	λ/μ =0.9	λ/μ =1.0	λ/μ =1.1
	EXPECT	.588	1.045	2.836	50.682(±.358)	97.28
	MR-PI	.589(±.001)	1.048(±.004)	2.84(±.04)	49.19(±2.52) [3.0%]	97.27(±.05)
	MR-D$_0$	.589(±.001)	1.047(±.004)	2.83(±.04)	45.41(±2.31) [10.4%]	92.53(±.20) [4.9%]
100	MR-D$_{20}$	.588(±.002)	1.048(±.004)	2.86(±.05)	48.60(±2.71) [4.1%]	97.32(±.05)
	MR-D$_{50}$	.590(±.002)	1.043(±.006)	2.81(±.05)	48.99(±3.3) [3.3%]	97.29(±.06)
	MR-D$_{90}$	.589(±.001)	1.041(±.013)	2.81(±.14)	48.34(±4.5) [4.6%]	97.13(±.16)
	MR-PI	.588(±.001)	1.03(±.003)	2.84(±.04)	49.53(±1.31) [2.3 %]	97.29(±.04)
	MR-D$_0$	.587(±.001)	1.046(±.004)	2.81(±.04)	29.12(±.80) [42.5%]	51.37(±.57) [47.2%]
1000	MR-D$_{20}$	.588(±.002)	1.046(±.004)	2.80(±.04)	34.40(±.97) [32.1%]	61.55(±.67) [36.7%]
	MR-D$_{50}$	.590(±.002)	1.044(±.006)	2.83(±.05)	37.76(±1.20) [25.5%]	75.61(±.73) [22.3%]
	MR-D$_{90}$	.591(±.005)	1.035(±.01) [1.7%]	2.84(±.10)	41.17(±1.36) [18.8%]	89.35(±.64) [8.1%]
	MR-PI	.582(±.001)	1.027(±.003) [1.7%]	2.45(±.02) [13.6%]	51.63 (±.41) [1.9%]	97.74(±.02)
	MR-D$_0$	.578(±.001) [1.7%]	1.018(±.003) [2.6%]	2.37 (±.02) [16.2%]	14.09(±.13) [72.2%]	8.66(±.06) [91.1%]
10000	MR-D$_{20}$	.589(±.002)	1.053(±.004) [2.6%]	2.54(±.02) [10.2%]	15.67(±.15) [69.0%]	9.86(±.07) [89.9%]
	MR-D$_{50}$	.588(±.002)	1.049(±.006)	2.68(±.03) [5.3%]	17.60(±.18) [65.3%]	12.01(±.09) [87.5%]
	MR-D$_{90}$	.590(±.004)	1.086(±.01) [3.9%]	2.76(±.04) [2.5%]	2 0.22(±.24) [60.1%]	17.15(±.11) [82.4%]

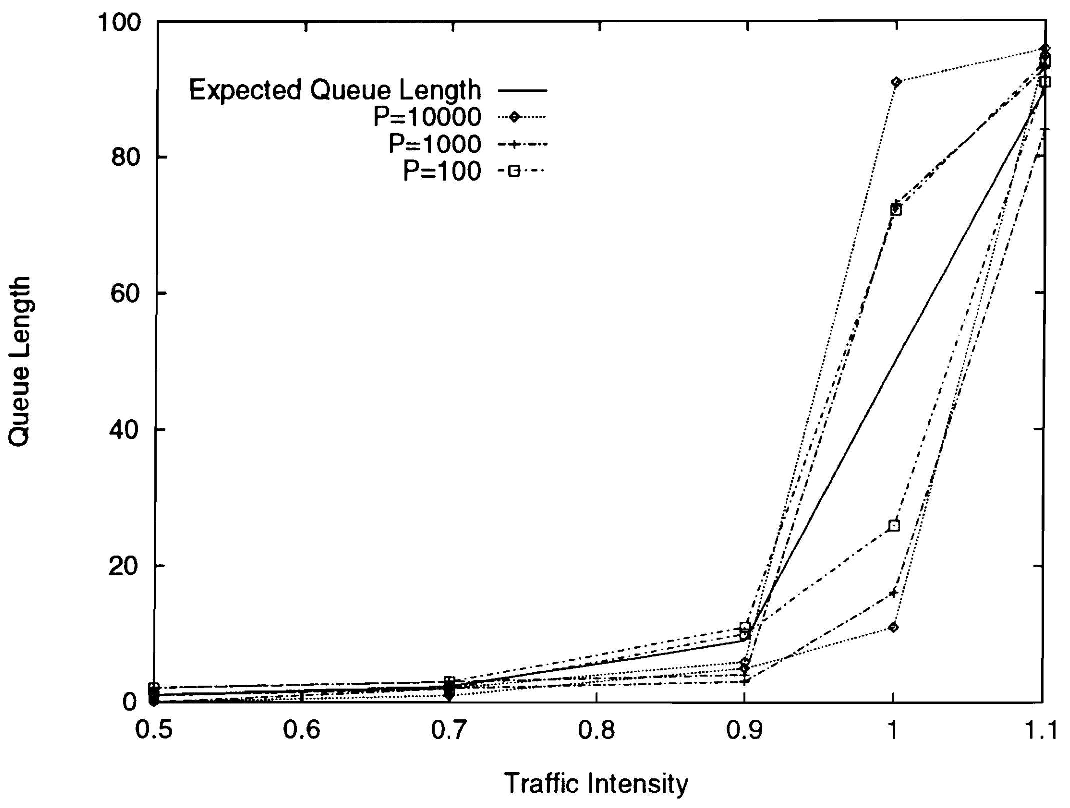


Figure 4.4: The estimated intervals $[a, b]$ for s_{opt} computed by the MR-PI approach for the M/M/1/100 queue.

and $\lambda/\mu > 0.7$, the errors are significant regardless of the deletion period length.

Our experiments also suggest that except when $\lambda/\mu = 1$ (when the steady-state queue length is unstable), the PI pilot simulation can very accurately locates the steady-state queue length even with a very short pilot run length. This is illustrated in Figure 4.4.

4.4.2 Virtual Circuit

Consider a virtual circuit (VC) of three hops as shown in Figure 4.5 where $N_W = 20$ and $\lambda_0 = \mu_1 = \mu_3 = 1.25\mu_2$. Assume that the average number of packets of, the bottleneck node (i.e., node 2) is of interest. Other parameters in our experiment are: $\lambda_0 = 1$, the desired simulation length $\tau = 10^5$ (i.e., the guideline stopping time is $10^5/P$), and $K = 30$

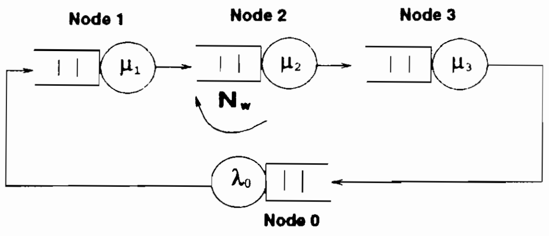


Figure 4.5: A virtual circuit with a sliding-window control, in which N_W is the window size.

Table 4.3: The results of the PI pilot simulation for the VC model.

	$P = 100$	$P = 1000$	$P = 10000$
$[a, b]$	[14,18]	[15,16]	[15,16]

for all nodes. Also, in our model, the packet lengths follow the $\Gamma(2)$ distribution and each packet is assigned a new length each time when it arrives at a queue.

For the MR-PI simulation, we let $T'_P = 0$ and $T_P = 10$ for all P's. Each run starts at a state where the packets are evenly distributed among all nodes. Table 4.3 shows the results of the pilot simulation. The MR-PI production runs and the MR-D runs use the same initial states as the PI pilot runs.

Experimental results of the VC model are shown in Table 4.4. Comparing the MR-PI and the MR-D approaches, it appears that the PI pilot simulation significantly reduces the initial transient bias. For the MR simulation, the errors are large for $P \geq 1000$.

4.5 Summary

This chapter investigates multiple replication simulation which is suitable for small models (Table 2.1 and Figure 2.2). This approach is appealing due to its simplicity. However, this approach is subject to bias caused by the initial transient. The impact of initial transient on the simulation results increases as the number of processors increases. An algorithm

Table 4.4: The estimated queue lengths of node 2 with approximate 90% confidence intervals. Each entry in column 'EXPECT' is estimated by a simulation of 10 independent replications, each of 10^6 simulation time units in length. The values in the square brackets are the normalized errors against the 'EXPECT' values. Absence of a value in square brackets denotes an error less than 1.0%.

Approach	$P = 100$	$P = 1000$	$P = 10000$
EXPECT	13.177(± .024)	13.177(± .024)	13.177(±.024)
MR-PI	13.26(± 0.15)	13.76(± 0.10)[3.7%]	14.58(± .03)[10.6%]
MR-D_0	12.85(± .14)[2.4%]	11.16(± .12)[15.3%]	7.86(± .04)[40.3%]
MR-D_{20}	13.29(± .16)	12.38(± .14)[6.0%]	8.51(± .05)[35.4%]
MR-D_{50}	13.20(± .23)	13.03(± .15)[1.1%]	9.33(±.06)[29.2%]
MR-D_{90}	13.25(± .35)	13.65(± .19)[3.5%]	11.30(± .08)[14.2%]

called MR-PI that uses a pilot simulation to find initial states that are representative of the steady-state conditions is proposed to reduce the initial transient. Empirical results (Tables 4.1, 4.1,4.2, and 4.4) suggest that for a class of models the proposed approach can produce very accurate results while allowing short simulation run lengths.

Chapter 5

TIME-PARALLEL SIMULATION

This chapter discusses time-parallel simulation. The value of time parallel simulation is particularly significant when the number of processors is large (i.e., massively parallel simulation), in which case other methods can not provide sufficient parallelism (Table 2.1 and Figure 2.2). However, time-parallel simulation has not been fully explored by the research community partly because time-domain parallelism is implicit and is often difficult to exploit. In this chapter, we first review some related work and then we consider a variety of simulation models for which we propose a set of time-parallel algorithms.

5.1 Related Work in Time-Parallel Simulation

This section reviews some related time-parallel simulation algorithms: (1) Greenberg, Lubachevsky, and Mitrani's (GLM) prefix algorithm [34], (2) Lin's network multiplexer algorithm [59], (3) Nikolaidis and Fujimoto's bursty arrival algorithm [67], and (4) Andradóttir and Ott's time segmentation algorithm [4]. The GLM algorithm is discussed with more detail because it forms the basis of our approximation algorithms.

5.1.1 The GLM Algorithm

Before discussing the GLM algorithm, we have to first review the *prefix problem*. Let $\Re$ be a domain and $\circ$ be any associative operator on $\Re$. Let N be any positive integer and let $x_1, x_2, \ldots, x_N$ be a sequence, where $x_j \in \Re$ for $1 \leq j \leq N$. The prefix problem [55] is to compute $s_j = x_0 \circ x_1 \circ \ldots \circ x_j$, for $1 \leq j \leq N$. Greenberg et al. [34] show that the task of simulating some queueing networks can be transformed into a prefix problem. The GLM algorithm is in fact a parallel algorithm that solves the prefix problem efficiently.

Now we review how the GLM algorithm is applied to a FCFS G/G/1/∞ queue. Let D_j^∞ denotes the departure time of job j from a G/G/1/∞ queue. If A_1 (i.e., the arrival time of the first job) is given, the arrival and departure time sequences $A_1, A_2, \ldots, A_N$ and $D_1^\infty, D_2^\infty, \ldots, D_N^\infty$ are the solution of the following recurrence relations [65]:

$$A_j = A_{j-1} + \alpha_{j-1} \quad 1 < j \le N, \tag{5.1}$$

$$D_j^\infty = \begin{cases} A_j + \delta_j & j = 1, \\ \max(D_{j-1}^\infty, A_j) + \delta_j & 1 < j \le N. \end{cases} \tag{5.2}$$

From (5.1), $A_j = A_1 + \alpha_1 + \ldots + \alpha_{j-1}$. Thus, solving $A_1, \ldots, A_N$ is a prefix problem. For the departure time sequence, equation (5.2) can be rewritten by a matrix recurrence relation [34] to which a prefix algorithm can be applied directly. In the rest of this paper, a sequence $x_a, x_{a+1}, \ldots, x_{b-1}, x_b$, where $a < b$, will be denoted by $\langle x_a, x_b \rangle$.

In a G/G/1/∞ model, an *event time* sequence $\langle E_1^\infty, E_{2N}^\infty \rangle$ is the result of merging two sequences $\langle A_1, A_N \rangle$ and $\langle D_1^\infty, D_N^\infty \rangle$ in ascending time order. *Event* i (for $1 \le i \le 2N$) is a job arrival or a departure corresponding to E_i^∞. When there are N arrivals, a total of $2N$ events will be simulated. Let t^- and t^+ represent the instances immediately *before* and *after* t, respectively, such that $t^+ - t = t - t^- \approx 0$ and no event occurs in the intervals $[t^-, t)$ and $(t, t^+]$. For example, $E_i^{\infty+}$ represents the instance immediately after the occurrence of event i and A_j^- represents the instance immediately before job j arrives. Then the G/G/1/∞ queue length at $E_i^{\infty+}$, denoted L_i^∞, is obtained by solving the following recurrence relation:

$$L_i^\infty = \begin{cases} L_{i-1}^\infty + 1 & \text{if event } i \text{ is an arrival,} \\ L_{i-1}^\infty - 1 & \text{if event } i \text{ is a departure,} \end{cases} \tag{5.3}$$

where L_0^∞ is defined to be the initial queue length of the G/G/1/∞ queue. Therefore, the computation of the queue length sequence is again a prefix problem.

Suppose that the number of processors, denoted P, divides N evenly, and there exists an identity element $\iota \in \Re$, such that $\iota \circ x_j = x_j$, for $1 \leq j \leq N$. To compute $s_j = x_0 \circ x_1 \circ \ldots \circ x_j$ for $1 \leq j \leq N$ in parallel, the GLM algorithm performs the following steps:

Algorithm GLM

input: $\langle x_1, x_N \rangle$.

onput: $\langle s_1, s_N \rangle$.

1. Partition the sequence $\langle x_1, x_N \rangle$ into P batches evenly such that batch k, $1 \leq k \leq P$ contains $\langle x_{((k-1)N/P)+1}, x_{kN/P} \rangle$.
2. Compute $f_k = x_{((k-1)N/P)+1} \circ \ldots \circ x_{kN/P}$ for all $1 \leq k \leq P$.
3. Compute
$$\beta_k = \begin{cases} \iota & k = 1, \\ f_1 \circ \ldots \circ f_{k-1} & 1 < k \leq P. \end{cases}$$
4. Let $q(j)$ and $r(j)$ be the quotient and the remainder of $\frac{j}{N/P}$, respectively. For $1 \leq j \leq N$, compute
$$s_j = \begin{cases} \beta_{q(j)+1} & r(j) = 0, q(j) \leq P, \\ \beta_{q(j)+1} \circ x_{q(j)(N/P)+1} \circ \ldots \circ x_j & r(j) \neq 0, q(j) < P. \end{cases}$$

The GLM algorithm can be directly applied to compute $\langle A_2, A_N \rangle$, and $\langle D_1, D_N \rangle$ (by solving (5.1) and (5.2)) in time $O(N/P + \log P)$. The GLM algorithm can be seen as a time-parallel algorithm because it partitions a sequence, for example $\langle A_2, A_N \rangle$, into P batches (i.e., $\langle A_1, A_{N/P} \rangle, \langle A_{N/P}, A_{2N/P} \rangle, \ldots \langle A_{(P-1)N/P}, A_N \rangle$) such that each batch is computed concurrently. Computing $\langle E_1^\infty, E_N^\infty \rangle$ requires merging two N-element sorted lists (i.e., $\langle A_1, A_N \rangle$ and $\langle D_1, D_N \rangle$). This can be done in time $O(N/P + \log N)$ using a parallel merging algorithm described in [3]. While computing $\langle E_1^\infty, E_N^\infty \rangle$, we associate each event time with its corresponding event type (i.e., arrival or departure). Then the GLM algorithm can again be applied to compute $\langle L_1^\infty, L_{2N}^\infty \rangle$ (by solving (5.3)) in time $O(N/P + \log P)$. In total, the GLM algorithm requires time $\Theta(N/P + \log P + \log N)$ to compute the job arrival time sequence, the job departure time sequence, the event time sequence, and the queue length sequence for a FCFS G/G/1/∞ queue [34].

5.1.2 The Network Multiplexer Algorithm

Lin [59] has developed an approximate trace-driven algorithm for simulating network multiplexers which are modeled by loss G/D/1/K queues. The algorithm first partitions the input trace consisting of N arrivals into P subtraces as evenly as possible. Then the algorithm computes a lower bound and an upper bound on the number of lost jobs by assuming, respectively, at each of the partition time points the queue length is zero (for the lower bound), and the remaining service time of the first job in the queue has a full service time quantum (for the upper bound). Overall, Lin's algorithm requires time $\Theta(N/P + P)$.

When $N \gg P$, Lin's algorithm produces accurate results and can achieve near-linear speedup. However, with a fixed trace length, the accuracy decreases as P increases because when fewer jobs are simulated by each processor, the initial state approximation will increasingly affect the accuracy of the simulation.

5.1.3 The Bursty Arrival Algorithm

Nikolaidis and Fujimoto [67] model a network multiplexer as a finite FCFS queue which receives packets from several sources. The packets generated by each source arrive at the queue in *bursts*. In each burst, the packet lengths and interarrival times are assumed to be constant so that the input trace can be *compressed* by recording the arrivals on a burst basis rather than tracking each individual packet. This characterization of the workload allows us to obtain a set of simulation times (through a pilot simulation) when the queue is guaranteed to be empty or full regardless of the initial queue length. The input trace is then partitioned into subtraces at these time points, hence permitting time-parallel simulation. Therefore, the amount of parallelism obtained by this algorithm relies on the workload of the system. In a circumstance where the multiplexer becomes idle or saturated very often, the amount of parallelism permitted by the bursty arrival could be significant, and hence good speedup can be achieved. However, the bursty arrival simulation assumes constant interarrival times and constant job service times within each traffic burst which, in turn, limits the applicability of this approach.

5.1.4 The Time Segmentation Algorithm

Recently, Andradóttir and Ott [4] have proposed a time-parallel algorithm for systems that can be modeled by a positive recurrent Markov Chain. Andradóttir and Ott show that for certain queueing systems, there exists a finite time beyond which the trajectory of the system is guaranteed to converge regardless of the initial state of the system. Therefore, it is possible to correctly 'predict' a state at time $t > 0$ without simulating the system for the entire time interval $[0, t)$, hence permitting temporal parallelism.

5.2 Time-Parallel Simulation for Loss Queueing Networks

Consider a single loss G/G/1/K queue. Let N be the number of arrivals. Also, let α_j denote the interarrival time between job j and job $j+1$ to the queue, and δ_j denote the service time of job j, where $1 \leq j \leq N$. Let A_j, D_j, and $L(t)$ denote the arrival time of job j, the departure time of job j, and the queue length at simulation time t, respectively, for all t in the simulation time interval $[0, \tau]$ (where $\tau > 0$). Job interarrival and service times are continuous random variables whose values can be pre-sampled. Given $\alpha_j > 0$ and $\delta_j > 0$, we wish to find A_j and D_j, for all $1 \leq j \leq N$, and $L(t)$, for all t in $[0, \tau]$, so that measures such as average queue length and job loss rate can be obtained. We assume that an arrival and departure do not occur simultaneously.

5.2.1 The GG1K Algorithm

This section presents the GG1K algorithm for loss FCFS G/G/1/K queues, where the job service times are independent and identically distributed random variables. Let $L(t)$

denote the loss FCFS G/G/1/K queue length at time t. For convenience, the departure time of a lost job is defined to be its arrival time because a lost job leaves the queue immediately upon arrival. The departure time of any job j is:

$$D_j = \begin{cases} \max(D_1, \ldots, D_{j-1}, A_i) + \delta_j & L(A_j^-) < K, \\ A_j & L(A_j^-) = K. \end{cases} \tag{5.4}$$

An open question is whether there exists an efficient exact parallel algorithm to solve (5.4). The GG1K algorithm presented next provides an approximate solution to (5.4).

The GG1K algorithm (stated in Section 5.2.1.2) consists of two phases. The first phase relaxes the finite buffer constraint and simulates a FCFS G/G/1/∞ queue using the GLM algorithm. If the queue length computed in the first phase never exceeds K, phase two is skipped because the trajectory generated by phase one is also an exact trajectory for the corresponding FCFS G/G/1/K queue. Otherwise, the second phase is performed, which takes the finite buffer storage into account and *transforms* the phase one trajectory into an approximate trajectory for the corresponding G/G/1/K queue.

The transformation technique is illustrated in Figure 5.1 and 5.2. This transformation is based on the following assumptions:

A1: Whenever an arrival occurs in the G/G/1/∞ queue, an arrival will also occur in the corresponding G/G/1/K queue.

A2: Whenever a departure occurs in the G/G/1/∞ queue, a departure will also occur in the corresponding G/G/1/K queue except when the G/G/1/K queue is empty. No departure will occur in the G/G/1/K queue at any other times.

For the models of interest (i.e., acyclic networks of G/G/1 queues), A1 is always true. However, when job losses occur in the G/G/1/K queue, A2 may not hold (Figure 5.2). Hence the GG1K algorithm is approximate. The next section shows that A2 enables us to approximately simulate a G/G/1/K queue efficiently with unbounded parallelism. Sections 5.2.1.3 and 5.2.4 show that the error arising from A2 is small in general.

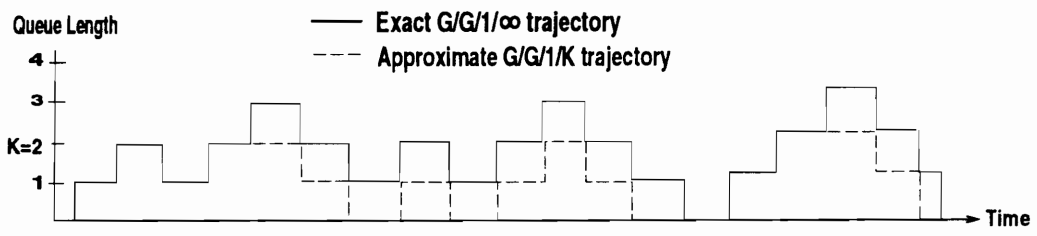


Figure 5.1: An example of computing an approximate trajectory based on the transformation technique.

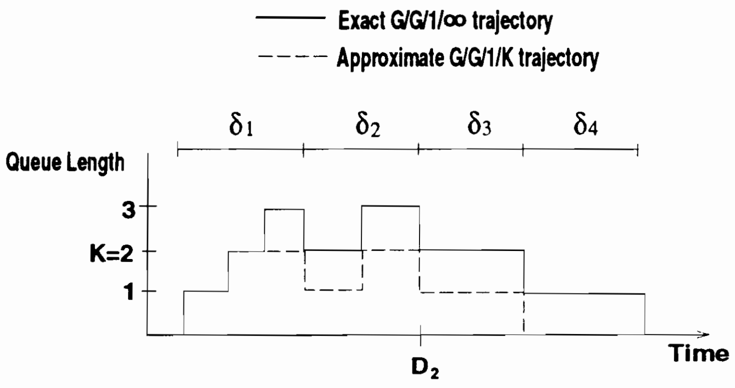


Figure 5.2: An example of computing an approximate trajectory based on the transformation technique. In the exact G/G/1/K queue, because job 3 is lost, job 4 should depart at $D_2 + \delta_4$. Thus, A2 holds if and only if $\delta_3 = \delta_4$ (e.g., when service times are constant)

.

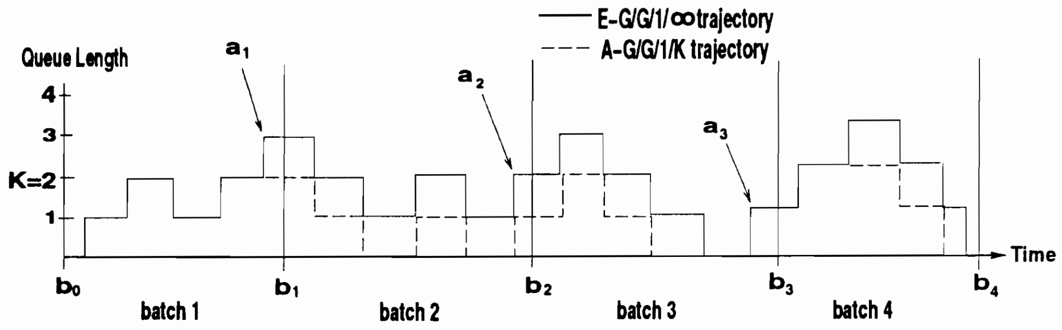


Figure 5.3: Each of the P=4 batches contains 5 events. The last event of each batch k is labeled by a_k, for $1 \leq k < 4$.

5.2.1.1 Determining the Initial Queue Lengths

Before presenting the algorithm, we consider a key problem: how to determine the initial queue length for each batch required by time-parallel simulation. This section presents an efficient *approximate* solution to this problem. The rest of this paper uses the FCFS queueing discipline. In addition, we use prefixes "A-" and "E-" to distinguish between an approximate simulation, which uses the GG1K algorithm (Section 5.2.1.2), and its corresponding exact simulation. For instance, "A-G/G/1/K trajectory" and "A-G/G/1/K queue length" represent the approximate trajectory and queue length, respectively, resulting from the GG1K algorithm, while their exact G/G/1/∞ counterparts will be referred to as "E-G/G/1/∞ trajectory" and "E-G/G/1/∞ queue length," respectively. Finally, *all approximate quantities are denoted by bold (and non-italics) font.*

Let $\mathbf{L}_i^K$ denote the A-G/G/1/K queue length at $E_i^{\infty+}$. Let $\Delta\mathbf{L}_i^K$ denote the difference between the E-G/G/1/∞ queue length (L_i^{∞}) and the A-G/G/1/K queue length ($\mathbf{L}_i^K$) at $E_i^{\infty+}$. Formally:

$$\Delta\mathbf{L}_i^K = L_i^{\infty} - \mathbf{L}_i^K. \tag{5.5}$$

Let $a_0 = 0$ and let event a_k be the last event of batch k, for $1 \leq k < P$, as illustrated in Figure 5.3. Formally, a_k satisfies $a_k \in \{1, 2, \ldots, 2N\}$; $a_1 < a_2, \ldots, < a_{P-1}$; and $E_{a_k}^{\infty} < b_k < E_{a_k+1}^{\infty}$. Our goal is to efficiently determine $\Delta\mathbf{L}_{a_k}^K$, for $1 \leq k < P$, as a function only of quantities known at the end of an exact G/G/1/∞ simulation using the GLM algorithm: $\langle L_0^\infty, L_{2N}^\infty \rangle$ and K. From $L_{a_k}^\infty$ and $\Delta\mathbf{L}_{a_k}^K$, equation (5.5) yields the desired quantity: $\mathbf{L}_{a_k}^K$ (the initial A-G/G/1/K queue length of batch $k+1$). Our approach computes $\Delta\mathbf{L}_{a_k}^K$ by deriving lower and upper bounds on $\Delta\mathbf{L}_{a_k}^K$. The bounds are computed using only K and the E-G/G/1/∞ queue lengths in batch k (i.e., $\langle L_{a_k-1}^\infty, L_{a_k}^\infty \rangle$).

Consider for a moment the problem of finding the initial queue length of batch $k+1$ (i.e., $\mathbf{L}_{a_k}^K$) from batch k. Consider any event i in batch k (i.e., $a_{k-1} < i \le a_k$). Let $\underline{\Delta\mathbf{L}}_{a_{k-1},i}^K$ (respectively, $\overline{\Delta\mathbf{L}}_{a_{k-1},i}^K$) denote a lower (upper) bound on $\Delta\mathbf{L}_i^K$ computed using only K and the E-G/G/1/∞ queue length in $\langle L_{a_{k-1}}^\infty, L_i^\infty \rangle$. Formally,

$$\underline{\Delta\mathbf{L}}_{a_{k-1},i}^K \le \Delta\mathbf{L}_i^K \le \overline{\Delta\mathbf{L}}_{a_{k-1},i}^K \quad \text{for all } a_{k-1} \le i \le a_k. \tag{5.6}$$

Then $\underline{\Delta\mathbf{L}}_{a_{k-1},i}^K$ and $\overline{\Delta\mathbf{L}}_{a_{k-1},i}^K$, for $i = a_k$, yields the desired quantity, $\Delta\mathbf{L}_{a_{k-1}}^K$. The subsequent four equations will express $\underline{\Delta\mathbf{L}}_{a_{k-1},i}^K$ and $\overline{\Delta\mathbf{L}}_{a_{k-1},i}^K$ as recurrences.

The first recurrence for the lower bound arises because immediately after event a_{k-1}, for any $1 \le k < P$, if the E-G/G/1/∞ queue has more than K jobs in the buffer, then the E-G/G/1/∞ queue length and the A-G/G/1/K queue length must differ by at least the amount by which the E-G/G/1/∞ queue length exceeds K. Otherwise, they differ by at least 0. Formally:

$$\underline{\Delta\mathbf{L}}_{a_{k-1},a_{k-1}}^K = \max\{0, L_{a_{k-1}}^\infty - K\}. \tag{5.7}$$

The first recurrence for the upper bound arises because immediately after event a_{k-1}, the queue length difference must be no more than the E-G/G/1/∞ queue length. Formally:

$$\overline{\Delta\mathbf{L}}_{a_{k-1},a_{k-1}}^K = L_{a_{k-1}}^\infty. \tag{5.8}$$

Now, consider any event i in batch k (i.e., $a_k < i \le a_{k+1}$). To compute a lower and an upper bound on $\Delta\mathbf{L}_i^K$, we consider two cases: (1) event i is an arrival, (2) event i is a departure. In case (1), after event i occurs, the queue length difference either remains the same or increases, and must be no less than the amount by which the E-G/G/1/∞

queue length exceeds K. In case (2), the queue length difference either remains the same or decreases, and must be no more than the E-G/G/1/∞ queue length. These observations are formally stated in (5.9) and (5.10). For $a_{k-1} < i \leq a_k$,

$$\underline{\Delta \mathbf{L}}^K_{a_{k-1},i} = \begin{cases} \max\{L^\infty_i - K, \underline{\Delta \mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i > L^\infty_{i-1} \\ \min\{L^\infty_i, \underline{\Delta \mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i < L^\infty_{i-1} \end{cases} \tag{5.9}$$

and

$$\overline{\Delta \mathbf{L}}^K_{a_{k-1},i} = \begin{cases} \max\{L^\infty_i - K, \overline{\Delta \mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i > L^\infty_{i-1} \\ \min\{L^\infty_i, \overline{\Delta \mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i < L^\infty_{i-1}. \end{cases} \tag{5.10}$$

The fact that $\underline{\Delta \mathbf{L}}^K_{a_{k-1},i}$ and $\overline{\Delta \mathbf{L}}^K_{a_{k-1},i}$ are indeed lower and upper bounds on $\Delta \mathbf{L}^K_i$ (i.e., (5.6) holds) can be proven from (5.7) to (5.10) by induction on i.

Lemma 1 states that the difference between the E-G/G/1/∞ and the A-G/G/1/K initial queue lengths for batch k (i.e., $\Delta \mathbf{L}^K_{a_k}$) can be one of the following three values: the lower bound $\underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}$, the difference between the E-G/G/1/∞ and the A-G/G/1/K initial queue lengths for batch $k-1$ (i.e., $\Delta \mathbf{L}^K_{a_{k-1}}$), or the upper bound $\overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}$. These three cases occur when, respectively, $\Delta \mathbf{L}^K_{a_k}$ is in the interval $(0, \underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k})$, $[\underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}, \overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}]$, or $(\overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}, +\infty)$ (Figure 5.4).

Lemma 1 *For all* $1 \leq k \leq P-1$,

$$\Delta \mathbf{L}^K_{a_k} = \max\{\min\{\Delta \mathbf{L}^K_{a_{k-1}}, \overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}\}, \underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}\}. \tag{5.11}$$

Proof: See Appendix A.1.

Equation (5.11) is a linear recurrence in the semiring on non-negative integers, where *max* is the addition operator with identity $-\infty$, and *min* is the multiplication operator with identity $+\infty$. That is, (5.11) can be expressed as:

$$\begin{bmatrix} \Delta \mathbf{L}^K_{a_k} \\ \infty \end{bmatrix} = \begin{bmatrix} \overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k} & \underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k} \\ 0 & \infty \end{bmatrix} \cdot \begin{bmatrix} \Delta \mathbf{L}^K_{a_{k-1}} \\ \infty \end{bmatrix}. \tag{5.12}$$

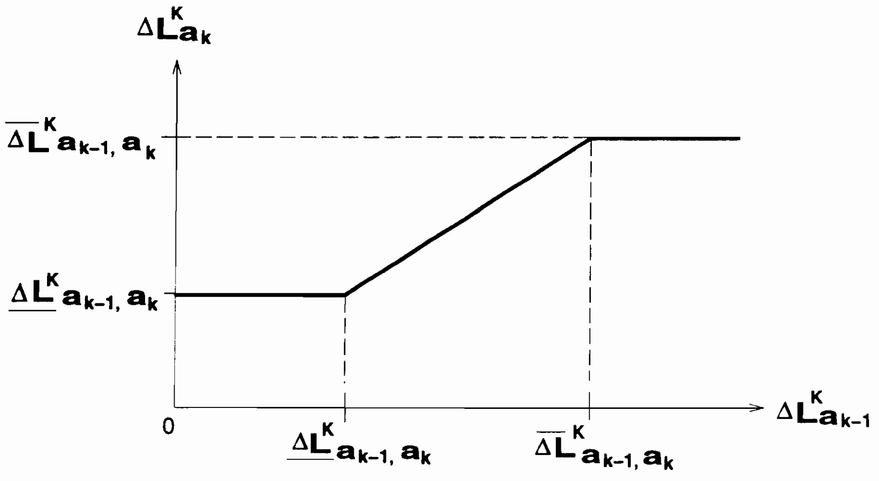


Figure 5.4: Relationship of $\Delta \mathrm{L}^K_{k-1}$ and $\Delta \mathrm{L}^K_{a_k}$, which are the initial queue length differences of the E-G/G/1/∞ and the A-G/G/1/K queues for batches k and $k+1$, respectively

.

Solving equation (5.12) is a parallel prefix problem. Using Stone's combining scheme [82, pp. 200-206] to compute $P-1$ values of $\Delta \mathrm{L}^K_{a_k}$ for all k requires $\Theta(\log P)$ of execution time. Note that we only compute $P-1$ initial queue lengths because the initial state of the first batch is the initial state of the system. We next describe how to use (5.5) to (5.12) to compute the initial A-G/G/1/K queue lengths.

Algorithm IQL (to compute initial A-G/G/1/K queue lengths):

input: $\langle L^\infty_0, L^\infty_{2N}\rangle$, K.

output: $\langle \mathrm{L}^K_{a_1}, \mathrm{L}^K_{a_{P-1}}\rangle$.

1. Compute $\underline{\Delta \mathrm{L}}^K_{a_{k-1},a_k}$ and $\overline{\Delta \mathrm{L}}^K_{a_{k-1},a_k}$ using (5.7) to (5.10), for all $1 \leq k < P-1$.
2. Compute $\Delta \mathrm{L}^K_{a_k}$, for all $1 \leq k < P$, using (5.12).
3. Compute $\mathrm{L}^K_{a_k}$ for all $1 \leq k < P$, (i.e., the initial A-G/G/1/K queue lengths of batches 2 to P) using (5.5).

Table 5.1
Computing $\underline{\Delta \mathbf{L}}^K_{a,i}$ and $\overline{\Delta \mathbf{L}}^K_{a,i}$ for the trajectory in Figure 3.

(a,i)	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	0	0	0	0	0	1
$\underline{\Delta \mathbf{L}}^K_{a,i}$	0	0	0	0	0	1
(a,i)	(5,5)	(5,6)	(5,7)	(5,8)	(5,9)	(5,10)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	3	2	1	1	1	1
$\underline{\Delta \mathbf{L}}^K_{a,i}$	1	1	1	1	1	1
(a,i)	(10,10)	(10,11)	(10,12)	(10,13)	(10,14)	(10,15)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	2	2	2	1	0	0
$\underline{\Delta \mathbf{L}}^K_{a,i}$	0	1	1	1	0	0

Assume that each batch contains approximately the same number of events. Then with P processors, step 1 requires time $\Theta(N/P)$. Step 2 requires $\Theta(\log P)$ as discussed above. Step 3 requires constant time. In total, the algorithm takes $\Theta(N/P + \log P)$.

We use the example in Figure 5.1 to illustrate the computation of the initial A-G/G/1/K queue lengths using the algorithm described above. Suppose $P = 4$. Thus, the trajectory is partitioned into four batches (Figure 5.3). Each batch contains 5 events. The last event of the first three batches are numbers 5, 10, and 15, respectively. Therefore, we are solving for $\Delta \mathbf{L}^K_5$, $\Delta \mathbf{L}^K_{10}$, and $\Delta \mathbf{L}^K_{15}$. Each of these is computed by a processor independently. We first (step 1 of algorithm IQL) compute in parallel three pairs of upper and lower bounds: $(\underline{\Delta \mathbf{L}}^K_{0,5}, \overline{\Delta \mathbf{L}}^K_{0,5})$, $(\underline{\Delta \mathbf{L}}^K_{5,10}, \overline{\Delta \mathbf{L}}^K_{5,10})$, and $(\underline{\Delta \mathbf{L}}^K_{10,15}, \overline{\Delta \mathbf{L}}^K_{10,15})$ by applying (5.7) to (5.10) as shown in table 5.1. Then (step 2 of IQL) we apply (5.12) to obtain $\Delta \mathbf{L}^K_5=1$, $\Delta \mathbf{L}^K_{10}=1$, and then $\Delta \mathbf{L}^K_{15}=0$, respectively. Note from Figure 5.3 that $\mathbf{L}^K_5$ and $\mathbf{L}^K_{10}$ differ by one from the exact values while $\mathbf{L}^K_{15}$ matches the exact value. Finally, (step 3 of IQL) using (5.5), we obtain $\mathbf{L}^K_5=2$, $\mathbf{L}^K_{10}=1$, and $\mathbf{L}^K_{15}=1$.

5.2.1.2 Algorithm GG1K

The complete GG1K algorithm is given below:

Algorithm GG1K

input: $\langle \alpha_1, \alpha_N \rangle$, $\langle \delta_1, \delta_N \rangle$, K, A_1, L_0^∞.

output: $\langle E_1^\infty, E_{2N}^\infty \rangle$, $\langle \mathbf{L}_1^K, \mathbf{L}_{2N}^K \rangle$.

Phase I: G/G/1/∞ Simulation

1. Use algorithm GLM to compute $\langle E_1^\infty, E_{2N}^\infty \rangle$, and $\langle L_0^\infty, L_{2N}^\infty \rangle$.

2. If there exists no i that satisfies $L_i^\infty > K$, for $1 \le i \le 2N$, skip phase II.

Phase II: Transformation

1. Partition $\langle L_0^\infty, L_{2N}^\infty \rangle$ evenly into P batches such that each batch contains about the same number of events.

2. Use algorithm IQL (Section 5.2.1.1) to compute $\mathbf{L}_{a_1}^K, \mathbf{L}_{a_2}^K, \ldots, \mathbf{L}_{a_{P-1}}^K$ (i.e., the A-G/G/1/K initial queue lengths for $P-1$ batches).

3. Each processor k, for $1 \le k \le P$, performs the following computation in parallel:

$$\mathbf{L}_i^K = \begin{cases} \mathbf{L}_{i-1}^K + 1 & \text{if } \mathbf{L}_{i-1}^K < K \text{ and event } i \text{ is an arrival,} \\ K & \text{if } \mathbf{L}_{i-1}^K = K \text{ and event } i \text{ is an arrival,} \\ \mathbf{L}_{i-1}^K - 1 & \text{if } \mathbf{L}_{i-1}^K > 0 \text{ and event } i \text{ is a departure,} \\ 0 & \text{if } \mathbf{L}_{i-1}^K = 0 \text{ and event } i \text{ is a departure.} \end{cases}$$

In the GG1K algorithm, phase I requires time $\Theta(N/P + \log N + \log P)$. In phase II, step 1 requires constant time and step 3 requires time $\Theta(N/P)$. Section 5.2.1.1 shows that step 2 requires time $\Theta(N/P + \log P)$. Therefore, in total, the GG1K algorithm requires time $\Theta(N/P + \log N + \log P)$, which is the same as the GLM algorithm. Thus, near-linear

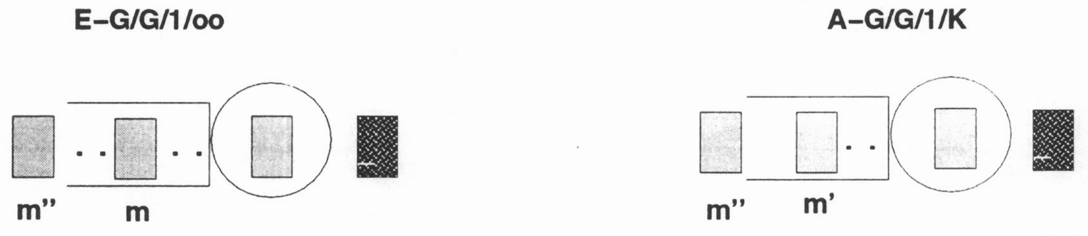


Figure 5.5: Case 1: When job m'' arrives, the E-G/G/1/∞ and the A-G/G/1/K queue lengths are both greater than zero.

speedup can be achieved when $N \gg P$. In addition, because the GG1K algorithm allows arbitrary batch partitioning (Section 5.2.1.1), "unbounded" parallelism (as defined in [34]) is achieved.

5.2.1.3 Analysis of the Approximation Technique

This section analyzes the approximation technique of the GG1K algorithm. Consider some simulation time t at which both the E-G/G/1/∞ simulation and the A-G/G/1/K simulation experience a departure. Let jobs m and m' be the departing jobs, respectively. Recall from A2 that for each departure in a A-G/G/1/K trajectory, there exists a departure in the corresponding E-G/G/1/∞ trajectory at the same simulation time. Let job m'', where $m'' \geq m' + 1$, be the first non-lost job that arrives after job m' arrives in the A-G/G/1/K simulation. Computing the departure time of job m'' can be described by the following four cases:

Case 1: When job m'' arrives, the E-G/G/1/∞ and the A-G/G/1/K queue lengths are both greater than zero (Figure 5.5).

In this case, the departure time of job m'' should occur at time $t + \delta_{m''}$ (i.e., $\delta_{m''}$ time units after the previous departure). However, the GG1K algorithm will assign $t + \delta_{m+1}$ as the departure time for job m''. If $m'' \neq m+1$, the algorithm effectively *reassigns* the service time of job $m+1$ to job m''. Assume that job service times are independent and identically

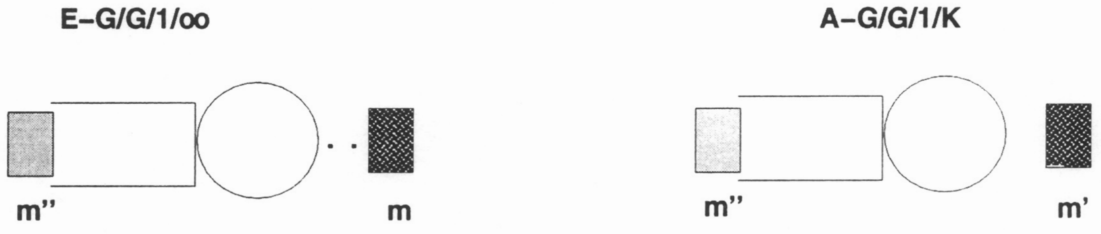


Figure 5.6: Case 2: When job m'' arrives, the E-G/G/1/∞ and the A-G/G/1/K queue lengths are both zero.

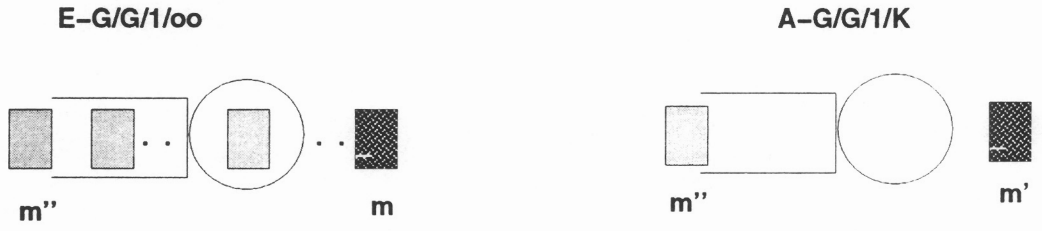


Figure 5.7: Case 3: When job m'' arrives, the E-G/G/1/∞ queue length is greater than zero while the A-G/G/1/K queue length is zero.

distributed (IID) random variables and are uncorrelated with job arrival times. Then such re-association of service times is unbiased since the asymptotic job service time distribution will not be changed. Asymptotically, no approximation error is introduced in this case.

Case 2: When job m'' arrives, the E-G/G/1/∞ and the A-G/G/1/K queue lengths are both zero (Figure 5.6).

In this case, the departure time of job m'' should occur at time $A_{m''} + \delta_{m''}$ (i.e., $\delta_{m''}$ time units after job m'' arrives). Since both systems are idle at $A_{m''}$, the GG1K algorithm will assign $A_{m''} + \delta_{m''}$ as the departure time for job m''. Therefore, no approximation error will be introduced.

Case 3: When job m'' arrives, the E-G/G/1/∞ queue length is greater than zero while the A-G/G/1/K queue length is zero (Figure 5.7).

In this case, the departure time of job m'' should occur at time $A_{m''}+\delta_{m''}$ (i.e., $\delta_{m''}$ time units after job m'' arrives). Let t' denote the simulation time of the first departure that occurs in the E-G/G/1/∞ queue after job m'' arrives. In this case, the GG1K algorithm will assign t' as the departure time for job m''. In other words, the GG1K algorithm effectively replaces the service time of job m'' with the remaining service time of the job that is in service in the E-G/G/1/∞ server when job m'' arrives. If job service times are constant, the approximation is biased since a service time will be under calculated each time this case occurs.

Case 4: When job m'' arrives, the A-G/G/1/K queue length is greater than zero while the E-G/G/1/∞ queue length is zero.

This case is impossible with the GG1K algorithm.

Therefore, only case 3 causes an approximation error. As a result, the occurrence frequency of case 3 governs the accuracy of the GG1K simulation. In the rest of this paper, case 3 is referred to as the *case-3 condition.* Also, $A_{m''}$ is referred to as the *occurrence time* of the case-3 condition, and job m'' is said to *cause* the case-3 condition.

5.2.1.4 Error Analysis

This section considers the approximation error of the GG1K algorithm in terms of the number of job losses for G/D/1/K queues. Let A_c and $A_{c'}$ be two simulation times at which the following conditions are satisfied:

1. $A_c < A_{c'}$.

2. Case-3 condition occurs (hence a job arrives at A_c and $A_{c'}$, respectively).

3. The E-G/G/1/K queue is empty immediately before the time (i.e., $L(A_c^-) = 0$ and $L(A_{c'}^-) = 0$).

4. There exists no other simulation time that satisfies conditions two and three.

Let θ and θ^K denote the exact and the approximate *loss count* (i.e., the number of lost jobs) computed by the GG1K algorithm, respectively, in the time interval $(A_c, A_{c'})$. Then for any pair of A_c and $A_{c'}$, the following lemma holds:

Lemma 2 *If* $\delta_0 = \delta_1 =, \ldots, \delta_N$, $\theta^K \leq \theta \leq \theta^K + 1$.
Proof: See Appendix A.2.

Let $\Delta\theta^K_{t,t'}$ denote the amount by which the E-G/G/1/K loss count exceeds the A-G/G/1/K loss count in the interval $[t, t']$. Also, let $C_{t,t'}$ denote the number of case-3 condition occurrences in $[t, t']$, and I_0 be the number of times that the E-G/G/1/K queue becomes idle in $[0, \tau]$. The following lemma gives an upper bound on the approximation error in terms of the total loss count for the entire simulation.

Lemma 3 $0 \leq \Delta\theta^K_{0,\tau} \leq \min\{I_0, C_{0,\tau}\}$.
Proof: Follows from Lemma 2.

Lemma 3 suggests that the GG1K algorithm produces less error when the value of λ/μ is either very large or small, where λ and μ are the average job arrival rate and service rate, respectively. This is because when λ/μ is large (e.g., 2) both the A-G/G/1/K queue and the E-G/G/1/K queue will rarely become idle. Thus, both I_0 and $C_{0,\tau}$ will be small. When λ/μ is small (e.g., 0.5), the G/G/1/∞ system becomes idle more frequently so that the probability of G/G/1/∞ queue length being greater than the A-G/G/1/K queue length (required by the case 3 condition) is low. Thus, $C_{0,\tau}$ will be small.

5.2.2 The IGG1K algorithm

In this section, we discuss a simple alternative for the GG1K algorithm. This approach is identical to the GG1K algorithm except that an iterative approach is used to determine the batch initial states. More specifically, in this approach, we use a 'guessed' initial state for each batch and then conduct a time-parallel simulation as described in step 3 of Phase II in the GG1K algorithm. After the computation of all batches is completed, if the new initial state of any batch differs from its previous initial state, the batch is recomputed based on the new initial state. This iterative process continues until all batch initial states converge.

Let $\mathbf{L}^K_{i,j}$ denote the GG1K queue length computed by the iterative approach at iteration j. Then algorithm (referred to as the IGG1K algorithm) can be obtained from the GG1K algorithm with a small modification as shown below:

Algorithm IGG1K

input: $\langle \alpha_1, \alpha_N \rangle$, $\langle \delta_1, \delta_N \rangle$, K, A_1, L^∞_0.

output: $\langle E^\infty_1, E^\infty_{2N} \rangle$, $\langle \mathbf{L}^K_1, \mathbf{L}^K_{2N} \rangle$.

Phase I: G/G/1/∞ Simulation

identical to the Phase I of the GG1K algorithm.

Phase II: Transformation

1. Partition $\langle L^\infty_0, L^\infty_{2N} \rangle$ evenly into P batches such that each batch contains about the same number of events.

2. j=0.

3. For $1 \le k < P$, compute $\mathbf{L}^K_{a_1,j}, \mathbf{L}^K_{a_2,j}, \ldots, \mathbf{L}^K_{a_{P-1},j}$ in parallel:

$$\mathbf{L}^K_{a_k,j} = \begin{cases} min(L^\infty_{a_k}, K) & j = 0, \\ \mathbf{L}^K_{a_k,j-1} & \text{otherwise.} \end{cases}$$

4. Each processor k, for $1 \leq k \leq P$, performs the following computation in parallel:

$$\mathbf{L}_{i,j}^{K} = \begin{cases} \mathbf{L}_{i-1,j}^{K} + 1 & \text{if } \mathbf{L}_{i-1,j}^{K} < K \text{ and event } i \text{ is an arrival,} \\ K & \text{if } \mathbf{L}_{i-1,j}^{K} = K \text{ and event } i \text{ is an arrival,} \\ \mathbf{L}_{i-1,j}^{K} - 1 & \text{if } \mathbf{L}_{i-1,j}^{K} > 0 \text{ and event } i \text{ is a departure,} \\ 0 & \text{if } \mathbf{L}_{i-1,j}^{K} = 0 \text{ and event } i \text{ is a departure.} \end{cases}$$

5. If there exists some k in $\{1, \ldots P\}$ such that $\mathbf{L}_{a_k,j}^{K} \neq \mathbf{L}_{a_k,j-1}^{K}$ then $j = j + 1$ and go to step 3. Otherwise, $j_{con} = j$ and exit.

In phase two, step 3 assigns each batch an initial queue length which is obtained from the final queue length of the preceding batch resulting from the previous iteration except for the first iteration at which a guessed initial queue length is used. The guessed initial queue length of each batch is obtained by computing the minimum of the corresponding E-G/G/1/∞ queue length and the buffer size since the queue length can not exceed the buffer size. Step 4 is the same as step 3 of the GG1K algorithm. Step 5 checks the convergence of the simulation on queue lengths and advances the simulation to the next iteration if the simulation is not converged.

Phase two requires $O(j_{con}(N/P + logP))$ to execute with P processors because step 1 to 3 take a constant of time; step 4 takes $O(N/P)$; step 5 requires $O(logP)$. Therefore, the total execution time of the IGG1K algorithm is $O[(N/P + logP + logN) + j_{con}(N/P + logP)]$ in which the first term is the time required by the GLM algorithm in phase one.

5.2.2.1 Convergence Analysis

The efficiency of the IGG1K algorithm relies on the convergence speed (i.e., the number of iterations required to reach convergence). In this section, we study the convergence speed of the IGG1K algorithm both experimentally and analytically.

An experiment of the algorithm described above is carried out for an M/M/1/K model. The results (Table 5.2) show that, in general, it takes only a few iterations to converge

Table 5.2: Numbers of iterations for an M/M/1/K model using the G/G/1/K substate matching algorithm. Each data point is an average of 10 runs. Each run simulates 10^5 jobs which are divided into 2^{10} batches. For each entry (a, b), a is the average iteration number and $(a - b, a + b)$ is the 90-percent confidence interval for a.

K	λ/μ	P=4	P=16	P=64	P=256	P=1024
	.1	1.2,0.2	1.7,0.3	2.0,0.0	2.0,0.0	2.0,0.0
	.3	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.5	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.7	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.9	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
1	.95	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.05	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	1.1,0.2	1.6,0.3	2.0,0.0	2.0,0.0	2.0,0.0
	.9	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.2,0.2
10	.95	1.9,0.2	2.0,0.0	2.0,0.0	2.0,0.0	2.3,0.3
	1.0	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.7,0.3
	1.05	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.6,0.3
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.6,0.3
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	2.1,0.2
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.9	1.0,0.0	1.4,0.3	2.0,0.0	2.6,0.3	7.7,1.1
	.95	1.6,0.3	2.0,0.0	2.1,0.2	4.7,0.5	15.8,2.1
50	1.0	2.0,0.0	2.0,0.0	2.5,0.3	6.3,0.4	21.1,1.5
	1.05	2.0,0.0	2.0,0.0	2.1,0.2	5.1,0.5	18.2,1.2
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	4.1,0.4	13.3,1.6
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	3.8,0.4
	.1	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.3	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.5	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.7	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0	0.0,0.0
	.9	0.1,0.2	0.1,0.2	0.2,0.4	0.3,0.5	1.0,1.8
	.95	1.1,0.3	1.3,0.4	2.1,0.5	4.8,1.6	17.0,6.1
100	1.0	2.0,0.0	2.2,0.2	4.9,0.6	17.3,1.7	66.6,6.6
	1.05	2.0,0.0	2.0,0.0	3.2,0.5	9.9,1.1	36.5,4.9
	1.1	2.0,0.0	2.0,0.0	2.0,0.0	4.9,0.5	16.5,1.5
	1.3	2.0,0.0	2.0,0.0	2.0,0.0	2.0,0.0	3.8,0.4

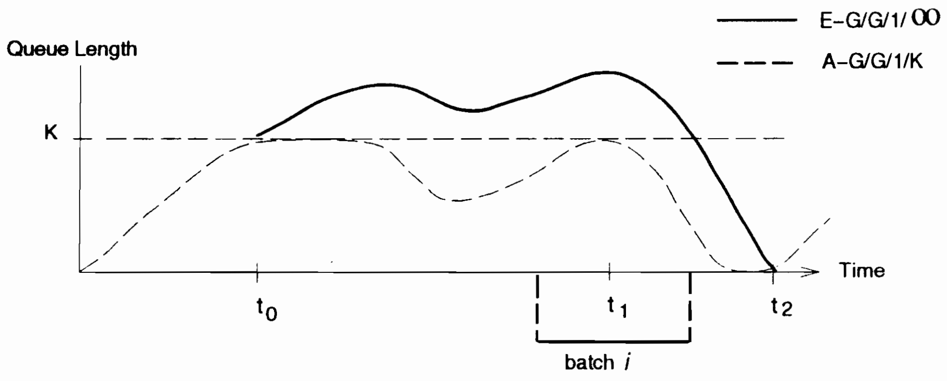


Figure 5.8: The first loss occurs at t_0. An E_F occurs at t_0 and t_1 and an E_E occurs at t_2. Sub-trajectories from t_0 to t_1 and from t_1 to t_2 are two propagation segments.

(mostly no more than 2 extra iterations). Many iterations are required only when both the ratio of λ to μ is close to 1 and the buffer size is large. We discuss this phenomenon next.

Recall that phase two of the IGG1K algorithm (and the GG1K algorithm as well) is a process of transforming an initial estimated trajectory into a more accurate trajectory. When there is no job being lost, phase-two computation is not necessary. Otherwise, phase-one trajectory after the first loss is modified using the GG1K transformation technique. Based on the GG1K transformation technique, we can see that the following property holds:

Property 1 *For all $1 \leq i \leq 2N$ and $1 \leq j < j_{con}$, $L_i^{\infty} \geq \mathbf{L}_{i,j}^K \geq \mathbf{L}_{i,j_{con}}^K$. That is, the queue length as a function of time of any iteration is a lower bound on the queue length of the previous iterations.*

Consider the example shown in Figure 5.8. At t_0 where the G/G/1/K queue is full and a job is lost. A change in queue length at t_0 has to be made and the change needs to be propagated along the queue length trajectory. At time t_1 the queue becomes full again. By property 1, the queue length at time t_1 will be changed to K in the first iteration in phase two regardless of the initial queue length of batch i which contains t_1. Therefore, no matter how many iterations it takes for a change to propagate from t_0 to t_1, it is guaranteed that the propagation *segment* between t_1 to t_2 would start with a correct initial queue length, namely K, at any iteration. Similarly, at t_2, where the G/G/1/∞ queues become idle the estimated queue length at any iteration in phase two will always be zero as well. Thus, the propagation starting from t_1 will not pass beyond t_2. We call time points such as t_1 and t_2 ***synchronization*** points. A ***propagation segment*** is a trajectory fragment enclosed by two neighboring synchronization points. An event which leads to a synchronization point is called a *synchronization event.* Let E_F denote a (job arrival) event immediately after whose occurrence the A-G/G/1/K queue becomes full, and let E_E denote a (job departure) event immediately after whose occurrence the E-G/G/1/∞ queue becomes empty. Then, E_F and E_E are synchronization events of the queue length trajectory.

It becomes evident that the number of iterations required for convergence is bounded by the number of batches spanned by the longest propagation segment. Therefore we have:

$$1 \leq j_{con} \leq \lceil \frac{max[d(E_F, E_E), d(E_F, E'_F)]}{2N/P} \rceil + 1$$

where (E_F, E_E) and (E_F, E'_F) are any pair of neighboring synchronization events, $2N/P$ is the batch length (assuming P divides $2N$ evenly), and $d(E_x, E_y)$ is the number of events between the occurrence of E_x and E_y.

Therefore, when some losses occur, if the traffic intensity is low or is very high, E_E and E_F tend to occur more frequently, respectively, and hence the maximum propagation length is shorter. As a result, the number of iterations required for convergence becomes small. When the traffic intensity is neither high or low, both synchronization events occur more sparsely and the longest propagation length increases. Thus, the number of iterations increases. Once the propagation length becomes longer than the batch length, adding more processors becomes useless because the number of iterations will grow linearly with the number of processors used. This situation can be seen when K is 50 and 100, and λ/μ is close to 1 in Table 5.2.

In Table 5.2, the worst M/M/1/K simulation performance always occurs when $\lambda/\mu = 1$, regardless of the number of processors and the buffer capacity. Actually, when $\lambda/\mu = 1$, it is least likely that the queue will become empty or full. This can be derived as follows.

Let $\rho = \lambda/\mu$. If P_0 is the probability that the M/M/1/∞ queue is idle (when an E_E occurs) then P_0 is given by:

$$P_0 = (1-\rho) \quad 0 \le \rho < 1.$$

If P_K is the probability that the M/M/1/K queue is full (when an E_F occurs), then P_K is given by (Kleinrock 1975):

$$P_K = \frac{\rho^K - \rho^{K+1}}{1-\rho^{K+1}} = \frac{\rho^K}{\sum_{i=0}^{K} \rho^i}.$$

Because when the M/M/1/∞ queue is empty, the corresponding M/M/1/K must also be empty, the joint probability of E_E and E_F occurrence can be given as:

$$P_s(\rho, K) = \begin{cases} P_0 + P_K & 0 \le \rho < 1, \\ P_K & \text{otherwise.} \end{cases}$$

Lemma 4 *$P_s(\rho, K)$ has a minimum at $\rho = 1$ for any $K \ge 1$.*

Proof: In the first step of our proof, we show that for $0 \le \rho < 1$, $P_s(\rho, K)$ is strictly decreasing. That is, for $0 \le \rho < 1$

$$\begin{aligned} \frac{\partial P_s}{\partial \rho} &= \frac{K\rho^{K-1} - (K+1)\rho^K}{1-\rho^{K+1}} + \frac{(K+1)\rho^K(\rho^K - \rho^{K+1})}{(1-\rho^{K+1})^2} - 1 \\ &= \frac{[K\rho^{K-1} - (K+1)\rho^K](1-\rho^{K+1}) + (K+1)\rho^K(\rho^K - \rho^{K+1}) - (1-\rho^{K+1})^2}{(1-\rho^{K+1})^2} < 0. \end{aligned}$$

Because $(1-\rho^{K+1})^2 \ge 0$, the above inequality holds if the numerator of the left-hand-side expression is less than zero. This is derived as follows:

$$\begin{aligned} &[K\rho^{K-1} - (K+1)\rho^K](1-\rho^{K+1}) + (K+1)\rho^K(\rho^K - \rho^{K+1}) - (1-\rho^{K+1})^2 \\ &= (K\rho^{K-1} - K\rho^K - \rho^K)(1-\rho^{K+1}) + (\rho^K + K\rho^K)(\rho^K - \rho^{K+1}) - (1 - 2\rho^{K+1} + \rho^{2K+2}) \\ &= -\rho^{2K+2} + \rho^{2K} + 2\rho^{K+1} - K\rho^K - \rho^K + K\rho^{K-1} - 1 \\ &= \rho^{2K}(1-\rho)(1+\rho) + K\rho^{K-1}(1-\rho) - \rho^K(1-\rho) - (1-\rho)(\textstyle\sum_{i=0}^{K} \rho^i) \\ &= (1-\rho)[\rho^{2K}(1+\rho) + K\rho^{K-1} - \rho^K - \textstyle\sum_{i=0}^{K} \rho^i] \\ &= (1-\rho)[(\rho^{2K+1} - \rho^K) + (\rho^{2K} - \rho^K) + (K\rho^{K-1} - \textstyle\sum_{i=0}^{K-1} \rho^i)] < 0. \end{aligned}$$

The second step of the proof shows that for $\rho \geq 1$, $P_s(\rho, K)$ is non-decreasing. When $\rho > 1$, $P_0 = 0$. Thus, $P_s(\rho, K) = P_K$. Otherwise, we take the first partial derivative of P_K with respect to ρ:

$$\frac{\partial P_K}{\partial \rho} = \frac{(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K)}{(1-\rho^{K+1})^2}.$$

Because $(1-\rho^{K+1})^2 > 0$ for $\rho > 1$, we need to show that:

$$(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K) > 0 \quad \rho \geq 1.$$

Dividing both sides of the above inequation by ρ^{K-1} and let $g(\rho, K)$ be the resulting left-hand-side expression, we have:

$$g(\rho, K) = (K - K\rho - \rho + \rho^{K+1}).$$

Because $g(1, K) = 0$ and $\frac{\partial g}{\partial \rho} = KX^K + X^K - K - 1 > 0$, clearly, $g(\rho, K) \geq 0$ for all $\rho > 1$ and $K \geq 1$. The lemma follows immediately from the results of steps 1 and 2. □

The IGG1K algorithm provides a simple alternative for the GG1K algorithm. However, the convergence speed (and hence the simulation execution time) of the IGG1K algorithm relies on the occurrence frequencies when the A-G/G/1/K queue becomes full and the E-G/G/1/∞ queue becomes empty.

5.2.3 The GD1K Algorithm

We have shown that the GG1K algorithm generates biased results for G/D/1/K queues. This section extends the GG1K algorithm and presents a GD1K algorithm for G/D/1/K queues that can produce highly accurate results. The GD1K algorithm improves the accuracy of the GG1K algorithm by accurately estimating the initial queue length for each batch.

Before deriving the GD1K algorithm, we first examine the differences between an A-G/D/1/K trajectory and its corresponding E-G/D/1/K trajectory. It can be observed that when no case-3 condition occurs during the simulation, these two trajectories are the same. Therefore, we consider only the situations where the case-3 condition occurs at least once. Suppose job k causes the first occurrence of case-3 condition. Then the GG1K algorithm will effectively *underestimate* the service time of job k by some amount $\delta'_k < \delta$, where δ is the job service time. Thus, in the A-G/D/1/K trajectory the departure time of job k should be *delayed* for $d_k = \delta'_k$ time units to compensate the error. This delay should also be ***propagated*** along the A-G/D/1/K trajectory for the subsequent departures by following the rules below:

R1: The propagation of a departure delay should stop at the point when the the corresponding E-G/G/1/∞ queue becomes empty.

R2: When a job k' $(k' > k)$ which causes another case-3 condition arrives, and of the shortfall of the service time $\delta'_{k'}$ is greater than the current delay d_k, then $\delta'_{k'}$ should be used as the delay for the subsequent departures until either R1 or R2 condition is met again.

For R1, because when the E-G/G/1/∞ queue becomes empty the A-G/D/1/K queue will become empty too. Therefore, the effect (which causes the delay propagation) of the case-3 condition disappears. To illustrate R2, we consider the example shown in Figure 5.9, in which a case-3 condition occurs at A_{k+2} and the shortfall of the service time (of job $k+2$) δ'_{k+2} is greater than the current propagation delay δ'_k. Therefore, δ'_{k+2} should be used as the new departure time delay for the subsequent jobs (including job $k+2$) until either R1 or R2 condition is met again. It is easy to show that the modified trajectory resulted from R1 and R2 is exact.

The system state of the G/D/1/K queue can be specified by $\{QL(t), FRST(t)\}$, where $QL(t)$ and $FRST(t)$ denote the length and the first job remaining service time of the queue at simulation time t, respectively. The GD1K algorithm is based on a key property of the GG1K algorithm: In any A-G/D/1/∞ trajectory, the queue length immediately after any

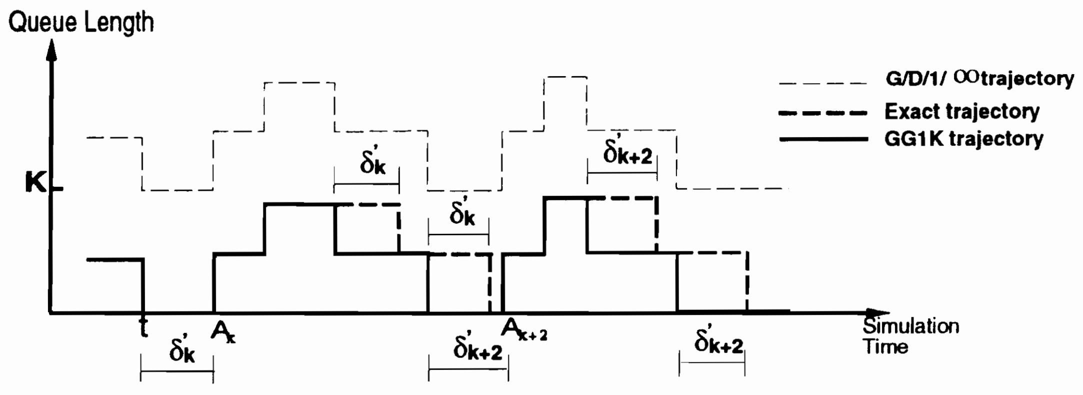


Figure 5.9: At A_k a case-3 condition occurs. The departure delay δ'_k propagates along the trajectory until another case-3 condition occurs at A_{k+2} causing a new departure delay δ'_{k+2}.

departure event is either the same or exactly one less than the exact queue length. This is formally stated in the following lemma:

Lemma 5 *For any A-G/D/1/K trajectory, if $E_j^{\infty+}(1 \leq j \leq 2N)$ corresponds to a departure event in the A-G/D/1/K trajectory, then $L(E_j^{\infty+}) - 1 \leq \mathbf{L}_j^K \leq L(E_j^{\infty+})$.*

Proof: See Appendix A.3.

Lemma 5 enables us to predict the queue length at any GG1K-computed departure time with a maximum error of one. Given lemma 5, the GD1K algorithm can be described as follows:

Algorithm GD1K

1. Compute an A-G/D/1/K trajectory using the GG1K algorithm. If no case-3 condition occurs in the resultant GD1K trajectory, the trajectory is exact; hence exit.

2. Partition the resultant A-G/D/1/K trajectory (approximately) evenly into P batches such that each pair of batches are partitioned at a time point immediately after an occurrence of a (GG1K-computed) departure.

3. For each batch i, $1 \leq i \leq P$, apply lemma 5 to compute the batch initial queue lengths $\{\ell_i - 1, \ell_i\}$, where the exact initial queue length is in $\{\ell_i - 1, \ell_i\}$.

4. For each processor $i, 1 \leq i \leq P$, sequentially compute batch i using the initial states $\{\ell_i, \delta\}$, and $\{\ell_i - 1, \delta\}$, respectively.

5. Construct a complete trajectory in the following manner: Let B_i and B_i' be the subtrajectories computed by processor i obtained from the previous step. For all $1 \leq i < P$, select a subtrajectory $B_i'' \in \{B_i, B_i'\}$ such that the final queue length of B_i'' is equal to the initial queue length of B_{i+1}''.

In step 4, we approximate the first job remaining service time with δ. Thus, the GD1K algorithm produces approximate results. Nevertheless, because each batch is re-computed using a sequential simulation, case-3 approximation error is eliminated. The total execution time required by the GD1K algorithm is $O(N/P + P + logP + logN)$, in which step 1, steps 2, 3, and 4 combined, and step 5 take time $O(N/P + logP + logN)$, $O(N/P)$, and $O(P)$, respectively. When $N \gg P$, near-linear speedup can be obtained.

5.2.4 Application and Experimental Results

5.2.4.1 Multiplexers

The GG1K simulations are compared with a sequential simulation for multiplexers modeled by loss G/G/1/K queues. Normalized approximation errors of the average queue lengths for three models are shown in table 5.3, in which $\Gamma(2)$ represents a gamma distribution with a shape parameter equal to 2.

The results show that the approximation errors are negligible except when the buffer size is smaller than 5 for $\Gamma(2)/\Gamma(2)/1/K$ and M/D/1/K models. The errors decrease as the value of K increases. Also, for all values of K, the approximation errors peak when the ratio of λ/μ is 1. This is because the case-3 condition occurs more frequently as λ/μ approaches 1. Consequently the error increases. Also, as indicated in Section 5.2.1.3,

Table 5.3

Normalized approximation errors on the average queue length using the GG1K algorithm. Each table entry is the average value of 100*(exact value - approximate value)/(exact value) obtained from 10 simulation runs of 10^6 jobs each.

	λ/μ	K=2	K=5	K=20	K=100
M/M/1/K	.1	0.03	0.0	0.0	0.0
	.5	-0.01	0.01	0.0	0.0
	.9	0.17	0.83	-0.35	0.0
	1.0	1.2	0.39	1.35	0.49
	1.3	0.17	0.38	0.08	0.16
$\Gamma(2)/\Gamma(2)/1/K$	.1	0	0.0	0.0	0.0
	.5	0.93	0.007	0.0	0.0
	.9	5.99	2.24	0.22	0.0
	1.0	7.68	3.84	0.99	0.002
	1.3	4.64	1.04	-0.005	0.005
M/D/1/K	.1	0.028	0.0	0.0	0.0
	.5	2.90	0.80	0.0	0.0
	.9	12.5	4.66	0.17	0.0
	1.0	15.32	8.0	2.22	0.36
	1.3	9.0	1.73	0.02	0.003

when the service times are constant, the approximation is biased. Thus, the results for the M/D/1/K model in table 5.3 are in general the worst. Because G/D/1/K is an important class of queueing models (e.g., useful in modeling ATM switches), we evaluate the GG1K algorithm by also comparing the errors on job loss rates and average job waiting times with sequential simulations for some G/D/1/K queues. Table 5.4 shows results for some M/D/1/K queues. The GG1K algorithm produces very accurate results except when the buffer size is very small (e.g., $K = 5$). Table 5.5 shows that the GD1K algorithm significantly improves the accuracy of the GG1K algorithm when K is small. However, for the GD1K algorithm the approximation error grows as P increases due to the increasing effect of the approximate initial $FRST$.

5.2.4.2 Datagram Networks

The GG1K algorithm can simulate acyclic feed-forward, finite buffer queueing networks that can be used to model datagram networks (see Chapter 3). In an acyclic feed-forward queueing network, jobs enter the system from *source* queues and depart the system from *destination* queues. Because the network contains no cycles, each job visits each queue at most once.

We present results of simulating three networks: a fork, merge, and tandem network as shown in Figure 3.3 using the GG1K algorithm. Each of these three models contains seven homogeneous M/M/1/K queues. We assume that a job departing from a fork queue has equal probability of joining any of the queues connected to the server output. Also, the service times of a job at each server are *independent*. That is, each job is assigned with a new service time when arriving at a server. For any of these three models, because there are no cycles, the GG1K algorithm can be applied queue by queue in a breadth-first order starting from any of the the source queues.

In our experiments, we let $\lambda = 100$ for all source queues and $\mu = 100$, $K = 10$ for all queues. Each model is simulated 10 times. Each run simulates 10^5 external arrivals. In the merge network model, queues 1 to 4 are independent source queues, each admitting

Table 5.4

A comparison of normalized approximation errors on job loss rate and average waiting time for M/D/1/K queues for $N = 10^5$. The numbers in column 'SEQ' are the estimates (with 90% confidence intervals) obtained from sequential simulations, and each number in columns 'GG1K' is the average value of 100*(approximate value - sequential estimate) /(sequential estimate) from 10 simulation runs.

	Loss Rate			Avg. Waiting Time	
	λ/μ	SEQ(90%c.i.)†	GG1K	SEQ(90%c.i.)†	GG1K
K=5	.3	$9.8(\pm 2)\times 10^{-3}$	.0	1.215	-.08
	.5	.22(±.02)	-.04	1.484	-.13
	.9	6.5(±.06)	-5.46	2.527	-4.94
	.95	8.3(±.07)	-7.27	2.685	-6.92
	1.0	10.4(±.07)	-9.35	2.841	-8.87
	2.0	50.0(±.06)	-.03	4.375	-.04
K=10	.3	0	0	1.215	.0
	.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
	.9	1.7(±.03)	-1.35	3.930	-10^{-3}
	.95	3.1(±.05)	-2.74	4.618	-2.70
	1.0	5.1(±.07)	-4.50	5.345	-4.62
	2.0	50.0(±.06)	.0	9.371	-.04
K=20	.3	0	0	1.215	.0
	.5	0	0	1.502	.0
	.9	.20(±.01)	-.14	5.189	-.13
	.95	.86(±.04)	-.72	7.415(±.1)	-.76
	1.0	2.6(±.07)	-2.18	10.379(±.1)	-2.22
	2.0	50.0(±.06)	.0	19.362	.04
K=100	.3	0	0	1.215	.0
	.5	0	0	1.502	.0
	.9	0	0	5.602(±.1)	.0
	.95	0	0	10.89(±.89)	.0
	1.0	.58(±.10)	-.27	51.40(±3.9)	-.34
	2.0	49.9(±.06)	.0	99.211	.0

† The 90% half confidence intervals of those not shown are negligible.

Table 5.5: Normalized approximation errors of the GD1K algorithms in job loss rate and in average waiting time for M/D/1/K queues for $N = 10^5$. The numbers in column 'SEQ' are the estimates (with 90% confidence intervals) obtained from a sequential simulation, and each number in column 'GD1K' is the average value of 100*(approximate value - sequential estimate) /(sequential estimate) from 10 simulation runs.

			Loss Rate		Avg. Waiting Time	
		λ/μ	SEQ(90%c.i.)†	GD1K	SEQ(90%c.i.)†	GD1K
		.3	$9.8(\pm 2)\times 10^{-3}$	.0	1.215	.0
		.5	.22(±.02)	-.04	1.484	.0
	K=5	.9	6.5(±.06)	.05	2.527	.0
		.95	8.3(±.07)	.07	2.685	.0
		1.0	10.4(±.07)	.13	2.841	.07
		2.0	50.0(±.06)	.02	4.375	.04
		.3	0	0	1.215	0
		.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
	K=10	.9	1.7(±.03)	.04	3.930	.0
		.95	3.1(±.05)	.09	4.618	.06
		1.0	5.1(±.07)	.34	5.345	.11
P= 100		2.0	50.0(±.06)	-10^{-4}	9.371	.0
		.3	0	0	1.215	0
		.5	0	0	1.502	0
	K=20	.9	.20(±.01)	.0	5.189	.0
		.95	.86(±.04)	0.10	7.415(±.1)	.02
		1.0	2.6(±.07)	.72	10.379(±.1)	.24
		2.0	50.0(±.06)	-10^{-4}	19.362	.04
		.3	0	0	1.215	0
		.5	0	0	1.502	0
	K=100	.9	0	0	5.602(±.1)	0
		.95	0	0	10.89(±.89)	0
		1.0	.58(±.10)	1.5	51.40(±3.9)	.39
		2.0	49.9(±.06)	-10^{-4}	99.211	.0
		.3	$9.8(\pm 2)\times 10^{-3}$	-1.02	1.215	-.16
		.5	.22(±.02)	-.22	1.484	-.13
	K=5	.9	6.5(±.06)	.39	2.527	.11
		.95	8.3(±.07)	.80	2.685	.29
		1.0	10.4(±.07)	1.36	2.841	.63
		2.0	50.0(±.06)	.22	4.375	.22
		.3	0	0	1.215	0
		.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
	K=10	.9	1.7(±.03)	.26	3.930	.07
		.95	3.1(±.05)	1.08	4.618	.38
		1.0	5.1(±.07)	3.04	5.345	1.10
P=1000		2.0	50.0(±.06)	-10^{-3}	9.371	.0
		.3	0	0	1.215	0
		.5	0	0	1.502	0
	K=20	.9	.20(±.01)	-.14	5.189	-.03
		.95	.86(±.04)	1.02	7.415(±.1)	.24
		1.0	2.6(±.07)	5.77	10.379(±.1)	1.81
		2.0	50.0(±.06)	-10^{-3}	19.362	.04
		.3	0	0	1.215	0
		.5	0	0	1.502	0
	K=100	.9	0	0	5.602(±.1)	0
		.95	0	0	10.89(±.89)	0
		1.0	.58(±.10)	4.19	51.40(±3.9)	.55
		2.0	49.9(±.06)	-10^{-3}	99.211	.0

† The 90% half confidence intervals of those not shown are negligible.

Table 5.6

A comparison of the queue lengths for the feed-forward queueing network models in which K=10 for all queues. Each number is the average of 10 runs.

	Fork		Merge		Tandem	
Queue	GG1K	Exact	GG1K	Exact	GG1K	Exact
1	5.00	5.01	5.02	5.01	4.97	4.99
2	0.82	0.82	5.01	5.02	4.05	4.09
3	0.82	0.81	4.95	5.04	3.59	3.58
4	0.29	0.29	5.05	5.00	3.30	3.30
5	0.29	0.29	8.79	8.79	3.08	3.09
6	0.29	0.29	8.79	8.78	2.93	2.92
7	0.29	0.29	8.96	8.97	2.79	2.77

25,000 arrivals for each simulation run. Table 5.6 compares the mean number of jobs in the system from the GG1K simulation to an exact simulation. The table shows that the GG1K simulations produce very accurate results for all three models. The approximation errors are mostly well under 1%, and never exceeds 1.8%.

5.3 Time-Parallel Simulation for Markov Chains

Markov chains are an important class of stochastic processes. Let N be the number of events to be simulated and let $t_i, 1 \leq i \leq N$, be the simulation time at which event i occurs. If the simulation model, represented by process $X = \{X(t), t \geq 0\}$, has a discrete state space and the probability distribution of $X(t_i), 1 < i \leq N$, depends only on $X(t_{i-1})$, then the simulation model is a Markov chain. This section describes two time-parallel approaches for Markov chains.

5.3.1 The Parallel Prefix Approach

It is well known that if $\mathbf{X}=\{X(t), t \geq 0\}$ is an irreducible, and positive recurrent Markov chain with a state space $S \in \{1, 2, \ldots\}$, then there exists a random variable X with a mass distribution $\pi = \{\pi_i, i \in S\}$, such that $X(t)$ converges to X in distribution as $t \to \infty$. That is, π is the stationary distribution of $\mathbf{X}$. Suppose that the purpose of the simulation is to study the long term behavior of the target system. Then we are interested in estimating $E(X)$.

Let A be the $m \times m$ transition probability matrix of the Markov chain. Assume that the Markov chain to be simulated is ergodic (i.e., each state of the Markov chain is positive recurrent and aperiodic). Then the stationary distribution of the Markov chain π can be determined by solving the following equations [6]:

$$\begin{array}{ll} \pi A = \pi & \text{and} \\ \sum_{k=1}^{m} \pi_k = 1. & \end{array} \tag{5.13}$$

For ergodic Markov chains, the limiting distribution is asymptotically equivalent to the stationary distribution and can be determined by the following equation:

$$\lim_{n \to \infty} A^n = \begin{bmatrix} \pi \\ \pi \\ \bullet \\ \bullet \\ \bullet \\ \pi \end{bmatrix}. \tag{5.14}$$

Here, (5.13) provides a set of linear equations. Let $A_{i,j}$ denote the j^{th} element of row i in A. Then for (5.14), a lower bound n', where n' is a positive integer, can be found such that for all $k > n', 1 \leq i,j \leq m$, $|A^k_{i,j} - A^\infty_{i,j}| \leq \epsilon$ for some small value $\epsilon > 0$ [6]. Computing $A^{n'}$ (and hence π) is a *prefix* problem. Therefore, the GLM algorithm (Section 5.1.1) can be applied to solve (5.14).

The parallel prefix approach described above is simple. However, when the state space of the target system is very large (or even infinite in many cases), using equations (5.13) and (5.14) to obtain the stationary distribution is computationally prohibitive. In this case, we present another time-parallel approach that is applicable to a more general semi-Markov chain.

5.3.2 The Recurrent State Approach

The memoryless property of a (continuous) Markov chain requires that the state holding times of the process have an exponential distribution. When this requirement is relaxed (i.e., to permit arbitrary state holding time distribution), the resultant random process is referred to as a semi-Markov chain.

Let $T_j(s), 1 \leq j \leq N+1$, denote the simulation time of the j^{th} occurrence of state $s \in S$ and let $r_m(s)$ denote $T_{m+1}(s) - T_m(s), 0 < m \leq N$, which is the simulation time between the m^{th} and the $(m+1)^{th}$ occurrence of state s. Assume that for each $s \in S$, $r_m(s)$ are identically and independently distributed random variables for all m. Then the *recurrence period* of state s is defined to be:

$$E(r_m(s)) = \frac{1}{\pi_s}. \tag{5.15}$$

State s is said to be *positive recurrent* when $E(r_m(s)) < \infty$. If there exists some $c > 1$ such that $E(r_m(s)) < \frac{N}{c}$, then temporal parallelism may be obtained through partitioning the simulation time interval into sub-intervals (or batches) at the time points where state s occurs. Hence, the batch initial and the final states (in this case, state s) are determined. The states used for partitioning the simulation time interval are called *matching states* (because they determine the matching condition between two batches). To obtain a high degree of parallelism, we are interested in finding a frequently occurring state (FOS). If ***a priori*** knowledge of such a state is unavailable, we propose to use a pilot simulation described below to obtain an estimated FOS:

FOS Pilot Simulation

1. For all i, $1 \leq i \leq P$, processor i randomly chooses an initial state and simulates the system for some short time period. Record the occurrences of each state.

2. For each processor i, identify the most frequently occurring state, denoted by $\mathbf{r}_i$, in the step 1 simulation run.

3. Select the state that appears the most times in $\{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_P\}$. Break a tie arbitrarily.

The state obtained in step 3 will be used as the matching state for the time-parallel simulation. This heuristic pilot simulation is not guaranteed to select the most frequently occurring state which has the minimum recurrence period. However, for our time-parallel simulation, it is beneficial as long as the selected state has a relatively high occurrence rate. Note that in the time-parallel simulation described above, although only one matching state is used, multiple matching states can also be used if necessary.

The recurrent state approach can be easily extended to non-Markovian models. Consider an arbitrary simulation model. Suppose we know that when the simulation model starts from an initial state s_1, it will reach states s_2 and then s_3, s_4 and so on such that the number of state transitions between each pair s_i and s_{i+1}, where $i \geq 1$, is relatively small. Assume that, for all i, the state transition probabilities after the occurrence of s_i are independent of any state that occurs before s_i. That is, the process at the instances when states $s_2, s_3, \ldots$ occur is an *imbedded Markov chain* [51]. Note that this assumption does not imply that the model is a Markov chain because the process between the occurrences of s_i and s_{i+1} can be non-Markovian. To achieve parallel simulation for such a model, we may simply partition the trajectory into batches by allowing processor i to independently simulate the model from states s_i and terminate when s_{i+1} occurs.

The approach described above requires a prediction of state occurrences that take place in the (simulation time) future. Unfortunately, such prediction is usually difficult (if not impossible) in practice. Consider a FCFS G/G/1/K queue whose state at time t is defined by: $\langle NAT(t), QL(t), RST_1(t), RST_2(t), \ldots, RST_K(t)\rangle$, where $NAT(t)$ denotes the simulation time between t and the next arrival time; $QL(t)$ denotes the queue length at t; $RST_i(t)$ denotes the remaining service time of the i^{th} job in the queue at t. Because the system has a continuous state space, unless the queue becomes empty periodically, a future-state prediction for the time-parallel approach described above is unlikely to be achieved. To deal with this problem, we present a *substate matching* technique which allows approximate simulation results in the next section.

5.4 Time-Parallel Simulation Using Substate Matching

The basic idea of substate matching (SSM) is that we relax the requirement that the final state of each time-partitioned batch has to exactly match the initial state of the following batch when the simulation terminates. Instead, only a subset of state variables are matched. To illustrate the SSM approach, we use the FCFS G/G/1/K model as an example. In this model, if we consider only the 'imbedded' process corresponding to the time points when a job arrives, the state of the imbedded process at a job arrival time t' can be described by $\langle QL(t'), RST_1(t'), RST_2(t'), \ldots, RST_K(t')\rangle$ (because $NAT(t') = 0$).

Assume that the behavior of the FCFS G/G/1/K queue is relatively insensitive to slight changes (or 'perturbation' [41]) in the values of $RST's$. Then we may conduct a time-parallel simulation for the FCFS G/G/1/K model with the following steps:

1. Processor i, for each $1 \leq i \leq P$, initializes the state variables $QL, RST_1, RST_2, \ldots,$ and RST_k appropriately, and then starts simulating the system independently. Let $q_i \in \{0, 1, \ldots, K\}$ denote the initial value of QL corresponding to processor i.

2. Processor i terminates when $QL = q_{i+1}$ and when all other termination conditions are satisfied.

In this approach, we allow batches to partially match on state variable QL (as opposed to the entire state vector).

Suppose we know that the queue length will return to k, for some $1 \leq k \leq K$, relatively frequently. Then in step one, we may simply let $q_1 = q_2 = \ldots q_P = k$. Let τ be the total simulation length. Then the amount of parallelism is determined by the number of times that the queue length returns to k in the time interval $[0, \tau]$. To find a frequently occurring queue length k, again, we may apply the FOS pilot simulation.

The SSM approach described above can enhance the applicability of time-parallel simulation. The tradeoff, however, is that it generates approximate simulation results. The selection of the matching state variable(s) (QL in the above example) has to be balanced between parallelism and approximation errors. Fewer state variables, in general (but not necessary), permits a higher degree of parallelism but is likely to cause more approximation errors.

5.4.1 Application

We have shown that the SSM approach can be applied to a G/G/1/K queue. This section applies the SSM approach to a central server system.

5.4.1.1 Central Server System

Consider the central server system shown in Figure 3.4 which consists of a CPU, two disks, and an arbitrary number of jobs which circulate around the system. Each job is associated with a *length*, which is a constant defined to be the product of the job's service time and the device's service rate. The length of each job follows some probability distribution and remains *unchanged* as it travels around the system. Thus, this model does not have a product form solution. The buffer sizes of the devices are infinite and the time required for the job to move between devices is ignored.

A queueing model with feedback causes both optimistic and conservative algorithms to perform poorly [56]. For this central server model, Wagner and Lazowska [87] have shown that for any conservative parallel algorithms, 3.67 is the upper bound of speedup. A number of researchers [25, 45, 77, 86] have used this model as a benchmark for their parallel simulation protocols and none of them could achieve a speedup better than 3.

To apply the SSM approach to the central server system, we let batches partially match on state variables $\{QL_1(t), QL_2(t), QL_3(t)\}$. In our experiments, we first let the system contain only 3 jobs. The job lengths are generated by an exponentially distributed random variable with a mean of 0.01. All three devices have the same job service rate. Since there are only 3 jobs, the service times are effectively 3 small numbers.

Let μ_i denote the job service rate of queue i, and $QL_i(n)$ denote the queue length of queue i immediately after the occurrence of event n, respectively. A simulation of **QL** $=\{QL_1(n), QL_2(n), QL_3(n), n > 0\}$ can be largely simplified by making the following assumptions:

Assumption 1: The departure processes of all queues are mutually independent such that the probability that the next job departure will occur at queue i, denoted p_i, is given by:

$$p_i = \begin{cases} 0 & QL_i(n) = 0, \\ \frac{\mu_i}{\sum_{j=1}^{M} \mu_j,\ QL_j(n)>0} & QL_i(n) > 0, \end{cases} \tag{5.16}$$

where M is the number of queues in the system.

Assumption 2: The state holding times are constant.

Violation of these assumptions does not prohibit use of the SSM approach, but reduces the accuracy of the FOS algorithm. With assumption 1, $QL(n)$ can be easily modeled by a Markov chain. With assumption 2, the computation of job arrival and departure times is not required. Also, the pilot simulation does not require an event list because the newly created event is always the next event to be executed. These assumptions will reduce the real execution time required for the pilot simulation.

Using the approximation technique described above, the queue length process **QL** can be modeled by a Markov chain of 10 states. Figure 5.10 shows the state transition matrix of the resulting Markov chain. Solution of equations (5.13) or (5.14) yields the stationary distribution: $\langle \frac{3}{10}, \frac{2}{10}, \frac{2}{10}, \frac{1}{10}, \frac{1}{15}, \frac{1}{15}, \frac{1}{45}, \frac{1}{45}, \frac{1}{90}, \frac{1}{90} \rangle$. State 0 (all three jobs are in the CPU's queue), which has the the smallest recurrence period, is hence chosen as the matching state.

To validate the accuracy of the occurrence frequencies obtained from the approximate Markov chain, we execute the original simulation model 10 times using a sequential simulation. Each run uses a different seed and simulates 10^5 arrival events. A comparison of these

$$A = \begin{array}{c|cccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1/2 & 1/6 & 0 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1/2 & 0 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1/3 & 1/3 & 1/9 & 0 & 0 & 1/9 & 1/9 & 0 & 0 \\ 4 & 0 & 1/2 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 & 0 \\ 5 & 0 & 0 & 1/2 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 \\ 6 & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

0 : <3,0,0>	**1** : <2,1,0>	**2** : <2,0,1>	**3** : <1,1,1>	**4** : <1,2,0>
5 : <1,0,2>	**6** : <0,2,1>	**7** : <0,1,2>	**8** : <0,3,0>	**9** : <0,0,3>

Figure 5.10: The State Transition Matrix of the Central Server System. Each state of the Markov chain corresponds to a state $\langle QL_1, QL_2, QL_3\rangle$, where QL_1, QL_2, and QL_3 are the queue lengths of the CPU, Disk1, and Disk2, respectively.

Table 5.7: A comparison of the stationary probability distributions obtained from solution of the approximate Markov chain and an exact simulation.

States	Approximate Markov Chain	Exact Simulation
0	30%	28.6%
1	20%	21.4%
2	20%	21.1%
3	10%	9.9%
4	6.7%	6.8%
5	6.7%	6.8%
6	2.2%	1.7%
7	2.2%	1.6%
8	1.1%	1.0%
9	1.1%	1.1%

state distributions is given in Table 5.7. It shows that solution of the approximate Markov chain generates a very close probability distribution to the results of an exact simulation.

In our time-parallel simulation, we let each device have a job in its queue initially. Because state 0 is chosen to be the matching state, it serves as the final state of each simulation sub-interval as well as the initial state of each simulation sub-interval except the first one. Therefore, a final state is considered to be reached when all three jobs are in the CPU's queue regardless of the remaining service times

In the experiment, we let $P = 10^2$ and $N = 10^5$. Therefore, for a perfectly balanced loading, each processor should simulates 10^3 events. Because the expected number of events in each sub-interval is only 3.33 (i.e. $\frac{1}{3/10}$), each processor is expected to simulate a large number of sub-intervals. Therefore, for each processor, the termination condition is defined as, "state 0 occurs and at least 10^3 arrival events have been simulated by the processor." Although this simple algorithm will cause more than N arrival events to be simulated, because state 0 occurs with a very high frequency, experiment results show that only an average of 2.64 more arrival events are simulated by each processor (Table 5.8).

The experiment executes the model 10 times using the SSM approach and a sequential

Table 5.8: A comparison of average queue lengths of the central server model. Each number is an average of 10 runs.

Algorithm	CPU	Disk1	Disk2	Arrivals Simulated
Parallel	2.160	0.422	0.425	100264
Sequential	2.188	0.406	0.406	100000

simulation, respectively. Table 5.8 compares the expected queue lengths. The differences for the CPU, Disk1, and Disk2 queues are 1.2%, 3.8%, and 4.2%, respectively. Because the load of each processor is nearly perfectly balanced, and the cost of computing the stationary distribution (using equation (5.13) or (5.14)) is constant, a linear speedup can be achieved.

To illustrate that the SSM simulation can achieve massive parallelism and still maintain high accuracy, we vary the value of P, ranging from 10 to 10^5, and compare the results with a sequential simulation. We also increase the number of jobs and the number of events simulated from 3 and 10^5 to 300 and 10^7, respectively. In this case, each queue contains 100 jobs initially. The service times of the jobs are generated by an exponentially distributed random variable with a mean of 0.01. All other parameters and assumptions of the system remain unchanged from the previous experiment.

The number of possible states of the imbedded queue length process of this model is about 4.5×10^4. Because of the large state space, we use the FOS pilot simulation to estimate a frequently occurring substate. Table 5.9 compare the expected queue lengths obtained from the two simulations. It shows that the SSM approach produces highly accurate results.

5.5 Summary

Time-parallel approaches can potentially exploit massive parallelism for a small simulation model. Several time-parallel approaches are proposed in this chapter. In Section 5.2, we present the GG1K, IGG1K and GD1K algorithms for acyclic feed-forward networks of loss FCFS G/G/1/K and G/D/1/K queues, respectively. The GG1K and GD1K algo-

Table 5.9: A comparison of the average queue lengths of the central server model. In the Table, SSM and SEQ represent the substate matching and the sequential simulations, respectively.

P	Simulation	CPU	Disk1	Disk2
	SEQ	299.05	0.473	0.473
10	SSM	299.06	0.471	0.471
100	SSM	299.06	0.467	0.468
1000	SSM	299.06	0.470	0.469
10000	SSM	299.06	0.470	0.469

rithms exploit unbounded parallelism and require time $\Theta(N/P + \log P + \log N)$ to simulate N arrivals using P processors. The IGG1K algorithm is a simple, iterative alternative of the GG1K algorithm.

The non-iterative GG1K algorithm consists of two phases. In the first phase, a similar system (i.e., a G/G/1/∞ queue) is simulated using the GLM algorithm. Then the resultant trajectory is transformed into an approximate G/G/1/K trajectory in the second phase. The closeness of the approximation is investigated analytically and experimentally. The experimental results show that generally the GG1K algorithm achieves highly accurate results for G/G/1/K queues where the service times are IID (i.e., G/GI/1/K). However, the approximation errors become more significant due to the case-3 condition when both K is small (e.g., 5) and the job service times are constant.

The IGG1K algorithm is identical to the GG1K algorithm except that the IGG1K algorithm reaches convergence iteratively. The convergence speed is analyzed. For M/M/1/K queues, we prove that the worst performance (in terms of the number of iterations) of the IGG1K algorithm occurs when $\lambda/\mu = 1$. Experimental results show that except when $\lambda/\mu \rightarrow 1$, the IGG1K algorithm converges very quickly.

We show that for constant service times, the initial queue length computed by the GG1K algorithm can differ from the exact value by at most one. Based on this property, we develop the GD1K algorithm, in which we use the GG1K algorithm to estimate accurate batch initial states. Then each processor performs a regular sequential simulation concurrently with others, so that the case-3 approximation bias can be removed.

In Section 5.3, two time-parallel approaches (i.e., parallel prefix and recurrent state) for Markov chains are presented. In the parallel prefix approach, we show that computing the stationary distribution of an ergodic Markov chain is in fact a prefix problem. Hence, a parallel prefix algorithm (such as the GLM algorithm) can be applied to obtain the stationary distribution of a Markov chain. For a more general semi-Markov chain, we describe a recurrent state approach which partitions the Markov chain trajectory into batches at the instances where a pre-determined state occurs. The amount of parallelism depends on the occurrence frequency of the pre-determined state. We present a heuristic pilot simulation algorithm that may help identify a frequently occurring state.

The SSM approach relaxes the batch matching requirement by allowing two consecutive batches to partially match on a subset of the state variables. Substate matching enhances the applicability of time-parallel simulation and can potentially exploit massive parallelism. However, the trade-off is that this technique leads to approximate simulation results. Nevertheless, the degree of approximation error may be assessed by evaluating the *gradient* (of the interested measure) due to a change in the initial values of the state variables that are not used for the convergence matching. Some gradient estimation techniques are given by Law and Kelton [54, Chapter 12].

Chapter 6
PARALLEL REGENERATIVE SIMULATION

This chapter discusses parallel regenerative simulation which utilizes the regenerative structure of the target system to exploit parallelism for steady-state simulation (Table 2.1 and Figure 2.2).

The time evolution of a simulation model can be viewed as a realization of a random process $\mathbf{X} = \{X(t), t \geq 0\}$. Assume that $X(t)$ converges in distribution to some random variable X as $t \to \infty$. Suppose that the purpose of simulation is to estimate the steady-state mean $\mu = E(X)$. Crane and Iglehart [20] showed that if the random process is regenerative, a simple procedure can be applied to obtain a strongly consistent point estimator and a confidence interval for μ.

The underlying regenerative structure of a random process provides potential parallelism because the trajectory of the process consists of a set of consecutive IID regeneration cycles, each of which can be independently computed by a processor. The amount of parallelism thus relies on the regeneration frequency. Unfortunately, for many systems of interest the expected regeneration cycle length is very long (or infinite). In this case, parallel regeneration simulation is not directly applicable.

In this chapter we investigate parallel regenerative simulation and propose an approximation approach for models that do not have a short expected regeneration cycle length. The rest of the chapter is organized as follows: Sections 6.1 and 6.2 review regenerative simulation using one and multiple processors, respectively. Section 6.3 addresses the implications of using different regeneration states. Section 6.4 presents an approximation parallel regeneration algorithm. The proposed algorithm is evaluated experimentally with some G/G/1/K queues and a virtual circuit system in Section 6.5.

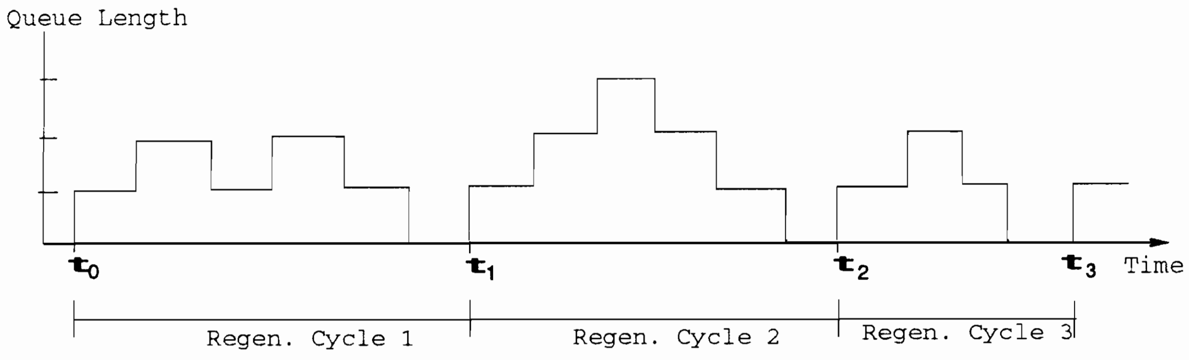


Figure 6.1: A GI/G/1 queue trajectory. The system regenerates (i.e., a job arrives at an empty queue) at t_0, t_1, t_2, and t_3.

6.1 Regenerative Simulation

A random process is said to be *regenerative* if there exists a state $\mathbf{r}$ such that the portions of the process defined in the time intervals $[\beta_1, \beta_2), [\beta_2, \beta_3), \ldots$ are IID replications, where β_n is the n^{th} time point when the system enters state $\mathbf{r}$. The state $\mathbf{r}$ is referred to as a *regeneration state.* The time interval between two consecutive *regeneration times* (i.e., the entrance time points to state $\mathbf{r}$) is called a *regeneration cycle.* For example, in a GI/G/1 queue, the system regenerates whenever a job arrives at an empty queue (Figure 6.1).

Define $\omega_n = \beta_{n+1} - \beta_n$, and

$$Y_n = \int_{\beta_n}^{\beta_n+1} X(t)dt.$$

That is, ω_n is the *length* of the n^{th} regeneration cycle and Y_n is the n^{th} observation computed by the processor assigned to the simulation time interval $[\beta_n, \beta_{n+1})$. Then $\{(Y_n, \omega_n), n \geq 1\}$ are IID random vectors and $\mu = E(Y_n)/E(\omega_n)$ [20]. Let $\aleph$ be the total number of regeneration cycles simulated. Then it can be shown that the following estimator for μ, denoted $\hat{\mu}_2(\aleph)$, is *strongly consistent* for μ as $\aleph$ increases (i.e., $\hat{\mu}_2(\aleph) \to \mu$ with probability one as $\aleph \to \infty$) [20]:

$$\hat{\mu}_2(\aleph) = \frac{\sum_{i=1}^{\aleph} Y_i}{\sum_{i=1}^{\aleph} \omega_i}.$$

To construct a confidence interval for $\hat{\mu}_2(\aleph)$, two well-known methods are the *classical* approach [20] and the *jackknife* approach [63]. We briefly describe the classical approach below.

Let $Z_i = Y_i - \mu\omega_i$. Then Z_i's are IID random variables with mean 0. Let σ_Z^2 denote the variance of Z_i. By the central limit theorem,

$$\frac{\bar{Z}(\aleph)}{\sqrt{\sigma_Z^2/\aleph}} \Rightarrow \mathbf{N(0,1)} \text{ as } \aleph \to \infty, \tag{6.1}$$

where '$\Rightarrow$' denotes convergence in distribution. The sample variance of Z_i, denoted $\hat{\sigma}_Z^2(\aleph)$, is given by:

$$\hat{\sigma}_Z^2(\aleph) = \hat{\sigma}_Y^2(\aleph) + (\hat{\mu}_2(\aleph))^2\hat{\sigma}_\omega^2(\aleph) - 2\hat{\mu}_2(\aleph)\hat{Cov}(Y_i, \omega_i),$$

where $\hat{\sigma}_Y^2(\aleph) = \sum_{i=1}^{\aleph}(Y_i - \bar{Y}(\aleph))^2/(\aleph-1)$, $\hat{\sigma}_\omega^2(\aleph) = \sum_{i=1}^{\aleph}(\omega_i - \bar{\omega}(\aleph))^2/(\aleph-1)$, and $\hat{Cov}(Y_i,\omega_i)$ $= \sum_{i=1}^{\aleph}(Y_i - \bar{Y}(\aleph))(\omega_i - \bar{\omega}(\aleph))/(\aleph - 1)$ are respectively, the sample variance of Y_i, the sample variance of ω_i, and the sample covariance of Y_i and ω_i. It can be shown that $\hat{\sigma}_Z^2(\aleph) \to \sigma_Z^2$ as $\aleph \to \infty$. Because

$$\frac{\bar{Z}(\aleph)}{\bar{\omega}(\aleph)} = \frac{\bar{Y}(\aleph)}{\bar{\omega}(\aleph)} - \mu = \hat{\mu}_2(\aleph) - \mu,$$

from (6.1), we have:

$$\frac{\hat{\mu}_2(\aleph) - \mu}{\sqrt{\hat{\sigma}_Z^2(\aleph)/\aleph(\bar{\omega}(\aleph))^2}} \Rightarrow \mathbf{N(0,1)} \text{ as } \aleph \to \infty, \tag{6.2}$$

Then, an approximate 100(1-α) percent confidence interval for μ is given by:

$$\hat{\mu}(\aleph) \pm \frac{z_{1-\alpha/2}\sqrt{\hat{\sigma}_Z^2(\aleph)/\aleph}}{\bar{\omega}(\aleph)}.$$

6.2 Parallel Regenerative Simulation

A key property for a regenerative system is that at a regeneration time point the system becomes totally independent of any previous states. This property provides potential parallelism for simulation because each processor can compute one or more regeneration cycles independently. This section reviews an estimator for μ based on a parallel regenerative approach proposed by Glynn and Heidelberger.

To illustrate parallel regenerative simulation, consider the GI/G/1 trajectory illustrated in Figure 6.1. The trajectory in the time interval $[t_0, t_3)$ can be partitioned into three subtrajectories at simulation time points t_0, t_1 and t_2, respectively. Because these subtrajectories are stochastically identical, each subtrajectory can then be computed by a processor independently. Therefore, parallel regenerative simulation can be treated as a special case of *time-parallel* simulation (e.g., [13, 34, 88, 89, 90]) in which parallelism is obtained through partitioning the time domain of the simulation model.

Let τ_r be the *real* (i.e., wall clock) simulation execution time for a simulation time length τ. Assume that the real execution time is linear to the simulation time length. That is, there exists a constant $\eta > 0$, such that $\tau_r = \eta\tau$. Suppose τ is the *desired* simulation time length. For balancing the workload it is desirable that each processor simulates τ/P time units (or equivalently τ_r/P real time units). Unfortunately, because a regenerative simulation requires that an integer number of regeneration cycles be simulated, and because the regeneration cycle length is a random variable, the simulation run length of each processor is also a random variable. Therefore, the perfect workload partitioning (i.e., all processors execute exactly τ/P simulation time units) can not be achieved.

In parallel regenerative simulation, a key problem is how to stop the processors. There exists a number of 'stopping rules', each resulting in a different estimator for μ. An example of a stopping rule is the *static computation assignment* [7], in which each processor completes a fixed number of regeneration cycles and then stops. Estimators based on a variety of stopping rules have been studied by Glynn and Heidelberger [31, 32, 38]. They showed that some intuitive stopping rules could cause severely biased results. Glynn and Heidelberger [32, 38] have proposed a strongly consistent estimator [32, 38] that uses τ/P as a *guideline* (because processor execution time is a random variable) stopping time. This estimator forms the basis of our PR approach which will be discussed later. For consistency with our notations, this estimator is denoted by $\hat{\mu}_3(P, \frac{\tau}{P})$.

For $\hat{\mu}_3(P, \frac{\tau}{P})$, all processors start simultaneously to simulate the system from a fixed regeneration state. For each processor, when the *system enters the regeneration state and at least τ/P time units have been elapsed*, the processor stops. Let $N_i(\tau/P)$ denote the number of regeneration cycles completed on processor i in τ/P time units. Then processor i will simulate exactly $N_i(\tau/P)+1$ regeneration cycles. Let Y_{ij} and ω_{ij} denote the j^{th} observation and the j^{th} regeneration cycle length (RCL), respectively, on processor i. Then $\hat{\mu}_3(P, \frac{\tau}{P})$ is obtained by:

$$\hat{\mu}_3(P, \frac{\tau}{P}) = \frac{\sum_{i=1}^{P}\sum_{j=1}^{N_i(\tau/P)+1} Y_{ij}}{\sum_{i=1}^{P}\sum_{j=1}^{N_i(\tau/P)+1} \omega_{ij}}. \tag{6.3}$$

It can be shown that $\hat{\mu}_3(P, \tau/P)$ is strongly consistent for μ as P increases [38]. Because $\sum_{j=1}^{N_i(\tau/P)+1}(Y_{ij} - \mu\omega_{ij})$, for $1 \leq i \leq P$, are IID random numbers, based on the central limit theorem, as $P \to \infty$,

$$\frac{E(\omega_{ij})E(N_i(\tau/P)+1)(\hat{\mu}_3(P, \frac{\tau}{P}) - \mu)}{\sigma(\tau/P)/\sqrt{P}} \Rightarrow \mathbf{N(0,1)}, \tag{6.4}$$

where $\sigma^2(\tau/P)$ is the variance of $\sum_{j=1}^{N_i(\tau/P)+1}(Y_{ij} - \mu\omega_{ij})$. Therefore, an approximate $100(1-\alpha)$ percent confidence interval can be obtained by:

$$\hat{\mu}_3(P, \frac{\tau}{P}) \pm z_{1-\alpha/2}\frac{\hat{\sigma}(\tau/P)}{\sum_{i=1}^{P}\sum_{j=1}^{N_i(\tau/P)+1} \omega_{ij}/\sqrt{P}},$$

where $\hat{\sigma}^2(\tau/P)$ is the sample variance of $\sum_{j=1}^{N_i(\tau/P)+1}(Y_{ij} - \mu\omega_{ij})$. Let $T(P)$ denote the completion time of the regenerative simulation using P processors. Also, let γ_i denote the residual time at τ/P for processor i. That is,

$$\gamma_i = \left(\sum_{j=1}^{N_i(\tau/P)+1} \omega_{ij}\right) - \frac{\tau}{P}.$$

Then

$$T(P) = \max_{1\leq i\leq P}\{\sum_{j=1}^{N_i(\tau/P)+1} \omega_{ij}\} = \tau/P + \max_{1\leq i\leq P}\{\gamma_i\}.$$

It can be shown that [22, Section 4.2]

$$\tau/P + \mu_\gamma(\tau/P) \leq E(T(P)) \leq \tau/P + \mu_\gamma(\tau/P) + \left(\frac{\sigma_\gamma(\tau/P)(P-1)}{\sqrt{2P-1}}\right), \qquad (6.5)$$

where $\mu_\gamma(\tau/P)$ and $\sigma_\gamma(\tau/P)$ are the mean and the variance of γ_i, respectively.

6.3 The Selection of Regeneration State

For a regenerative process, there usually exists multiple regeneration states. For example, all states in a homogeneous Markov chain are regenerative. Therefore, to apply the parallel regenerative simulation, one concern is how to choose a regeneration state. In this section, we first argue that it is advantageous to use regeneration states that have high occurrence frequencies. Then we present a pilot simulation that selects regeneration states which occur frequently.

For the parallel regenerative approach, we define the degree of parallelism to be the expected number of regeneration cycles in the simulation time interval $[0, \tau]$. That is, the degree of parallelism denotes the expected number of processors that we can use so that each processor can finish at least one regeneration cycle for a total simulation length τ. Apparently, to obtain a high degree of parallelism, it is desirable that we use a regeneration state that occurs relatively frequently.

There are two other potential advantages using a frequently occurring regeneration state (FORS). Consider $\hat{\mu}_3(P, \frac{\tau}{P})$, when regeneration occurs frequently, after the guideline stopping time has passed, a processor may quickly reach a regeneration point and terminates. Therefore, a short simulation completion time results.

Consider a process which has $r > 1$ regeneration states. Let $\gamma_{i,k}$ denote the residual time at τ/P for processor i when using regeneration state k, $1 \leq k \leq r$. Assume that the RCL corresponding to each regeneration state k, denoted Ω_k, is exponentially distributed with a mean $1/\lambda_k$. Due to the memoryless property of an exponential distribution, $\gamma_{i,k}$ and Ω_k have the same distribution. Therefore, for regeneration state k, the simulation completion time, denoted $T_k(P)$, is the P^{th} order statistic of the exponential distribution with mean

$1/\lambda_k$. It can then be shown that

$$E(T_k(P)) = P \int_{t=0}^{\infty} te^{-\lambda_k t}(1 - e^{-\lambda_k t})^{P-1} dt = \frac{1}{\lambda_k} \sum_{i=1}^{P} \frac{1}{i}.$$

Therefore, in this case, the expected simulation completion time strictly decreases as we use a regeneration state that has a smaller expected RCL. This relationship is also illustrated in Figures 6.2 and 6.3 by the queue length process of a loss M/M/1/10 queue where $\lambda/\mu = 0.7$. Note that the RCL of this process is not exponentially distributed.

Let $\overline{T_k(P)}$ and $\underline{T_k(P)}$, respectively, denote the upper and the lower bound on $T_k(P)$ given by (6.5). Then as a direct result of (6.5), we can see that if $E(\Omega_k) < E(\Omega'_k) \Longrightarrow \sigma_{\gamma_k} < \sigma_{\gamma_{k'}}$, where '$\Longrightarrow$' represents 'imply' and σ_{γ_k} denotes the variance of $\gamma_{i,k}$, then the following condition holds:

$$E(\Omega_k) < E(\Omega'_k) \Longrightarrow (\underline{T_k(P)} < \underline{T'_k(P)}) \wedge (\overline{T_k(P)} < \overline{T'_k(P)}).$$

The third potential advantage of using a FORS is that a FORS results in a large number of observations. This, generally, leads to a smaller confidence interval.

We have enumerated some potential advantages of using a FORS for the parallel regenerative simulation. When *a priori* knowledge of a FORS is unavailable, we may estimate one by using the FOS pilot simulation discussed in Chapter 5 (Section 5.3). The length of the pilot simulation can be determined somewhat arbitrarily because the purpose of the pilot simulation is simply to obtain a regeneration state that occurs relatively frequently. For the M/M/1/10 queue where $\lambda/\mu = 0.7$, our experiments show that the FOS pilot simulation almost always successfully selects the true most frequently occurring state even when each processor simulates as few as 20 job arrivals when $P \geq 100$.

The regenerative structure of a random process provides statistical convenience for constructing an estimate for μ, and allows us to partition the simulation easily for multiple processor execution. However, the regenerative approach requires that the expected RCL be relatively short so that sufficient IID observations can be obtained in the desired simulation interval. In practice, many systems of interest (e.g., a congested network) do not have

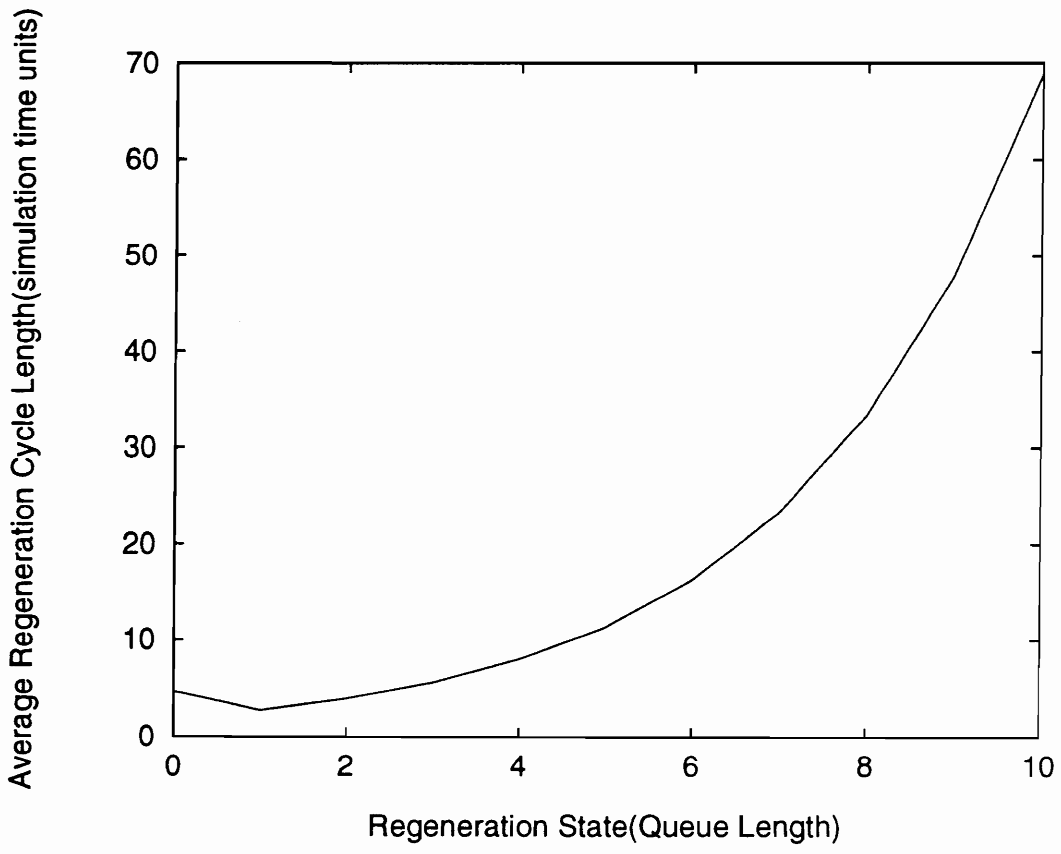


Figure 6.2: Average RCL for the M/M/1/10 queue, where $\lambda = 70$, $\mu = 100$, and $P = 100$. The guideline stopping time is 100.

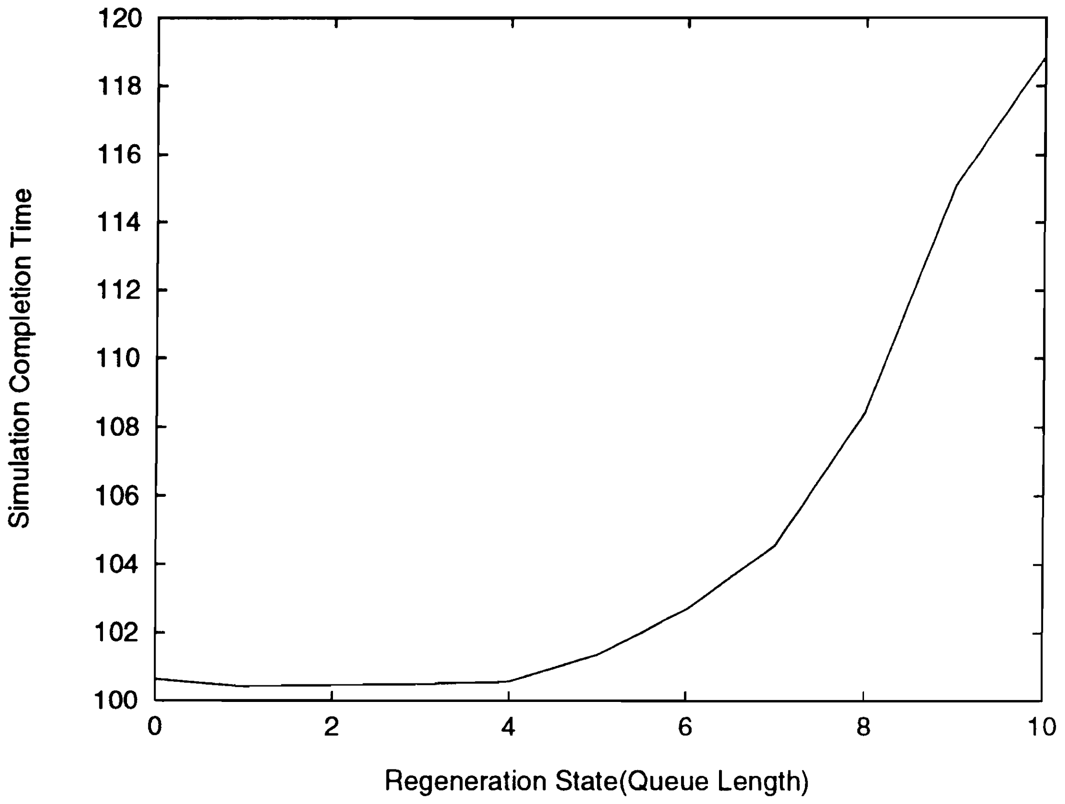


Figure 6.3: Parallel regenerative simulation completion time for the M/M/1/10 queue, where $\lambda = 70$, $\mu = 100$, and $P = 100$. The guideline stopping time is 100.

a short (or even finite) expected RCL. In such cases, in order to obtain enough observations, we may use approximate regeneration cycles by relaxing the regeneration condition. The concept of approximate regeneration was originally suggested by Crane and Iglehart [21]. In the next section, we present an approximate regeneration technique using substate matching which is inspired by the SSM approach described in the previous chapter (both cases achieve parallelism by relaxing the matching condition).

6.4 Parallel Regenerative Simulation with Substate Matching

Let $V = \{v_1, v_2, \ldots, v_m\}$ be the set of state variables of the target simulation model and $v_k(t)$, $1 \leq k \leq m$, denotes the value of v_k at simulation time t. A state variable $v_k \subset V$ is *redundant* if v_k can be uniquely determined by any other state variables in V. Assume that no state variables in V are redundant. Then we say that process $\mathbf{X}$ can be sufficiently described by the process $\mathbf{V}$=$\{v_1(t), v_2(t), \ldots, v_m(t), t \geq 0\}$ if there exists a set $W = \{v_{k_1}, v_{k_2}, \ldots, v_{k_{m'}}\} \subseteq V$, $1 \leq m' \leq m$, and a function f such that $X(t) = f(W(t))$ for all $t \geq 0$, where $W(t) = (v_{k_1}(t), v_{k_2}(t), \ldots, v_{k_{m'}}(t))$. If there does not exist any set $W' \subset W$ and a function f' such that $X(t) = f'(W'(t))$ for all $t \geq 0$ is satisfied then we refer to set W as an *essential* set for $\mathbf{X}$. That is, the state of $\mathbf{X}$ directly depends on the state variables in an essential set.

Assume that process $\mathbf{V}$ is regenerative. This is not a restrictive assumption because Crane and Iglehart [20] have shown that with appropriate arrangement of the state vector, a simulation model can always be structured as a regenerative process. If process $\mathbf{V}$ enters regeneration state $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ at time t (i.e., $r_k = v_k(t)$ for all $1 \leq k \leq m$), we say that the process regenerates on $\mathbf{r}$ at t.

Define *substate* $\mathbf{r}_W$ to be the portion of state $\mathbf{r}$ corresponding to set W. For example, if $W = \{v_1, v_2, v_4\}$, $\mathbf{r}_W$=$\{r_1, r_2, r_4\}$. We say that the system *partially regenerates*[1] on $\mathbf{r}_W$ at time t if $v_{k_j}(t) = r_{k_j}$ for all $1 \leq j \leq m'$. A partial regeneration cycle length (PRCL) is

[1]The term 'partial regeneration' was introduced in [58].

the time interval between two consecutive occurrences of partial regeneration. Then we can see that for any regeneration state $\mathbf{r}$, the expected RCL on $\mathbf{r}$ is no larger than the expected PRCL $\mathbf{r}_W$. This is obvious because if the system partially regenerates on $\mathbf{r}$ at any time t, the system must also regenerate on $\mathbf{r}_W$ at t, and the converse is not true. Therefore, if the expected PRCL corresponding $\mathbf{r}_W$ is finite and when $t \to \infty$, it is almost certain that number of partial regeneration cycles (corresponding to $\mathbf{r}_W$) will be larger than the original regeneration cycles (corresponding to $\mathbf{r}$). Based on this property, we consider the following approximate regenerative approach:

Partially Regenerative (PR) Simulation

1. Identify an essential set W and a substate $\mathbf{r}_W$ such that the target system partially regenerates on $\mathbf{r}_W$ relatively frequently.

2. Initialize the state variables corresponding to W with substate $\mathbf{r}_W$.

3. Initialize the state variables corresponding to $W' = V - W$ (where '-' is the set difference operator) appropriately.

4. For all i, $1 \le i \le P$, processor i independently simulates the target system until both of the following conditions are satisfied:

   (1) The system partially regenerates on $\mathbf{r}_W$.

   (2) At least τ/P simulation time units have been simulated.

5. Let $N_i'(\tau/P)$ denote the number of partial regeneration cycles completed on processor i in τ/P simulation time units and let Y_{ij}' and ω_{ij}' denote the j^{th} observation and the j^{th} PRCL, respectively, on processor i. Compute an estimate for μ as follows:

$$\hat{\mu}_4(P, \frac{\tau}{P}) = \frac{\sum_{i=1}^{P} \sum_{j=1}^{N_i'(\tau/P)+1} Y_{ij}'}{\sum_{i=1}^{P} \sum_{j=1}^{N_i'(\tau/P)+1} \omega_{ij}'}.$$

The PR approach obtains an estimate for μ using (6.3) by treating the partial regeneration cycles (i.e., on $\mathbf{r}_W$) as though they are true regeneration cycles. We refer $\mathbf{r}_W$ as the *matching substate* because a partial regeneration cycle is determined by a match between the system substate corresponding to W and $\mathbf{r}_W$. Due to the relaxation of the regeneration condition, the partial regeneration cycles are unlikely to be IID. Consequently, the simulation results could be biased. Nevertheless, by applying Gunther's *almost renewal process* theorem [35], it can be shown that $\hat{\mu}_4(P, \frac{\tau}{P})$ is a strongly consistent estimator for μ as $\tau \to \infty$. When strong correlation is expected among the partial regeneration cycles, we may apply the *batch means* method by collecting only one observation for every $b > 1$ consecutive partial regeneration cycles. When b is large, under some mild conditions, it can be shown that the resultant observations are approximately uncorrelated [53].

Application

To illustrate the PR approach, we consider an example shown in Figure 6.4 which represents a packet-switched local area network (LAN) of four hosts (i.e., nodes 1 to 4) and a gateway (i.e., node 5) that connects the LAN to other networks. Each node is modeled by a loss G/G/1/K queue, where arriving packets that find the queue full are dropped, representing a packet loss due to a buffer overflow. For each departing packet from a host, the next node to enter is chosen randomly from those connected to the host.

Let QL_i denote the queue length of node $i, 1 \le i \le 5$. Suppose we are interested in the average congestion level (i.e., the queue length) of the gateway. Then the essential set $W = \{QL_5\}$. For the matching substate $\mathbf{r}_W$, observe that for any G/G/1/K queue where $K < \infty$, the queue length being zero (respectively, K) will always have a relatively high occurrence frequency in the long run when λ/μ is sufficiently smaller (larger) than 1. Therefore, we may simply let $\mathbf{r}_W = K$ and $\mathbf{r}_W = 0$ for $\lambda/\mu > 1$ and $\lambda/\mu < 1$, respectively. Otherwise, the FOS pilot simulation can be applied to estimate a frequently occurring substate for $\mathbf{r}_W$. Finally, the queues corresponding to nodes 1 to 4 have to be initialized appropriately to avoid large initial transient. This can be achieved by using the PI pilot

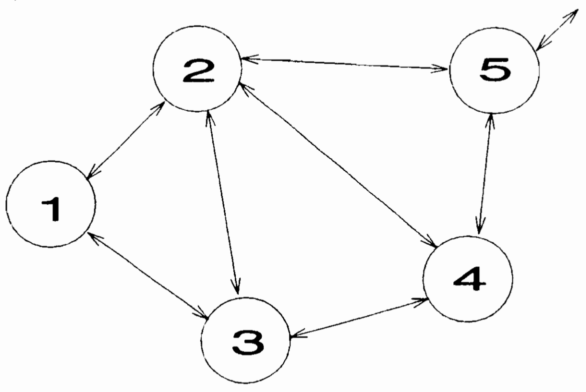


Figure 6.4: An open queueing network of five nodes. Each node contains a queue and a server to retain and serve the arriving jobs.

simulation discussed in Chapter 4.

6.5 Experimental Results

In this section, we use the multiplexer and the virtual circuit (VC) models considered in Section 4.4 to evaluate the performance of the PR approach.

6.5.1 Multiplexers

Recall our discussion in Chapter 4 that a multiplexer modeled by a loss G/G/1/K queueing can be represented by a regenerative process $\{NAT(t), QL(t), ST_j(t), t \geq 0\}$, where NAT, QL, and $ST_j, 1 \leq j \leq K$ are the time interval before the next arrival time, queue length, and the remaining service time of the j^{th} job in the buffer ($ST_j = 0$ for $j > QL$), respectively. Note that any process that contains only a proper subset of these three state variables is not a regenerative process for general interarrival and service times. Supposed we are interested in the average queue length. Then for the PR approach, the essential set $W = \{QL\}$. For the PR runs, the initial value of NAT is drawn from the interarrival time distribution, while the initial values of ST_i, $1 \leq j \leq QL$, are drawn independently from the service time distribution. For the MR-D runs, the queue are idle initially.

The M/M/1/100 queue and $\Gamma(2)$/D/1/100 queue are used, where $\Gamma(2)$ denotes the gamma distribution with a shape parameter 2. For the PR simulation, the desired simulation length is $\tau = 10^4$ simulation time units. Thus, the guideline stopping time is $10^4/P$ simulation time units. The matching substate $\mathbf{r}_W$ in each case is obtained by applying the FOS pilot simulation. The PR approach is compared with the MR-D approach. For a fair comparison, the two approaches use the same overall simulation time (including the pilot simulations). That is, the simulation lengths for the MR-D runs are given by

$$\tau_{MR-D}(P) = \frac{\sum_{i=1}^{P} \sum_{j=1}^{N_i'(\tau_r/P)+1} \omega_{ij}' + \Delta\tau_{PR}}{P},$$

where τ_{PR} denotes the pilot simulation run length for the PR simulation. In our experiments, τ_{PR} is equal to 20% of the guideline stopping time of the PR runs (i.e., $2000/P$ simulation time units).

Experimental results are given in Tables 6.1 and 6.2, where MR-D$_x$ represents MR-D simulation with the first x% of the simulation time interval deleted. The number in the square brackets in the tables are the normalized simulation errors which indicate the statistical efficiencies of the simulation runs.

The results show that the PR approach produces highly accurate estimates for all tested cases. Typical errors against the expected values are well under 1.0%. The MR-D simulation is competitive in accuracy only when the traffic load is low (e.g., $\lambda/\mu = 0.5$). The major disadvantage of the PR approach is that the simulation completion time of each processor is a random variable. The MR-D approach, on the other hand, has a fixed completion time. Nevertheless, for the PR simulation, except when $\lambda/\mu \to 1$, each processor reaches a partial regeneration point very quickly. Therefore, the completion time of the two approaches are approximately the same. When λ/μ is near one, however, the queue length becomes unstable, some processors may take a long time to reach a partial regeneration point. This is illustrated in Figures 6.5, where the simulation completion time increases dramatically when $\lambda/\mu \to 1$.

Table 6.1: The average queue lengths with approximate 90% confidence intervals of the M/M/1/100 model. Values in row 'EXPECT' are computed analytically. The values in the square brackets show the normalized errors against the expected values. Absence of a value in square brackets denotes an error no more than 1.0%.

Average Queue Length (M/M/1/100)

P	Approach	λ/μ =0.5	λ/μ =0.7	λ/μ =0.9	λ/μ =1.0	λ/μ =1.1
	EXPECT	1.0	2.333	8.997	50.0	90.007
100	PR	1.00(±.01)	2.34(±.03)	9.03(±.32)	48.87(±2.92) [2.3%]	90.01(±.43)
	MR-D$_0$	1.01(±.01)	2.35(±.03)	9.20(±.33) [2.2%]	47.13 (±1.67) [5.7%]	86.54(±.40) [3.8%]
	MR-D$_{20}$	1.00(±.01)	2.34(±.03)	9.15(±.30) [1.7%]	51.85(±2.12) [3.7%]	90.01(±.33)
	MR-D$_{50}$	1.00(±.01)	2.32(±.03)	8.94(±.39)	50.30(±2.23)	90.06(±.38)
	MR-D$_{90}$	1.00(±.02)	2.31(±.07)	9.98(±1.1) [10.9%]	48.56(±3.91) [2.9%]	89.51(±.84)
1000	PR	.99(±.01)	2.63(±.03)	8.83(±.31) [1.9%]	49.03(±1.59) [1.9%]	90.05(±.31)
	MR-D$_0$	1.00(±.01)	2.31(±.02)	8.33(±.21) [7.4%]	38.16(±.84) [23.7%]	60.69(±.73) [32.6%]
	MR-D$_{20}$	1.00(±.01)	2.35(±.03)	8.56(±.23) [4.9%]	42.19(±.99) [15.6%]	68.44(±.82) [43.5%]
	MR-D$_{50}$	1.00(±.01)	2.34(±.03)	9.31(±.32) [3.4%]	46.31(±1.16) [7.4%]	81.53(±.78) [9.4%]
	MR-D$_{90}$	1.03(±.01) [3.0%]	2.37(±.06) [1.7%]	9.12(±.38) [1.3%]	48.08(±1.42) [3.6%]	87.04(±.74) [3.3%]
10000	PR	.99(±.01)	2.63(±.03)	8.98(±.25)	49.28(±.98) [1.4%]	89.94(±.27)
	MR-D$_0$	.96(±.01) [4.0%]	2.08(±.02) [10.7%]	5.62(±.06) [37 .5%]	27.40(±.24) [45.2%]	14.14(±.12) [84.3%]
	MR-D$_{20}$	1.01(±.01)	2.27(±.02) [2.6%]	6.11(±.07) [32.1%]	30.37(±.28) [39.3%]	16.76(±.15) [81.4%]
	MR-D$_{50}$	.99(±.01)	2.30(±.03) [1.3%]	6.86±.09) [23.7%]	35.3 4(±.36) [28.9%]	20.10(±.24) [77.5%]
	MR-D$_{90}$	1.00(±.01)	2.46(±.04) [5.6%]	7.44(±.11) [17.3%]	38.96(±.43) [22.1%]	24.71(±.24) [72.5%]

Table 6.2: The average queue lengths of the $\Gamma(2)/D/1/K$ model with approximate 90% confidence intervals. Each entry in row 'EXPECT' is estimated by a simulation of 10 independent replications for a total of 10^8 arrivals. The values in the square brackets are the normalized errors against the 'EXPECT' values. Absence of a value in square brackets denotes an error less than 1.0%.

Average Queue Length ($\Gamma(2)/D/1/100$)

P	Approach	λ/μ =0.5	λ/μ =0.7	λ/μ =0.9	λ/μ =1.0	λ/μ =1.1
	EXPECT	.588	1.045	2.836	50.682(±.358)	97.28
	PR	.588(±.001)	1.046(±.004)	2.83(±.04)	50.89(±2.61)	97.31(±.05)
	MR-D_0	.589(±.001)	1.047(±.004)	2.83(±.04)	45.41(±2.31) [10.4%]	92.53(±.20) [4.9%]
100	MR-D_{20}	.588(±.002)	1.048(±.004)	2.86(±.05)	48.60(±2.71) [4.1%]	97.32(±.05)
	MR-D_{50}	.590(±.002)	1.043(±.006)	2.81(±.05)	48.99(±3.3) [3.3%]	97.29(±.06)
	MR-D_{90}	.589(±.001)	1.041(±.013)	2.81(±.14)	48.34(±4.5) [4.6%]	97.13(±.16)
	PR	.588(±.001)	1.045(±.004)	2.83(±.04)	50.71(±1.81)	97.27(±.05)
	MR-D_0	.587(±.001)	1.046(±.004)	2.81(±.04)	29.12(±.80) [42.5%]	51.37(±.57) [47.2%]
1000	MR-D_{20}	.588(±.002)	1.046(±.004)	2.80(±.04)	34.40(±.97) [32.1%]	61.55(±.67) [36.7%]
	MR-D_{50}	.590(±.002)	1.044(±.006)	2.83(±.05)	37.76(±1.20) [25.5%]	75.61(±.73) [22.3%]
	MR-D_{90}	.591(±.005)	1.035(±.01) [1.7%]	2.84(±.10)	41.17(±1.36) [18.8%]	89.35(±.64) [8.1%]
	PR	.587(±.001)	1.045(±.004)	2.79(±.03) [1.8%]	50.21 (±1.08)	97.29(±.04)
	MR-D_0	.578(±.001) [1.7%]	1.018(±.003) [2.6%]	2.37 (±.02) [16.2%]	14.09(±.13) [72.2%]	8.66(±.06) [91.1%]
10000	MR-D_{20}	.589(±.002)	1.053(±.004) [2.6%]	2.54(±.02) [10.2%]	15.67(±.15) [69.0%]	9.86(±.07) [89.9%]
	MR-D_{50}	.588(±.002)	1.049(±.006)	2.68(±.03) [5.3%]	17.60(±.18) [65.3%]	12.01(±.09) [87.5%]
	MR-D_{90}	.590(±.004)	1.086(±.01) [3.9%]	2.76(±.04) [2.5%]	2 0.22(±.24) [60.1%]	17.15(±.11) [82.4%]

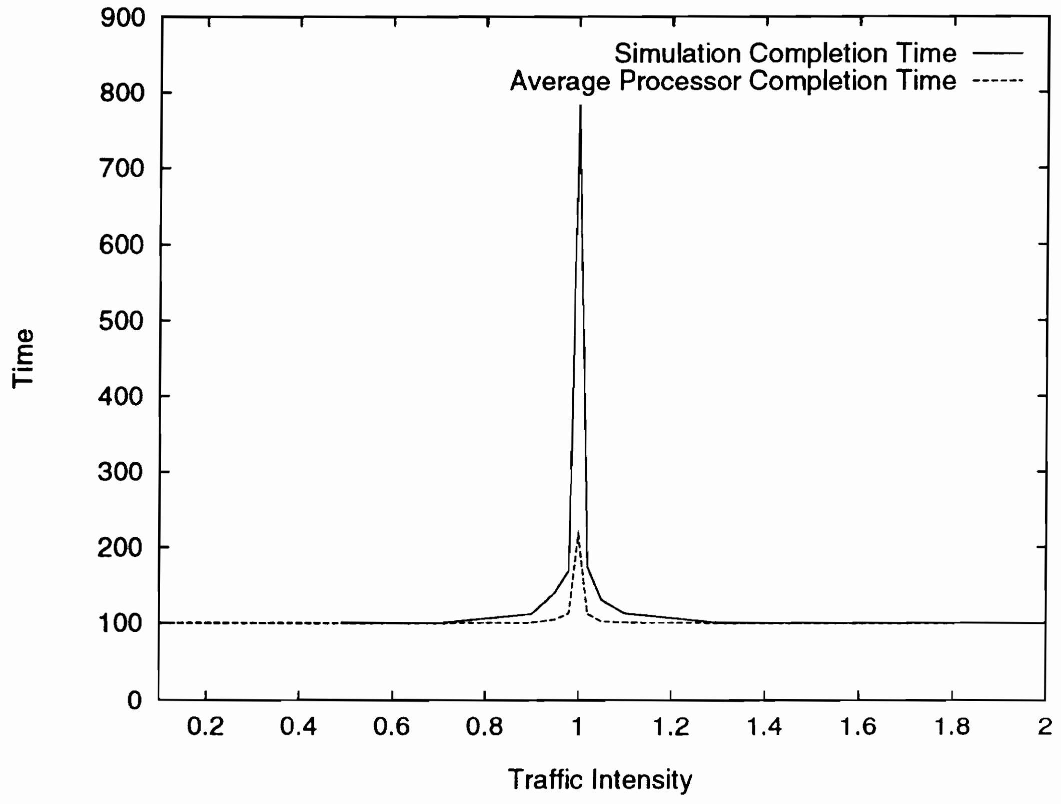


Figure 6.5: The PR simulation completion time for the M/M/1/100 model for $P = 100$. The guideline stopping time is 100 (simulation time units).

Table 6.3: The most frequently occurring state of QL_2 for the VC model.

	$P = 100$	$P = 1000$	$P = 10000$
$\mathbf{r}_W$	16	16	16

6.5.2 Virtual Circuit

We consider the VC model that we use to evaluate the PI approach in Section 4.4. Compared to the G/G/1/K model in the previous section, the VC model has a much larger state vector and the state variables (e.g., the queue lengths) belonging to different nodes are correlated. To apply the PR simulation, the essential set $W = \{QL_2\}$. To determine the matching substate $\mathbf{r}_W$, again, we apply the FOS pilot simulation. The lengths for the FOS runs are 10 simulation time units (i.e., 1% of the guideline stopping time when $P = 100$) for all P's. The results of the FOS pilot simulation are given in Table 6.3. For the MR-D simulation, each run starts at a state where the packets are evenly distributed among all nodes.

Experimental results of the VC model are shown in Table 6.4. It shows that the PR simulation produces highly accurate and consistent results (less than 1.0% of error) in all cases. For the MR simulation, the errors are large for $P \geq 1000$. However, the trade-off for the high accuracy is the extra completion time (Figure 6.6).

6.6 Summary

Regenerative simulation is an appealing alternative for parallel simulation because of its potential in producing unbiased results. A problem in applying the parallel regeneration approach is that for many applications of interest there does not exist a regeneration state that results in a short expected regeneration cycle length. To enhance the applicability of the regenerative simulation, we present a partial regeneration (PR) technique which uses a substate matching technique (similar to the SSM approach in Section 5.4) to increase

Table 6.4: The estimated queue lengths of node 2 with approximate 90% confidence intervals. Each entry in column 'EXPECT' is estimated by a simulation of 10 independent replications, each of 10^6 simulation time units in length. The values in the square brackets are the normalized errors against the 'EXPECTED' values. Absence of a value in square brackets denotes an error less than 1.0%.

Approach	$P = 100$	$P = 1000$	$P = 10000$
EXPECTED	13.177(± .024)	13.177(± .024)	13.177(±.024)
PR	13.29(± 0.15)	13.16(± 0.13)	13.21(± 0.10)
MR-D_0	12.85(± .14)[2.4%]	11.16(± .12)[15.3%]	7.86(± .04)[40.3%]
MR-D_{20}	13.29(± .16)	12.38(± .14)[6.0%]	8.51(± .05)[35.4%]
MR-D_{50}	13.20(± .23)	13.03(± .15)[1.1%]	9.33(±.06)[29.2%]
MR-D_{90}	13.25(± .35)	13.65(± .19)[3.5%]	11.30(± .08)[14.2%]

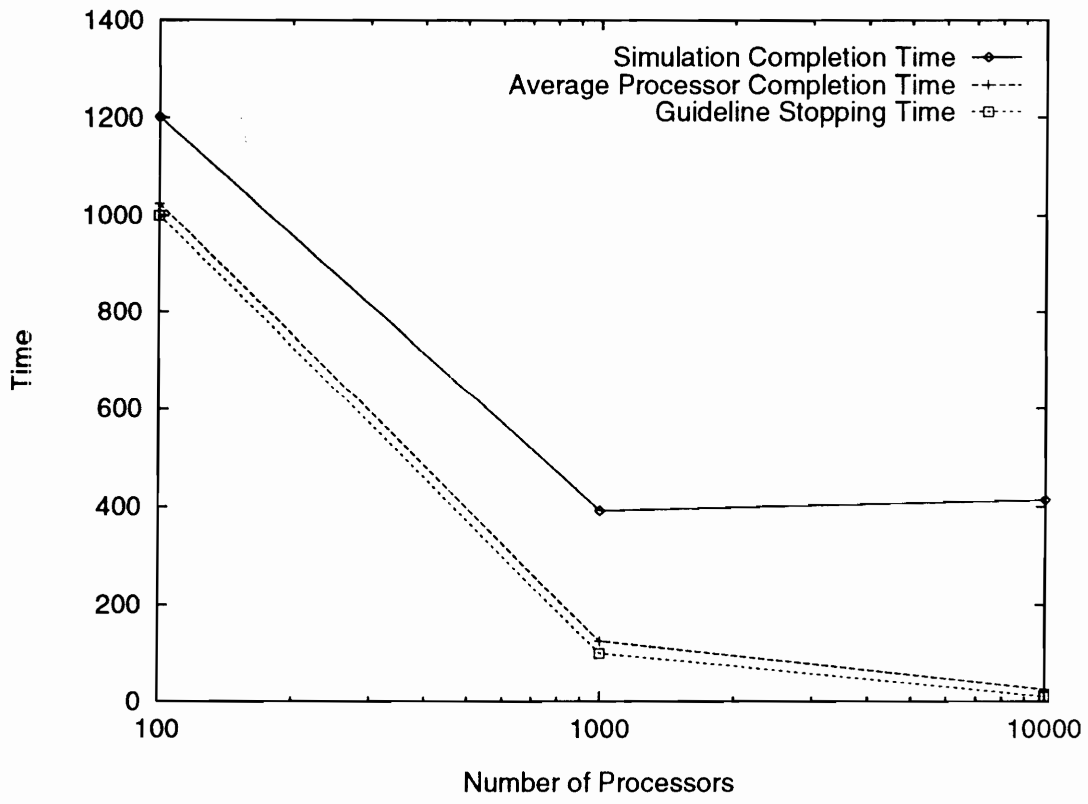


Figure 6.6: Completion times of the PR simulation for the VC model.

the number of observations. Our experiments with some queueing models show that the PR approach has better statistical efficiency compared to the MR-D approach (and the PI approach in Chapter 4 as well). The percent error of the PR approach is negligible in almost all cases run.

Note that in many cases, the problems of long expected regeneration intervals can not be solved by the substate matching technique along. For example, in the G/G/1 queue model, when $\lambda/\mu \to 1$, the system becomes unstable and some processors may take a long time to reach partial regeneration. Also, when the essential set consists of a continuous state variable, a matching substate that results in a finite expected PRCL may not exist. In such cases, we may further relax the regeneration condition by using a state aggregation technique [21], in which we group the state space corresponding to the essential set W into some finite aggregated states. Then a partial regeneration is considered to be reached whenever the system partially regenerates on any substate in a pre-selected aggregated state.

A major disadvantage of the PR approach (as well as any other regenerative simulation approach) is that the simulation completion time is a random variable. Therefore, the execution time of the simulation can not be accurately predicted before the simulation. Another concern of applying the PR approach is the assessment of the simulation accuracy. This can be achieved by conducting a pilot simulation and then evaluating the correlation among the partial regeneration cycles generated by the pilot runs. The amount of the correlation provides an indication of the accuracy of the resultant PR simulation.

Chapter 7

A STUDY OF CONSERVATIVE SPACE-PARALLEL SIMULATION

In the previous chapters, we have explored multiple replication, time-parallel, and parallel regenerative simulations. In these methods, each processor simulates a replication of the entire simulation model. However, when the target simulation model is large, the model may not fit in the memory of the processor, and thus these methods are not appropriate. In this case, parallelism can be achieved by partitioning the simulation model into smaller components in the space domain of the simulation model such that each component contains a disjoint subset of state variables. Consider simulating a computer network that has a large number of nodes using P processors. A space-parallel simulation partitions the nodes into P sets of nodes; each set of nodes are simulated by a single processor concurrently.

Space-parallel simulations have been studied extensively in the past decade. The major two space-parallel approaches, namely optimistic and conservative, are evaluated rigorously in the literature. Generally speaking, for the optimistic approach, good performance (in terms of speed up) can be obtained only if the roll-back overhead is low [15, 27, 60]. For the conservative approach, many previous empirical studies have suggested that good performance relies on good lookahead capability [25, 27, 69, 77]. Although the performance of conservative simulation has been the interest in many previous studies (e.g., [57, 70, 72]), there has been a lack of formal arguments to quantify the impact of lookahead on conservative simulation performance. In this chapter, we develop stochastic models to analyze the relationship between the amount of lookahead and the performance of a conservative simulation. A review of conservative simulation is given first in section 7.1. In Section 7.2, the relation between the simulation time and lookahead is analyzed for both open and closed simulation models. Conclusions are given in Section 7.3.

7.1 Conservative Simulation

In conservative simulation, the communication channel between any pair of logical processes (LP) is modeled by a *link* which is specified statically. That is, all possible communication between two LPs has to be identified before the simulation run. For any pair of LPs, say LP_a and LP_b, we say that LP_b is a *descendent* of LP_a (likewise, LP_a is a *precedent* of LP_b) if LP_a can send LP_b messages. In each LP, incoming messages through each link are stored in a message queue in FIFO order. Each incoming link is associated with a clock whose value is equal to the timestamp of the first message in the corresponding message queue or the timestamp of the last received message if the corresponding message queue is empty. The term *least-valued link* of an LP refers to the incoming link that has the smallest clock value. Also, *logically-next message* refers to the first message in the message queue corresponding to the least-valued link, if the queue is non-empty. Let $t(m)$ denote the timestamp of a message m. In the conservative simulation, each LP repeatedly performs the following steps:

1. Identify the logically-next message m and process all local events (from the local event queue of the LP) that have a timestamp no greater than $t(m)$ in the order of their timestamps. For each event executed, update the local clock of the LP to the timestamp of the event.

2. Process message m (which may in turn generate some local events and output messages for other LPs). Place all newly-created local events and output messages, respectively, in the local event queue and output message queues.

3. Process all output messages (in output message queues) that have a timestamp no greater than $t(m)$ in the order of their timestamps.

4. Repeat steps 1 to 3 until the simulation termination condition is satisfied.

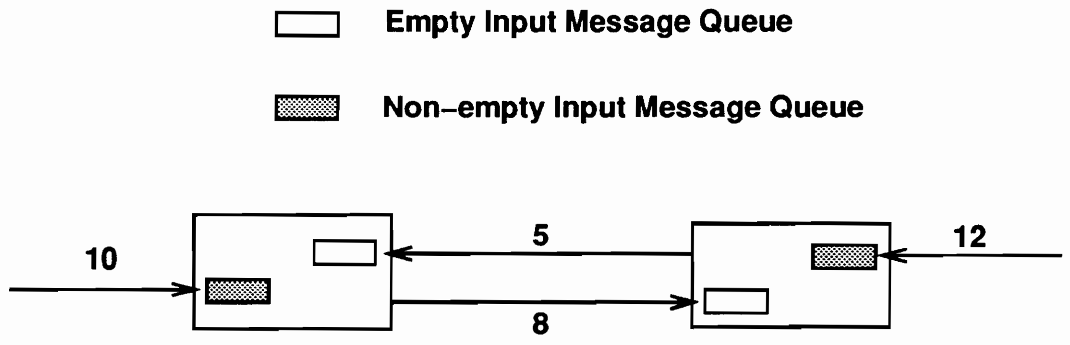


Figure 7.1: A deadlock. Processes A and B are waiting for each other to send messages. The number labeled with each link is the clock value of the link.

In this paper, we assume message processing is non-preemptive. Note that in step 1, when the message queue corresponding to the least-valued link is empty, the logically-next message can not be determined and thus the LP has to *block* until some messages arrive at that message queue. A deadlock occurs if such blocking forms a circular dependency. An example of a deadlock is given in Figure 7.1.

Two approaches have been used to handle a deadlock, namely *deadlock avoidance* and *deadlock detection and recovery* (or simply *deadlock recovery*). Deadlock avoidance algorithms [10, 11] use *null messages* to prevent deadlocks from occurring. This is achieved in a way that whenever an LP finishes processing an event, a null message, whose timestamp indicates a lower bound on the simulation time of the next message created by the LP, is sent to all its descendants. It can be shown that as long as there does not exist any cycles in which the collective timestamp increment of the null messages is zero, no deadlock will occur [64]. Note that, a null message does not have any corresponding object in the target simulation model.

The use of null messages may pose a great amount of overhead. To reduce null message traffic, a conservative approach in which null messages are created on a demand basis has been proposed [64]. In this approach, a null message is created only if any of an LP blocks.

Unlike the deadlock avoidance approach, the deadlock recovery approach does not use any null messages and allows deadlocks to occur. Instead, a special mechanism is applied to detect and remove deadlocks. Lin [57] has shown that the deadlock recovery approach is in general inefficient. Unless mentioned otherwise, we assume that the deadlock avoidance approach is used for conservative simulation. Throughout this paper, the terms 'simulation time' and 'simulation execution time' refer to the *simulated* time and the *real* (i.e., wall clock) computation time for the simulation, respectively.

7.2 The Impact of Lookahead on Performance

An LP is said to have a *lookahead* ℓ if the clock value of the LP is t and the LP can predict that the timestamps of all events that arrive later are no smaller than $t + \ell$, for all $t \geq 0$. A number of empirical studies (e.g., [25, 27, 69]) have suggested that lookahead plays an important role in the performance of the conservative simulation. That is, good lookahead ability (i.e, a larger lookahead value) results in shorter execution time. Intuitively, this is because a good lookahead reduces the processor blocking time. Consider the example shown in Figure 7.2, where the LP has two input links with clock values 8 and 10, respectively. The LP has processed a message from the top message queue and updated its local clock value to 8. Because the top message queue becomes empty, the LP has to block if the LP does not have a lookahead of at least 2. Therefore, it appears that a larger lookahead implies a smaller blocking probability, hence a shorter execution time.

7.2.1 Does Lookahead Help?

Before we analyze the impact of lookahead on simulation performance with respect to different system topologies, this section first argues that a larger lookahead guarantees no worse simulation performance.

Suppose that the target simulation model is mapped into P LPs, denoted LP$_1$, LP$_2$, ..., LP$_P$. Let G^a and G^b be two conservative simulation implementations for an arbitrary

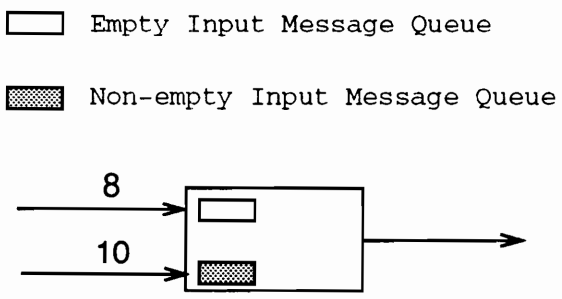


Figure 7.2: An LP which has two input links with clock values 8 and 10, respectively. Unless the LP has a lookahead of at least 2, the LP has to block until a message arrives at the empty input message queue.

simulation model. Let ℓ_i^a (respectively, ℓ_i^b) denote the lookahead of the i^{th} LP for G^a (G^b). Suppose that G^a and G^b are identical (including all random variates involved in the simulation) except that $\ell_i^a \geq \ell_i^b$ for all i, $1 \leq i \leq P$. For convenience, the same set of symbols are used for variables in G^a and G^b except that each symbol is superscripted with 'a' or 'b' for distinction. Let T_r denote the total simulation execution time. Then T_r^a and T_r^b represent the simulation execution times for G^a and G^b, respectively.

Assume that G^a and G^b are run simultaneously on two identical, dedicated (i.e., to only the execution of G^a and G^b) multiprocessor systems. Let $e_{i,1}, e_{i,2}, \ldots$ be the sequence of events (ordered by their timestamps) that are executed by LP$_i$ during the simulation. Also, let $t_r(e_{i,j})$, where $1 \leq j$, denote the real time at which event $e_{i,j}$ is executed.

Lemma 6 *For any simulation model, $t_r^a(e_{i,j}) \leq t_r^b(e_{i,j})$, for all $1 \leq i \leq P$ and $1 \leq j$.*

Proof: Assume that there exists an event $e_{i,j}$ such that $t_r^a(e_{i,j}) > t_r^b(e_{i,j})$. Then there must exist an event $e_{i',j'}$, where $1 \leq i' \leq P, j' \leq j$, such that both of the following conditions are satisfied:

1. For all events $e_{i'',j''}$ such that $t_r^b(e_{i'',j''}) < t_r^b(e_{i',j'})$, $t_r^b(e_{i'',j''}) \geq t_r^a(e_{i'',j''})$.

2. $t_r^a(e_{i',j'}) > t_r^b(e_{i',j'})$.

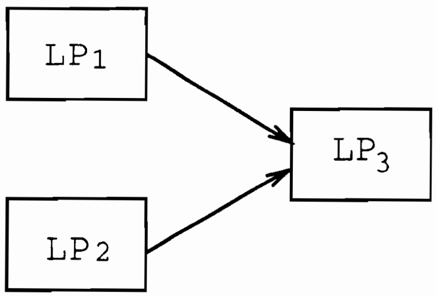


Figure 7.3: A simple simulation model of three LPs.

Given condition 1, condition 2 can be true only if $LP^a_{i'}$ blocks and $LP^b_{i'}$ does not block immediately before $e_{i',j'}$ is executed. However, this is impossible because $\ell^a_{i'} \geq \ell^b_{i'}$. □

Corollary 1 $T^a_r \leq T^b_r$ *for any given simulation model.*

Proof: Follows directly from Lemma 6. □

Therefore, a larger lookahead guarantees no worse performance. Examples showing that a larger lookahead does indeed reduce simulation execution time can be easily constructed. A question arises immediately is that how much such improvement can we expect? We answer this question in the following sections.

7.2.2 Open Models

We first consider the simple three-LP model shown in Figure 7.3. This simple model is referred to as the *basic* model. Note that, for an acyclic model a deadlock can not occur and hence null messages are unnecessary. Assume that the simulation terminates immediately after the local clock of LP_3 reaches time τ.

To evaluate the effect of lookahead, we consider two implementations G^O and G^Z where G^O has an *optimal* lookahead while G^Z has zero lookahead. That is, in G^O, all LPs can exactly predict the timestamps of all messages that have not arrived at any time. Thus, it is guaranteed that no blocking occurs in G^O for any LP only if the message queue corresponding to the least-valued link is empty. For G^Z, on the other hand, no prediction

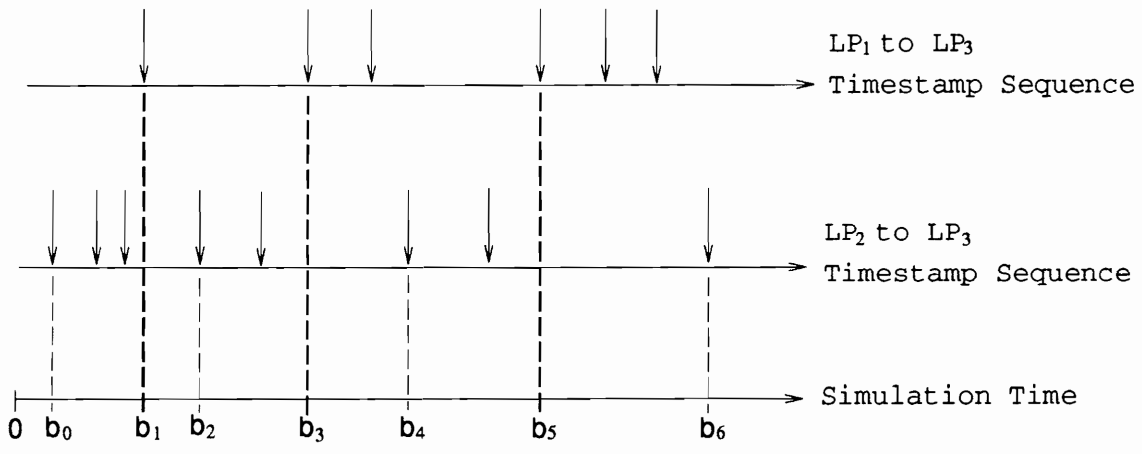


Figure 7.4: Timestamp sequences for the basic model. Each simulation time interval $[b_i, b_{i+1})$ consists of a set of timestamps corresponding to messages that are sent by the same LP.

for the timestamps of future messages is made. Based on Corollary 1, we see that G^Z and G^O, respectively, give the upper bound and the lower bound on the simulation execution time for the conservative simulation with any amount of lookahead.

Let $[0, \tau]$ be the simulation time interval. We may obtain a set of subintervals divided by a set of $n (n \geq 1)$ increasing simulation time points $b_0, b_1, \ldots b_n$, such that: (1) each b_i, $0 \leq i \leq n$, corresponds to the timestamp of a message received by LP_3 and (2) all messages whose timestamps in the interval $[b_i, b_{i+1})$ are sent by the same LP (i.e., LP_1 or LP_2). An example of such a partition for the basic model is given in Figure 7.4. Each interval $[b_i, b_{i+1})$ will be referred to as a 'b-cycle'.

Analyzing the effect of lookahead requires some definitions. For simplicity, we assume that no two messages arrive simultaneously. Let $t_r(m)$ denote the real time at which message m arrives. Let $m(t)$ denote the message that arrives at simulation time t. Let R_i be the real time interval between the arrivals of messages $m(b_i)$ and $m(b_{i+1})$ (i.e., $m(b_i)$ and $m(b_{i+1})$ are the messages that arrive at simulation times b_i and b_{i+1}, respectively). That is, $R_i = t_r(m(b_{i+1})) - t_r(m(b_i))$. Also, let $T_r^h(t)$ (respectively, $T_r^l(t)$) denote the real time at which the simulation clock corresponding to G^O (G^Z) advances to simulation time t. Then the 'execution time gap' (i.e., the real execution time gap between G^O and G^Z) when the simulation clock advances to t is given by:

$$\Delta T_r(t) = T_r^l(t) - T_r^h(t) \geq 0.$$

Assume that the message transmission time is negligible. Then we can show that (in Lemma 7 and Corollary 2) $\Delta T_r(t)$ is bounded by $\max_{0 \leq i \leq n}\{R_i\}$ for all $t \geq 0$.

Lemma 7 *For all* $n \geq 0, \Delta T_r(b_n) \leq \max_{0 \leq i \leq n}\{R_i\}$.

Proof: This lemma can be proved by induction on n. When $n = 0$, $T_r^h(b_0)$ and $T_r^l(b_0)$ correspond to the real times that LP_3^h (i.e., LP_3 in G^O) and LP_3^l (i.e., LP_3 in G^Z) start processing their first message. It is clear that $T_r^h(b_0) \geq 0$ and $T_r^l(b_0) \leq t_r(m(b_1)) - t_r(m(b_0)) = R_0$. Thus, $\Delta T_r(b_0) \leq R_0$ (induction base). Assume that, $\Delta T_r(b_i) \leq \max_{0 \leq k \leq i}\{R_k\}$ for some $i \geq 0$ (induction hypothesis). We need to show that $0 \leq \Delta T_r(b_{i+1}) \leq \max_{0 \leq k \leq i+1}\{R_k\}$. To derive $\Delta T_r(b_{i+1})$, we consider the following three possible cases that could occur when LP_3^l processes the messages in the interval $[b_i, b_{i+1})$:

Case 1: LP_3^l runs continuously without blocking. Because no time has been spent in blocking, in this case, when LP_3^l starts processing $m(b_{i+1})$, the execution time gap between G^Z and G^O will not increase (It decreases if LP_3^h blocks while LP_3^l processing the messages in $[b_i, b_{i+1})$). Therefore, $\Delta T_r(b_{i+1}) \leq \Delta T_r(b_i)$.

Case 2: LP_3^l blocks for a message, say $m(b)$, in the interval $[b_i, b_{i+1})$. Note that, for LP_3^l and LP_3^h, the real message arrival times are the same for all messages. Thus, in this case, LP_3^h also has to block for message $m(b)$. Therefore, at this point, the progress of G^Z 'catches up with' G^O. That is, $\Delta T_r(b_{i+1}) = 0$.

Case 3: LP_3^l blocks for message $m(b_{i+1})$. At this point, LP_3^l has completed all messages in $[b_i, b_{i+1})$. Because $m(b_{i+1})$ has not arrived yet, we know that G^O must be ahead of G^Z for less than R_{i+1} real time units. Because LP_3^l can not resume computing until $m(b_{i+1})$ arrives, thus $\Delta T_r(b_{i+1}) = R_{i+1}$ in this case.

Combining the above cases yields $0 \leq \Delta T_r(b_{i+1}) \leq \max_{0 \leq k \leq i+1}\{R_k\}$. □

Corollary 2 *For all t in $[0,\tau]$, $\Delta T_r(t) \le \max_{0 \le i \le n}\{R_i\}$.*

Proof: This corollary follows directly from the proof of Lemma 7. □

Corollary 2 gives an upper bound (i.e., $\max_{0\le i \le n}\{R_i\}$) on the execution time gap between G^O and G^Z at any point during the simulation. Assume that $\{R_0, R_2, \ldots\}$ and $\{R_1, R_3, \ldots\}$ are two sets of IID (independent and identically distributed) random variables. Then an expected upper bound for $\Delta T_r(t)$ can be derived easily as discussed below.

Suppose the simulation terminates after LP_3 simulates n b-cycles. For convenience, we assume that n is an odd integer. Then from Lemma 7, we have

$$E(\Delta T_r(b_n)) \le max\{E(R_0^{(n/2)}), E(R_1^{(n/2)})\}, \tag{7.1}$$

where $R_0^{(n/2)}$ and $R_1^{(n/2)}$ are the $(n/2)^{th}$ order statistic with respect to R_0 and R_1, respectively. Let $E(R_u) = max\{E(R_0), E(R_1)\}$ and $E(R_u^{(n/2)}) = max\{E(R_0)^{(n/2)}, E(R_1)^{(n/2)}\}$. Then it can be shown that [22, Ch.4.2]

$$E(R_u^{(n/2)}) \le E(R_u) + \frac{\sigma_u(n/2-1)}{\sqrt{n-1}}, \tag{7.2}$$

where σ_u^2 is the variance of R_u. Suppose R_u is exponentially distributed with a mean 1. Then it can be derived that

$$E(\Delta T_r(b_n)) \le E(R_u^{(n/2)}) = \frac{n}{2}\int_0^\infty te^{-t}(1-e^{-t})^{\frac{n}{2}-1}dt = \sum_{i=1}^{n/2}\frac{1}{i}.$$

Lemma 8 $E(\Delta T_r(b_n)) \le E(R_u) + \frac{\sigma_u(n/2-1)}{\sqrt{n-1}} = O(\sqrt{n})$.

Proof: Follows directly from (7.1) and (7.2) and the fact that $E(R_u)$ is a constant with respect to n. □

Lemma 8 gives an upper bound on the expected execution time gap between G^O and G^Z.

Corollary 3 $\lim_{n\to\infty} \frac{E(\Delta T_r(b_n))}{n} \to 0$.

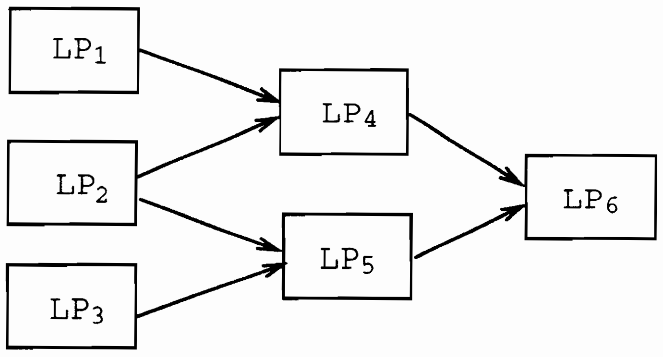


Figure 7.5: An open acyclic model.

Proof: Follows directly from Lemma 8. □

Corollary 3 states an important result: *no matter how large the lookahead is (assuming that it is finite), the performance improvement due to the lookahead becomes insignificant when the simulation length is long.* Note that, with arguments similar to those given above, we can easily show that this result also applies to the cases where LP_3 has more than two input message links.

The results obtained for the basic model extend to a more general acyclic model. Define a *source* LP to be one that has no precedent LP. We say that an LP has *depth d* (where $d \geq 0$) if the longest path from a source to the LP consists of d links. Consider the example in Figure 7.5. LP_1, LP_2 and LP_3 are sources with depth zero while LP_4 and LP_5 have depth one and LP_6 has depth two.

Assume that, except for source LPs, each output message is caused by exactly one input message and the output message waiting time is negligible. For example, in a queueing model, if each LP contains a single queue, then an output message (representing a job departure) can be sent to the decedent LP immediately after it is created (thus no output waiting time). Assume that the simulation terminates after an LP, say LP_x, simulates n b-cycles.

Let d_x denote the depth of LP_x. Then, it is expected that $E(\Delta T_r(n))$ increases as d_x increases. This is because in G^Z, messages are 'propagated' from the sources to other LPs

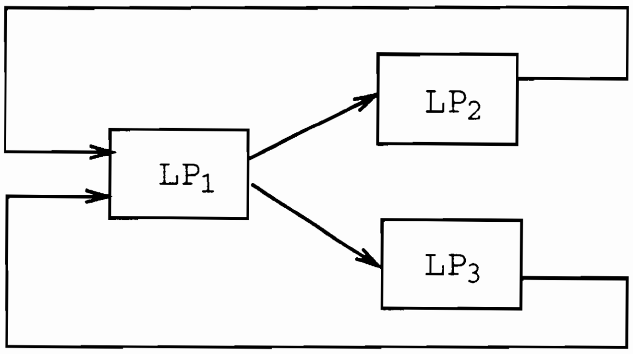


Figure 7.6: A basic closed model.

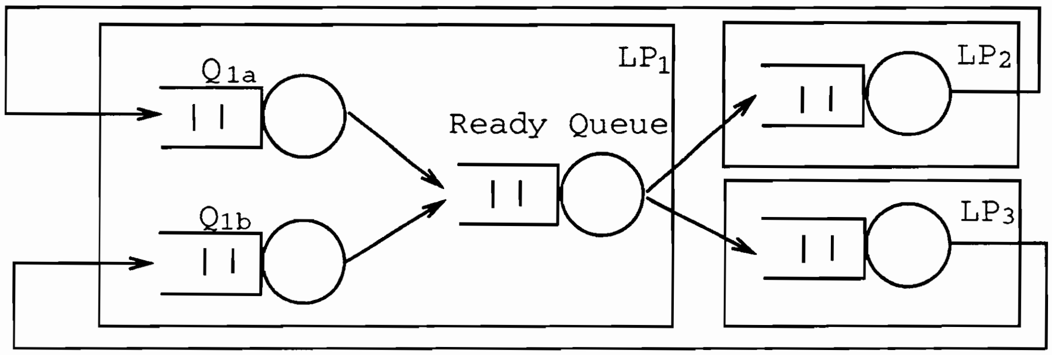


Figure 7.7: A queueing network representation of the basic closed model. Each queue corresponds to a message queue in the cyclic model. The extra queue for LP_1 is the 'ready' queue which buffers jobs that are ready for execution.

at a lower speed due to the lack of lookahead. However, from Corollary 2 and Lemma 8, we can see that for each extra level of depth for LP_x., $E(\Delta T_r(n))$ can increase by at most $c\sqrt{n}$ for some constant $c > 0$. That is,

$$E(\Delta T_r(n)) = O(d_x\sqrt{n}).$$

In this case, again, $\lim_{n\to\infty} \frac{E(\Delta T_r(n))}{n} \to 0$. Thus, for acyclic models when the simulation length is long, the performance gain due to an increased lookahead becomes insignificant regardless of the amount of the lookahead.

7.2.3 Closed Models

This section examines the effect of lookahead for closed models. For simplicity, we consider the simple closed model in Figure 7.6. To derive a performance model, we view this conservative simulation system as a closed queueing network (shown in Figure 7.7 in which the ready queue buffers jobs that are ready for execution) such that each message queue and a message, respectively, corresponds to a queue and a job in the queueing network. In this section, the terms 'message' and 'job' are used interchangeably.

To study the effect of lookahead for such a system, again, we consider two implementations G^a and G^b. The definitions of G^a and G^b are identical to those given in Section 7.2.1 (i.e., all LPs in G^a have a larger lookahead than that in G^b). Define the *response* time of the queueing network to be the time required for a job to circulate the network exactly once (i.e., the time interval between two consecutive visits to LP_1). Then we *mark* an arbitrary job and let W_i denote the i^{th} response time (i.e., the time spent in the i^{th} circulation) with respect to this marked job. Assume that, for all i greater than some sufficiently large i_0, the stochastic processes $\{W_i^a, i \geq i_0 > 0\}$ and $\{W_i^b, i \geq i_0 > 0\}$ are approximately covariance stationary. That is, for $i \geq i_0$, there exist two random variables W^a and W^b such that $E(W_i^a) \approx E(W^a)$ and $E(W_i^b) \approx E(W^b)$. Let $\Delta W = W^b - W^a$. From the result of Lemma 6, we know that $E(\Delta W) \geq 0$.

Consider queues Q_{1a} and Q_{1b} which correspond to the two message queues of LP_1 in Figure 7.7. Normally, if neither queue is empty, the jobs are passed to the ready queue with zero service times (representing no blocking). When one of the queues, say Q_{1b}, becomes empty and if the timestamp of the first message in Q_{1a} is greater than the sum of the lookahead and the clock value of LP_1, messages in Q_{1a} have to wait for some time δ (representing message blocking) until a job arrives at Q_{1b}. Then the waiting time δ can be treated as though it is the service time of the first job in Q_{1a}. Therefore, the larger the lookahead the less likely it is that a job will be assigned a service time greater than zero. As a result, a smaller response time is expected. Therefore, ΔW increases as the lookahead difference between G^a and G^b increases. Suppose the simulation terminates after the marked job circulates through the network n times, where $n \gg i_0$. Then the expected execution time gap (i.e., $E(\Delta T_r(n))$) between G^a and G^b is given by:

$$E(\Delta T_r(n)) \approx nE(\Delta W) = \Theta(n). \tag{7.3}$$

Then $E(\Delta T_r(n))$ grows linearly with the simulation length. Therefore, the amount of lookahead is critical to the simulation performance of closed models. From (7.3), the expected speedup of G^a over G^b due to a larger lookahead is given by:

$$E\left(\frac{T_r^b(n)}{T_r^a(n)}\right) \approx \left(\frac{E(T_r^a(n)) + nE(\Delta W)}{E(T_r^a(n))}\right) = 1 + E\left(\frac{\Delta W}{W^a}\right).$$

Therefore the speedup is independent of n.

Note that in our analysis null messages are ignored. With null messages, however, it will only increase the performance gap between G^a and G^b because G^b is likely to require more null messages. This agrees with our conclusion that lookahead is important for conservative simulation for closed models.

7.3 Summary

The effect of lookahead to conservative simulation is investigated. For open acyclic models, we derive an upper bound on the performance improvement due to lookahead. We show that when the simulation length is long, increasing the amount of lookahead will not improve the simulation performance significantly. For closed models in which message departure times determine future message arrival times, the amount of lookahead is critical to simulation performance. In particular, the speedup due to a larger lookahead converges to a constant as the simulation length increases.

Chapter 8
CONCLUSIONS

Simulation is one of the most important tools to study the performance of communication networks as well as many other systems. Because of its intense computational requirement, simulation often becomes a bottleneck in a system performance study. As computer hardware becomes more affordable, parallel simulation, which uses multiple processors to share the computation of a simulation becomes an appealing means to reduce the simulation execution time. This research addresses problems that arise as we attempt to parallelize simulation execution. By investigating a variety of parallel simulation methods, solutions to overcome some of these problems are proposed. The simulation models considered in this research focus on queueing systems that are representative of communication network systems. Specific contributions are summarized by chapter in the following sections.

8.1 Contributions

8.1.1 Parallel Simulation Primer

Chapter 2 categorizes parallel simulation approaches into four methods: multiple replication, time-parallel, parallel regenerative, and space-parallel. For a given simulation model, a parallel simulation method has to be chosen carefully to obtain adequate simulation performance. We identify the advantages and limitations of each of these methods (Table 2.1). This characterization provides a method selection guideline in a parallel simulation design (Figure 2.2).

8.1.2 Multiple Replication Simulation

Multiple replication simulation, in which each processor simulates an independent replication of the system with a fixed initial state and a fixed simulation length, is the most intuitive (and perhaps the simplest) way of conducting a parallel simulation. However, the result of a multiple replication simulation is subject to initial transient bias. To address the initial transient problem, we develop a polling initialization (PI) technique (Section 4.3) which uses a pilot simulation to select appropriate initial states that are representative of the system steady-state condition. The PI technique is best suited for the case when the estimator $\hat{\mu}(s,T)$ converges monotonically toward the steady-state u. We show that the accuracy of the PI technique (in terms of the closeness between the resultant initial states and u) increases as the pilot simulation length or the number of processors increases. Experimental results with some communication network systems suggest that the MR-PI approach can largely reduce the initial transient bias.

8.1.3 Time-Parallel Simulation

Time-Parallel simulation, which partitions the execution of a simulation model in time domain, can potentially exploit massive parallelism that can not be achieved by other methods. Therefore, when the target simulation model is small (i.e., in terms of the size of the state vector) while a large number of processors are available, time-parallel simulation is an appealing approach. Time-parallel simulation has not been fully explored by the research community partly because time domain parallelism is implicit and is often difficult to discover. In Chapter 5, we investigate time-parallel simulation and propose a set of approximate time-parallel algorithms (i.e., GG1K, IGG1K, GD1K, and SSM) for a variety of simulation models.

The GG1K and GD1K algorithms (Section 5.2) exploit unbounded parallelism for acyclic networks of G/G/1/K and G/D/1/K queues, respectively. The GG1K and GD1K algorithms produce approximate results. The closeness of the approximation is investigated analytically and experimentally. We show that the approximation errors are in general negligible except when the both K is small (e.g., 5) and the job service times are constant, or when the job service times are correlated. Experimental results with a variety of queueing systems show that the approximation errors are usually well under 1%. An algorithm called IGG1K (Section 5.2.2), which is a simple but iterative alternative for the GG1K algorithm, is also presented.

In this chapter we also show that computing the stationary distribution of an ergodic Markov chain can be transformed into a parallel prefix problem (and hence solvable through a parallel prefix algorithm). We also describe an intuitive recurrent state approach (Section 5.3.2) which allows time-parallel simulation for semi-Markov chains. In this recurrent state approach, we use a pilot simulation to identify a frequently occurring state (FOS), and partition the simulation trajectory at the occurrence time points of the resultant FOS to exploit time domain parallelism. Finally, for a more general class of simulation models, we present a substate matching (SSM) technique (Section 5.4). Substate matching results in approximate results. However, experimental results with a central server system (Section 5.4.1.1) and a virtual circuit [89] illustrate that the SSM approach can largely enhance the applicability of time-parallel simulation while still maintains high accuracy.

8.1.4 Parallel Regenerative Simulation

Parallel regenerative simulation obtains parallelism through partitioning the model trajectory at a set of regeneration time points where the system 'starts over'. When applicable, parallel regenerative simulation is an appealing approach because it is simple to implement and can produce highly accurate estimates. Previous research on this approach has focused on statistical implications with respect to different stopping rules. However, to apply this approach some practical issues have to be addressed. For example, how do we select an appropriate regeneration state? What can we do if the target simulation model does not have any regeneration state that occurs sufficiently frequently? Chapter 6 considers these questions and propose using FORS pilot simulation and PR simulation, respectively, to handle these problems. Experimental results of the PR simulation with some communication network models show that the PR approach outperforms straightforward MR-D simulation in accuracy. However, the major disadvantage of the PR approach is that the simulation completion time is a random variable.

8.1.5 A Study of Space-Parallel Simulation

Previous experimental work has suggested that lookahead plays an important role in the performance of conservative space-parallel simulation. In Chapter 7, we develop stochastic models to study the relationship between the amount of lookahead and the speedup of the simulation. We show that for open acyclic simulation models, if the simulation length is sufficiently long, the amount of lookahead is not influential to the performance of the simulation. For closed models, on other hand, the amount of lookahead is decisive to the speedup.

8.2 Future Research

It is clear that to achieve good parallel simulation performance, the parallel simulation algorithm must be carefully devised. In this research, we endeavor to provide parallel algorithms for simulation practitioners. However, the algorithms presented here (summarized in Table 8.1) by no means fully cover the range of possible models. Also, a number of assumptions are made by our algorithms. For example, in multiple replication simulation, the MR-PI approach assumes that the interested measure converges monotonically toward the steady-state. In the GG1K algorithm, independent service times are required to avoid possible bias. Efficient parallel simulation for the cases where the assumptions made by our algorithms do not hold requires further investigation. Also, in the SSM approach we choose the matching substate somehow heuristically. To enhance the applicability of the SSM algorithm, a systematic approach for the selection of the matching substate is desirable. This also leaves room for future study. Finally, in this dissertation, four methods are considered individually. A potential research direction for new algorithms is to exploit hybrid methods based on those considered here.

Table 8.1: A summary of the proposed algorithms, where PI, SSM, and PR represent polling initialization, parallel prefix, recurrent state, substate matching, and parallel regenerative, respectively. The term 'distributed system' refers to loosely coupled computers (e.g., workstations) interconnected by a communication network.

Algorithm	*Characteristics*
PI	**Method**: Multiple Replication **Application**: models exhibiting monotonic convergence **Limitations**: steady-state simulation only **Merits**: simplicity, generate IID observations, fixed completion times **Efficiency**: workload evenly distributed among processors **Suitable Architectures**: MIMD, distributed system
GG1K & IGG1K	**Method**: Time-Parallel **Application**: acyclic feed-forward networks of loss FCFS G/GI/1/K queues **Limitations**: require independent job service times, generate approximate results **Merits**: unbounded parallelism (for GG1K) **Efficiency**: workload evenly distributed among processors **Suitable Architectures**: MIMD, SIMD, distributed system
GD1K	**Method**: Time-Parallel **Application**: acyclic feed-forward networks of loss FCFS G/D/1/K queues **Limitations**: generate approximate results **Merits**: unbounded parallelism **Efficiency**: workload evenly distributed among processors **Suitable Architectures**: MIMD, SIMD, distributed system
SSM	**Method**: Time-Parallel **Application**: general **Limitations**: approximate results **Merits**: potential massive parallelism **Efficiency**: workload of each processor is a random variable **Suitable Architectures**: MIMD, distributed system
PR	**Method**: Parallel Regenerative **Application**: regenerative systems **Limitations**: steady-state simulation **Merits**: potential massive parallelism, generate (approximate) IID observations **Efficiency**: workload of each processor is a random variable **Suitable Architectures**: MIMD, distributed system

REFERENCES

[1] Abrams, M. The Object Library for Parallel Simulation (OLPS). *Proceedings of the 1988 Winter Simulation Conference* (Dec. 1988), 210-219.

[2] Abrams, M. A Model of TCP/IP Suitable for Parallel Simulation of Large Internets. Technical Report, TR 92-10, Computer Science Department, Virginia Tech (1992).

[3] S.G. Akl. *The Design and Analysis of Parallel Algorithms* (Prentice Hall, 1989).

[4] S. Andradóttir and T.J. Ott. Parallel Simulation of Communication Networks through Time Segmentation, working paper.

[5] Bain, W. L., Scott, D. S. An Algorithm for Time Synchronization in Distributed Discrete Event Simulation. *Proceedings of the SCS Multiconference on Distributed Simulation 19*, 3 (July 1988), 30-33.

[6] Bhat, U. N. *Elements of Applied Stochastic Process*, 2nd ed., Jhon Wiley & Sons, 82-98.

[7] Bhavsar V. C., Isaac J. R. Design and Analysis of Parallel Monte Carlo Algorithms. *SIAM J. Sci. Stat. Comput.* 8, (1987), s73-s95.

[8] Bertsekas, D., Gallager, R. *Data Networks.* 2nd ed. (1992), Prentice Hall.

[9] Bolot, J., Shankar, A. U. Analysis of a Fluid Approximation to Flow Control Dynamics, Technical Report 2553, Dept. of Computer Science, Univ. of Maryland (Oct. 1990).

[10] R. E. Bryant. Simulation of Packet Communication Architecture Computer Systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[11] Chandy, K. M., Misra, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Trans. on Softw. Eng. SE-5*, 5, (Sept. 1979), 440-452.

[12] Chandy, K. M., Misra, J. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Commun. ACM* 24, 11 (Nov. 1981), 198-205.

[13] Chandy, K.M., Sherman, R. Space-Time and Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation* (March 1989), 53-57.

[14] Chang, C., Heidelberger, P. Fast Simulation of Packet Loss Rates in a Shared Buffer Communications Switch. ICASE Report 93-79, ICASE, Hampton, VA (1993).

[15] Cleary, J., Gomes, F., Thudt, R. Cost of State Saving & Rollback, *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation* 94-101.

[16] Comer, R. B. *Internetworking with TCP/IP.* vol. 1 (1991), Prentice Hall.

[17] Cooper, R. B. *Introduction to Queueing Theory.* 2nd ed. (1981), North Holland.

[18] Corson, M. S. A Distributed, Object-Oriented Communication Network Simulation Testbed *Proceedings of the 1992 Winter Simulation Conference* (Dec. 1992), 672-679.

[19] Crane, M., A., Iglehart, D. L. Simulating Stable Stochastic Systems, II: Markov Chain. *J. Assoc. Comput. Mach*, 21, (1974), 114-123.

[20] Crane, M., A., Iglehart, D. L. Simulating Stable Stochastic Systems, III: Regenerative Process and Discrete-Event Simulations. *Operations Research*, 23, 1, (Jan. 1975), 33-45.

[21] Crane, M., A, Iglehart D. L. Simulating Stable Stochastic Systems, IV: Approximation Techniques. *Management Science*, 21, 11, (July 1975), 1215-1244.

[22] David, H. A. *Order Statistics*, 2nd ed., John Wiley, New York, 1981.

[23] Earnshaw R. W., Hind, A. A Parallel Simulator for Performance Modeling of Broadband Telecommunication Networks. *Proceedings of the 1992 Winter Simulation Conference* (Dec. 1992), 1365-1373.

[24] Fujimoto, R. M. Performance Measurements of Distributed Simulation Strategies. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation* (July 1988), 14-20.

[25] Fujimoto, R. M. Lookahead in Parallel Discrete Event Simulation. *Proceedings of International Conference on Parallel Processing.* St. Charles, IL (August 1988)

[26] Fujimoto, R. M. Time Warp on a Shared Memory Multiprocessor. *Trans. Soc. for Comput. Simul.*, 6(3), (July 1989), 211-239.

[27] Fujimoto, R. M. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (Oct. 1990), 31-53.

[28] Fujimoto, R. M. Parallel Discrete Event Simulation: Will The Field Survive? *Submitted for publication.*

[29] Gafni, A. Rollback Mechanisms for Optimistic Distributed Simulation Systems. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation, 19* (July 1988), 61-67.

[30] Gaujal, B., Greenburg, A. G., Nicol, D. M. A Sweep Algorithm for Massively Parallel Simulation of Circuit-Switched Networks. *Submitted for publication*

[31] Glynn, P., Heidelberger, P. Analysis of Parallel Replicated Simulations Under a Completion Time Constraint. *ACM TOMACS.* 1(1), (Jan. 1991), 3-23.

[32] Glynn, P., Heidelberger, P. Analysis of Initial Transient Deletion for Parallel Steady-State Simulations. *SIAM J. Stat. Comput.* 14(4), (July 1992), 904-922.

[33] Gordon, Gordon, W. L., Newell, G. F. Closed Queueing Systems with Exponential Servers. *Opns. Res*, Vol. 15, No. 2, 254-265

[34] Greenberg, A. G., Lubachevsky, B. D., Mitrani, I. Unboundedly Parallel Simulations via Recurrence Relations. *Proceedings of the Conference on Measurement and Modeling of Computer Systems.* Boulder, Colorado, (May 1990), 1-12.

[35] Gunther, F., L., Wolff, R. W. The Almost Regenerative Method for Stochastic System Simulation. ORC 75-21, University of California, Berkeley, 1975.

[36] Heidelberger, P. A Renewal Theoretic Approach to Bias Reduction In Regenerative Simulations. *Management Science*, 28, 2, (Feb. 1982), 173-181.

[37] Heidelberger, P. Statistical Analysis of Parallel Simulations. *Proceedings of the 1986 Winter Simulation Conference* (Dec. 1986), 290-295.

[38] Heidelberger, P. Discrete Event Simulations and Parallel Processing: Statistical Properties. *SIAM J. Stat. Comput.* 9(6), (Nov. 1988), 1114-1132.

[39] Heidelberger, P., Stone H. S. Parallel Trace-Driven Cache Simulation by Time Partitioning. *Proceedings of the 1990 Winter Simulation Conference* (Dec. 1990), 734-737.

[40] Heidelberger, P. Parallel Trace-Driven Cache Simulation by Time Partitioning. *Proceedings of the 1990 Winter Simulation Conference* (Dec. 1990), 734-737.

[41] Ho, Y. C., Cao, X. R. *Perturbation analysis of discrete event dynamic systems.* Kluwer Academic Publishers, 1991.

[42] Jackson J. R. "Jobshop-like Queueing Systems". *Management Science*, 10, (1963), 449-460.

[43] Jefferson, D. R., Sowizral H. Fast Concurrent Simulation using the Time Warp Mechanism, Part I: Local Control. Technical Report N-1906-AF, RAND Corporation, (Dec. 1982).

[44] Jefferson, D. Virtual Time. *ACM Transactions of Programming Languages and Systems* 7, 3 (July 1985), 404-425.

[45] Jones, D. W., Chou C., Renk, D., Bruell S. C. Experience with Concurrent Simulation *Proceedings of the 1989 Winter Simulation Conference* (Dec. 1989), 756-764.

[46] Jow Y. L. An Approximation Method for Tandem Queues with Blocking *Opns. Res*, 36, 1, 73-83.

[47] Kaudel, F. J. A literature Survey on Distributed Discrete Event Simulation. *Simuletter* 18, 2 (June 1987), 11-21.

[48] Kelton, W. D., Law, A. M. A New Approach for Dealing with the Startup Problems in Discrete Event Simulation. Naval Res. Logist. Quart., 30,(1983) 641-658.

[49] Kelton, W. D. Random Initialization Methods in Simulation. *IIE Trans*, 21, (1989), 335-367.

[50] Kleijnen, L. *Statistical Techniques in Simulation.* Vol. 1,2 (1974), Marcel Dekker, Inc.

[51] Kleinrock, L. *Queueing Systems.* Vol. 1,2 (1975), Wiley-Interscience.

[52] Konheim, A. G., Reiser M. Finite Capacity Queueing Systems with Applications in Computer Modeling. *SIAM J. Comput.* 7, 210-229.

[53] Law, A. M., Carson, J. S. A Sequential Procedure for Determining the Length of a Steady-State Simulation. *Operations Research*, 27, (1979), 1011-1025.

[54] Law, A. M., Kelton, W. D. *Simulation Modeling & Analysis.* (1991), McGraw Hill, 2nd Edition.

[55] Lander, R. E., Fischer, M. J. Parallel Prefix Computation, . *J. of the ACM*, 27, (1980), 831-838.

[56] Leung, E., Cleary, J., Lomow, G., Baezner, D., Unger, B. The Effect of Feedback on the performance of conservative algorithms. *Proceedings of 1989 SCS Multiconference on in Distributed Simulation*, 44-49.

[57] Lin, Y. B. Understanding the Limits of Optimistic and Conservative Parallel Simulation. Technical Report 90-08-02, Department of Computer Science and Engineering, University of Washington, 1990.

[58] Lin, Y. B., Lazowska, E. A. Time-Division Algorithm for Parallel Simulation. *ACM TOMACS* 1, 1 (Jan. 1991), 73-83.

[59] Lin, Y. B. Parallel Trace-Driven Simulation for Packet Loss in Finite-Buffered Voice Multiplexers. to appear in *Parallel Computing.*

[60] Lin Y. B., Preiss, B. R., Loucks, W. M., Lazowska E. D. Selecting the Checkpoint Interval in Time Warp Simulation. *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation* 3-10.

[61] Liu, L. Z., Tropper, C. Local Deadlock Detection in Distributed Simulations. *Proceedings of 1990 SCS Multiconference on Advances in Distributed Simulation 22*, 1, (Jan 1990), 64-69.

[62] Madisetti V., Walrand. J., Messerschmitt, D. Wolf: A Rollback Algorithm for Optimistic Distributed Simulation Systems. *Proceeding of the 1988 Winter Simulation Conference.* (Dec. 1988), 296-305.

[63] Miller, R. G., The Jackknife-A Review. *Biometrika*, 61, (1974), 1-15.

[64] Misra, D. Distributed-Discrete Event Simulation. *ACM Comput. Surv. 18*, 1, (Mar. 1986), 39-65.

[65] D. Mitra and I. Mitrani. Control and Coordination Policies for System with Buffers. ACM SIGMETRICS Performance Evaluation Review 17(1989)156-164.

[66] Nance, R. The Time and State Relationships in Simulation Modeling. *CACM*, (April 1981), 173-180.

[67] Nikolaidis I., Fujimoto R. Parallel Simulation of High-Speed Network Multiplexers. Unpublished draft. (Oct. 1992)

[68] Nicol, D. M., Reynolds, P. F., Jr. Problem Oriented Protocol Design. *Proceeding of the 1984 Winter Simulation Conference.* (Dec. 1984), 471-474.

[69] Nicol, D. M. Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks. *SIGPLAN Not.* 23, 9 (Sept. 88), 124-137

[70] Nicol, D. M. Performance Bounds on Parallel Self-initiating Discrete-Event Simulations. *ACM TOMACS* 1, 1 (Jan. 1991), 24-50.

[71] Nicol, D. M., Greenberg, A. MIMD Parallel Simulation of Circuit Communication Networks. *Proceeding of the 1992 Winter Simulation Conference.* (Dec. 1992), 629-635.

[72] Nicol, D. M. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. *Journal of the Association for Computing Machinery*, Vol. 40, No. 2, 304-333, April 1993.

[73] Peacock, J. K., Wong, J. W., Manning Concurrent Discrete-Event Simulation Tools. *IEEE Journal of Selected Areas in Communications.* 9, 3, (April 1991), 477-485.

[74] Phillips, C. I., Cuthbert, L. G. Concurrent Discrete-Event Simulation Tools. *IEEE Journal of Selected Areas in Communications.* 9, 3, (April 1991), 477-485.

[75] Pool, R. Massively Parallel Machines. *Technology Edge.* (Oct. 1992), 13-15.

[76] Prasad S., Deo N. An Efficient and Scalable Parallel Simulation for Discrete-Event Simulation *Proceeding of the 1991 Winter Simulation Conference.* (Dec. 1991), 652-658.

[77] Reed, D. A., Malony, A. D., McCredie, B. D. Parallel Discrete Event Simulation Using Shared Memory. *IEEE Transaction on Software Engineering* Vol. 14, No. 4, (Apr. 1988), 541-553.

[78] Reiher, P., Jefferson D. Limitation of Optimism in the Time Warp Operating System *Proceeding of the 1989 Winter Simulation Conference*, (Dec. 1989), 765-770.

[79] Schwarz, M. *Telecommunication Networks.* (1987), Addison Wesley.

[80] Singh S., Agrawala A. K., Keshav S. Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits, Technical Report 2490, Dept. of Computer Science, Univ. of Maryland (June 1990).

[81] Sokol, L. M., Briscoe, D. P., Wieland, A. P. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. *Proceedings of 1988 SCS Multiconference on Distributed Simulation. 19*, 3 (July 1988), 34-42.

[82] H.S. Stone. *High Performance Computer Architecture* (Addison Wesley, 1987).

[83] Su, W. K., Seitz, C. L. Variants of the Chandy-Misra-Bryant Distributed Discrete Event Simulation Algorithm. *Proceedings of 1989 SCS Multiconference on Advances in Parallel and Distributed Simulation. 21*, 2, (Mar. 1989), 79-84.

[84] Tallieu F., Verboven F. Using Time Warp for Computer Network Simulations on Transputers. *IEEE 1991*, 112-117.

[85] Verbiest, W., Pinnoo, L., and Voeten, B. The Impact of the ATM concept on Video Coders. *IEEE Journal on Selected Areas in Communications*, 6(9), (Dec. 1988), 1623-1631.

[86] Wagner, D. B., Lazowska, E. D. Parallel Simulation of Queueing Network: Limitation and Potentials. *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE* (May 1989), 146-155.

[87] Wagner, D. B., Lazowska, E. D., Bershad B. N. Techniques for Efficient Shared-Memory Parallel Simulation. *Distributed Simulation 1989.* Society for Computer Simulation International, San Diego, CA (March 1989)

[88] Wang, J., Abrams, M. Approximate Time-Parallel Simulation of Queueing Systems with Losses. *Proceedings of the 1992 Winter Simulation Conference* (Dec. 1992), 700-708.

[89] Wang, J., Abrams, M. Determining Initial States of Time-Parallel Simulations. *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation* (May 1993), 19-26.

[90] Wang, J., Abrams, M. Massively Time-Parallel, Approximate Simulation of Loss Queueing Systems. *Annals of the Operations Research,* Vol. 53, (1994), 553-576.

[91] Wang, J., Abrams, M. Massively Parallel Simulation with Application to Networks. *submitted to Parallel Computing.*

[92] Wang, J., Abrams, M. The Impact of Lookahead on Conservative Simulation. *submitted to the International Journal of Computer Simulation.*

[93] West, D. Optimizing Time Warp: Lazy Rollback and Lazy Re-evaluation. *M.S. Thesis, University of Calgary*, Jan. 1988.

Appendix A
PROOFS

A.1 Proof of Lemma 1

The correctness of (5.11) can be proven by showing that for any pair of integers a and a', for $1 \leq a \leq a' \leq 2N$, given $\mathbf{L}_a^K$ the following equation always holds:

$$\Delta\mathbf{L}_{a'}^K = \max\{\min\{\Delta\mathbf{L}_a^K, \overline{\Delta\mathbf{L}}_{a,a'}^K\}, \underline{\Delta\mathbf{L}}_{a,a'}^K\}. \tag{A.1}$$

The proof uses induction on a'. It is clear that equation (A.1) holds (induction base) for $a = a'$ for any $1 \leq a \leq 2N$ because $\underline{\Delta\mathbf{L}}_{a,a}^K \leq \Delta\mathbf{L}_a^K \leq \overline{\Delta\mathbf{L}}_{a,a}^K$. Assume that equation (A.1) holds for some integer y satisfying $a < [a' = y]$ (induction hypothesis). Then:

$$\Delta\mathbf{L}_y^K = \max\{\min\{\Delta\mathbf{L}_a^K, \overline{\Delta\mathbf{L}}_{a,y}^K\}, \underline{\Delta\mathbf{L}}_{a,y}^K\}. \tag{A.2}$$

The proof completes if we show that (A.1) holds for $a' = y + 1$. Consider event $y + 1$. There are four possible cases:

Case 1: Event $y + 1$ is an arrival and $\mathbf{L}_y^K < K$

In this case, $L_{y+1}^\infty = L_y^\infty + 1$. From A1, $\mathbf{L}_{y+1}^K$=$\mathbf{L}_y^K$+1 and hence $\Delta\mathbf{L}_{y+1}^K$=$\Delta\mathbf{L}_y^K$. Because $\mathbf{L}_y^K < K$, from (5.6) and (5.5), we have $\overline{\Delta\mathbf{L}}_{a,y}^K \geq \Delta\mathbf{L}_y^K \geq L_y^\infty - K + 1 = L_{y+1}^\infty - K$. Thus, $\overline{\Delta\mathbf{L}}_{a,y}^K \geq \Delta\mathbf{L}_y^K \geq L_{y+1}^\infty - K$. Applying (5.10) yields:

$$\overline{\Delta\mathbf{L}}_{a,y+1}^K = \overline{\Delta\mathbf{L}}_{a,y}^K. \tag{A.3}$$

Because event $y + 1$ is an arrival, from (5.9), we have:

$$\underline{\Delta\mathbf{L}}_{a,y+1}^K \geq \underline{\Delta\mathbf{L}}_{a,y}^K. \tag{A.4}$$

Now consider ΔL_a. There are three possible situations:

1. $\Delta L_a < \Delta \mathbf{L}_y^K$: From (5.6) and (A.2), $\underline{\Delta\mathbf{L}}_{a,y}^K = \Delta\mathbf{L}_y^K$. Because $\underline{\Delta\mathbf{L}}_{a,y+1}^K \leq \Delta\mathbf{L}_{y+1}^K$, from (A.4), $\underline{\Delta\mathbf{L}}_{a,y+1}^K = \underline{\Delta\mathbf{L}}_{a,y}^K$. Thus, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \underline{\Delta\mathbf{L}}_{a,y+1}^K =$ dLiKy.

2. $\Delta L_a > \Delta \mathbf{L}_y^K$: From (5.6) and (A.2), $\overline{\Delta\mathbf{L}}_{a,y}^K = \Delta\mathbf{L}_y^K$. From (A.3), $\overline{\Delta\mathbf{L}}_{a,y+1}^K = \overline{\Delta\mathbf{L}}_{a,y}^K$. Thus, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \overline{\Delta\mathbf{L}}_{a,y+1}^K = \Delta\mathbf{L}_y^K$.

3. $\Delta L_a = \Delta \mathbf{L}_y^K$: From (A.3) and (A.2), $\underline{\Delta\mathbf{L}}_{a,y}^K \leq \Delta\mathbf{L}_y^K \leq \overline{\Delta\mathbf{L}}_{a,y}^K$. But because $\overline{\Delta\mathbf{L}}_{a,y}^K = \overline{\Delta\mathbf{L}}_{a,y+1}^K$ (A.3) and $\underline{\Delta\mathbf{L}}_{a,y+1}^K \leq \Delta\mathbf{L}_y^K$, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \Delta\mathbf{L}_y^K$.

Case 2: Event $y+1$ is an arrival and $\mathbf{L}_y^K = K$

In this case, $L_{y+1}^\infty = L_y^\infty + 1$. From A1, $\mathbf{L}_{y+1}^K = \mathbf{L}_y^K = K$. Thus, $\Delta\mathbf{L}_{y+1}^K = \Delta\mathbf{L}_y^K + 1 = L_{y+1}^\infty - K$. Now consider ΔL_a. There are three possible situations:

1. $\Delta L_a < \Delta \mathbf{L}_y^K$: From (5.6) and (A.2), $\underline{\Delta\mathbf{L}}_{a,y}^K = \Delta\mathbf{L}_y^K = L_y^\infty - K$.

From (5.9), $\underline{\Delta\mathbf{L}}_{a,y+1}^K = L_y^\infty + 1 - K = \underline{\Delta\mathbf{L}}_{a,y}^K + 1$. Thus, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \underline{\Delta\mathbf{L}}_{a,y+1}^K = \underline{\Delta\mathbf{L}}_{a,y}^K + 1 = \Delta\mathbf{L}_y^K + 1$.

2. $\Delta L_a > \Delta \mathbf{L}_y^K$: From (5.6) and (A.2), $\overline{\Delta\mathbf{L}}_{a,y}^K = \Delta\mathbf{L}_y^K = L_y^\infty - K$. From (5.10), $\overline{\Delta\mathbf{L}}_{a,y+1}^K = L_y^\infty + 1 - K = \overline{\Delta\mathbf{L}}_{a,y}^K + 1$. Thus, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \overline{\Delta\mathbf{L}}_{a,y+1}^K = \overline{\Delta\mathbf{L}}_{a,y}^K + 1 = \Delta\mathbf{L}_y^K + 1$.

3. $\Delta L_a = \Delta \mathbf{L}_y^K$: From (5.6) and (A.2), $\underline{\Delta\mathbf{L}}_{a,y}^K \leq \Delta\mathbf{L}_y^K = L_y^\infty - K$. From (5.9), $\underline{\Delta\mathbf{L}}_{a,y+1}^K = L_y^\infty + 1 - K$. Thus, (A.1) yields $\Delta\mathbf{L}_{y+1}^K = \underline{\Delta\mathbf{L}}_{a,y+1}^K = L_y^\infty + 1 - K = \Delta\mathbf{L}_y^K + 1$.

An analogous argument holds for case 3 (event $y+1$ is a departure and $\Delta L_y^K > 0$) and case 4 (event $y+1$ is a departure and $\mathbf{L}_y^K = 0$). $\square$

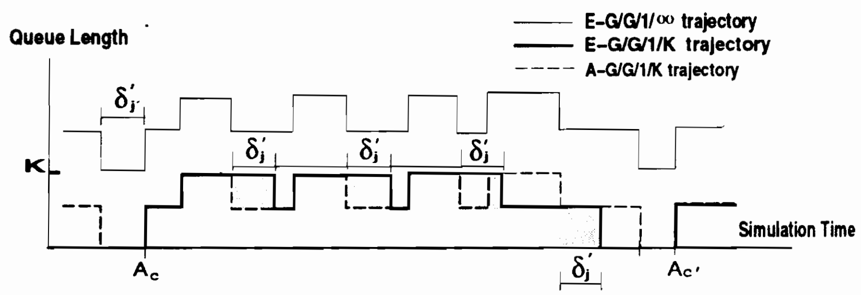


Fig. A.1: Sample trajectories for E-G/G/1/∞, E-G/G/1/K, and A-G/G/1/K que ues.

A.2 Proof of Lemma 2

Let δ denote the constant job service time. Let $\hat{\delta}(t)$ denote the remaining service time of the first job in the E-G/G/1/K queue at simulation time t. That is, $\hat{\delta}(t)$ is the amount of time between t and the next departure time. Define $R(t)$ to be the *total remaining service time* of all jobs in the E-G/G/1/K queue at simulation time t. Formally,

$$R(t) = \begin{cases} \hat{\delta}(t) + (L(t) - 1)\delta & L(t) \geq 1, \\ 0 & L(t) = 0. \end{cases}$$

Analogously, define $\mathbf{R}^K(t)$ to be the total remaining service time of all jobs in the A-G/G/1/K queue at simulation time t. Let $\mathbf{L}^K(t)$ denote the A-G/G/1/K queue length at simulation time t. Then the following corollary holds:

Corollary 4 *$R(t) \geq \mathbf{R}^K(t) \Rightarrow L(t) \geq \mathbf{L}^K(t)$; $R(t) \leq \mathbf{R}^K(t) \Rightarrow L(t) \leq \mathbf{L}^K(t)$.*

A proof of Corollary 4 follows directly from the fact that the service times are constant.

Suppose a case-3 occurs at A_j (hence $\mathbf{L}^K(A_j^-)$=0). Assume $L(A_j^-) = 0$. Based on the GG1K transformation technique (section 5.2.1.3), the GG1K algorithm will *underestimate* the service time of job j by some amount δ', where $0 < \delta' < \delta$ (as illustrated in fig. A.1).

Then $R(A_j^+) = \mathbf{R}^K(A_j^+) + \delta_j'$.

Corollary 5 *For all t in the time interval $(A_c, A_{c'})$, exactly one of the following situations must be true:*

S1: $0 \leq R(t) - \mathbf{R}^K(t) < \delta$ *and* $\Delta\theta^K_{A_c,t}$*=0.*

S2: $0 < \mathbf{R}^K(t) - R(t) < \delta$ *and* $\Delta\theta^K_{A_c,t}$*=1.*

Corollary 5 can be proved by induction on t. Because $\Delta\theta^K_{A_c^+,A_c^+}=0$ and $R(A_c^+) > \mathbf{R}^K(A_c^+)$, S1 holds at A_c^+ (induction base). Assume that an event occurs at some time t, $A_c^+ < t < A_{c'}^-$, and S1 or S2 is true at t^+ (induction hypothesis). Now, consider time t', $t < t' < A_{c'}$, when the next event occurs. Then this event can be an arrival (to both the E-G/G/1/K and the A-G/G/1/K queues) or a departure (to either or both the E-G/G/1/K and the A-G/G/1/K queues). For the event that occurs at t', there are four cases:

Case 1: S1 holds at t^+ and an arrival occurs (to both queues) at t':

S1 holds in $[t^+, t']$. From Corollary 4, if the job that arrives at t' is lost in the E-G/G/1/K queue and is not lost in the A-G/G/1/K queue, then S2 holds at t'^+. Otherwise, S1 holds at t'^+.

Case 2: S1 holds at t^+ and a departure occurs (to either queue) at t':

From Corollary 4, S1 holds in $[t^+, t'^+]$.

Case 3: S2 holds at t^+ and an arrival occurs (to both queues) at t':

S2 holds in $[t^+, t']$. From Corollary 4, if the job that arrives at t' is lost in the A-G/G/1/K queue and is not lost in the E-G/G/1/K queue, then S1 holds at t'^+. Otherwise, S2 holds at t'^+.

Case 4: S2 holds at t^+ and a departure occurs (to either queue) at t':

From Corollary 4, S2 holds in $[t^+, t'^+]$.

Therefore, Corollary 5 holds. Note that because both queues are empty at A_c^- and $A_{c'}^-$, the jobs arrive at A_c and $A_{c'}$ are not lost. Therefore, Lemma 2 follows directly from Corollary 5. □.

A.3 Proof of Lemma 4

Proof: A proof can be carried out by induction on departure events. Suppose event j_0 is the first departure event in the corresponding E-G/D/1/∞ trajectory. It is clear that the $\mathbf{L}_{j_0}^K = L(E_{j_0}^{\infty +})$ (induction base). Let event j and j', $j < j'$, be any two consecutive departure events in the E-G/D/1/∞ trajectory. Assume that $L(E_j^{\infty +}) - 1 \leq \mathbf{L}_j^K \leq L(E_j^{\infty +})$ holds for some j, $1 \leq j < N$ (induction hypothesis). The proof completes if we show that $L(E_{j'}^{\infty +}) - 1 \leq \mathbf{L}_{j'}^K \leq L(E_{j'}^{\infty +})$ holds. Let n_j denote the number of jobs that arrive in the simulation time interval $(E_j, E_{j'}]$. If $n_j = 0$, clearly, $L(E_{j'}^{\infty +}) - 1 \leq \mathbf{L}_{j'}^K \leq L(E_{j'}^{\infty +})$. If $n_j > 0$, there are two cases:

Case 1: $\mathbf{L}_j^K = L(E_j^{\infty +})$

If $n_j + L(E_j^{\infty +}) \leq K$, no job will be lost in $(E_j, E_{j'}]$ and $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +})$. If $n_j + L(E_j^{\infty +}) > K$, at least one job will be lost. Let d_j and a_j denote the departure delay for event j and the last job arrival time in $(E_j, E_{j'}]$, respectively. Then if $d_j = 0$, obviously, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +})$. Otherwise, if $E_j^{\infty} + d_j > a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +})$. On the other hand, if $E_j^{\infty} + d_j < a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +}) - 1$.

Case 2: $\mathbf{L}_j^K = L(E_j^{\infty +}) - 1$

If $n_j + L(E_j^{\infty +}) \leq K$, no job will be lost in $[E_j, E_{j'})$ and $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +}) - 1$. Otherwise, at least one job will be lost . If $E_j^{\infty} + d_j > a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +})$. On the other hand, if $E_j^{\infty} + d_j < a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty +}) - 1$.

$\square$

VITA

Jain-Chung J. Wang was borned in Nanto, a small town in central Taiwan, Republic of China, on 24 January 1964. He received his engineering diploma in civil engineering from National Taipei Institute of Technology, Taipei Taiwan, in 1983. After completing his military service as a platoon leader, he started his graduate study in computer science in 1986 and subsequently received an M.S. degree from the State University of New York in 1988. Since 1989, he has been a graduate student in the Department of Computer Science at Virginia Polytechnic Institute and State University.

Mr. Wang is a member of Sigma Xi, Association for Computing Machinery, and the special interest group on Simulation. Mr. Wang has served as a referee in the area of parallel discrete event simulation for IEEE Transactions on Parallel and Distributed Systems, 7th, 8th, 9th Workshop of Parallel and Distributed Simulation, and the Third International Symposium on High-Performance Distributed Computing. His research interests include parallel discrete event simulation, system modeling and performance analysis, and communication network systems.

[illegible]