ANALYSIS OF THE AGGRESSIVE GLOBAL WINDOWING ALGORITHM

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Phillip Mathew Dickens

January 1993

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

The sheer magnitude of many discrete event simulation problems makes the use of conventional sequential techniques impractical. For this reason researchers have turned to the use of multiprocessor architectures to provide the power necessary to simulate these large systems. Unfortunately, after fifteen years of research there is very little analytic work that allows us to understand under what conditions certain approaches to parallel discrete event simulation (PDES) perform well - or not so well.

Recently, there has been much interest in the parallel simulation community in a new class of synchronization protocols which blend aspects of existing aggressive and non-aggressive mechanisms. The goal of this research is to maximize the advantages, and minimize the disadvantages, of each approach. Preliminary studies have suggested this approach to synchronization offers the potential for excellent performance.

This dissertation significantly broadens and extends the body of analytic results established in PDES by providing the first in-depth analysis of the costs/benefits of this new approach. Our research demonstrates this type of approach offers a significant increase in the level of parallelism over non-aggressive approaches. Also, we have developed a set of theoretical results which represent a significant step towards proving system-level scalability. Further, the results of our simulation studies closely match the predictions of our analytic model, even when the simulated application does not meet some of the major assumptions of our model. We discuss these ideas more fully after providing the context for this research.

1.1. Background and Motivation

In a parallel discrete event simulation (PDES) the physical system is partitioned into a set of physical processes (PP) and each PP is modeled by a corresponding logical process (LP). Logical processes communicate through the use of timestamped messages where each message represents a change to the

state of the system being simulated. The timestamp of the message represents the time the system state is changed.

Parallel discrete event simulation poses very difficult synchronization problems due to the underlying sense of logical time. Each LP maintains its own logical clock representing the time up to which the corresponding PP has been simulated. The fundamental synchronization problem in PDES is in the determination of when it is permissible for an LP to advance its logical clock. If an LP advances its logical clock too far ahead of any other LP in the system it may receive a message with a timestamp in its logical past. When an LP receives a message with a timestamp in its logical past it is termed a *causality error* or *fault* and may lead to incorrect results.

Most of the synchronization protocols developed for parallel discrete event simulation fall into two basic categories. One category (using the terminology developed by Reynolds 1988) is protocols that are *accurate, non-aggressive* and *without risk* (also known as "conservative" protocols, e.g. Chandy and Misra 1979, Lubachevsky 1988, Nicol 1993 and Peacock, Manning and Wong 1978). The second category is protocols that are *accurate, aggressive* and *with risk,* also known as "optimistic", e.g. Time Warp (Jefferson 1985). Protocols that are non-aggressive and without risk do not allow an LP to process a message with timestamp t if it is possible that it will receive another message with a timestamp less than t at some point in the future. Protocols that are aggressive allow an LP to process any event it receives, and any causality errors that result from this aggressive processing are corrected through a *rollback* mechanism.

There has been much debate in the literature about which is the best approach for parallel simulation. Empirical results suggest both approaches perform well in certain situations. Non-aggressive approaches tend to perform well when information about the simulation problem can be exploited by the protocol. Aggressive approaches have been shown to perform well when state saving costs are low and/or there is hardware support for saving state.

Both approaches have inherent problems however. Non-aggressive protocols are criticized for not allowing a computation to proceed because there exits the *possibility* of a causality error. Thus computations that can possibly result in an error, but generally do not, will not be allowed to proceed. This tends

to leave processors idle due to the overly pessimistic synchronization constraints. Aggressive protocols are criticized for the high overhead costs associated with state saving and rollback, and because of the possibility of cascading rollbacks, a situation where one rollback causes a chain of rollbacks and the number of participants increases without bounds (Lubachevsky *et al.* 1989). Thus aggressive processing can lead to thrashing due to an overly optimistic view of synchronization requirements.

Throughout this debate there have been few theoretical results to help predict, explain or bound the behavior of either approach. Many of the models involving Time Warp assume a two processor system (e.g. Lavenburg, Muntz and Samadi 1983, Mitra and Mitrani 1984, Felderman and Kleinrock 1991, Felderman and Kleinrock 1992). Other models assume the cost of saving state and rollback are free (e.g. Felderman and Kleinrock 1991a, Gupta, Akylidiz and Fujimoto 1991, Lin and Lazowska 1990). While these results give some useful insights into the performance of aggressive protocols, they are clearly limited by the size of the system they model and/or the lack of overhead costs.

Most of the analyses of non-aggressive approaches have been of synchronous protocols (e.g. Nicol 1993, Lubachevshy 1988, Lubachevsky 1989a). These studies have established the very important property that this type of protocol scales well as the size of the problem and the architecture grow. It is important to note that these are the only studies which have shown important scalability results *when the cost of the synchronization mechanism is included in the model.* We discuss this important point in detail below. Other studies of non-aggressive approaches include a two processor model (Felderman and Kleinrock 1992), and a study of non-aggressive protocols for systems with no *lookahead*, which is the ability of a logical process to predict its future behavior (Lin *et al.* 1990a

Also, there have been some analytic studies which compare the two basic approaches. Many of these studies do not include the costs of either approach (e.g. Lin and Lazowska 1990), include only some of the costs (e.g. Mizell and Lipton 1990), or are restricted to a two processor system (Felderman and Kleinrock 1992). Nicol (1991) gives the only analysis which includes most of the overhead costs of each approach. This analysis however is in the context of a *self-initiating* model where a logical process schedules its own times to re-evaluate its state rather than doing so in response to the receipt of an event from some other logical process in the system.

Given the inherent limitations of both approaches, researchers have begun to investigate other possible approaches for the synchronization of parallel simulation. Reynolds (1988) was the first to demonstrate that the aggressive/non-aggressive dichotomy is too restrictive a view of parallel simulation. Rather, there exists a spectrum of options which includes, but is not limited to, these two approaches. More recently, researchers have investigated the benefits of adding aggressiveness to existing non-aggressive protocols in order to maximize the advantages, and minimize the disadvantages of each approach (Dickens *et al.* 1992, Turner and Xu 1992, Steinman 1992, Madisetti *et al.* 1992, Dickens and Reynolds 1991, Dickens and Reynolds 1990, Lubachevsky *et al.* 1989). This research is focused on increasing the amount of parallelism of a non-aggressive protocol, and decreasing the thrashing of an aggressive protocol.

One of the primary motivations of this research is to decrease the thrashing behavior of a fully aggressive protocol. Thus one important goal of this research is to theoretically demonstrate that cascading rollbacks will not develop in a system with limited aggressiveness. A related issue is the relationship between the level of aggressiveness and the probability of a rollback. Another motivation for this research is the need to increase the amount of parallelism available in a non-aggressive protocol. Thus an important issue is the extra parallelism made available as a function of the level of aggressiveness. Perhaps one of the most important feature of any synchronization mechanism is how well the protocol scales as the size of the application and the architecture increase.

Some of these issues have begun to be addressed in the literature, but a number of important issues have not yet been addressed and are the focus of this dissertation. We define a new protocol, the Aggressive Global Windowing Algorithm, which adds aggressiveness to an existing class of non-aggressive protocols. The class of protocols chosen for this new approach is known as the Global Windowing Algorithms, and this class is chosen because it is the only one for which important scalability results have been established (Nicol 1993, Lubachevsky 1989, Lubachevsky 1989a).

We define a simple mechanism, *the aggressive window*, to control the level of aggressiveness exhibited by the system. Then we develop an analytic model to predict the increase in parallelism *as a function of the level of aggressiveness.* Further, we develop our model to predict the probability of a causality

error also as a function of the level of aggressiveness. This is the first time the relationship between the level of aggressiveness and the expected improvement in parallelism, and the increase in the probability of a causality error, has been established.

This research gives theoretical and simulation results which strongly suggest that our aggressive algorithm maintains the important scalability properties of the non-aggressive protocol. We demonstrate that the probability of a causality error at a given LP does not increase *as the number of LPs approaches infinity*. Further, we show that the probability of a given LP starting a rollback chain does not increase as the number of LPs approaches infinity. Even though these results apply only to a given LP, we believe they represent a significant step towards proving system-level scalability.

The research presented here is one of only two studies which compares the performance of two protocols and includes the major costs *of each approach* (the other being Nicol 1991). As discussed, the major cost of an aggressive protocol is saving state and the potential for cascading rollbacks. A major cost of a synchronous algorithm such as a Global Windowing Algorithm is the cost of global synchronization. In our model we derive the expected improvement in performance *as a function of the cost of saving state and the cost of global synchronization.* Thus our model studies the improvement in performance as a function of the costs of each approach *and* the level of aggressiveness. Overall, the research presented in this dissertation makes a significant contribution to the small but growing set of analytic results for parallel simulation.

1.2. Dissertation Organization

In this dissertation we develop an analytic model to examine the costs and benefits of adding aggressiveness to an existing non-aggressive synchronization mechanism for parallel discrete event simulation. We now describe the major contribution of each of the remaining chapters.

Chapter 2:

In this chapter we build the foundation for our work by developing the model for parallel discrete event simulation. We describe the problems encountered in moving from a sequential to a parallel simulation, and discuss the various approaches to solving these problems. Finally, we give a detailed review

of the major empirical and analytic results accomplished thus far in order to put our work into perspective.

Chapter 3:

This chapter lays the groundwork for all of our subsequent analyses. We define our approach of adding aggressiveness to an existing non-aggressive protocol through a mechanism we term the *aggressive window*. We develop a preliminary analytic model under the assumption of a closed queueing system which is heavily loaded. We use this model to investigate the probability of a causality error (at a given LP), and the *upper bound* on the expected improvement in performance, as a function of the level of aggressiveness. Also, we prove that the probability of a causality error at a given LP does not increase *as the number of LPs approaches infinity.*

The results given in this chapter are preliminary in that we do not define a mechanism to correct causality errors that occur as a result of aggressive processing. For this reason we do not include the cost of such a mechanism in our model, and the results therefore represent an upper bound on the potential improvement.

Chapter 4:

In this chapter we define a simple state saving and rollback mechanism to correct the causality errors that occur as a result of aggressive processing. We discuss one component of this correction mechanism, the *anti-message*, which is a necessary ingredient in a rollback chain. We extend our model to predict the probability that a given LP will produce an anti-message *as a function of the level of aggressiveness.* We then demonstrate that the probability of a given LP producing an anti-message does not increase as the number of LPs approaches infinity. Finally, we extend our model to predict the probability a given LP produces an *Nth* generation anti-message as a function of the level of aggressiveness.

Chapter 5:

In this chapter we relax our assumption of a closed queueing system that is heavily loaded, and extend our model to include an open system which can be lightly loaded. The analysis presented in this chapter is important for two reasons. First, it allows us to extend our analysis to a lightly loaded system.

Second, it demonstrates how quickly our analysis becomes intractable as we move away from a system that allows us to make many simplifying assumptions.

Chapter 6:

The analysis in all of the preceding chapters examines the behavior of a "typical" LP in a system using our synchronization mechanism. In this chapter we extend our analysis to investigate *system level performance* rather than the behavior of a "typical" LP. We define the processing cost of the two approaches as the number of messages that must be processed in order to complete one unit of logical time. Note that this processing cost includes the cost of global synchronization and the cost of saving state. Our model makes a set of best case assumptions for the non-aggressive version of the protocol, and a set of pessimistic assumptions for our protocol which permits limited aggressive processing. Also, we show the expected improvement in performance as a function of the cost of global synchronization, the cost of saving state and the level of aggressiveness permitted by our protocol. Finally, we investigate the expected improvement in performance given communication topologies other than the one assumed in our model.

Chapter 7:

In this chapter we give our conclusions and our thoughts for future research.

CHAPTER 2

Background

In order to establish a better understanding of the context of our research we provide a detailed description of the issues encountered in parallel simulation, and the mechanisms developed to deal with these issues. We begin this review by describing the model of parallel simulation. Then we discuss the issues which make parallel discrete event simulation such a difficult problem, followed by a review of the synchronization mechanisms developed to solve these problems. Where available, we discuss the empirical studies which investigate the performance of these various protocols.

We follow the review of synchronization mechanisms with a discussion of the theoretical results which have been developed for parallel simulation. As will be seen, most of the earlier analytic studies of a given protocol did not include the primary costs of the synchronization mechanism. Many other models are limited to two processor systems. More recently however there have been some theoretical results which include the cost of the protocol being studied. There have been no studies however that deal with the issue in which we are interested: the expected increase in parallelism, and the probability of a causality error, as a function of the level of aggressiveness. We begin our review with a description of the model for parallel discrete event simulation.

2.1. Simulation Model

The following model of a simulation is given by Misra (1986). A physical system to be simulated is modeled as a set of communicating physical processes that interact with each other through messages. Each physical process (PP) represents some component of the physical system. The messages exchanged between the PP's represent changes to the state of the system, and each message contains a timestamp that represents the time the change to the state of the system occurs. These messages that represent such changes are called *events*. Note that we frequently use the terms messages and events interchangeably.

Once the system has been modeled as a set of physical processes, there are two basic approaches to simulation. Common to both approaches is a clock which holds the time up to which the system has been simulated. In a *time-stepped* simulation, at each step the clock is incremented by one tick where a tick is one unit of time in the simulation. As an example, if the system being simulated is a computer chip, one tick may represent a pico-second. At each tick all of the events scheduled to occur at that time are simulated. The simulation is driven by advancing the clock by some preset, fixed amount and simulating the activity at each time step.

The problem with a time-stepped simulation is that there may be many ticks of the simulation clock for which no events are scheduled to occur. An approach that deals with this problem is *discrete event simulation*, where the clock is set to the time of the next event in the system rather than being incremented by some fixed amount. This work is concerned only with discrete event simulation.

In a sequential discrete event simulation, all events to be processed are maintained in a data structure called the *event list.* At each step of the simulation the event with the smallest timestamp is removed from the event list, the clock is set to the time of the event, and the effects of the event are simulated. The simulation of an event may cause other events to be added to the event list, or previously scheduled events to be removed from the event list. Simulation of an event may also cause the *state variables* of the simulation to be modified. The state variables represent the state of the system being simulated. As an example, if the system being simulated is a logic network, the state variables would include the current state of all input lines and all outputs of the gates in the network. An event may represent a change to the state of one of the lines in the system.

In a parallel discrete event simulation (PDES) each of the physical processes is modeled by a corresponding logical process (LP). The LP simulates the activity of the associated PP. The LP's are connected through logical communication channels or links such that if PP_i can send a message to PP_j in the physical system, then there is a link between LP_i and LP_j in the simulation. The messages sent between LP's contain a timestamp indicating the time when the event specified by the message is scheduled to occur.

Each LP maintains its own logical clock representing the time up to which the corresponding PP has been simulated. It is assumed that LP_i can correctly simulate the activity of PP_i up to any time *t*, given that LP_i knows the initial state of PP_i and all of the messages received by the PP up to time *t*. Stated another way, an LP can only guarantee that it can correctly simulate the corresponding PP up to time *t* if it knows all of the messages received by the PP up to time *t*. Knowing when it is permissible to advance the clock of an LP is a very difficult problem. In the next section we give a brief discussion of the difficulties associated with PDES.

2.2. Synchronization Problems of PDES

In the traditional approach to parallel discrete event simulation there is no global clock synchronizing all of the LPs. Each LP maintains its own logical clock representing the time up to which the corresponding PP has been simulated. It is therefore likely that, at any instant, the LPs will have progressed up to different points in the simulation. This asynchronous approach leads to difficult synchronization problems.

The fundamental synchronization problem of PDES is that it is very difficult to know if the execution of event A on LP_i will, in some way, affect an event B scheduled on some other LP_j. As discussed by Fujimoto (1990), the scenario in which one event may affect another event may be quite complex. Further, whether one event affects another event is dependent upon the timestamps of the events. To see this consider the following example.

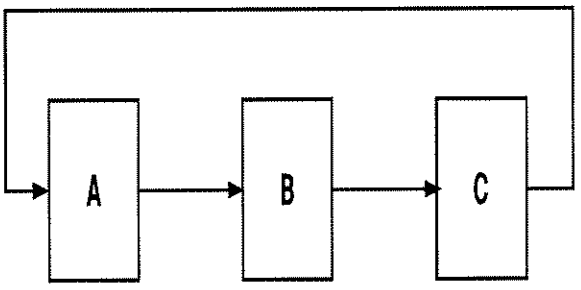

Figure 2.1 - Example of Event Dependencies

The system being modeled is a simple queueing network with three servers and their associated queues. Figure 2.1 shows the three LPs; the arcs between the LPs represent the inputs and outputs of the server. An input of a server represents LPs that may send it a message. An output of a server is an LP to which the server may send messages. Recall that a message is an event with a timestamp signifying when the event is to occur. An event in this system is either the arrival of a "job" at a server, or the completion of service for a job. Completion of a job may cause zero or more jobs to be scheduled at the current server, or at one of the other servers. The service discipline is FCFS. Assume that LP_A has a job scheduled to begin service at logical time ten ($event_1$), that LP_B has a job to begin service at logical time fifteen ($event_2$) and LP_C has a job scheduled to begin service at logical time twenty five ($event_3$). Further assume that the execution of $event_1$ causes the scheduling of a job to arrive at LP_B at time thirteen ($event_4$), and that $event_4$ in turn schedules a job to arrive at LP_C at time twenty one ($event_5$). As can be seen, it would be incorrect for the three LPs to execute their initial events concurrently. Consider LP_B. If it immediately simulates the effects of $event_2$ and advances its simulation time to fifteen, it will at some later point receive $event_4$ with a timestamp of thirteen. At the point LP B receives the event with a timestamp of thirteen, it will recognize that it has incorrectly simulated the behavior of the PP it is modeling since it has serviced the jobs in the wrong order. It recognizes its error when it receives an event that occurs in its past. Note also that LP_C must wait before simulating $event_3$. It will, at some point, receive $event_5$ which must be serviced before $event_3$.

As can be seen, the scenario in which one event may affect another can be a complex chain of events. In the above example, $event_1$ affects $event_3$ by scheduling $event_4$ at LP_B, which in turn schedules $event_5$ at LP_C. It may also have been the case that none of the events would have had an effect on each other. If $event_4$ were scheduled to occur at logical time sixteen instead of at logical time thirteen, and $event_5$ were scheduled at logical time twenty eight rather than at logical time twenty one, then there would have been no violation of event dependencies and the events could have all been executed concurrently. In general it is very difficult to determine which events can in some way affect other events and thus it is very difficult to determine which events may be executed in parallel.

In a sequential simulation all of the event dependencies are automatically maintained since the event with the smallest timestamp is always chosen for execution. All events are therefore executed in the correct order. In a similar fashion a parallel simulation will correctly maintain all event dependencies if *each* LP executes all of its events in non-decreasing timestamp order. Fujimoto (1990) terms this the *local-causality constraint* and discusses how it is a sufficient condition for correct execution. This is a difficult problem as there is not necessarily any correlation between the timestamps of events received by an LP and the real time order in which the events are received. In the next section we discuss the synchronization protocols that have been developed to solve this problem.

2.3. Synchronization Protocols

Most researchers classify synchronization protocols as either *conservative* or *optimistic*. As shown by Reynolds (1988), this classification is too narrow and there exists a spectrum of options. Reynolds developed a set of design variables and used these variables to classify existing synchronization protocols and to explore new approaches.

Although Reynolds defined ten design variables, three of these are most pertinent to the current discussion. These are *accuracy, aggressiveness* and *risk*. A parallel simulation is accurate if the results of the simulation are the same as those produced in a sequential simulation. An accurate protocol ensures that all event dependencies are maintained. A protocol that is aggressive allows an LP to process messages based on incomplete information. That is, the LP is allowed to process a message and advance its clock even though it may at some later point receive a message in its past. If a protocol is to be accurate and allow aggressive processing, there must be some kind of state saving and rollback mechanism to ensure accuracy. We discuss state saving and rollback in some detail below. Risk has to do with passing computation based on aggressive processing to other LPs. Thus a protocol that is aggressive but without risk (Dickens and Reynolds 1990, Steinman 1992) allows locally aggressive processing, but requires that the results of this aggressiveness not be passed along until it can be guaranteed to be correct.

2.3.1. Protocols That Are Accurate, Non-aggressive and Without Risk

What is generally considered as conservative protocols are accurate, non-aggressive and without risk. A non-aggressive LP cannot advance its simulation time until it can be guaranteed that it will not receive a message in its logical past. This can be guaranteed if two conditions are met. First, it is required that all messages sent between any pair of LPs be in non-decreasing timestamp order. Second, it is required that an LP block until it has received a message from each of the LPs which may send it a message. Once it has received a message from each of its input LPs, the LP may choose the message with the smallest timestamp and safely process this message. This is because it has received a message from all of its input LPs, and none of these input LPs may send a message with a smaller timestamp than one already sent. The LP will therefore never receive a message with a smaller timestamp than the one it chooses to process.

Even though this approach guarantees accuracy, it can lead to deadlock since a cyclic waiting condition can arise where each LP in the cycle is blocked waiting to receive a message from some other LP in the cycle. Much of the early research in parallel discrete event simulation focused on the deadlock issue. We briefly present the results of that early research.

Peacock *et al* (1979) showed that two conditions are necessary for deadlock to occur. Before these results are given, some definitions are needed. A link is a communication channel between two LPs. The link time of a given link is defined as the timestamp of the last message sent along that link. An empty link is one where there is no message waiting to be consumed by the receiving LP. We now give the results established by Peacock *et al* (1979).

> Theorem. A deadlock exists if and only if there is a cycle consisting of empty links which all have the same link time, and nodes which are all blocked because of these links.

This theorem establishes that deadlock will not occur if there are no cycles in the communication links. Even if cycles do exist, deadlock will not occur *unless* all of the links have the same link time. Thus one way to avoid deadlock is to ensure that all of the links in a cycle do not have the same link time. This can be guaranteed if, when an LP receives a message with timestamp t, any message that it sends in the future will have timestamp at least $t + \varepsilon$ where $\varepsilon > 0$. This requires that an LP that has received the

messages of the corresponding PP up to time t be able to predict the behavior of the PP up to some time greater than t. This ability to predict the future action of the PP is termed *lookahead.* An example of a system with the ability to look ahead is a non-preemptive FCFS queueing simulation where there is some minimum service time $M > 0$. Once the LP begins servicing a job at logical time t, it can predict that it will generate no events before logical time $t+M$.

Our discussion of the deadlock problem is by necessity brief. A thorough discussion of this issue can be found in Misra (1986) or Chandy and Misra (1979). Detailed examples of how deadlock occurs can also be found in Peacock *et al* (1979) or Peacock *et al* (1979a).

The non-aggressive, accurate protocols that we discuss in the remainder of this section are based upon the results established above. It is important to note that protocols based on these results require that the communication links be static and known a priori since an LP must know all LPs that may send it a message. If the communication links contain cycles, then the system must possess lookahead capabilities. These two factors restrict the class of problems for which this type of protocol may be used. For example, if the communication links contain cycles, then a queueing simulation with a minimum service time of zero may deadlock using this approach. This is because such a system has no lookahead capabilities.

For a system with no cycles in the communication graph (feedforward system), an LP can block until it has received a message from all of its input LPs without deadlock occurring. In Kumar (1986), a simple protocol based on this idea is presented. Even though deadlock cannot occur using this scheme, each LP must have infinite buffer space as there is no way to predict how many messages must be buffered until a message is received from each input LP. Further, the parallelism of the system may be greatly reduced if many of the LPs remain blocked for long periods of time waiting for messages from some of its input LPs. The authors developed an analytic model showing that the system can attain good speedup over sequential simulation. The model assumes however a high message flow rate to each LP thus assuring that the probability that any LP will have empty input links is low.

If the network topology does contain cycles, and the synchronization protocol is non-aggressive and accurate then there must be some mechanism to either avoid or detect and resolve deadlock. One

approach is to send non-event messages to help the LPs advance their simulation time. A non-event message is any message not sent in the physical system being simulated. Two protocols that are accurate and non-aggressive that employ non-event messages are the Null Message protocol (Bryant 1977, Chandy and Misra 1979) and the Link Time protocol (Peacock 1979). In these protocols, non-event messages convey information pertaining to the logical time of the LP sending the message. This information is used by the receiving LP to advance its simulation time. The primary problem with this type of approach is that the message traffic can be dominated by the non-event messages.

Peacock (1979) also introduced the Blocking Table algorithm that is accurate and non-aggressive. The major distinction of this approach is that non-event messages are sent only upon demand. An LP is defined as blocked if it has any empty communication links. Each blocked LP_i determines all LP_j such that there is a path of empty communication links from LP_j to LP_i. LP_i sends a request to each such LP_j, and remains blocked until it receives a response from each LP_j stating that it is permissible to proceed. Each LP repeats this algorithm every time it becomes blocked. As can be seen, even though non-event messages are sent only upon demand, the communication costs of this algorithm can be extremely high.

The appointment protocol (Nicol 1984, Nicol and Reynolds 1984, Nicol 1988) is also demand driven. In this approach a writer (an LP that sends a message to another LP) establishes an appointment with its readers (LPs with which the writer communicates). The appointment is the greatest lower bound on the logical time of the next message transmission from the writer to a reader. When a reader becomes blocked it demands an appointment from its writers to help it advance its simulation time (another approach is to have the writer maintain appointment times without the reader demanding it, see Nicol 1984 and Nicol and Reynolds 1984). As we discuss below, the performance of this type of protocol is governed by how well the writer can predict its future behavior.

In Chandy and Misra (1981), the authors propose allowing the simulation to deadlock and then detect and resolve the deadlock. This protocol was introduced in response to studies (cited in their paper) indicating that the number of non-event messages generated using the Null Message approach is unacceptably high. While this approach allows accuracy and non-aggressiveness without the use of non-event messages, its performance is obviously dependent upon how often the simulation deadlocks and how

expensive it is to detect and break deadlock.

In Lin and Lazowska (1990a), it is shown that systems without lookahead capabilities using the deadlock detection and resolution algorithm will result in a partial deadlock after each message processed. For these types of systems the authors describe a method of transforming such a network into a feedforward network thus avoiding deadlock (recall that a cyclic network is required for deadlock to occur). Once the network is a feedforward network, each LP simply blocks until it receives a message from all of its input LPs. Other approaches to deadlock detection and resolution can be found in Liu and Tropper (1990), Groselj and Tropper (1988), Groselj and Tropper (1989) and Wagner and Lazowska (1989).

2.3.2. Protocols That Are Aggressive, With Risk and Potentially Inaccurate

At least two researchers (Reynolds 1982, Sokol *et al* 1988) have investigated synchronization protocols that are aggressive, use risk and potentially inaccurate. The potential inaccuracies arise because no rollback mechanism is used to correct any out of order message processing that occurs as a result of the aggressive processing.

In SRADS (Reynolds 1982), an LP makes two aggressive assumptions. The first is that messages will arrive only at regularly occurring intervals. Using this assumption an LP will only synchronize with another LP that may send it a message at predetermined intervals. In between these intervals (referred to as the polling interval) LPs progress at their own rate. The second aggressive assumption is that it is permissible to advance the time of an LP to the next anticipated message arrival time. Thus if an LP has no more work to perform at the current time it will assume it can safely advance its clock.

In Moving Time Window (Sokol 1988), all events falling between the lower and upper bounds of a chosen time interval are available for execution. Events with timestamps outside of the window are not considered for execution. The aggressive assumption in Moving Time Window is that it is acceptable to process all of the events in the window interval concurrently. As there is no synchronization for the execution of events within the window interval it is easy to demonstrate that inaccuracies can occur. The authors point out however that by choosing the window interval to be small enough such that it is less

than the time between any two events in the sequential simulation, that no inaccuracies will occur.

Whether the potential inaccuracies are acceptable depends upon the system being modeled. In Theofanos (1984), it was shown that the inaccuracies in SRADS had little effect on mean value statistics for queueing models. In Sokol (1988), the authors discuss the tradeoffs between accuracy and speed of execution. Both of these potentially inaccurate protocols have introduced versions that are guaranteed to be accurate (Nicol and Reynolds 1984, Nicol 1984, Sokol 1990).

2.3.2.1. Use of Knowledge About the System

An important issue is how and whether knowledge about the system being simulated is embedded in the simulation, and how knowledge about the state of the simulation is shared throughout the logical system. In preceding sections we discussed the necessity of lookahead in preventing deadlock in protocols that are accurate, non-aggressive and without risk. The calculation of the lookahead value requires an LP to be able to predict its future behavior and thus involves knowledge about the system being modeled. Research has clearly demonstrated that the better the lookahead value, and thus the more knowledge about the system being modeled is exploited, the better the performance of these protocols (Fujimoto 1988, Fujimoto1989, Lin and Lazowska 1989).

As an example consider the appointment protocol discussed above. This protocol employs the use of knowledge in that the appointment is calculated based on the writer's knowledge of the system being simulated. For FCFS queueing network models, Nicol (1988) introduced the futures list as a way of obtaining a much better lookahead value. This allows the LP to calculate a sharper lower bound on the next message transmission time. The technique is based on the writer's knowledge of the service discipline employed in the network. As discussed by Nicol, the use of this knowledge about the system greatly enhanced performance. In Wagner and Lazowska (1988) and Lin and Lazowska (1989), the futures list technique is expanded to include a wider class of service disciplines.

An in depth discussion of the role of knowledge in parallel simulation is beyond the scope of this review. An excellent analysis of knowledge in parallel simulation can be found in Chandy and Misra (1987). Reynolds (1988) gives an in depth discussion of the use of knowledge in existing synchronization

protocols. Nicol and Reynolds (1984) and Lin *et al* (1988) discuss using knowledge about the system being simulated in the simulation protocol. Chandy and Sherman (1989) introduce a protocol based on converting conditional knowledge into unconditional knowledge. Wagner and Lazowska (1988) give a good summary of optimization techniques based on knowledge of the system being simulated.

Before leaving this section it should be mentioned that there are disadvantages to employing knowledge about the system being modeled in the simulation protocol. If the protocol requires that information about the simulation problem be exploited, then the protocol is obviously restricted to classes of problems where such information is available. As discussed by Fujimoto (1990), information about behavior such as minimum timestamp increments may be difficult to derive for some complex simulations.

It should also be noted that the types of information required by some of these protocols cannot be obtained for all applications. As an example, the futures list technique requires the presampling of service times. There are some applications which do not lend themselves to this presampling. One such application is a queueing network where the service time is dependent upon the state of the LP at the time the job enters service. In the next section we discuss Time Warp which is a much more general protocol. The generality of Time Warp is regarded as a strength by its designers.

2.3.3. Protocols That Are Accurate, Aggressive and With Risk

The accurate and non-aggressive algorithms discussed above use blocking to guarantee that no synchronization errors can occur. As discussed, the blocking mechanism can lead to deadlock, and solutions to the deadlock problem are expensive. An alternative approach is to allow an LP to aggressively process any message it receives, and to then use a rollback mechanism to correct any synchronization errors that occur as a result of this aggressive processing. This is the approach taken in the Time Warp protocol introduced by Jefferson (1985). Time Warp is accurate, aggressive and uses risk. It is generally considered an optimistic protocol in the literature because it optimistically processes any message it receives without any blocking.

Time Warp uses state saving and rollback to ensure accuracy. An LP periodically saves its state. When a synchronization error is discovered, the LP restores a previous state that was saved before the synchronization error occurred. The restoration of a previous state is termed a rollback. Once the previous state has been restored the LP again begins processing forward in time.

If an LP discovers a synchronization error at logical time t, then any messages it has sent with a timestamp $t' > t$ are potentially in error. To ensure accuracy, any erroneous messages must be cancelled. This is done through the use of *anti-messages*. If a message sent at logical time t with content c sent to LP_i is found to be incorrect, then an anti-message with logical time t and content c is sent to LP_i to cancel the original message. This anti-message may cause the receiving LP to rollback, which may in turn cause it to send anti-messages causing other rollbacks. Thus one rollback may cause a cascade of rollbacks. An LP will never rollback past the *Global Virtual Time*, defined as the minimum logical time over all logical clocks in the system and all events that have been sent but not yet received (Jefferson 1985). In Chapter 4 we discuss anti-messages in relation to our synchronization mechanism.

Two types of anti-message schemes have been investigated. *Aggressive Cancellation* is the sending of an anti-message as soon as the synchronization error is discovered. Using *lazy cancellation* the LP that is in error waits to send the anti-message until it is sure that it will not produce a message that is the same as the one to be cancelled. If the LP does indeed produce the same message as the one to be cancelled, then this message will have been sent ahead of schedule. In effect the right message has been sent for the wrong reason, and sent ahead of schedule. Thus performance is enhanced (Lin and Lazowska 1990). The disadvantage is that it delays the cancellation of incorrect messages thus allowing the LP to continue to process incorrectly for a longer period of time. The two rollback mechanisms are studied in Lin and Lazowska (1989a) and Gafni (1988).

Time Warp requires more memory than non-aggressive protocols due to state saving. The cost of performing state saving is very expensive. Fujimoto (1989a) found that as state size is increased by two kilobytes, the performance of Time Warp is degraded by 50%. For this reason hardware support has been suggested for state saving and rollback (Fujimotot 1988a, Buzzel *et al* 1990).

As discussed by Lubachevsky (1989), no theoretical arguments have been presented to show that Time Warp is scalable. That is, it is not known whether as the size of the simulation problem grows the frequency of rollbacks will increase at a greater rate. Protocols that limit the aggressiveness of Time Warp in order to reduce the frequency of rollbacks are discussed below.

2.3.4. Global Windowing Algorithms

All of the protocols discussed above, with the exception of Moving Time Window, are asynchronous. That is, the LPs never synchronize globally and there is no attempt to bound the difference in logical times between the various LPs in the system. Several researchers have investigated protocols that are loosely synchronous (see Fox *et al* 1988): The LPs proceed asynchronously for some time and then synchronize (Nicol 1993, Lubachevsky 1988, Ayani 1989, Chandy and Sherman 1989). These protocols are often referred to as windowing algorithms since the LPs cooperatively define a simulation window as we discuss below. In this section we briefly discuss the most important windowing algorithms.

The windowing algorithms under consideration proceed iteratively, most of them consisting of three phases. In the first phase the LPs cooperatively determine the minimum event in the system, and then determine which events are eligible to be considered for processing. In the next phase some mechanism is used to determine which of the eligible events are safe to process. By *safe* we mean the event can be processed without any possibility of a causality error. In the third phase the events determined to be safe are processed concurrently. Each of the phases is separated by a barrier synchronization. The primary difference between any of these windowing algorithms is the method used to determine which events are safe to process concurrently.

The Bounded Lag algorithm (Lubachevsky 1988) uses what Lubachevsky terms the bounded lag restriction. This restriction is that the difference between the times of any events processed concurrently is bounded from above by a known constant, the bounded lag. The algorithm progresses in the three phases described above. The events that are eligible to be considered for execution are those that fall within the simulation window, where the floor of the window is the smallest timestamp among all events in the system and the ceiling of the window is the floor plus the bounded lag. The purpose of the bounded

lag is to restrict the amount of computation that must be performed to determine which of the eligible events are safe to process. Lubachevsky (1988) gives an excellent discussion of how the bounded lag reduces the computation required to determine which events are safe to process.

Chandy and Sherman (1989) developed a protocol that is based on the idea of converting *conditional* events into *unconditional* events. A conditional event is one that will occur as long as no other events that can affect it are received. An example of a conditional event is the departure of a low priority customer in a preemptive queueing network. The job will depart at the scheduled time as long as no higher priority job is received before the time of the departure. An unconditional event is one that will occur regardless of any other events received.

The basic algorithm consists of three steps. First, the LP computes all unconditional events possible given its conditional events and all messages it has received. In the next step the LP sends a message to *each* of the other LPs. This message contains the LPs minimal event time, as well as any definite events it has computed. The LP then blocks until it has received a similar message from *all* other LPs. Given that all LPs have exchanged messages, and that each message contains the time of the earliest event of the sending LP (as well as all definite events computed), each LP can determine the earliest event time in the system. This earliest event time, as well as knowledge about all definite events that have been produced in the system, allows the LPs to convert conditional events into unconditional events. The way this is done is very application dependent. For any simulation problem however, the LP with the earliest event in the system can always safely process that event.

Ayani (1989) introduced a synchronization algorithm based on the distance between objects. The distance between objects is essentially the same concept as *lookahead* discussed above. The distance algorithm consists of three phases. In the first phase the earliest event in each LP is marked as a candidate for execution. In a sense this establishes a time window where the floor of the window is the smallest event time in the system and the ceiling is the maximum event time over all of the minimum events of the LPs. Note that each LP has exactly one event eligible for execution. In the next phase each LP examines the possible effect of executing its candidate event on all other candidate events. If its event can affect another event (i.e. potentially lead to a causality error), that event is marked as no longer being a

candidate for execution. Note that in the Bounded Lag approach an LP determines if its events are safe to process. In this algorithm an LP determines if the events of other LPs are safe to process. An LP assumes its candidate event is safe to process if no other LP has marked it as unsafe. In the third phase all events that are still eligible for execution are processed concurrently.

Note that in this version of the algorithm each LP checks its candidate event against all other LPs candidate events. Thus the complexity of this operation is $O(N^2)$ with N LPs. It also requires that each LP maintain the minimum propagation delay between itself and all other LPs. Due to these costs, Ayani modified the algorithm so that each LP maintains the minimum propagation delay between itself and its immediate successors only. When an LP determines if its candidate event can affect other events, it initially only searches the events of its immediate successors. Only if it can affect events on these LPs does it continue the search for other LPs the candidate event may affect.

The windowing algorithm developed by Nicol (1993) is the algorithm on which we base our research. We give a brief description of his algorithm in this section and discuss it in more detail in the next chapter. It is similar to Lubachevsky's algorithm except for the mechanism used to determine which events are eligible for processing. As discussed by Nicol, it is also a specific application of the conditional event approach, with a different mechanism to convert conditional into unconditional events.

Nicol's approach consists of three phases. In the first phase the LPs synchronize and determine the simulation window. The floor of the window is the minimum event time in the system. Given this minimum event time the LPs determine the ceiling of the simulation window. This ceiling is chosen such that all events between the floor and ceiling can be executed concurrently without the possibility of any causality errors. In the second phase of the algorithm, the LPs concurrently process all of their events with timestamps falling within the simulation window concurrently. In the third phase the LPs receive any messages generated by the processing of these events. Note that we term the simulation window defined by Nicol's protocol the *lookahead window*, since it is defined such that there will be no event dependencies within the window.

The ability to determine this lookahead window requires three assumptions about the system being modeled. First, when an event begins service at an LP, any events caused by the completion of the given

event must be known. The serving LP must inform the LPs that will receive such caused events at the time service begins. To clarify this idea assume that LP_A is to begin processing *event* 1 at logical time t. Further assume the completion of *event* 1 will cause *event* 2 to be scheduled for LP_B at logical time $t+x$. The protocol requires that LP_A inform LP_B that it will receive *event* 2 at logical time $t+x$ *at the real time at which* LP_A *begins processing event* 1. We term these messages sent to inform an LP of a future event a "pre-sent" completion message, and note they are important in the definition of our aggressive windowing algorithm. This ability to inform an LP of a future event requires that routing decisions be made at the time an event begins service. Thus this protocol cannot be used in systems where this routing decision is somehow dependent on the events enqueued at the LP when service is completed. The second assumption required by the protocol is that when an LP is informed that it will be receiving an event, it must be able to determine a lower bound on the service time (or the service time itself) for the event at the time of this notification. Third, it is assumed that once an event begins service, no other event in the system can affect its completion time. This implies that the protocol cannot be used to simulate a preemptive queueing system.

Nicol (1992) extends his basic algorithm to include preemptive queueing systems under certain conditions. The requirement is that each job in the network belong to one of C priority classes, and that a job never changes its class as it moves between queues. Nicol is able to obtain the powerful lookahead required by his algorithm by concurrently simulating different job classes in non-overlapping regions of simulation time. Higher priority jobs are not affected by lower priority jobs, and thus can be simulated far into the future. This implies that when a lower priority job enters service, the behavior of higher priority jobs that will affect it is already known. The protocol still requires however that the destination queue be known at the time a job enters service, and that its service time can be presampled when it enters a queue.

This concludes our review of the synchronization protocols which, with the exception of Moving Time Window and SRADS, would be categorized as either aggressive or non-aggressive. We now discuss some of the problems associated with these protocols. In the final section we discuss the most current research, including the research presented in this dissertation, which is investigating the blending of aggressive and non-aggressive protocols.

2.4. Problems of Existing PDES Synchronization Protocols

Each of the approaches discussed above has its own problems that make it ineffective when used to simulate some systems. Before leaving this section on the synchronization mechanisms it is important to point out the major weaknesses of the various approaches. It is beyond the scope of this review to give an in-depth critique of each synchronization protocol. The interested reader is directed to Fujimoto (1990) for an excellent discussion.

Protocols that are non-aggressive and accurate require that an LP not process an event until it can be guaranteed that no other event can possibly affect it. If there is *any* possibility that one event will affect another event, the two events must be executed sequentially. As discussed by Fujimoto (1990), if it is the case that two events that can affect each other rarely do, then most of the time sequential execution is unnecessary. An LP that blocks even though it could proceed without compromising the integrity of the simulation is, using Reynolds' (1982) terminology, *artificially blocked* It is blocked unnecessarily because it has insufficient information to determine it can safely advance its clock. This artificial blocking can greatly reduce parallelism.

Another criticism of non-aggressive and accurate mechanisms is that knowledge about the system being modeled often needs to be used by the synchronization mechanism in order to achieve good performance (Fujimoto 1990). As discussed above, research has clearly demonstrated that the better the lookahead value, and thus the more knowledge about the system being modeled that is embedded in the protocol, the better the performance. This has two implications. First, the simulation programmer must be concerned with details of the synchronization mechanism. Second, if the system contains inherently poor lookahead ability, these protocols will not perform well (Fujimoto 1990).

Proponents of aggressive and accurate protocols argue that the simulation programmer should not have to deal with issues of the synchronization mechanism (Fujimoto 1990). Time Warp (Jefferson 1985) does not require that the simulation programmer be nearly as concerned with this level of detail. It should be noted however that the programmer is not completely alleviated of concern with low level details using Time Warp. The programmer must ensure, for instance, that the machine has enough resources (such as memory) to support the application.

Another advantage of Time Warp is that research has demonstrated that good lookahead is *not* necessary in order to achieve good performance (Fujimoto 1990a). Further, Time Warp does not suffer from artificial blocking; it never blocks if there is any work to do.

Time Warp does however suffer three primary problems. First, state saving is an expensive operation. As discussed above, Fujimoto (1989a) found that as the size of the state increased from 100 bytes to 2000 bytes, performance was degraded by 50%. While several performance studies have shown Time Warp to perform well (Presley 1989, Wieland *et al* 1989, Ebling *et al* 1989), none of these authors discusses the size of the state vector for the given problems. Hardware support for saving state has become available (Buzzell *et al* 1990, Fujimoto *et al* 1988a).

The second problem with Time Warp is the possibility that it can become very inefficient due to *echoing* or *cascading rollbacks* (Lubachevsky *et al* 1989). Echoing is a pattern of self-perpetuating rollbacks where the amplitude of the rollbacks increases without bound. That is, a pattern is established where LPs are constantly resynchronizing each other, and the amount of logical time that an LP must roll back increases with each resynchronization. Cascading is a situation where one rollback causes other rollbacks, and the number of participants increases without bound. If either of these phenomena occur in a simulation, Time Warp will make very very little forward progress. Turner and Xu (1992) have reported cascading rollbacks and echoing which significantly degraded performance in their telephone switching network simulation, and Lubachevsky (1989), Mitra and Mitrani (1984) and Aahlad and Brown (1988) have all developed models to show that echoing can occur.

The third problem with Time Warp is it requires much more memory than other protocols in order to save state and to save copies of messages that have been sent.

As discussed, non-aggressive protocols suffer from artificial blocking and high costs for deadlock prevention, and aggressive protocols suffer from excessive rollbacks, high state saving costs and large memory requirements. We now review current research which combines these two approaches in order to improve performance.

2.4.1. Protocols Which Blend Aggressive and Non-Aggressive Behavior

As noted, there are problems inherent in a protocol which is purely aggressive or purely non-aggressive. For this reason researchers have begun to investigate protocols which blend aspects of both approaches. The goal of this research is to maximize the advantages, and minimize the disadvantages of each approach.

Reynolds (1988) was the first to suggest that the aggressive/non-aggressive dichotomy is too restrictive, and showed that there exists a spectrum of options for parallel simulation. As an outgrowth of this research, Dickens and Reynolds (1990) developed the SRADS with Local Rollback protocol. The concept behind a protocol with local rollback is that an LP can perform aggressive computation, but the results of this computation cannot be sent until it can be determined that the computation is correct. Thus if a computation is found to be incorrect, the LP will perform a local rollback but the error will not spread through the system. This allows an LP to perform potentially useful processing when it may otherwise be blocked due to synchronization constraints. More recently, Steinman (1992) has modified his non-aggressive Time Bucket Synchronization algorithm to include local rollback.

Turner and Xu (1992) have developed the Bounded Time Warp Algorithm to address the problem of excessive rollback of unbounded Time Warp. The Bounded Time Warp protocol defines a simulation window in which the LPs can process aggressively, and does not allow an LP to process beyond this upper bound. In this sense it is similar to a global windowing algorithm, but uses a two phase token passing mechanism rather than a barrier synchronization to synchronize the LPs. The authors developed this protocol in response to the problem of excessive rollbacks when they simulated a telephone switching system using unbounded Time Warp. The authors note that in a large telephone switching system, it took much longer to perform a single run (from say logical time 0 to logical time 500), then it did to cover the same virtual time period in five separate runs (i.e. 0-100, 100-200,300-400,400-500). This degradation in performance was due to excessive rollbacks. The authors noted an improvement of up to 78% when they bounded the aggressiveness of Time Warp.

The Bounded Time Warp protocol is similar to the Filtered Rollback algorithm developed by Lubachevsky *et al* (1989). This algorithm is essentially an aggressive version of the Bounded Lag

Algorithm discussed above, where the simulation window may be defined such that all processing within the window is *not* guaranteed to be correct. Any causality errors are corrected through the Time Warp state saving and rollback mechanism. The goal of this algorithm is two fold. First, it decreases the amount of blocking that may be required in his non-aggressive version of the algorithm. Second, it decreases the probability of cascading rollbacks in an unbounded Time Warp. Lubachevsky developed a model to show that the Filtered Rollback algorithm maintains the important scalability properties of the Bounded Lag algorithm.

Madisetti, Hardaker and Fujimoto (1992) developed the Mindix Operating System which blends characteristics of aggressive and non-aggressive protocols. The synchronization mechanism uses what they term *probabilistic synchronization*, where resynchronization messages are periodically sent through the system according to some probability distribution. When an LP receives a resynchronization message, it deletes all of its input and output with a timestamp greater than that of the resynchronization message. This mechanism is used to keep the LPs from getting too far out of synchronization with each other, thus avoiding cascading rollbacks. This mechanism is not a windowing mechanism in that the LPs are not forced to block at the end of a simulation window. Rather, the LPs are allowed to continue processing without blocking, but are forced to resynchronize periodically. The authors give empirical results which demonstrate that keeping the system loosely synchronized can offer on the order of a magnitude of speedup over unbounded Time Warp.

As can be seen, there is much interest in blending aspects of aggressive and non-aggressive protocols. The research presented in this thesis builds on the windowing algorithm developed by Nicol (1993), where we define a mechanism to add aggressiveness to his non-aggressive protocol. We give the details of this new algorithm in the next chapter. Our algorithm is similar to the other such algorithms discussed above, and is most closely related to the Bounded Time Warp algorithm introduced by Turner and Xu (1992), and the Filtered Rollback algorithm developed by Lubachevsky *et al.* (1989). Thus we do not claim that our algorithm is unique.

What separates our research from the other research discussed above is our *analysis* of this new approach. We derive the expected improvement in performance of the aggressive windowing algorithm

compared with the non-aggressive version of the algorithm *as a function of the level of aggressiveness.* Further, we derive the probability of a causality error, the probability that a given LP starts a rollback chain by issuing an anti-message, the number of times an LP must save state, the expected number of messages that must be reprocessed due to rollbacks and the probability that the longest rollback chain observed in the system reaches depth $D = d$ *as a function of the level of aggressiveness.* Thus while many researchers have now accepted the idea of blending aspects of aggressive and non-aggressive protocols, this is the first time the relationship between *the level* of aggressiveness and these various performance characteristics has been established. Also, the research presented here is one of only two models (the other being Nicol 1991) which compares the performance of two competing approaches and includes the major costs of each approach.

In the next section we review the other important analytic results which have been established for parallel simulation.

2.5. Analytic Results

There is a small but growing set of analytic results for the expected performance of parallel simulation protocols. It is generally agreed that this is a difficult problem, particularly as it relates to asynchronous protocols. Much of the early modeling of Time Warp assumed a two processor system. Recent work includes the development of multiprocessor models to predict the behavior of Time Warp, but these models generally assume negligible overhead costs. Other models have compared the performance of the Chandy/Misra/Bryant Null Message protocol (a non-aggressive protocol) with Time Warp, and still others have investigated optimality results for Time Warp. Again however these studies tend to assume negligible overhead costs and are therefore of limited utility. In this section we review the major analytic results established for parallel simulation.

As discussed by Lubachevsky (1989), a very important issue is if, as the simulation problem grows, the overhead associated with the synchronization protocol grows at a much faster rate. Obviously models that exclude or minimize overhead costs cannot answer this question. As pointed out by Lubachevsky both Time Warp and the Chandy/Misra/Bryant Null Message protocol have the potential for explosive

overhead costs: Time Warp because of the potential for cascading rollbacks and increasing state saving costs, and Null Messages because of the potential for an explosion of null messages. As mentioned above, these types of scalability issues have been addressed for only a narrow class of synchronization protocols.

2.5.1. Models Involving Time Warp

Mitra and Mitrani (1984) developed a two processor model to study the behavior of Time Warp. A two processor model is chosen so as to avoid the complications of cascading rollbacks. In their model rollback costs are assumed to be proportional to the rollback distance and the cost of state saving is not considered in the model. Based on their model the authors conclude that if the rollback costs are sufficiently high, the optimal decision is to not perform the task at all. Also they show that as the costs of rollback get more expensive there is more incentive to slow down the faster of the two processors. This model has not been extended to more than two processors.

Aahlad and Browne (1988) develop a model showing that the echoing phenomenon can occur in Time Warp. In their study an expression is derived showing the relationship between the cost of rollback and forward processing that will lead to echoing. Their model is based on the assumption that the cost of rollback is proportional to the rollback distance. Only when the rollback cost is sufficiently high will echoing occur.

Lubachevsky (1989) developed an example using three processors demonstrating the echoing phenomenon. He showed that as the example simulation progresses, the lengths of simulated time intervals that must be rolled back and the number of incorrect messages processed also increase. Thus as the simulation progresses, the amount of logical time advanced per unit of physical time decreases. Lubachevsky assumes the cost of rolling back a process is the same as forward computation. Also he ignores the cost of state saving during forward computation.

As noted in the last section, these models showing that echoing can occur have been validated in studies by Turner and Xu (1992) where this phenomenon has been observed.

Other studies involving Time Warp assume either zero or negligible overhead costs. As would be expected, these studies show the potential for excellent performance for Time Warp.

Lin and Lazowska (1990) present a model comparing the performance of Time Warp and the Null Message protocol. State saving and rollback costs are assumed to be zero. Before giving the results of their analysis three definitions are needed. A simulation is *conservative optimal* if it completes in the time given by the critical path of the event-precedence graph. *True computation* is correct computation and *false computation* is incorrect computation caused by aggressive processing. Assuming no overhead cost, and assuming that true computation is never rolled back by false computation, the authors show that Time Warp is conservative optimal using aggressive cancellation. They extend these results to show that Time Warp using lazy cancellation can out-perform conservative optimal. This is because some correct computation may be performed ahead of schedule due to aggressive computation. In essence correct computation is performed for the wrong reason. The authors then relax the assumption that no correct computation is ever rolled back by incorrect computation, but maintain the assumption of no overhead costs. Under these assumptions they show that Time Warp always out-performs Null Messages for feed-forward networks. Finally the authors prove that Time Warp out-performs the Null Message protocol in most feedback networks without lookahead. However, as noted, they assume zero cost for rollback and state saving.

Mizell and Lipton (1990) construct a model to perform a worst case analysis between Time Warp and the Null Message protoco. The model assumes the state of an LP is saved after every message processed, but there is no state saving cost in the model. The cost of rolling back a process is assumed to be constant and independent of the rollback distance. Using this model they show that there exists a simulation such that Time Warp can arbitrarily out-perform the Null Message protocol. According to the author's definition Time Warp arbitrarily out performs the Null Message protocol if, in a simulation involving P processors, Time Warp is proportional to P times faster than the Null Message protocol. The authors then prove that the converse is not true: There is no simulation problem such that the Null Message protocol arbitrarily out-performs, in the same sense, Time Warp.

Madisetti, Walrand and Messerschdmitt (1990) develop a model for an optimistic system similar to Time Warp in order to derive the average rate of progress of the system. In this model LPs are categorized as either fast or slow, where the fast LPs progress at a rate at least twice as fast as the slow LPs. Thus their model describes a highly unbalanced system. They study two rollback techniques: successive and concurrent rollback. In the concurrent rollback scheme, when an LP in the fast set is rolled back by an LP in the slow set, all other LPs in the fast set are also rolled back. This is done so as to avoid cascading rollbacks. The successive rollback scheme is the basic Time Warp approach where LPs are resynchronized through the use of antimessages. They assume rollback costs are constant and independent of the rollback distance.

The authors assume that any rollbacks caused by an LP in the fast set communicating with another LP in the fast set takes no time. Given this assumption, the authors state that resynchronizations among the LPs in the fast set has no effect upon the rate of progress of the system. Using this model the authors show that it is advantageous to force the fast set of LPs to resynchronize concurrently when one of the members of the set is resynchronized by a slow LP.

Felderman and Kleinrock (1990) derive an upper bound on the improvement of Time Warp over a time stepped simulation. In the model it is assumed Time Warp incurs no state saving or rollback costs. The model of the time stepped simulation assumes that each LP has an event to process at each time step. Thus there are no idle processors at the beginning of any time increment. Under these assumptions the authors show that, when the event computation time is exponentially distributed, the maximum improvement of Time Warp over time stepped simulation is ln (P) with P processors. When event computation time is uniformly distributed the maximum speedup is shown to be approximately two, independent of the number of processors.

Felderman and Kleinrock (1991) studied the improvement in performance of a two processor Time Warp system over a single processor using the probability of an interaction between the processors as a parameter to the model. Also, they added state saving costs to this model and derived the conditions under which Time Warp would out-perform a sequential simulation. In this model they did not include message queueing, and thus any messages received in the logical future of an LP are discarded. This two

processor model was extended to include message queueing in Feldermnan and Kleinrock (1992a). Also, the authors (Felderman and Kleinrock 1992) develop a two processor model for a non-aggressive synchronization mechanism and use their two processor model of Time Warp to compare the approaches.

Felderman and Kleinrock (1991a) develop a multiprocessor model to study the behavior of Time Warp in the context of a *self-initiating* system with no lookahead. A self-initiating system is one where the LP itself determines when it will re-evaluate its state, it is not determined by the receipt of external messages. In the self-initiating model assumed by the authors, messages are used only for synchronization purposes and do not carry any work. Thus when an LP receives a message in its logical past it performs a rollback, and when it receives a message in its future the message is discarded. Under these assumptions, and the assumption of no state saving costs, the model predicts the rate of progress of a Time Warp system. Under these same assumptions the authors demonstrate that the rate of progress of the system scales well as the number of processors is increased. This represents the first scalability argument for Time Warp, but as noted the model does not include state saving costs and the messages exchanged between the LPs do not contain any work.

Gupta, Akylidiz and Fujimoto (1991) develop a model to derive various performance measures for Time Warp under the assumption of negligible overhead costs. They derive measurements such as the fraction of processed events that are committed, the probability of a rollback, the expected length of a rollback, and the fraction of time a processor remains idle. They then compare their results with an operational Time Warp system. The authors note that this is the first time the comparison between a model and an actual implementation has been performed. The results obtained in their analysis are validated by their empirical results.

2.5.2. Model Comparing Null Message Protocol and Time Stepped

Bailey and Snyder (1989) develop a model to compare the asynchronous Null Message protocol (Chandy and Misra 1979) and a synchronous (time stepped) algorithm for parallel logic-level simulation. Using their model they show that the asynchronous approach will always complete in time at least as fast as the synchronous approach.

Bailey and Snyder state also that they use their model to compare Time Warp with the asynchronous and synchronous approaches. They do not do this directly, but rather point to the Lin and Lazowska paper discussed above (Lin and Lazowska 1990). Bailey and Snyder state, for a certain class of circuits, that Lin and Lazowska proved the optimality of Time Warp. The class of circuits they discuss are those without cycles, and thus fall into the general category of feedforward networks discussed by Lin and Lazowska (1990). These results were established under the assumption of no overhead costs for Time Warp for state saving and rollback, and assuming unlimited processors. Bailey and Snyder do develop an example to show, without unlimited processors, that either the Null Message protocol or Time Warp can complete faster depending on event scheduling.

2.5.3. Analysis of Global Windowing Algorithms

As discussed above an important question is whether the overhead associated with a given protocol grows much faster than the size of the simulation problem. An important question pertaining to the windowing protocols that are accurate and non-aggressive is whether a window that is small enough to exclude event dependencies admits enough events to keep processors busy (Nicol 1993). In this section we discuss analytic results that answer these questions for two iterative algorithms. Then we discuss other important results established for one of the algorithms in this class.

Lubachevsky proved analytically two important properties of his Bounded Lag Algorithm. In Lubachevsky (1988), it was shown that given P processors, the average number of instructions executed for each event processed is O(log P). This includes the synchronization time of the P processors, as well as all idling and busy waiting. Thus the amount of overhead associated with processing each event goes up very slowly as the size of the architecture increases.

Also, Lubachevsky (1989a) addresses the issue of the scalability of the Bounded Lag Algorithm. Before reviewing the results, it is important to state the major assumptions of his analysis. First, it is assumed that given K nodes, on average at least an order K nodes have events within the window per iteration. Second, it is assumed that the total number of events to be processed in the simulation run is $O(K)$ for K processors. Using these assumptions, as well as other less important assumptions, Lubachev-

sky proves the scalability of the Bounded Lag approach. It is shown that if it takes time T to complete a simulation on one processor, then to simulate a K times larger problem using K processors it will take $O(T \log K)$ time to complete. Thus the proof is addressing the situation where both the problem size and the size of the architecture grow proportionately while other system parameters such as event density remain constant.

Lubachevsky *et al* (1989) extend these scalability results to the Filtered Rollback algorithm, which as discussed in the previous section, adds aggressiveness to the Bounded Lag Algorithm. The authors develop a model of a rollback tree in order to derive conditions under which cascading rollbacks will not occur. They use these results, and the results established for the Bounded Lag algorithm, to demonstrate that the Filtered Rollback algorithm maintains the log scalability features of the non-aggressive version of the algorithm. They make no theoretical comparison of the performance of the Bounded Lag algorithm with that of the Filtered Rollback algorithm.

Nicol (1993) also demonstrates the scalability of his windowing algorithm, but in the context of a fixed size architecture. He shows as the problem size grows relative to a fixed architecture, the per event overhead approaches $O(\log T)$ where T is the total number of events in the system. The event overhead includes synchronization costs, event list manipulation, lookahead calculation and processor idle time due to synchronization constraints. Thus the overhead of this method approaches the overhead of a sequential simulation as the problem size increases.

Also Nicol demonstrates analytically that as the problem size grows the number of events available for execution in a given iteration also grows. Given some constant minimum delay greater than zero, the average number of events processed per window grows at least linearly as the total event rate in the simulation increases. As pointed out by Nicol this shows that for large problems executed on medium size machines there will generally be enough work to keep most processors from being idle.

Nicol (1991) gives an in-depth study of his iterative algorithm in the context of self-initiating simulation models. Recall that a self initiating model is one where the LP itself determines when it will re-evaluate its state, it is not determined by the receipt of external messages. For this class of simulations Nicol develops upper and lower bounds on optimal performance, as well as upper bounds on Time

Warp's performance and lower bounds on the iterative algorithm's performance. The analysis of Time Warp includes state saving and rollback costs. Also Nicol investigates the impact of message fan-out, lookahead ability and the time-increment distribution on performance. To the best of our knowledge this is the first paper to compare Time Warp and a non-aggressive protocol that includes most of the overhead associated with both approaches. Also it is the first paper to give an in-depth analytic study of the sensitivity of the protocol to factors such as message fan out and lookahead ability. For this reason we feel it is important to outline the main results of the paper. We begin by describing the model.

In the model an LP advances its simulation time by executing some simulation activity termed a *cycle*. At the end of one cycle the LP schedules the end of the next cycle. This scheduling is done without regard to any messages received from any other LPs. After execution of a cycle the simulation time is advanced by a random amount drawn from some probability distribution. For purposes of this discussion this probability distribution is termed the time-increment distribution. After an LP re-evaluates its state it sends its new state to other LPs. The message fan-out is the number of LPs to which any given LP may send its new state. The type of lookahead discussed in the paper is termed *full lookahead* meaning that both the time and the *content* of the message can be predicted in advance. As pointed out by Nicol the results established can be extended to systems with only *time-lookahead* where the time of the message, but not the content, can be predicted.

The first result pertains to the effect of message fan-out for time-increment distributions such as exponential plus a non-negative constant. The first results are obtained under the assumption that no lookahead is available. He shows the optimal processor utilization is $\leq 1/K$ where K is the message fan-out. If the time-increment distribution is geometric with mean $1/p$, then as $(1-p)^K \rightarrow 0$, processor utilization is $\leq p$. If however one cycle full-lookahead is available, and the time-increment distribution is exponential plus a constant, then processor utilization is $\leq \frac{1}{\sqrt{K}}$. Thus it can be seen that both the message fan-out and the ability to lookahead can dramatically affect performance.

Lower bounds for the performance of the iterative algorithm are given assuming the time-increment distribution is constant plus an exponential and one cycle full-lookahead is used. The cost of

global synchronization is excluded for these results. It is shown that if the ratio of the constant to the mean of the distribution is 0.25, the lower bound on performance is 20% processor utilization. If this ratio is 1 then processor utilization is 50% and if it is 10 processor utilization is at least 91%.

Nicol also established an upper bound for the performance of Time Warp for this model. It is assumed that Time Warp saves state before every cycle, that no lookahead is used, that rollback costs are constant and that the costs of cascading rollbacks are ignored. It is shown that the upper bound on processor utilization under these assumptions is

$$\frac{1}{(C_s(C_{Rh}+1)K)}$$

where C_s is the cost of state saving, C_{Rh} is the cost of rollback divided by two and K is the message fan-out. Then Nicol uses the upper bound for Time Warp and the lower bound for the conservative protocol to establish conditions under which the conservative protocol will out-perform Time Warp.

This is a very important paper in that it develops non-trivial upper bounds for the performance of Time Warp while including the costs of state saving and rollback. Also is is important because guarantees are developed for minimum performance of the iterative protocol.

As we have demonstrated, there is still much to be accomplished in the development of analytic models for parallel simulation. The model developed in this dissertation makes a significant contribution to this small set of analytic results. First, we develop an analytic model to compare the performance of the non-aggressive version of Nicol's algorithm with our aggressive version. Our model is developed under the assumption of a *message-initiating* model where an LP updates its state based on the receipt of external messages. Thus in our model external messages carry work to be performed, and messages with a timestamp in the logical future are queued for later processing. As discussed, we include the major costs of both approaches in our model, and this represents the first time this is done for a message-initiating simulation model. In fact, we compare the two approaches as a *function* of the primary costs of each approach.

Another unique feature of this analysis is that we investigate the improvement in parallelism *as a function of the level of aggressiveness.* As the level of aggressiveness increases, it is expected that the

costs associated with aggressive processing (such as causality errors and state saving costs) will increase. We develop our model such that we can determine the relationship between the level of aggressiveness and the costs associated with aggressive processing. This too is the first time this issue has been addressed.

In this analysis we make significant progress towards extending the scalability results obtained by Nicol (1993) to the aggressive version of the algorithm. We theoretically demonstrate that the probability of a causality error at a given LP, and the probability that an LP initiates a rollback chain by producing an anti-message, do not increase *as the number of LPs approaches infinity.* Finally, we give simulation results which agree with the predictions of our model.

In the next chapter we define the Global Windowing algorithm, and develop a preliminary model to study its performance. Also, we derive results which represent an important step towards proving system-level scalability of our algorithm. Then we give simulation results which support the predictions of our analytic model.

CHAPTER 3

The Aggressive Global Windowing Protocol

In the previous chapter we discussed the difficult synchronization issues encountered in parallel discrete event simulation. Also, we discussed the various approaches developed to deal with these difficult issues. As noted, a purely non-aggressive or a purely aggressive mechanism has inherent limitations which we seek to overcome by developing a protocol which blends aspects of both approaches.

In this chapter we investigate the improvement in parallelism that is possible when aggressiveness is added to an existing non-aggressive windowing algorithm. We define a simple mechanism, the *aggressive window*, to introduce aggressiveness into the Global Windowing Algorithm. We study two basic issues. First, we look at the extra parallelism made available by adding aggressiveness. Clearly this extra parallelism is not as valuable if the aggressive processing leads to causality errors. For this reason we also investigate the probability of a causality error as a result of aggressive processing. We do not define a protocol to correct causality errors that occur as a result of this aggressive processing. This is done in later chapters. In this chapter we develop an analytic model to show that significant gains in parallelism are possible *given a very limited amount of aggressive processing*. Also, we show that the probability of a causality error at a given LP is low because of the limited amount of aggressive processing we allow.

The rest of this chapter is organized as follows. In section 3.1 we give a brief description of non-aggressive Global Windowing Algorithms and discuss the modifications required to make them aggressive. In section 3.2 we develop our model to evaluate the performance of our approach. In section 3.3 we calculate the probability of a fault at a given LP. In section 3.4 we calculate the expected improvement in parallelism as a result of aggressive processing. In section 3.5 we derive results which represent a significant step towards proving the scalability of our approach. In section 3.6 we give simulation results which test the predictive power of our model and in section 3.7 we give our conclusions.

3.1. The Aggressive Global Windowing Algorithm

As discussed in previous chapters, if events are not processed in correct timestamp order it is *possible* that dependencies among the events are violated. We say possible because it is not necessarily the case that two events processed in non-increasing timestamp order have any dependence relationship. Since it is not known *a priori* whether out of order processing results in a violation of event dependencies, we make the worst case assumption that in each such case it does. Recall that we term this out of order processing a *causality error* or *fault*. In previous chapters we have described many of the approaches used to prevent the violation of event dependencies, and other approaches that correct these violations when they occur. In this section we begin by describing the approach used by one class of synchronization protocols, the Global Windowing Algorithms, to ensure that all event dependencies are maintained. Then we describe how to modify this basic algorithm in order to enhance parallelism.

The windowing protocols under discussion generally proceed in three phases. In the first phase, the LPs cooperatively determine the synchronization window. The floor of the window is the minimum timestamp over all unprocessed messages in the system. The ceiling of the window is chosen such that all messages within the window can be executed concurrently without any possibility of a causality error. We term this simulation window defined by the protocol the *lookahead* window and again note that it is defined such that all events within the window can be processed concurrently without violating any event dependencies. In the second phase, each LP executes all of its events with timestamps falling within the lookahead window. In the third phase, the events generated as a result of execution within the lookahead window are exchanged. Each phase is separated by a barrier synchronization. The primary difference among the various windowing protocols is the mechanism used to determine the lookahead window. Only those messages with timestamps falling within the lookahead window (where there is no possibility of a causality error) are considered for execution.

Windowing algorithms, as all non-aggressive protocols, are criticized for not fully exploiting all of the parallelism available in the simulation application (Fujimoto 1990). This is because an event will not be executed if it is *possible* that some other event in the system can affect its execution. Thus if it is possible that one event's execution can affect another event, but rarely does so, then the two events will be

executed sequentially even though most of the time they could be executed concurrently.

One way to gain more parallelism in a windowing protocol is to extend the lookahead window and allow the execution of *conditional events*, that is, those events that *may* have some event dependencies. The benefit of this approach is that events that may affect each other, but rarely do, can be executed in parallel. These events would be excluded from concurrent execution by the basic windowing algorithm because of the possibility of an error. The disadvantage of this approach is that there must be some mechanism to correct causality errors that do occur as a result of processing conditional events. Before investigating the probability of a causality error we describe the modifications to the basic Global Windowing Algorithm to allow *aggressive processing*: the processing of events that may lead to a causality error.

Our approach is to extend the simulation window past the lookahead window established by the non-aggressive protocol. Assume the system is synchronized at logical time T where T is the current window floor. The non-aggressive windowing algorithm defines the lookahead window from logical time T to logical time $T+L$, where L is the width of the lookahead window. Our modified algorithm defines an extended simulation window from the upper bound of the lookahead window (logical time $T+L$) to logical time $T+L+A$. We term this extension to the lookahead window the *aggressive* window and note that it has a length of A logical time units.

In our aggressive algorithm we allow events with timestamps falling within the aggressive window to be processed as well as those events with timestamps falling within the lookahead window. This is shown in Figure 3.1.

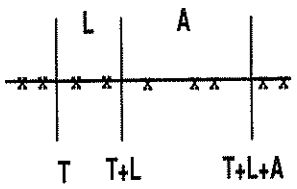

Figure 3.1 - The Aggressive Window

Our modified windowing algorithm proceeds in two phases. In the first phase the LPs synchronize to determine the lookahead window and the aggressive window. In the second phase, the LPs process all events concurrently within these two windows. When an LP completes all of its processing within both windows it blocks waiting for other LPs to similarly complete.* As a result of processing within the lookahead and the aggressive windows an LP may generate events to send to other LPs. We assume that these events are sent as soon as they are generated. Further note that Nicol's algorithm (and other windowing algorithms) can also be defined to proceed in two phases rather than three by sending new events as soon as they are generated.

As discussed above, all processing within the lookahead window is guaranteed to be correct. This occurs because the window is constructed such that no LP will receive an event with a timestamp falling within the lookahead window. Thus there can be no possibility of a causality error. Processing within the aggressive window however (recall that we term this *aggressive* processing) *can* lead to causality errors. This is because an LP can receive events with timestamps falling within the aggressive window such that the timestamp of the received event is less than the timestamp of an event already processed. If errors do occur as a result of processing within the aggressive window there must be some mechanism to correct the errors. We defer this important issue until Chapter 4 where we give a detailed description of our protocol.

Given this brief description of our approach, we are able to develop our analytic model to investigate the performance issues in which we are interested. We do this in the following sections.

3.2. Model

We are interested in the possible increase in available parallelism as a result of extending the lookahead window to allow aggressive processing. When we discuss the improvement in parallelism we are referring to the extra amount of processing that can be obtained between synchronization points with this extension to the lookahead window. We define the parallelism of the non-aggressive protocol as the ex-

*We are temporarily ignoring the important issue of how an LP determines it has completed all of its processing within the aggressive window. Further, we are temporarily ignoring how to implement a barrier synchronization given that an LP may discover that it has not performed all of its processing and thus needs to pull out of the barrier. These issues are discussed in detail in subsequent chapters.

pected number of messages processed within the lookahead window. We define the parallelism of the modified algorithm as the expected number of messages processed successfully (i.e. processed without later being invalidated by the receipt of another event with a smaller timestamp) in the aggressive window plus the number of messages processed in the lookahead window. We define the expected improvement in parallelism as the ratio of these two quantities. Note that this measure does not include the cost of correcting causality errors that occur as a result of aggressive processing. We modify our model to include this cost in the next chapter.

We are interested in two primary measurements. First, we seek the expected improvement in parallelism *as a function of the size of the aggressive window*. Second, we are interested in the probability that an event dependence is violated (i.e. a causality error) as we begin to process outside of the lookahead window. We are interested in this error probability as a function of the size of the aggressive window. This study represents the first time a model has been developed to study the expected increase in parallelism and the probability of a causality error *as a function of the level of aggressive processing*.

The model developed in this chapter investigates the possible improvement in parallelism and the probability of a causality error *at a given LP*. Thus the analysis in this chapter is not directly applicable to the more complicated issue of *system-level* performance. We note however that the analysis presented in this chapter is important in that it lays the foundation for the analysis of system-level performance which is addressed in subsequent chapters.

Our model is closely related to the one developed by Nicol (1993) although he has not studied processing outside of the lookahead window. Also his model is more general than the one presented here in that his captures general message exchange distributions. Our model is also closely related to the models developed by Akyildiz *et. al.* (1992) and Gupta *et. al.* (1991), and uses the same assumptions. Akyildiz and Gupta however are investigating the behavior of Time Warp which does not limit aggressive processing as we are proposing.

We analyze our model as a collection of servers where *activities* occur. There are N LPs, one server per LP and one LP per processor. The assumption of one LP per processor can be modified without difficulty. An activity begins, ends and upon its completion causes other activities. In our model

each completion causes exactly one other activity. We assume the completion causes an activity at a server that is picked at random where each server is equally likely to be chosen. This equally likely assumption is a common assumption in the modelling community and is used to make the analysis tractable (Felderman and Kleinrock 1991a, Gupta *et. al.* 1991, Nicol 1991, Akyildiz *et. al.* 1992).

The delay in simulation time between when an activity begins and ends is called the *duration* of the activity. We assume each server chooses the duration of an activity from an independent, identically distributed exponential distribution with mean $1/\lambda$. Our assumption of one server per LP implies that each LP will impose some queueing discipline.

The model presented in this paper assumes a closed queueing system. Also we assume the system is heavily loaded such that the probability of a server being idle is very close to zero. In Chapter 5 we relax the assumption of a closed system and model an open system with external Poisson arrival streams.

The width of the lookahead window (L) at a given iteration of the protocol is a random variable. We state without explanation that under our assumptions L is closely approximated by the minimum of N gamma distributed random variables, where each gamma is the sum of two independent, identically distributed exponential random variables with mean $1/\lambda$ (see Nicol 1993). In the analysis to follow we treat L as a constant equal to its expected value rather than as a random variable. This expected value can be obtained analytically or through sample simulation runs. Below we demonstrate analytically why using the expected value of L in our equations is very reasonable. Simulation studies support that using the expected value of L in our model provides excellent results.

As discussed above, there are two primary results in which we are interested. First, we seek the probability of a fault (at a given LP) as a function of the aggressive window size. Second, we seek the expected possible increase in the level of parallelism due to aggressive processing as a function of the size of the aggressive window. Before we are able to derive the results of interest it is necessary to develop the terminology we will use in the remainder of this chapter.

3.2.1. Terminology

Recall that the lookahead window is the simulation window defined by the basic windowing protocol, and is defined such that all events within the window can be computed concurrently without possibility of a causality error. This is guaranteed because the window is constructed such that an LP will never receive an event with a timestamp that falls within the lookahead window. When an LP completes computation all of its events within the lookahead window it blocks waiting for all other LPs to similarly complete. We define the aggressive window in order to increase the amount of processing the LPs perform before they enter into a global synchronization. While it is clear that a fully aggressive protocol allows the most processing (since it never blocks if there is processing to perform), we are also interested in maintaining the important scalability properties of the Global Windowing Algorithms. Also we are interested in minimizing the overhead costs associated with state saving and rollback. Thus as discussed above, we are interested in maximizing the advantages and minimizing the disadvantages of both the aggressive and non-aggressive approaches. For this reason we allow some aggressive processing, but place bounds on the level of aggressiveness.

Our analytic model focuses on the events within the aggressive window that an LP is allowed to process aggressively. As discussed above, these events are not processed in the non-aggressive version of the algorithm (at least not until the events fall within a lookahead window established at a later time). The processing of events within the lookahead and aggressive windows may result in the generation of new events. We make the assumption that an LP processes all of its events within both the lookahead and aggressive windows before receiving any events from the other LPs. This assumption may not be strictly true, but seems reasonable given a small aggressive window. Further, as we discuss below, this represents the worst case assumption in terms of the amount of damage created by a causality error.

We need to define several terms used in the remainder of the analysis. First we define the term "synchronization point" which is used frequently in our analysis. This refers to to the point in real time when all of the LPs have completed their processing within the previous simulation window, the LPs have synchronized globally and defined a new simulation window, and the LPs are ready to begin processing in the new simulation window. We often refer to the logical time of this global synchronization

(i.e. the ceiling of the most recently processed simulation window and the floor of the new window being established during the global synchronization) as logical time T.

Also we often make statements such as "the events in the lookahead (aggressive) window at the synchronization point". By this we are referring to those events with timestamps falling within the new lookahead (aggressive) window just established during the global synchronization. These events will not have been processed yet as they fall within the simulation window defined during the current global synchronization. As noted above, we frequently use the term logical time T to denote the floor of this new simulation window.

In order to define the other terms we frequently use it is necessary to discuss the aspects of Nicol's synchronization protocol that are critical to our analysis. In Nicol's protocol each LP "pre-sends" its completion messages. That is, the completion time of the activity, and the LP to receive this activity upon its completion, are both calculated at the time an activity enters into service. The LP to receive this activity upon its completion is notified of this future arrival *at the time the activity enters into service*. Thus for example assume activity A enters into service at LP_i at logical time t. At the point when the activity is taken off of the event list (or off of the server queue) and placed into service the completion time of the activity is computed. Assume this completion time is $s = t + d$. At this same time the LP determines the next LP to receive this activity upon its completion (logical time s). Assume the LP to receive this activity at logical time s is LP_j. At the wall-clock time that activity A is entered into service LP_i sends a message to LP_j with timestamp s. This message informs LP_j of the future arrival of activity A at logical time s. The message sent from LP_i to LP_j to inform it of the future arrival is termed a *"pre-sent" completion message*. Note that in our model these "pre-sent" completion messages are the only messages exchanged by the LPs.

In our model there are only two types of events. One is the "pre-sent" completion message described above which represents an arrival of an activity at a given LP. The other type of event is the "complete_service" message which we describe in detail below. Even though there are only two distinct event types in our system it is important to consider issues such as the real time these events are received by an LP, and into which simulation window (i.e. the aggressive or lookahead window) the timestamp of

a particular event falls into. For this reason we define terminology to classify events based on these types of considerations.

We refer to those events with timestamps that fall within the aggressive window at the synchronization point as *aggressive messages*. These are the events processed between global synchronization points in the aggressive version of the algorithm that are not processed in the non-aggressive version of the algorithm. As discussed above, new events (i.e. "pre-sent" completion messages) may be generated as a result of processing within the lookahead and the aggressive windows. Due to the construction of the lookahead window it is guaranteed that none of these new events will have a timestamp which falls within the lookahead window. It is possible however that a newly generated event will have a timestamp that falls within the aggressive window. Recall these events are sent at the physical time they are generated. Thus it is possible that an LP will have completed the processing of its events within the aggressive window, and then at a later point in real time receive another event (i.e. a "pre-sent" completion message) with a timestamp that falls within the aggressive window. We define an *arrival message* as an event generated as a result of processing within the simulation window (and given that all events are sent as soon as they are generated this event will be received by some LP in the system) such that the timestamp of the event falls within the aggressive window. Arrival messages are important because it is possible to receive such a message with a timestamp less than that of an event processed aggressively. Thus the receipt of the arrival message may constitute a causality error (i.e. out of order processing has occurred). An example should help clarify these ideas.

Assume the LPs are processing concurrently within the simulation window defined in the previous global synchronization. We show the event list of some LP_i in Figure 3.2. Note that the "X" represents an event and the number under the event represents its timestamp. Also we show the logical time of the current simulation window floor (logical time ten), the upper-bound of the lookahead window (logical time twelve), and the upper bound of the aggressive window (logical time fifteen). LP_i processes all of its events with timestamps between logical time ten and logical time fifteen (the lookahead and aggressive windows), but does not consider the events with timestamps greater than fifteen for execution. In the non-aggressive version of the algorithm LP_i would process only the event with timestamp eleven. Furth-

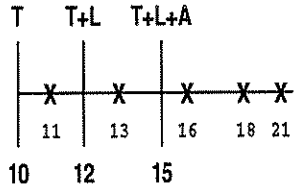

Figure 3.2 - Event List of LP_i

er, all events with timestamps less than ten will have been processed in previous iterations of the protocol. Assume LP_i has completed processing all of its events with timestamps between logical time ten and logical time fifteen, and is blocked waiting for the other LPs to similarly complete.

Assume some other LP in the system LP_j places an activity into service at logical time eleven which completes service at logical time fourteen. Further assume LP_i is to receive this activity upon its completion. When LP_j places the activity into service it calculates the completion time of the activity and the LP to receive the activity upon its completion. Thus LP_j will send a "pre-sent" completion message to LP_i with timestamp fourteen *at the time LP_j enters the activity into service*. The "pre-sent" completion message informs LP_i of this future arrival. Assume also that some LP_n sends LP_i a "pre-sent" completion message with timestamp twenty two.

The "pre-sent" completion message received by LP_i with timestamp fourteen is an arrival message because the timestamp of the message falls within the current aggressive window. While it is certainly possible that an arrival message can lead to a causality error this particular message does not since its timestamp is greater than any events processed aggressively. Thus no out of order processing has occurred. The "pre-sent" completion message sent by LP_n is *not* an arrival message since its timestamp falls outside of the current aggressive window. Furthermore, LP_i processes the arrival event when it is received (and the event from LP_n is not processed in this iteration of the algorithm).

Now assume the LPs have all completed their processing within the simulation window and have synchronized globally to determine the next simulation window. We term this the synchronization point, and the logical time of this synchronization is logical time fifteen, the upper bound of the previous simulation window (and the lower bound of the new simulation window to be defined). Assume the new loo-

kahead window extends from logical time fifteen to logical time seventeen and thus has a width of two logical units. Further assume the aggressive window extends from logical time seventeen to logical time nineteen. Now that the new simulation window is defined we refer to the floor of this new window as logical time T. Thus logical time T=15 in this iteration of the algorithm. This is shown in Figure 3.3.

At this point the LPs have synchronized globally, defined the new simulation window and are ready to begin processing concurrently. Note LP_i has one event within the new lookahead window and one event within the new aggressive window. These are the events we refer to as "being in the lookahead (aggressive) window at the synchronization point (or logical time T)". That is, these are the events with timestamps that fall within the new lookahead (aggressive) window just defined during the global synchronization. The event with timestamp eighteen is also referred to as *an aggressive message* since its timestamp falls within the aggressive window. It is termed an aggressive message because the (aggressive) processing of this event may result in a causality error. After the new simulation window is defined, the LPs again process their events with timestamps falling within the new simulation window.

Recall our assumption that all events within the aggressive window have been processed by the time an LP receives its first arrival message. This is the worst case assumption as we now demonstrate. Assume an LP has two events within the aggressive window, one with a timestamp of twelve, and one with a timestamp of fifteen. Further, assume the LP receives an arrival message with a timestamp of thirteen. If the LP has processed both aggressive messages before receiving the arrival message, then the event with timestamp fifteen will be invalidated. If we do not assume all aggressive messages are processed before the LP receives its first arrival message, then the LP may receive the arrival message *before*

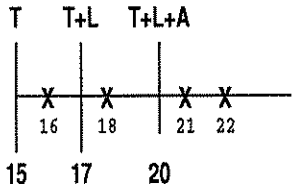

Figure 3.3 - Event List of LP_i

the event with a timestamp of fifteen is processed. In this case the arrival message would be processed first, and then the event with timestamp fifteen would be processed. Thus the receipt of the arrival message would not result in a causality error, and the event with a timestamp of fifteen would not have to be reprocessed. Since we cannot determine the real time order in which aggressive messages will be processed and the arrival messages received, we make the worst case assumption regarding this real time order. Now that we have defined our terminology, and clarified our assumptions regarding the real time ordering of events, we can proceed with our analysis.

In order to calculate the expected number of aggressive messages processed without a causality error, and to calculate the probability of a causality error at a given LP, there are several steps that must be accomplished. First, we need to determine the distribution of the number of messages in the aggressive window at the synchronization point (logical time T). This is the number of messages that an LP can process aggressively and represents the extra processing performed in the aggressive version of the algorithm between global synchronization points. Second, we need to determine the distribution of the number of messages in the lookahead window at logical time T. This determines the number of messages processed in the non-aggressive version of the algorithm between global synchronization points. Third, we need the distribution of the number of messages that subsequently arrive in the aggressive window (i.e. the arrival messages). As mentioned above, these messages are important because the receipt of such a message can potentially result in a causality error. Fourth, we need the distribution of the timestamps of messages in the aggressive window at logical time T. That is, we need to determine how the timestamps of messages in the aggressive window at the synchronization point are distributed within the window. Finally, we need the distribution of the timestamps of arrival messages. This tells us how the timestamps of arrival messages are distributed within the aggressive window. The timestamp distribution of the aggressive messages and the arrival messages determine the probability that an arrival message invalidates a message processed in the aggressive window. In the following sections we determine the distributions of each of these items.

3.2.2. Number of Messages in Aggressive Window at the Synchronization Point

For our analysis, the important aspect of the "pre-sending" of completion messages is that there will be one "pre-sent" completion message for every LP that is busy. Further, if the system is synchronized at some logical time T, each such "pre-sent" message will have a timestamp greater than or equal to T. This is because Nicol's algorithm (as well as our modified algorithm) guarantees that once the system synchronizes at logical time T, there are no unprocessed events in the system with timestamp less than T.

For the remainder of this chapter we define K as the random variable representing the number of "pre-sent" completion messages received by a given LP. Our first task in determining the distribution of the number of messages in the aggressive window at logical time T is to determine the probability that a particular LP receives $K=k$ "pre-sent" completion messages. Recall the assumption that the probability of an idle server is very low. This implies that with high probability each LP is busy with an activity that began service at logical time $s < T$, completes service at logical time $T' > T$, and whose completion message was pre-sent. As discussed above, this "pre-sent" completion message represents the completion time of the activity currently receiving service (and thus the time the activity will arrive at the given LP). Due to the assumption that each LP is equally likely to receive a given "pre-sent" completion message the number of such messages received by a given LP is binomially distributed.

$$P\{\ LP\ receives\ k\ completion\ messages\ \} = (1/N)^k\ (\frac{(N-1)}{N})^{N-k}\ \frac{N!}{k!(N-k)!} \tag{3.1}$$

In Equation (3.1) there is a large number of independent trials (N) and the probability of success (P) at any trial is small (1/N). Let $\lambda_1 = NP = 1$, the probability of success at any trial times the number of trials. As shown by Breiman (1986), the Binomial distribution is approximated very closely by that of the Poisson distribution (with rate $\lambda_1 = NP$) in the case where P is small and N is large. This is the case in Equation (3.1) and we conclude that the probability of a given LP receiving k pre-sent completion messages is closely approximated by the Poisson distribution with rate $\lambda_1=1$.

$$P\ \{\ LP\ receives\ k\ completion\ messages\ \} \approx \frac{e^{-1}}{k!} \tag{3.2}$$

Now consider the distribution of the timestamp of a "pre-sent" completion message that has been received by a given LP up to the time the LPs synchronize at logical time T. Due to the memoryless pro-

perty of the exponential distribution, the residual service time of the activity is also exponentially distributed. Thus the distribution of the timestamp of a "pre-sent" completion message, given that the timestamp is greater than logical time T, is also exponentially distributed. This implies we can view the system as probabilistically restarting at logical time T^*. For this reason in the equations that follow we define the aggressive window as falling between logical times L and $L+A$ rather than as falling between logical times $T+L$ and $T+L+A$ (i.e. we set $T=0$). Similarly, we often set $L = 0$ when we are discussing events that occur within the aggressive window. Thus we often define the aggressive window as falling in the interval $0 .. A$ rather than in the interval $L .. L+A$.

Given the timestamp distribution of "pre-sent" completion messages, and the distribution of the number of such messages received, we can compute the distribution of the number of messages with timestamps falling within the aggressive window $(L,L+A)$. Consider the probability that exactly one of the k "pre-sent" completion messages received by an LP falls within the aggressive window. In order for this to occur, exactly one of the k messages received by the LP must have a timestamp within the aggressive window, and all of the other $k-1$ messages must have timestamps outside of the aggressive window. Given that we know the timestamp distribution of "pre-sent" completion messages we can calculate the probability that such a message falls within the aggressive window. The probability that a "pre-sent" completion message with timestamp $X = x$ falls within the aggressive window is

$$P(x \text{ in Agg. Window}) = e^{-\lambda L} - e^{-\lambda(L+A)} \tag{3.3}$$

The probability that $X=x$ falls outside of the aggressive window is one minus the proability that it falls within the window.

$$P(x \text{ not in Agg. Window}) = 1-(e^{-\lambda L}-e^{-\lambda(L+A)}) \tag{3.4}$$

Let I be the random variable representing the number of messages received by a given LP with timestamps which fall within the aggressive window. The probability that $I = i$ messages fall within the aggressive window given that an LP receives $K=k$ messages is

*We note that logical time T is defined by the protocol and thus it is an approximation to state that the system probabilistically restarts at logical time T. As will be seen however this approximation provides excellent results. Nicol (1993) may be consulted for a discussion of why this is an approximation.

$$P(I=i \mid K=k) = (e^{-\lambda L}-e^{-\lambda(L+A)})^{i}\,(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{k-i}\frac{k!}{i!(k-i)!}. \tag{3.5}$$

In Equation (3.5), the $(e^{-\lambda L}-e^{-\lambda(L+A)})^{i}$ term is the probability of i independent messages falling within the aggressive window. The $(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{k-i}$ term is the probability that the other $k-i$ messages fall outside of the aggressive window. The $\frac{k!}{i!(k-i)!}$ term is the number of combinations of i messages out of k total messages that can fall within the aggressive window.

Equation (3.5) gives the probability of $I=i$ messages falling within the aggressive window given that k messages are received. In order to uncondition this expression we need to sum over all possible values of K times the probability of $K=k$.

$$P\{i \text{ messages in Agg. Window}\} = \sum_{k=i}^{N}\frac{e^{-1}}{k!}(e^{-\lambda L}-e^{-\lambda(L+A)})^{i}\,(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{k-i}\frac{k!}{i!(k-i)!}. \tag{3.6}$$

We can rewrite Equation (3.6) in the following way.

$$P\{i \text{ messages in Agg. Window}\} = \frac{e^{-1}}{i!}(e^{-\lambda L}-e^{-\lambda(L+A)})^{i}\sum_{k=i}^{N}\frac{(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{k-i}}{(k-i)!} \tag{3.7}$$

Let $j=k-i$ and rewrite the summation.

$$\sum_{k=i}^{N}\frac{(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{k-i}}{(k-i)!} = \sum_{j=0}^{N-i}\frac{(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{j}}{j!} \tag{3.8}$$

Recall the following identity.

$$\sum_{j=0}^{\infty}\frac{x^{j}}{j!} = e^{x}$$

Thus as N goes to infinity (and $i \ll N$), we have

$$\sum_{j=0}^{N-i}\frac{(1-e^{-\lambda L}+e^{-\lambda(L+A)})^{j}}{(j)!} \simeq e^{(1-e^{-\lambda L}+e^{-\lambda(L+A)})}. \tag{3.9}$$

We use this result as an approximation.

Equation (3.9) is very good for large N because $\sum_{j=0}^{\infty}\frac{x^{j}}{j!}$ converges to e^{x} very quickly when $x<1$. In Equation (3.8) $1-e^{-\lambda L}+e^{-\lambda(L+A)}$ is less than one as it represents the probability of an exponential random variable being outside of a given range.

Rewrite Equation (3.7) using the approximation given in Equation(3.9).

$$P\{i \text{ Messages in Agg. Window}\} \approx \frac{(e^{-\lambda L} - e^{-\lambda(L+A)})^i}{i!} \; e^{-(e^{-\lambda L} - e^{-\lambda(L+A)})} \tag{3.10}$$

We conclude that a very reasonable approximation of the probability distribution for the number of messages in the aggressive window at time T is the Poisson distribution with parameter $e^{-\lambda L} - e^{-\lambda(L+A)}$.

Also we are interested in the number of messages in the lookahead window at the synchronization point. Given (our approximation) that the system probabilistically restarts at logical time T, we can view the lookahead window as extending from logical time $0 .. L$. To compute the number of messages in a window extending from logical time $0 .. L$ we substitute 0 for L, and L for A in Equation (3.10). After making this substitution we see that a reasonable approximation for the number of messages in the lookahead window at the synchronization point is the Poisson distribution with parameter $1-e^{-\lambda L}$.

$$P\{j \text{ messages in lookahead window}\} \approx \frac{(1-e^{-\lambda L})^j}{j!} \; e^{-(1- e^{-\lambda L})} \tag{3.11}$$

3.2.2.1. Complete_Service Message

In the sections above we derived the probability distribution of the number of "pre-sent" completion messages in the aggressive window at logical time T. In addition to pre-sending the completion message to the receiving LP, an LP also schedules a completion message on its own event list. This second type of event we refer to as a "complete_service" message. This event represents the completion of the service requirements of the activity currently receiving service. The processing of the "complete_service" message consists of determining the next activity to receive service, determining the LP to receive this next activity upon its completion, sending the "pre-sent" completion message to the LP which will receive this activity upon its completion, and any statistics gathering required for the simulation. Thus for every server that is busy there is one "pre-sent" completion message (sent to the LP to receive the activity upon its completion) and one "complete_service" message scheduled on its own event list to signify that the activity has completed its service requirements. Note that since LPs "pre-send" their completion messages, there is no need to send another message when the "complete_service" message is processed. This is because the "pre-sent" completion message has already informed the LP to receive this activity upon its completion ion the future arrival.

Consider the "complete_service" message. If the timestamp of this message falls within the aggressive window it affects the number of messages processed aggressively. If it falls within the lookahead window it affects the number of unconditional messages processed. Due to the relatively small number of messages processed per window (per LP) it is important to consider the effects of the "complete_service" message.

Recall our assumption that the probability of an idle server is very low. Therefore it is with high probability that each LP will always have a "complete_service" message scheduled somewhere on its event list. Our analysis of the closed system assumes this will always be the case. As will be seen in Chapter 5, this assumption can be modified to reflect the probability of a "complete_service" message (i.e. the probability that a server is busy) without difficulty.

Due to the memoryless property of the exponential distribution, the residual service time of the activity receiving service at the synchronization point is also exponentially distributed*. Thus the timestamp of the "complete_service" message denoting this service completion is exponentially distributed from logical time T. The probability that the "complete_service" message falls within the lookahead window (given that the timestamp of the message is greater than T) is $1-e^{-\lambda L}$ and the probability that it falls within the aggressive window is $e^{-\lambda L}-e^{-\lambda(L+A)}$. Given this, we can recompute the distribution for the number of messages within the aggressive window at the synchronization point for a given LP.

Let J as the random variable representing the total number of messages within the aggressive window at the synchronization point, including the "pre-sent" completion messages and the "complete_service" message. In order to have $J=0$ messages in the aggressive window the LP must have no "pre-sent" completion messages *and* the "complete_service" message must fall outside of the aggressive window. We give this probability below.

$$P(J=0) = P(i=0,\overline{C}) = e^{-(e^{-\lambda L}-e^{-\lambda(L+A)})}(1-e^{-\lambda L}+e^{-\lambda(L+A)}). \tag{3.12}$$

In Equation (3.12) the $P(i=0)$ term is the probability of having zero "pre-sent" completion messages in the aggressive window (this probability is given in Equation (3.10)). The $\overline{C}$ term is the probability of the

* We again note that it is an approximation to state that the residual service is exponentially distributed. This is because the synchronization point T is chosen by the protocol, and thus there is some conditioning. As will be seen however the approximation is quite good.

"complete_service" message falling outside of the aggressive window.

Now consider the probability of $J = j$ messages within the aggressive window where $j \geq 1$.. One way this can occur is for the LP to have $i=j$ "pre-sent" completion messages within the aggressive window, and not have the "complete_service" message fall within the aggressive window. Alternatively, the LP may have $i=j-1$ "pre-sent" completion messages *and* the "complete_service" message within the aggressive window. This probability is given below.

$$P(J=j, j>0) = P(i=j-1, C) + P(i=j, \overline{C}) = \tag{3.13}$$

$$\frac{(e^{-\lambda L} - e^{-\lambda(L+A)})^{j-1}}{j-1!} e^{-(e^{-\lambda L} - e^{-\lambda(L+A)})} e^{-\lambda L} - e^{-\lambda(L+A)}$$

$$+ \frac{(e^{-\lambda L} - e^{-\lambda(L+A)})^{j}}{j!} e^{-(e^{-\lambda L} - e^{-\lambda(L+A)})} (1 - e^{-\lambda L} - e^{-\lambda(L+A)}).$$

In Equation (3.13) $I=i$ is the number of "pre-sent" completion messages within the aggressive window and C is the probability that the "complete_service" message falls within the aggressive window.

Similar arguments show that the probability of $M=0$ messages in the lookahead window is

$$P(M=0) = e^{-(1-e^{-\lambda L})} (1 - e^{-\lambda L}). \tag{3.14}$$

The probability of $M \geq 1$ messages in the lookahead window is

$$P(M=m, m \geq 1) = \frac{(1-e^{-\lambda L})^{m}}{m!} e^{-(1-e^{-\lambda L})} e^{-\lambda L} + \frac{(1-e^{-\lambda L})^{m-1}}{m-1!} e^{-(1-e^{-\lambda L})} (1 - e^{-\lambda L}). \tag{3.15}$$

It is important to remember that Equation (3.13) gives the probability of $M=m$ messages (events) in the aggressive window at the synchronization point. These messages are important because they represent the extra processing between global synchronization points performed in the aggressive version of the algorithm that are not performed in the non-aggressive version.

3.2.3. Distribution of Arrival Messages

As discussed in previous sections, the non-aggressive algorithm guarantees that once the lookahead window is defined an LP cannot receive an event with a timestamp that falls within the lookahead window. As noted, this implies that all LPs can process concurrently all events within the lookahead window without any possibility of a causality error. It is quite possible however that events processed within the lookahead window will generate events with timestamps that fall within the aggressive window. It is also

possible that processing events within the aggressive window will generate events with timestamps that again fall within the aggressive window. In either case, events that are received by an LP with a timestamp that falls within the aggressive window are termed *arrival* messages. We now determine the distribution of the number of arrival messages received by a given LP.

The duration of an activity at a server is exponentially distributed with mean $1/\lambda$. Due to our assumption of a heavily loaded system we know that the output of a given server is a Poisson process with rate λ. Due to the independence of the N servers in the system, the system output will be the merging of N independent Poisson streams. Thus the system output rate will be Poisson distributed with rate $N\lambda$. As each LP is equally likely to receive an activity upon completion the system output stream forks into N independent Poisson streams. The input rate to any given LP is therefore Poisson with rate $\frac{N\lambda}{N} = \lambda$.

We now have enough information to determine the probability distribution of the number of arrival messages. As discussed above, the total input process into a given LP is Poisson with rate λ. The aggressive window has a width of A logical time units, and thus the total number of messages that will fall within the aggressive is Poisson distributed with rate λA. There are two ways in which messages can land within the aggressive window. First, messages can be present in the aggressive window at the synchronization point. Recall in Equation (3.10) we showed that the number of messages present in the aggressive window at the synchronization point is Poisson distributed with rate $e^{-\lambda L} - e^{-\lambda(L+A)}$. Second, messages can arrive into the aggressive window as a result of the LPs processing through the aggressive window (i.e. the arrival messages).

Let X be the random variable representing the total number of messages that fall within the aggressive window, let Y be the random variable denoting the number of messages in the aggressive window at the synchronization point, and let Z be the random variable representing the number of arrival messages. Then $X = Y + Z$. By the additive property of the Poisson distribution, X is Poisson distributed with a rate equal to the sum of the rates of Y and Z. We know Y is Poisson distributed with rate $e^{-\lambda L} - e^{-\lambda(L+A)}$. Let $\lambda_{arrival}$ be the rate of Z. Then

$$\lambda A = \lambda_{Arrival} + (e^{-\lambda L} - e^{-\lambda(L+A)}) \tag{3.16}$$

We now solve for $\lambda_{Arrival}$.

$$\lambda_{Arrival} = \lambda A - [e^{-\lambda L} - e^{-\lambda(L+A)}] \tag{3.17}$$

We conclude that the distribution for the number of arrival messages is Poisson with rate $\lambda A - (e^{-\lambda L} - e^{-\lambda(L+A)})$. Let J be the random variable representing the number of arrival messages. Then

$$P(J=j) = \frac{(\lambda A - (e^{-\lambda L} - e^{-\lambda(L+A)}))^j}{j!} e^{-(\lambda A - (e^{-\lambda L} - e^{-\lambda(L+A)}))} \tag{3.18}$$

3.2.4. Timestamp Distribution

In this section we derive the timestamp distribution for the aggressive messages: those events with timestamps that fall within the aggressive window at the synchronization point. Also, we derive the timestamp distribution for arrival messages. We need these distributions in order to predict the probability of a causality error, and in order to predict the expected improvement due to aggressive processing. We begin by examining those events with timestamps falling within the aggressive window at the synchronization point.

We seek the timestamp distribution for the events within the aggressive window at logical time T. As discussed above, there are two types of events that may be in the aggressive window. First, there may be "pre-sent" completion messages representing the future arrival of an activity. The other type of event is the "complete_service" message which the given LP has scheduled on its own event list. Recall that this event denotes the completion of the activity currently receiving service at the given LP. We first consider the "pre-sent" completion messages, representing the future arrival of an activity to the given LP.

As noted, a "pre-sent" completion message (in the aggressive window at the synchronization point) represents the completion times of some activity that began service at logical time $s < T$ and completes service at logical time $T' > T$. We are interested in the distribution of the residual service time given that the activity is still in service at logical time T. Recall that all service times are *iid* exponential random variables with mean $1/\lambda$. Due to the memoryless property of the exponential, the residual service time will also be exponentially distributed. Thus if the timestamp of the "pre-sent" completion message falls within the aggressive window it will be a conditional exponential. That is, the timestamp is exponentially

distributed conditioned on being within the aggressive window. This distribution is given below.

$$pdf\,(x \mid L \le x \le L+A) = \frac{\lambda e^{-\lambda x}}{e^{-\lambda L} - e^{-\lambda(L+A)}} \tag{3.19}$$

Equation (3.19) is obtained from the definition of conditional probability. The numerator is the exponential density function, and the denominator is the probability that an exponentially distributed random variable falls within the aggressive window (given that it is greater than the synchronization point T). If we let $L = 0$ in Equation (3.19) we can rewrite the *pdf* in the following way.

$$pdf\,(x \mid 0 \le x \le A) = \frac{\lambda e^{-\lambda x}}{1 - e^{-\lambda A}}$$

Given the memoryless property of the exponential, we can use this alternate form of the *pdf* when we know that the timestamp in question lies within the aggressive window. We generally use this form of the *pdf* in the analysis that follows.

This same analysis applies to the timestamp of a "complete_service" message that falls within the aggressive window (at the synchronization point). The "complete_service" message represents the completion of an activity that begins service at logical time $s < T$, and completes service at logical time $T' > T$. Given that the service time is exponentially distributed, the residual service time is also exponentially distributed. Thus if the completion time of the activity falls within the aggressive window, the timestamp of the "complete_service" message denoting this completion will be a conditional exponential, again conditioned on being within the aggressive window. The timestamp distribution of a "complete_service" message that falls within the aggressive window will therefore also have the distribution given in Equation (3.19).

The timestamp distribution of an arrival message is more difficult to determine. Recall that an arrival message is a "pre-sent" completion message received by an LP such that the timestamp of the message falls within the aggressive window. An example should serve to demonstrate the timestamp distribution of an arrival message.

Assume LP_i has a "complete_service" message with timestamp t_1 that falls within the lookahead window. As discussed, this "complete_service" message represents the completion time of the activity currently receiving service at LP_i. Assume LP_i processes the "complete_service" message with times-

tamp t_1, and that there exists another activity, A_1, to enter into service. The processing of the "complete_service" message involves (among other things) determining the LP to receive activity A_1 upon its completion. The completion time of activity A_1 is the sum of logical time t_1 (the logical time it enters into service) and an exponential random variable with mean $1/\lambda$. Let $t_2 = t_1 + \xi$ be the completion time of activity A_1, where ξ is an exponential random variable with mean $1/\lambda$. Assume this completion time falls within the aggressive window. As discussed, one result of processing the "complete_service" message is that LP_i will send a "pre-sent" completion message with timestamp t_2 to the LP to receive activity A_1 upon its completion. Also, LP_i will schedule another "complete_service" message on its own event list with timestamp t_2 denoting the service completion of activity A_1. This is shown in Figure 3.4.

Note the "pre-sent" completion message sent by LP_i (with timestamp t_2) will be an arrival message to the LP which receives this message. We seek the timestamp distribution of this arrival message, or alternatively, we seek the distribution for timestamp t_2. We know t_2 falls within the aggressive window. Due to the memoryless property of the exponential, we know that the distribution of t_2, given that it falls within the aggressive window, is a conditional exponential, conditioned on being within the aggressive window. We define a *first generation arrival message* as an arrival message with a timestamp that is the sum of a) the timestamp of a "complete_service" message that falls within the lookahead window, and b) an exponential random variable with mean $1/\lambda$. Note the timestamp distribution of a first generation arrival message is the same as that given in Equation (3.19). Clearly the "complete_service" message with timestamp t_2 (the "complete_service" message which produced the first generation arrival message) has this same timestamp distribution. Also we term a "complete_service" message with a timestamp that is a

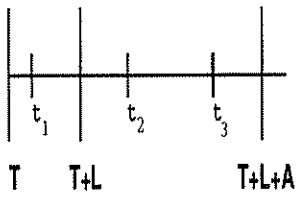

Figure 3.4 - Complete Service Messages

conditional exponential a *first generation "complete_service" message*. Thus first generation arrival messages (and first generation "complete_service" messages), and events with timestamps that fall within the aggressive window at the synchronization point, have the same timestamp distribution.

Now assume LP_i processes the first generation "complete_service" message (with timestamp t_2) and there exists another activity, A_2, to enter into service. As a result of processing this "complete_service" message LP_i will generate another "complete_service" message denoting the service completion of activity A_2. Let $t_3 = t_2 + \xi$ be the completion time of activity A_2, and thus the timestamp of the "pre-sent" completion message sent to the LP to receive A_2 upon its completion, as well as the "complete_service" message denoting the service completion of A_2. Assume t_3 also falls within the aggressive window, and thus the LP that receives activity A_2 upon its completion will receive an arrival message with timestamp t_3. We are interested in the timestamp distribution of this arrival message.

Timestamp t_3 is the sum of a conditional exponential and an exponential random variable with mean $1/\lambda$. Thus it is also a conditional exponential, in this case conditioned on being between the timestamp of the first generation "complete_service" message which produced it and the upper bound of the aggressive window. We term the arrival message with such a distribution a *second generation arrival message*. Similarly, we term the "complete_service" message with timestamp t_3 a *second generation "complete_service" message*. We give the *pdf* of this timestamp distribution below.

$$pdf\,(t_3) = \int_0^A \frac{\lambda e^{-\lambda t_3}}{e^{-\lambda t_2} - e^{-\lambda A}} \; \frac{\lambda e^{-\lambda t_2}}{1 - e^{-\lambda A}} \, dt_2 \tag{3.20}$$

In Equation (3.20) the first term is the conditional exponential distribution of t_3, conditioned on being between logical time t_2 and the upper bound of the aggressive window. Then we uncondition this distribution by integrating over all possible values of t_2 times the probability of t_2. This is the second term where t_2 is a conditional exponential, conditioned on being within the aggressive window.

There is one other way that an LP can produce an arrival message with the distribution given in Equation (3.20). Recall A_1 is the activity receiving service at the synchronization point. If A_1 completes service within the aggressive window rather than within the lookahead window, the timestamp of the corresponding "complete_service" message will also be a conditional exponential, and have the times-

tamp distribution given in Equation (3.19). For this reason it would also be what we term a first generation "complete_service" message. If the activity placed into service as a result of processing this "complete_service" message also falls within the aggressive window it would have the timestamp distribution given in Equation (3.20).

Clearly there can be third and higher order generation arrival messages as well. The probability of such an arrival message is dependent upon the size of the aggressive window.

As can be seen, the timestamp distribution of an arrival message is dependent upon the circumstances under which the arrival message is generated. If it is a first generation arrival message, it will have a timestamp that is a conditional exponential. If it is a second generation arrival message, it will have the timestamp distribution given in Equation (3.20) and so forth. We give the first three terms of this conditional distribution below. In this equation t is the timestamp of an arrival message, and timestamps t_2 and t_3 have the distributions given above.

$$pdf(t) = \tag{3.21}$$

$$\frac{\lambda e^{-\lambda t}}{1-e^{-\lambda A}} P(\textit{Arrival is 1st gen. arrival})$$

$$+\int_0^A \frac{\lambda e^{-\lambda t}}{e^{-\lambda t_2}-e^{-\lambda A}} \frac{\lambda e^{-\lambda t_2}}{1-e^{-\lambda A}} dt_2 \, P(\textit{Arrival is 2nd gen. arrival})$$

$$+\int_0^{AA}\int_{0t_3} \frac{\lambda e^{-\lambda t}}{e^{-\lambda t_3}-e^{-\lambda A}} \frac{\lambda e^{-\lambda t_3}}{e^{-\lambda t_2}-e^{-\lambda A}} \frac{\lambda e^{-\lambda t_2}}{1-e^{-\lambda A}} dt_3 \, dt_2 \, P(\textit{Arrival is 3rd gen. arrival})...$$

In order to uncondition this expression we need to calculate the probability of an *Nth* generation arrival message. We do this in the following section.

3.2.4.1. Probability of an *Nth* Generation Arrival Message

An LP produces a first generation arrival message when it processes a "complete_service" message with a timestamp that falls within the lookahead window and the completion time of the next activity to receive service falls within the aggressive window. Consider the probability that an LP has a "complete_service" message with a timestamp that falls within the lookahead window. Recall our assumption that with high probability each server will always be busy, and thus with high probability each

LP will have a "complete_service" message somewhere on its event list. We seek the probability that this "complete_service" message has a timestamp that falls within the lookahead window. Equivalently, we seek the probability that an activity which is receiving service at logical time T (the synchronization point) completes service in the interval $T,T+L$. Due to the memoryless property of the exponential, the probability that the activity completes in the interval $T,T+L$, given that the completion time of the activity is greater than T, is $1 - e^{-\lambda L}$.

Given a first generation "complete_service" message, the next activity to enter into service must complete within the aggressive window in order to produce a first generation arrival message. Note the lookahead window is constructed such that any* activity which begins service within the window will have a completion time greater than L, the upper bound of the lookahead window. Thus we seek the probability that the activity completes in the interval $L,L+A$, given that this completion time is greater than L. Due to the memoryless property of the exponential this probability is $1 - e^{-\lambda A}$. We can now calculate the probability that a given LP produces a first generation arrival message.

$$P(1st\ generation\ arrival) = (1 - e^{-\lambda L})\,(1 - e^{-\lambda A}). \tag{22}$$

The probability that an LP produces a second generation arrival message is computed in a similar manner. As discussed above, a second generation arrival message can be produced in two ways. First, the LP can process the "complete_service" message within the lookahead window, this activity completes within the aggressive window (thus producing a first generation arrival and "complete_service" message), and the next activity to enter into service also completes within the aggressive window. Second, the "complete_service" message representing the completion time of the activity receiving service at the synchronization point may fall within the aggressive window (and is therefore a first generation "complete_service" message). If the next activity to enter into service also completes within the aggressive window this will cause a second generation arrival message to be produced. In either case, the LP must process a first generation "complete_service" message, and the next activity to enter into service

* This is not strictly true since the lookahead window is defined such that exactly one activity will complete service at logical time $T + L$, i.e. the upper bound of the lookahead window. We ignore this one completion time for the moment and discuss it in more detail below. The reader interested in a detailed description of the construction of the lookahead window is directed to Nicol (1993).

must complete within the aggressive window. We first compute the probability that an activity placed into service as a result of processing a first generation "complete_service" message completes within the aggressive window.

We seek the probability that an activity which begins service as a result of processing a first generation "complete_service" message will have a completion time that again falls within the aggressive window. Recall that one aspect of processing a "complete_service" message is to determine the completion time of the next activity to enter into service. As we have shown, the completion time of the next activity to enter into service is the sum of the timestamp of the "complete_service" message and an exponential random variable with mean $1/\lambda$. Given a first generation "complete_service" message with timestamp $S=s$ within the aggressive window, we seek the probability that the new timestamp $s+\xi$ is less than A. For a particular $S=s$ this probability is shown below.

$$P(s+\xi < A \mid S=s) = 1-e^{-\lambda(A-s)}$$

In order to uncondition we integrate this probability over all possible values of $S=s$ multiplied by the probability of $S=s$. To determine the timestamp distribution of S, we note that (because it is a first generation "complete_service" message) it represents the completion time of an activity that entered into service at logical time $t \mid t < T+L$, and completes within the aggressive window. Due to the memoryless property of the exponential, the timestamp of a first generation "complete_service" message, given that this timestamp is greater than the lower bound of the aggressive window, will be a conditional exponential, conditioned on being within the aggressive window. This distribution is given in Equation (3.19). We compute the probability that an LP produces a second generation arrival message, given that it processes a first generation "complete_service" message, below.

$$P(2nd\ generation \mid 1st\ gen\ CS) = \int_0^A (1-e^{-\lambda(A-s)}) \frac{\lambda e^{-\lambda s}}{(1-e^{-\lambda A})} ds = \frac{1-(\lambda A e^{-\lambda A} + e^{-\lambda A})}{1-e^{-\lambda A}} \qquad (3.22)$$

We now compute the probability of a first generation "complete_service" message. As we have discussed, there are two ways that a first generation "complete_service" message can occur. First, the activity receiving service at the synchronization point completes within the lookahead window and the next activity to enter into service completes within the aggressive window. This is exactly the probability that

an LP produces a first generation arrival message given in Equation (3.22). Alternatively, the activity receiving service at the synchronization point may complete within the aggressive window rather than the lookahead window (or outside of both windows). Due to the memoryless property of the exponential, the probability that the activity completes within the aggressive window given that it is still in service at logical time T is $e^{-\lambda L} - e^{-\lambda(L+A)}$. The unconditioned probability that an LP produces a second generation arrival message is thus

$$P(2nd\ generation\ arrival) = ((1-e^{-\lambda L})(1-e^{-\lambda A}) + e^{-\lambda L} - e^{-\lambda(L+A)})\ (\frac{1-(\lambda A e^{-\lambda A} + e^{-\lambda A})}{1-e^{-\lambda A}}). \quad (3.23)$$

The probability that an LP produces a third or higher order generation arrival message is computed in a similar manner and is not discussed. The probability that an LP produces an arrival message is the sum of the probability of producing a first generation arrival message, plus the probability of producing a second generation arrival message, and so forth.

$$P(Arrival\ Message) = P(1st\ generation) + P(2nd\ generation) + P(3rd\ generation)... \quad (3.24)$$

We can now compute the distribution of the timestamp of an arrival message. Assume an arrival message has timestamp $X=x$. The distribution of x is a conditional exponential if the arrival message is a first generation arrival message. It is an exponential increment from a conditional exponential if it is a second generation arrival message, and so forth. We give this distribution below.

$$pdf(x) = \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda A}} \frac{P(1st\ gen)}{P(Arrival)} + \int_0^A \frac{\lambda e^{-\lambda x}}{e^{-\lambda t}-e^{-\lambda A}} \frac{e^{-\lambda t}}{1-e^{-\lambda A}} dt \frac{P(2nd\ generation)}{P(Arrival)} + \quad (3.25)$$

$$\iint_{0\,t}^{AA} \frac{\lambda e^{-\lambda x}}{e^{-\lambda s}-e^{-\lambda A}} \frac{e^{-\lambda s}}{e^{-\lambda t}-e^{-\lambda A}} \frac{\lambda e^{-\lambda t}}{1-e^{-\lambda A}} ds\, dt \frac{P(3rd\ Generation)}{P(Arrival)} \cdots$$

3.2.4.2. Timestamp Distribution Approximation

The timestamp distribution given in Equation (3.25) is quite complex, and using this distribution in our analysis would make the problem intractable. For this reason we seek an approximation to the timestamp distribution.

As can be seen from Equation (3.25), the complexity of the timestamp distribution increases rapidly as the generation of the arrival message increases. Clearly the probability of a second or higher order

generation arrival message is dependent upon the size of the aggressive window. Thus a reasonable approach to approximate the timestamp distribution is to constrain the size of the aggressive window such that the contribution of higher order generation arrival messages to the total probability of an arrival message is small. Given a small probability of higher order generation arrival messages, we can ignore these timestamp distributions in our calculations. Thus we can approximate the timestamp distribution by assuming all arrival messages are lower order arrival messages (i.e. either first or second generation). This simplifies the analysis significantly.

Clearly our analysis would be most simplified by assuming all arrival messages are first generation arrival messages. As will be seen, for most of our analysis this approximation gives excellent results for a fairly wide range of aggressive window sizes. In all cases this approximation bounds the measurement of interest. As will be seen, our analysis can become quite complex even when assuming all arrival messages are first generation arrival messages. In order to maintain the tractability of our analysis we generally limit our focus to first generation arrival messages. However, we do demonstrate the impact of assuming a more complex timestamp distribution. During the course of this investigation we make it clear whether we are assuming first or first and second generation arrival messages. Also, we make clear the impact of the approximation.

If we assume all arrival messages are first generation, the timestamp of an arrival message will be a conditional exponential. This is the same distribution as given in Equation (3.19) above. If we assume an arrival message is either first or second generation, we need to condition the timestamp distribution on the probability that the message will be a first or second generation arrival message. In this case we sum the probability of a first and second generation arrival message, and use this as an approximation to the probability of an arrival message. Then we assign the conditional distribution based on the relative contribution of the two components.

In order to demonstrate the error of this approximation we graph the probability that an arrival message is a first generation arrival message for aggressive window sizes between 10% and 100% of the mean service time (see Figure 3.5). These are the observed values for a system with N=1500 LPs. In Figure 3.6 we show the probability (for the same system) that an arrival message is either a first or second

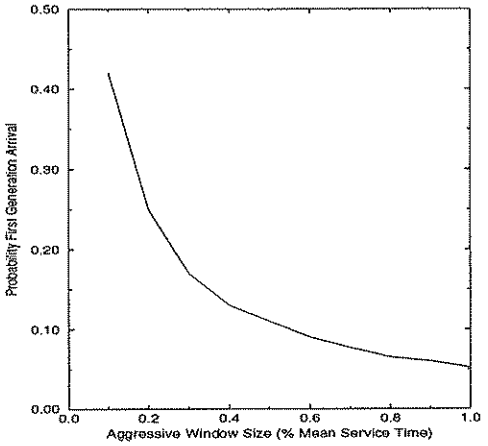

Figure 3.5 - Probability of a First Generation Arrival

generation message.

As can be seen, the probability that an arrival message is a first generation message decreases quickly as the size of the aggressive window is increased. This is due to two factors. First, recall that (with high probability) the server will be busy at the synchronization point. As the size of the aggressive window increases, the probability that the corresponding "complete_service" message falls within the aggressive window (rather than within the lookahead window) increases. As discussed above, a necessary

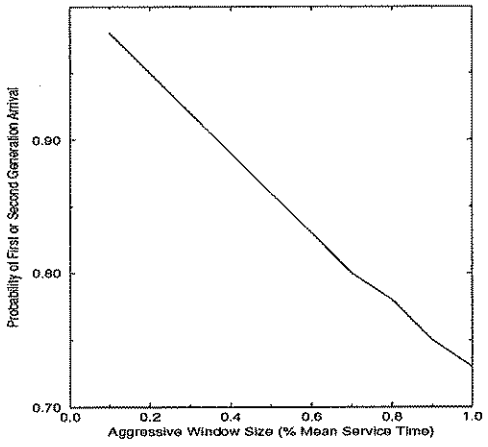

Figure 3.6 - Probability of a First or Second Generation Arrival

condition for a first generation arrival message is that the "complete_service" message fall within the lookahead window. Thus as the probability of this decreases, the probability of a first generation arrival message decreases. Secondly, as the size of the aggressive window increases, the probability that an activity which begins service within the aggressive window completes service within the aggressive window increases. Thus the probability of a second or higher order generation arrival message increases.

As the size of the aggressive window approaches the mean service time we begin to see more third and higher order generation arrival messages. With an aggressive window size equal to the mean service time approximately 28% of the arrival messages are third and higher order generation messages. Thus our approximation begins to break down as the size of the aggressive window approaches the mean service time. We discuss the effects of these higher order generation arrival messages.

The messages in the aggressive window at the synchronization point (recall we term these messages *aggressive messages*) are exponentially distributed within the aggressive window. This implies that aggressive messages are weighted towards the front of the aggressive window. As shown above, first generation arrival messages have this same timestamp distribution, and thus will also be weighted towards the front of the aggressive window. Second generation arrival messages however will be weighted more towards the back of the aggressive window since they will have timestamps that are the sum of a first generation "complete_service" message and an exponential random variable. Thus the probability that a second or third generation arrival message invalidates an aggressive message (which has the same timestamp distribution as a first generation arrival message), is less than the probability that a first generation arrival message will invalidate an aggressive message. Thus the assumption that all arrival messages are first generation messages (rather than first, second and higher order generation arrival messages) will tend to over-estimate the probability of a causality error.

In the analysis to follow we show the results of our predictions given aggressive window sizes up to the mean of the service time distribution. We give these larger aggressive window sizes even though our assumptions begin to break down as the window size approaches the mean service time. We do this for two reasons. First, our results are still quite good even for these larger aggressive window sizes. Second, we feel it is instructive to show the error of our approximation as the window size increases: it

helps to illustrate the tradeoff between the tractability of our analysis and the error of our approximations. For the rest of this analysis we assume the aggressive window size will vary between 0% and 100% of the mean service time ($0 < A \leq \frac{1}{\lambda}$).

Now that we have discussed the consequences of our timestamp distribution approximation, we need to discuss in more detail our approximation for the size of the lookahead window. We do this in the following section.

3.2.5. Using the Expected Value for L

Now we have obtained all of the distributions necessary to determine the probability of a fault and the expected improvement due to aggressive processing. All of our equations have treated L, the size of the lookahead window, as a constant. Recall that L is closely approximated by the minimum of N gamma distributed random variables, where N is the number of LPs in the system, and each gamma is the sum of two exponentially distributed random variables. Thus the value of L is a random variable rather than a constant. In this section we show that it is very reasonable to treat L as a constant, and to use the expected value of L in our equations, when $\lambda L \ll 1$. We note λL decreases very quickly as N increases, and this condition holds true for any reasonably large system (i.e. on the order of 50 or more LPs).

Recall the following identity.

$$e^{-\lambda L} = 1-\lambda L+\frac{\lambda L^2}{2!}-\frac{\lambda L^3}{3!}....$$

When $\lambda L << 1$ a good approximation for $e^{-\lambda L}$ is

$$e^{-\lambda L} \approx 1-\lambda L.$$

The most common term in our equations involving L is $e^{-\lambda L}-e^{-\lambda(L+A)}$. In this section we show why it is reasonable to use the expected value of L in this particular expression. Similar arguments can be made for other expressions involving L.

Rewrite

$$e^{-\lambda L} - e^{-\lambda(L+A)} = e^{-\lambda L}\ (1-e^{-\lambda A}).$$

Substituting our approximation for $e^{-\lambda L}$ we have

$$e^{-\lambda L} - e^{-\lambda(L+A)} \approx (1-\lambda L)(1-e^{-\lambda A})$$

$$= (1-e^{-\lambda A}) - \lambda(1-e^{-\lambda A})L.$$

For fixed A this is a linear function of L.

$$(1-e^{-\lambda A}) - \lambda(1-e^{-\lambda A})L = b + aL$$

Now take the expected value of the expression.

$$E[(1-e^{-\lambda A}) - \lambda(1-e^{-\lambda A})L] = E[b + aL]$$

$$= b + a\,E[L]$$

$$= (1-e^{-\lambda A}) - \lambda(1-e^{-\lambda A})E[L]$$

$$= (1-e^{-\lambda A})(1-\lambda\,E[L]).$$

Again using our approximation that $e^{-x} = 1-x$ for small x we rewrite $1-\lambda\,E[L]$ as $e^{-\lambda E[L]}$ (again noting that $E[L] \ll 1$).

$$(1-\lambda\,E[L])\,(1-e^{-\lambda A}) \approx e^{-\lambda E[L]}\,(1-e^{-\lambda A}) = e^{-\lambda E[L]} - e^{-\lambda(E[L]+A)}$$

This shows that using the expected value of L in our equations is very reasonable when $\lambda L \ll 1$.

3.2.6. Probability of a Fault

In order to fault, an LP must process at least one message aggressively and subsequently receive an arrival message with a timestamp in its past. There are many ways this can occur. For example, an LP can process one aggressive message and receive one arrival message with a timestamp in its past. Or an LP can process one aggressive message and receive two arrival messages where one or both messages have timestamps in its past. Theoretically there are an infinite number of ways a fault can occur. In practice however only a few such combinations have any significant associated probability.

Before enumerating the significant terms of the probability of a fault we note that one message in the system needs special consideration. We state without elaboration that the lookahead window is constructed such that exactly one activity in the system will complete service at logical time $T+L$, the upper edge of the lookahead window. Thus as a consequence of processing in the lookahead window exactly one LP will receive an arrival message with a timestamp of $T+L$. We term this message the *lookahead message*. Recall $T+L$ is the floor of the aggressive window. For this reason the LP that receives the lookahead message will fault *if it has processed any messages aggressively*.

Due to the assumption of equally likely routing of messages, the probability of a particular LP receiving the lookahead message is $(1/N)$. The probability that an LP processes at least one aggressive message is $1-P\{zero\ aggressive\ messages\}$ given in Equation (3.10). Let M be the random variable denoting the number of aggressive messages a given LP processes (recall that the distribution of M is given in Equation (3.10)). Then the probability of a given LP faulting due to the lookahead message is $1/N\ (1-P\ (M=0))$.

Below we enumerate the significant terms of the probability of a fault. The $P\ (Ar=X)$ term is the probability that an LP receives X arrival messages where the probability of receiving $X=x$ arrival messages has been shown to be Poisson distributed with rate $\lambda A-(e^{-\lambda L}-e^{-\lambda(L+A)})$. The $P\ (Inval)$ term is the probability that an arrival message invalidates an aggressive message. As discussed above, this probability is dependent upon the timestamp distribution of the arrival message. We elaborate on the $P\ (Inval)$ term below.

$$P\ (Fault) = \frac{1}{N}(1-P\ (M=0)) + \frac{(N-1)}{N}[P\ (M=1,Ar=1)\ P\ (Inval) \tag{3.26}$$

$$+\ P\ (M=2,Ar=1)\ (1-(1-P\ (Inval))^2) + P\ (M=3,Ar=1)\ (1-(1-P\ (Inval))^3)$$

$$+ P\ (M=1,Ar=2)\ (1-(1-P\ (Inval))^2) +$$

$$P\ (M=1,Ar=3)\ (1-(1-P\ (Inval))^3) + P\ (M=2,Ar=2)\ (1-(1-P\ (Inval))^4]...$$

The first term of Equation (3.26) is the probability that an LP has a causality error given that it receives the lookahead message, times the probability that it receives the lookahead message. The second term is the probability of a fault given that an LP does not receive the lookahead message, times the probability that it does not receive this message. Within the second term, the $P\ (M=1,Ar=1)\ P\ (Inval)$ term is the probability of a fault given that an LP processes one aggressive message and receives one arrival message. The $P\ (M=2,Ar=1)\ (1-(1-P\ (Inval))^2)$ term is the probability of a causality error given that an LP processes two aggressive messages and receives one arrival message. In this term the $(1-P\ (Inval))^2$ component is the probability that an arrival message has a timestamp greater than the timestamps of *both* aggressive messages. One minus this term is the probability that an arrival message

has a timestamp that is not greater than the timestamps of both arrival messages, and thus represents the probability of a causality error. The other terms are similarly derived.

The P (*Inval*) term in Equation (3.25) represents the probability that a given arrival message invalidates a given aggressive message. As discussed above, if we assume all arrival messages are first generation messages then both aggressive messages and arrival messages have the same timestamp distributions. In this case the probability that an arrival message has a timestamp less than that of an aggressive message is 50%. If an arrival message is a second generation message the probability of a causality error is slightly more complex. We compute this probability.

As shown in the previous section, the timestamp of an aggressive message is a conditional exponential. Given an aggressive message with timestamp t, the distribution of t is

$$pdf\,(t) = \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda A}}.$$

Now consider a second generation arrival message with timestamp $X=x$. We have shown the distribution of X is

$$pdf\,(x) = \int_0^A \frac{\lambda e^{-\lambda x}}{(e^{-\lambda t} - e^{-\lambda A})} \frac{\lambda e^{-\lambda t}}{(1 - e^{-\lambda A})} dt.$$

The probability that the timestamp of an arrival message is less than that of an aggressive message given that the arrival message is a second generation arrival message is

$$P\,(Fault \mid 2nd\ generation) = 1 - \int_0^A \int_0^t \int_0^x \frac{\lambda e^{-\lambda s}}{(1 - e^{-\lambda t})} \frac{\lambda e^{-\lambda x}}{e^{-\lambda t} - e^{-\lambda A}} \frac{\lambda e^{-\lambda t}}{(1 - e^{-\lambda A})} ds\, dx\, dt = 0.25$$

Thus the probability that an arrival message invalidates an aggressive message is

$$P\,(Inval) = P\,(1st\ generation)\ .5 + P\,(2nd\ generation)\ 0.25 \qquad (3.27)$$

We give predicted and observed values for the probability of a causality error given various aggressive window sizes below.

3.2.7. Number of Messages Successfully Completed

The second goal of this chapter is to capture analytically the improvement in parallelism made possible by aggressive processing. We measure gains in parallelism by comparing the number of messages

processed in the non-aggressive algorithm versus the number of messages processed successfully by adding aggressiveness. Recall the number of messages processed in the non-aggressive algorithm is the number of messages processed in the lookahead window. The number of messages processed in the aggressive algorithm is the number of messages processed in the lookahead window plus the number of messages processed successfully in the aggressive window. By processed successfully we mean processing that is not later found to be invalid. A crude measure of the improvement in parallelism is the ratio of these two quantities. We say crude because we are ignoring certain costs as discussed below. We begin by computing the expected number of messages processed successfully in the aggressive window.

The improvement in parallelism calculated as described above is an upper bound on the improvement that can be attained in practice. It is an upper bound because it does not account for the costs associated with a correction mechanism. In particular, it does not account for the cost of saving state or for the cost of reprocessing messages. In Chapter 4 we define a correction mechanism and account for its costs. For the moment however we are interested in the *potential* increase in parallelism due to aggressive processing. The given correction mechanism will determine how much of this potential gain in parallelism is offset by the costs of aggressive processing.

Given the distribution of the number of messages in the aggressive window at the synchronization point and the number of arrival messages it is possible to determine the number of aggressive messages processed successfully. Consider the ways in which an LP can process one aggressive message successfully. First, the LP can process one aggressive message and subsequently receive no arrival messages. In this case the aggressive processing cannot be invalidated. Second, the LP can process one aggressive message and receive one arrival message that does not invalidate the processing. Third, the LP can process two aggressive messages and receive one arrival message that invalidates one, but not both, aggressive messages. Clearly there are infinite combinations of events that could be considered. There are only a few such combinations however with any significant associated probability.

In Equation (3.28) we give the significant terms for the probability of successfully processing $M=1$ aggressive messages. In this equation $P(J=j)$ is the probability of processing j aggressive messages. The $P(Ar=ar)$ term is the probability that an LP receives ar arrival messages. In this equation we assume all

arrival messages are first generation arrival messages. This gives the worst case probability of invalidating a given aggressive message and thus represents a pessimistic assumption with regard to the number of messages processed successfully.

$$P(M=1) = P(J=1,Ar=0) + P(J=1,Ar=1)\ .5 + P(J=1,Ar=2)\ .25 \qquad (3.28)$$
$$+ P(J=1,Ar=3)\ .5^3 + P(J=2,Ar=1)\ .5 + P(J=2,Ar=2)\ .375$$
$$+ P(J=3,Ar=1)\ .375 + P(J=3,Ar=2)\ .5 \cdots$$

The $P(J=1,Ar=0)$ term is the probability of processing one aggressive message and receiving no arrival messages. In this case the one aggressive message is guaranteed to be processed successfully. The $P(J=1,Ar=1)\ .5$ term is the probability of processing one aggressive message, receiving one arrival message, and the arrival message not invalidating the aggressive message. The probability of the arrival message not invalidating the aggressive message is 50% given the assumption that the arrival message is a first generation arrival message. This is because, with the assumption of a first generation arrival message, both the aggressive message and the arrival message have the same timestamp distribution (conditional exponentials). The other terms are similarly derived.

Equation (3.29) gives the significant terms for processing two aggressive messages successfully.

$$P(M=2) = P(J=2,Ar=0) + P(J=2,Ar=1)\ .25 + P(J=2,Ar=2)\ .25^2 \qquad (3.29)$$
$$+ P(J=3,Ar=1)\ .375 \cdots$$

Finally we present the most significant terms for the probability of processing three aggressive messages successfully. The probability of processing four or more aggressive messages successfully is negligible for aggressive window sizes in the range of $0 \le A \le 1/\lambda$.

$$P(M=3) = P(J=3,Ar=0) + P(J=3,Ar=1).5^3 \cdots \qquad (3.30)$$

Now we have all of the equations necessary to derive the expected number of aggressive messages successfully processed. In Equation (3.31) below M is the number of messages processed successfully within the aggressive window.

$$E[M] = 1*P(M=1) + 2*P(M=2) + 3*P(M=3) \cdots \qquad (3.31)$$

Let LM be the random variable denoting the number of messages processed within the lookahead win-

dow. The expected number of messages processed in the lookahead window is

$$E[LM] = 1*P(LM=1)+2*P(LM=2)+3*P(LM=3)\cdots \tag{3.32}$$

Recall the probability of processing $LM=lm$ messages in the lookahead window is given in Equation (3.15). The expected improvement in performance due to aggressive processing is therefore:

$$E[I] = \frac{E[LM]+E[M]}{E[LM]}. \tag{3.33}$$

We calculate this expected improvement in performance for various window sizes below.

3.3. Scalability of Protocol

A very important issue in parallel discrete event simulation is how well a particular protocol scales as the number of LPs increases. As discussed by Lubachevsky (1989) both Time Warp and the Null Message protocol have the potential for explosive overhead costs as the size of the simulation grows. The overhead associated with Time Warp can increase significantly due to large state saving costs and the possibility of cascading rollbacks. The Null Message protocol can have significant overhead costs due to the proliferation of null messages. To date only Nicol (1993) and Lubachevsky (1989) have proven the scalability of their respective protocols.

We have not yet derived scalability results for a system such as ours. We have however developed a set of results which we feel represent a significant step towards establishing system-level scalability results. In this section we describe these results.

In order to begin our investigation of the scalability of our approach we examine the probability of a fault at a given LP as the size of the simulation grows. In this section we show that the upper bound on the probability of a fault at a given LP is approximately 25% *as the number of LPs in the system approaches infinity*. Further, we show that the probability of a fault reaches this maximum value for some particular value of N (the number of LPs in the system). That is, there is some particular value of $N = N^*$ such that once the number of LPs in the system is greater then N^* adding more LPs to the system actually *decreases* the probability of a causality error at a given LP. In this section we identify the value of N^*. We want to stress again that this discussion is about the probability of a fault at a given LP, not the probability of a fault in the system.

The equations given so far do not directly reflect N, the number of LPs in the system. Rather this is reflected in L, the expected value of the lookahead window. As discussed in previous sections, L is closely approximated by the minimum of N gamma distributed random variables, where each gamma is the sum of two exponentials. For details regarding the width of the lookahead window the interested reader is directed to Nicol (1993). What is important for this discussion is that as N approaches infinity, L approaches zero. In order to reflect the dependence of L on N we use the notation $L(N)$ when we are discussing the behavior of a given LP as N approaches infinity.

We seek the probability of a fault as the size of the architecture, and the number of LPs, approaches infinity. In order to study this issue we need to account for one overhead of our protocol that grows as the number of processors increases. In a Global Windowing algorithm, (whether aggressive or non-aggressive) the cost of global synchronization is $O(Log_2 P)$ in a system with P processors. (Costs are somewhat higher given an aggressive barrier synchronization, but we ignore this issue for the current discussion.) Thus in order to keep the workload per processor constant, we assume there are $J = P\ Log_2\ P$ LPs per processor (Nicol 1992b). Thus there will be $N = J\ P$ LPs in a system with P processors. We seek the probability of a fault as N approaches infinity.

In order to fault an LP must process at least one aggressive message *and* receive at least one arrival message. Both of these events are necessary, but not sufficient for a fault. Thus the probability of both events occurring gives an upper bound on the probability of a fault.

Recall that K is the random variable denoting the number of aggressive messages processed by a given LP. The probability of processing at least one aggressive message is $(1-P(K=0))$, where the $P(K=0)$ is given in Equation (3.12). Let $(Ar=ar)$ be the event that an LP receives ar arrival messages. The probability of receiving at least one arrival message is $(1-P(Ar=0))$ where $P(Ar=ar)$ has been shown to be Poisson distributed with rate $\lambda A-(e^{-\lambda L}-e^{-\lambda(L+A)})$. Noting the independence of these events, the probability of both events occurring is

$$P(F = K \geq 1, Ar \geq 1) = (1-[e^{(e^{-\lambda L}-e^{-\lambda(L+A)})}(1-(e^{-\lambda L}-e^{-\lambda(L+A)}))]) \quad (1-e^{-(\lambda A-(e^{-\lambda L}-e^{-\lambda(L+A)}))}). \tag{3.34}$$

In Figure 3.7 we plot Equation (3.34) as a function of $\lambda L(N)$. We set A to $1/\lambda$, the maximum aggressive window size considered. Note that $\lambda L(N)$ can only take on values between zero and two be-

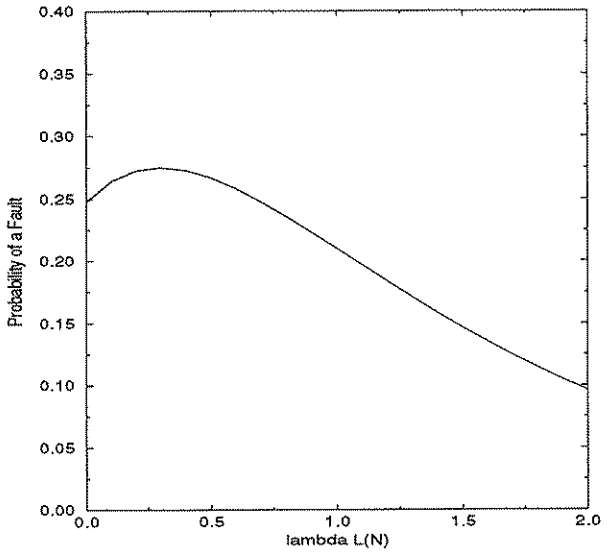

Figure 3.7 - Probability of a Fault as $\lambda L(N) \rightarrow 0$

cause the expected value of a gamma which is the sum of two exponentials is $2/\lambda$. Thus the maximum value of $L(N)$, which is closely approximated by the expected value of the minimum of N such gamma distributed random variables, cannot exceed $2/\lambda$. Also note that as $N \rightarrow \infty$ $\lambda L(N) \rightarrow 0$.

As can be seen from Figure 3.7, the maximum probability of fault (approximately 27%) is reached when $\lambda L(N) \approx 0.29$. It can also be seen that after this maximum value the probability of a fault decreases slightly as $\lambda L(N) \rightarrow 0$. The probability of a fault when $\lambda L(N) = 0$ is approximately 24.7%. Thus we have the very powerful result that the probability of a fault (at a given LP) decreases *as the number of LPs in the system approaches infinity.*

This point is important. The upper bound on the probability of a fault given in Equation (3.34) is a very loose upper bound. As discussed below, the maximum value we observed in a system with 1500 LPs was approximately 10%. This was for an aggressive window size set to its maximum value of 100% of the mean of the service time distribution. For an aggressive window size set to 50% of its maximum value, the probability of a fault was approximately 3%. For an aggressive window size set to 10% of its maximum value this probability was approximately 0.0008. As noted, these are the values for a system with 1500 LPs. As the number of LPs approaches infinity, these observed probabilities *will not increase.*

In fact, we can get a good estimate of the number of LPs required, such that adding more LPs to the system will cause the probability of a fault to decrease (although only slightly). We discuss this more fully below.

It can be seen from Figure 3.7 that the probability of a fault rises sharply for $.29 \leq \lambda L(N) \leq 2$. It would be desirable to either a) reduce this steep rise in the probability of a fault or b) show that the number of LPs (N) in the system for $.29 \leq \lambda L(N) \leq 2$ is sufficiently small that this steep rise is not critically important. We could reduce the slope of the fault curve by adjusting the size of the aggressive window. The approach we choose is to demonstrate that the number of LPs in the system for $\lambda L(N) = .29$ (N^*) is sufficiently small that $1 \leq N \leq N^*$ is not of critical importance.

We begin by noting that the expected value of the minimum of N gamma distributed random variable is a decreasing function of N. Without going through the derivation we note that the expected value of the minimum of N gamma distributed random variables (each with mean $2/\lambda$) is:

$$L(N) \approx \int_0^\infty N\lambda^2 y^2 (1+\lambda y)^{(N-1)} e^{-\lambda N y} dy. \tag{3.35}$$

Again note in Equation (3.35) L is closely approximated by the minimum of N gamma distributed random variables. Let $\xi = \lambda y$ and $d\xi = \lambda dy$. Rewrite Equation (3.35) using this variable transformation.

$$L(N) \approx \frac{1}{\lambda} \int_0^\infty N\xi^2 (1+\xi)^{(N-1)} e^{-N\xi} d\xi \tag{3.36}$$

Note the integral is a function of N only. Rewrite Equation (3.36).

$$L(N) \approx \frac{1}{\lambda} f(N) \tag{3.37}$$

Because $L(N)$ is a decreasing function of N, $f(N)$ in Equation (3.37) is also a decreasing function of N.

As shown in Figure 2, $\lambda L(N) \approx .29$ when the maximum probability of a fault is reached. Rewrite this as follows:

$$L(N) \approx \frac{1}{\lambda} .29. \tag{3.38}$$

Thus we see that $f(N)=.29$ when the maximum probability of a fault is reached. Since $f(N)$ is a decreasing function of N there is only one value of N, N^*, for which Equation (3.38) is true.

In order to find N^* we set $\lambda = 1$ and numerically determine the number of gamma distributed random variables (with mean 2) for which the expected value of the minimum of this number is approximately 0.29. We found this number to be between 23 and 24. Then we ran simulation studies to validate that the expected value of the minimum of twenty four gamma distributed random variables (each composed of the sum of two exponentials with mean 1) is approximately .29. We conclude that $N^* \approx 24$. This demonstrates the very important property that if there are more than twenty four LPs in the system, adding more LPs to the system will *decrease the probability of a fault (at a given LP).* This is important in that it shows the expected increase in parallelism at a given LP does not decrease (i.e. there is no higher probability of a causality error) as the size of the system grows without bound. Also, we can build on this result to establish system wide scalability arguments.

In this section we have investigated the *upper bound* on the probability of a fault given that the size of the aggressive window is set to its maximum value ($1/\lambda$). We used the upper bound of a fault, and the maximum aggressive window size, in order to to demonstrate the probability of a fault (as the system grows) under the worst possible conditions. As noted, the results obtained under these conditions are much higher than the results obtained using our explicit estimate of a fault given in Equation (3.26). In Table 1 we use Equation (3.26) to derive the maximum probability of a fault, the minimum probability of a fault (obtained when $\lambda L(N)=0$), and N^* for various aggressive window sizes.

A	P(F) Max	P(F) Min	N*
.1	.002	.0005	5
.3	.02	.009	7
.5	.053	.035	10
.7	.09	.077	15

Table 3.1 - Value of N^*

3.4. Simulation Results

In this chapter we have developed a model to investigate the improvement in potential parallelism made possible by adding aggressiveness to an existing non-aggressive protocol. In order to test the predictions of our model we ran a series of simulations using a simple FCFS queueing model. Our first queueing model meets the basic assumptions of our analysis including an exponential service time distribution, a heavily loaded system and each LP having an equal probability of receiving a given message. However, there are assumptions in our model that are not met in the system. Most importantly, we do nothing to interfere with an arrival message generated by an LP. Thus it is not guaranteed that an arrival message will be either a first or second generation message as assumed in our model. Also recall we use the observed expected value of L in our equations.

The simulation program used to gather these results does not allow causality errors to occur since we have not yet defined a mechanism to correct such errors. We do so in Chapter 4. To gather the empirical results we used the following approach. At the point when the LPs synchronized to define the lookahead and the aggressive windows, we made a copy of the messages in the aggressive window (the aggressive messages). As the simulation progressed and the LP received its arrival messages, we kept track of how the arrival messages would have affected the aggressive messages. Thus we monitored whether any of the aggressive messages would have been invalidated (and tagged the LP as faulting if this was the case) and kept track of the number of aggressive messages that would have been processed successfully. Thus the model and the simulation focus on the aggressive messages and how they would be affected by arrival messages.

We ran a series of simulations with 1500 LPs, a mean service time of $1/\lambda = 1$ and various aggressive window sizes. Then we compared the empirical results with the predictions of our model.

In Figure 3.8 we plot the predicted versus observed probability of a fault for various aggressive window sizes. One curve shows the predicted probability of a fault assuming all arrival messages are first generation messages. The other curve shows the predicted probability of a fault given that we account for second generation arrival messages. As can be seen the predictions are almost perfect when we account for both first and second generation arrival messages. Also it is clear that when we assume all arrival

messages are first generation messages we begin to over-predict the probability of a fault for larger aggressive window sizes. This is expected since the percentage of first generation arrival messages decreases as the aggressive window size increases. As we have discussed, the probability of a first generation arrival message causing a fault is higher than the probability of a second (or higher) generation message causing a fault. Also it can be seen that the probability of a fault at a given LP is quite low for a very reasonable range of aggressive window sizes.

In Figure 3.9 we plot the predicted versus observed expected improvement in parallelism for various aggressive window sizes. Recall we measure expected improvement as the ratio of the number of messages processed in the aggressive version and the number of messages processed in the non-aggressive version. Our model under-estimates the potential improvement in parallelism for larger aggressive window sizes. Again this is because it over-predicts the probability of a causality error. As can be seen there is significant potential for increased parallelism using our approach.

3.4.1. Predictive Power of Model When Assumptions Are Not Met

We are interested in the predictive power of our model given a system that does not meet our most significant assumption: a fully connected communication topology where each LP is equally likely to re-

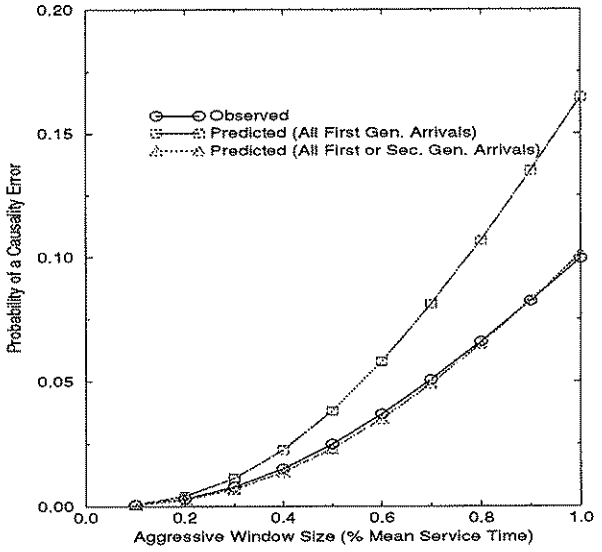

Figure 3.8 - Probability of a Causality Error

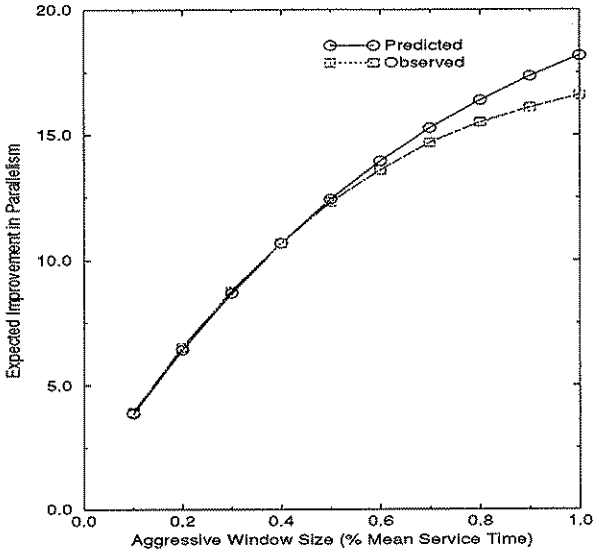

Figure 3.9 - Potential Expected Improvement in Parallelism

ceive a given message. We ran a series of experiments in order to study this issue. In each experiment (except where noted) we used a 2D toroidal mesh with 1024 LPs and λ=1. We varied the aggressive window size between 50% and 100% of the mean service time ($1/\lambda = 1$).

In the first experiment each LP communicated only with its nearest neighbor where each neighbor was equally likely to receive a given message. The second experiment involved a "hot spot" in the communication pattern. We used a nearest neighbor communication pattern where each neighbor had a 20% probability of receiving a given message. Also there was a 20% probability that a given message was sent to one of ten LPs in the system (where each of the ten LPs was equally likely). Thus approximately 1% of the LPs received 20% of the messages in the system. In the next experiment 20% of the messages were received by approximately 8% of the LPs (i.e. 20% probability of sending a given message to one of 80 LPs).

Then we compared the results of these experiments with the predictions of our model. We compared the results on both the probability of a fault and the upper bound on the expected improvement. The results are shown in Table 2. As can be seen our results are quite accurate for the nearest neighbor communication pattern. This makes sense as it represents essentially a uniform distribution of messages. The predictions are also reasonable given a hot spot where 1% of the LPs receive 20% of the messages.

	A = 50% MST				A = 100% MST			
	P(F) Pred	P(F) Obs.	E[I] Pred	E[I] Obs.	P(F) Pred	P(F) Obs.	E[I] Pred	E[I] Obs.
Nearest Neighbor	.024	.022	10.27	10.36	.103	.093	13.79	15.19
Hot Spot 1%	.024	.024	10.27	9.92	.103	.074	13.79	14.39
Hot Spot 8%	.024	.022	10.27	10.36	.103	.094	13.79	15.19
1500 LPs	.023	.0249	12.32	12.44	.101	.099	16.62	18.18
3000 LPs	.022	.0240	17.12	17.31	.099	.098	23.25	25.51

Table 3.2 - Effects of Various Communication Topologies

For an aggressive window size of 50% of the mean service time there is little difference between predicted and observed values. It is interesting to note that given an aggressive window size of 100% of the mean service time the probability of a fault is actually somewhat lower than predicted. We attribute this to the increased probability of third and higher generation arrival messages at this large aggressive window size. Consider that 99% of the LPs are receiving only 80% of the arrival messages. Thus it makes sense that they would not fault as much as predicted. The other 1% of the LPs (receiving 20% of the messages) are obviously not faulting enough to make up for the lower number of arrival messages received by the other LPs in the system. Most likely this is because at this large aggressive window size a greater proportion of arrival messages are third and higher order generation messages. Thus they are less likely to cause a fault. As can be seen, when the hot spot is reduced to 8% of the LPs in the system the observed values are again quite close to the predicted values.

Our final experiment was designed to test the probability of a causality error as the number of LPs is increased. We simulated two systems, one with 1500 LPs and one with 3000 LPs. In these experiments we used our basic assumption of a fully connected communication topology where each LP is equally

likely to receive a given message. As can be seen, both the predicted and observed value of a fault (at a given LP) are slightly lower in the system with 3000 LPs than in the system with 1500 LPs. This matches the predictions of our analytic model. The difference between the predicted values for the system with 1024 LPs and the systems with 1500 LPs and 3000 LPs is due to differences in the size of the lookahead window.

3.5. Limitations of Results

The model developed in this chapter is the foundation upon which we build a much more powerful model. For this reason it is important to discuss the limitations of the model as developed in this chapter. However, the results developed in this chapter are important in their own right. First, our model gives very accurate predictions given a system that meets the basic assumptions of the model (i.e. exponential service times, heavily loaded system, equally likely communication pattern, etc.). Further, these results show a tremendous *potential* for an increase in parallelism given a limited amount of aggressive processing. Also, we have shown this potential improvement in parallelism exists in systems with communication topologies other than those assumed in our model. Perhaps most importantly, this is the first time the potential increase in parallelism, and the probability of a causality error, has been investigated *as a function of the level of aggressiveness.* For these reasons the results presented in this chapter are very important.

As noted however, both the analytic model and the simulation system developed in this chapter are preliminary. We developed the model and the simulation in order to test our hypothesis that allowing a very limited level of aggressive processing can greatly increase parallelism without introducing a significant number of causality errors. To test this hypothesis, our model (and simulation) considers the messages in the aggressive window *at the synchronization point*, since these messages are processed in the aggressive version of the algorithm and are not processed in the non-aggressive version. Clearly the results obtained in this chapter are encouraging enough to warrant further development of the model in order to get a more realistic picture of the costs/benefits of this approach. We do this in the remaining chapters.

The preliminary nature of these results of course implies that there are limitations to the model and to the simulation developed in this chapter. The primary limitation of both the model and the simulation system is that neither addresses the costs of correcting causality errors that occur as a result of aggressive processing. The simulation system does not allow causality errors to occur since we have not yet defined a mechanism to correct such errors. Similarly, the model developed in this chapter does not address the costs associated with aggressive processing. Further, the simulation was designed to capture the effects of further processing on the messages in the aggressive window at the synchronization point, and does not capture any activity that does not involve these messages. For this reason it does not recognize a fault caused by one arrival message invalidating another, or the invalidation of a second generation "complete_service" message, since these messages are not in the aggressive window at the synchronization point. These issues are addressed in subsequent chapters. In Chapter 4 we define our correction mechanism, and give a set of simulation results obtained after implementing this mechanism. In these simulations we capture *all* of the causality errors that occur, and *all* of the costs of aggressive processing (such as reprocessing messages due to rollbacks, etc.). Also, we extend our model *to capture the costs associated with aggressive processing* such as saving state, reprocessing messages due to rollbacks, and so forth. It is important to note again that while the results presented in this chapter are preliminary, they are the foundation upon which we build more powerful results.

It is important to discuss the limitations of the simulation results related to the various communication topologies. The reported probability of a fault is the *average* probability of a fault taken over the whole system. Thus it can be assumed that an LP which receives many more messages than a "typical" LP (such as one of the LPs in a "hot spot") will exhibit a fault rate much higher than the average. This brings up the main limitation of the results presented in this chapter: It is the behavior of the *slowest* LP in the system, rather than the behavior of a "typical" LP, which determines the performance of a synchronous protocol such as ours. All of the LPs must block until the *slowest* LP in the system has completed all of its processing. The model developed in this chapter is not powerful enough to capture *system* level performance. In subsequent chapters we extend the results developed in this chapter in order to capture system level performance.

3.6. Conclusions

In this chapter we developed a model to study the effects of adding aggressiveness to an existing non-aggressive protocol. We have three very significant results from this work. First, we are able to predict the probability of a fault *as a function of the level of aggressiveness.* This is the first time this has been accomplished. Second, we have been able to demonstrate both theoretically and with simulation studies the significant potential for improvement in parallelism made possible by adding aggressiveness to a non-aggressive approach. Third, we have shown that the probability of a fault does not increase *as the number of LPs approaches infinity.* We feel this result represents an important step towards proving system-level scalability. Also, we have identified the number of LPs, N^*, such that $N^* < N_1 < N_2$ implies that the probability of a fault in a system with N_2 LPs is less than the probability of a fault in a system with N_1 LPs.

We have shown that the basic direction of our predictions is accurate even when the major assumptions of our model (such as communication topology and number of messages generated as a result of processing an event) are violated. The results that we have obtained in this chapter show that there is much *potential* for improvement in parallelism in our approach. However we still need to define a correction mechanism and account for the costs of this mechanism in our model. Also, we need to develop our simulation system such that it will allow causality errors to occur and be able to correct these errors. We accomplish both goals in remaining chapters.

CHAPTER 4

Error Correction Mechanism

The analytic model developed in the previous chapter lays the theoretical groundwork for the investigation of adding aggressiveness to an existing non-aggressive protocol. We demonstrated analytically and with simulation studies a significant potential for improvement in parallelism using this approach. The results presented in the previous chapter however are incomplete. First, the predicted improvement in parallelism is an *upper bound* on the improvement that can be attained in practice. This is because the model does not account for the costs associated with aggressive processing. Second, while we show that the probability of a causality error (at a given LP) is relatively small over the range of aggressive window sizes we consider, it is none the less greater than zero. Thus we need some mechanism to correct the causality errors that do occur.

In this chapter we define our error correction mechanism. We add the costs of this mechanism to our model in subsequent chapters. We propose a simple state saving and rollback scheme to correct the causality errors that occur as a result of aggressive processing. Also, we develop a model to predict the probability of anti-messages being produced, and show that the probability of cascading rollbacks is quite small. Then we use this model in later chapters to complete the analytic study of the costs of our approach. The result of this work is a model that gives a much clearer picture of the costs and benefits of adding aggressiveness to an existing non-aggressive protocol.

The rest of this chapter is organized as follows. In section 4.1 we we discuss the correction mechanism. In section 4.2 we develop our model to predict the probability an LP will produce an anti-message. In section 4.3 we give empirical results to validate our model. In section 4.4 we show that the probability of initiating a rollback chain at a given LP does not increase as the number of LPs approaches infinity. We feel this results represents a significant step towards proving system-level scalability. We give our conclusions in section 4.4.

4.1. Correction of Causality Errors

The approach is a simple state saving and rollback mechanism. Other approaches (such as some type of iterative technique) are also possible. We choose a state saving and rollback mechanism for two reasons. First, it is a widely accepted approach with significant development of hardware support (Reynolds *et al.* 1993, Buzzel *et al.* 1990). Second, the model developed in the previous chapter can be modified to account for the costs of this approach.

Our modified algorithm proceeds in two phases as follows. Assume the system is synchronized at logical time T. In the first phase of the algorithm the LPs cooperatively define the simulation window. As in the non-aggressive version, the lookahead window is defined such that all events within the window can be processed concurrently without the possibility of a causality error. In the aggressive version the simulation window is extended from logical time $T+L$ to logical time $T+L+A$ where A is the length of the aggressive window. In the second phase of the modified algorithm the LPs concurrently execute all of their events within the extended simulation window. We elaborate on this second phase of the algorithm.

We assume state is saved before each message processed in the aggressive window. If, during the course of processing within the aggressive window, an LP receives a message in its logical past it must perform a rollback. This involves restoring the state of the LP immediately before the timestamp of the message causing the rollback. Any messages sent as a result of processing that has been rolled back is potentially invalid and is cancelled through the use of an *anti-message*. An anti-message has the same timestamp as the message it is sent to cancel, and informs the receiving LP that the previous message is invalid. After the LP performs a rollback (and sends any anti-messages required) it processes forward from the logical time of the restored state. When an LP reprocesses a "complete_service" which had previously been rolled back (and therefore the "pre-sent" completion message had been cancelled by an anti-message), we assume the next LP to receive the "pre-sent" completion message is chosen at random where each LP is equally likely. Further, we assume that the duration of an activity is not affected by a rollback.

An anti-message can cause the receiving LP to roll back (if the timestamp of the message is less than the current logical time at the LP) and cancel messages it has sent. Thus one anti-message can lead

to a chain of such messages propagating through the system. Sending an anti-message as soon as the error is discovered corresponds to the *aggressive cancellation policy* in Time Warp (Reiher *et al.* 1990).

In the modified algorithm the LPs compute in the extended simulation window without synchronization. Any messages generated as a result of this processing are sent as soon as they are generated. Once an LP reaches the ceiling of the extended simulation window it blocks waiting for all of the other LPs to similarly complete. We require that all messages generated with timestamps in the simulation window be processed before the LPs establish another simulation window. This ensures that once a new simulation window is established no LP will ever have to roll back beyond this point. To guarantee that all messages generated within the aggressive window are processed requires two assumptions. First we assume that all messages sent are eventually received. Second, we assume there is some mechanism that allows the LPs to determine when this condition has been met. This is a very important issue which we now discuss in detail.

Recall that the three phases of a non-aggressive Global Windowing Algorithm are each separated by a barrier synchronization. Traditional barrier synchronization routines (such as the **gsync()** call available on an Intel iPSC multiprocessor) assume that a processor has completed all required processing before entering into the barrier. Once the call to the barrier synchronization is made the processor loses the thread of control.

A traditional barrier synchronization works well with a non-aggressive windowing algorithm because the processor can determine when all required processing is completed. This can be determined because the lookahead window is constructed such that no LP will receive a message with a timestamp that falls within the window (the interested reader is directed to Nicol 1993 to see how one such lookahead window is constructed). Thus when all of the LPs on the processor have completed processing within the lookahead window the processor can safely enter into a barrier synchronization. It is guaranteed that all necessary work is completed before the call is made.

If however an aggressive window is defined (such as in our modified algorithm) then it becomes a very difficult issue to determine when all required processing within the window is completed. This is because an LP can complete all known processing up to the upper edge of the aggressive window, and then

at some later point in real time receive a message with a timestamp that falls within the aggressive window. Thus all of the processing required within the window was not completed when the LP reached the upper bound of the window and the processor would be incorrect to make a call to the barrier routine at this point. Further, if the late message requires a rollback then the LP may send new messages to other LPs which have also processed up to the upper bound of the aggressive window. Thus they too will not have completed all of the processing required within the window and similarly would be incorrect in calling a traditional barrier routine when they reached the upper bound of the aggressive window. As can be seen, if it is possible to receive a message with a timestamp that falls within the simulation window, and the message passing activity is unpredictable, then traditional barrier synchronization routines are not powerful enough to use as the synchronization mechanism.

What is required is a mechanism that will allow an LP to "aggressively" enter into a barrier, such that it retains the ability to pull out of the barrier if it discovers new processing that must be performed. Alternatively, there can be some mechanism to keep track of enough global information to inform the LPs when the processing within the simulation window is complete. We discuss these two approaches.

Nicol (1992a) has developed a software solution to the problem of an aggressive barrier synchronization. In this algorithm an LP can enter the barrier optimistically before it is certain that it has completed all required processing within the window. If at some later point it discovers that it has not completed all required processing then it can roll out of the barrier. This is exactly what is required for our aggressive windowing algorithm. This optimistic barrier synchronization is a much more powerful routine than the traditional barrier synchronization and thus has a higher associated cost. As discussed by Nicol (1992a), the optimistic barrier synchronization is on the order of two to three times slower than an optimized traditional barrier synchronization. We note again however that a traditional deterministic barrier synchronization routine is not powerful enough to be used in our aggressive windowing algorithm.

An alternative solution to our optimistic barrier synchronization problem is to use hardware such as that currently being developed by Reynolds *et al.* (1993). Reynolds is building a parallel reduction network which will disseminate global information in order to help make parallel simulation algorithms complete in less time. For our purposes we can use such a network to determine when processing within

the simulation window is complete. We can do this by keeping a global count of the number of messages sent and the number of messages received. Every time an LP sends a message it increments a counter and this same counter is decremented when a message is received. Thus when the message count is zero, and all LPs have completed their processing within the simulation window, the LPs are ready to synchronize and determine the new simulation window.

As can be seen there are (at least) two solutions available to solve our problem of optimistic synchronization in the face of unpredictable message passing activity. These two approaches will have different costs for the synchronization mechanism. We discuss this issue in more detail when we develop the cost model for our algorithm.

Given a solution to the optimistic barrier synchronization problem our modifications to the non-aggressive version of the algorithm are minimal. Our modified algorithm guarantees that an LP will never have to roll back past the floor of the aggressive window established in the first phase of the algorithm. This implies that state only needs to be saved when processing within the aggressive window and that it can be discarded once a new simulation window is chosen. Thus the memory requirements are much less than in a fully aggressive approach.

The mechanism is essentially Time Warp (Jefferson 1985) with limited aggressiveness. This is also the same approach as the Bounded Time Warp protocol proposed by Turner and Xu (1992). The difference is that we analyze the performance of our system and they offer no analytic results at all. It is also similar to the filtered rollback algorithm proposed by Lubachevsky *et al.* (1989) in which he adds aggressiveness to his Bounded Lag Algorithm (Lubachevsky 1988). There are two primary differences between the research presented here and that presented by Lubachevsky. The first difference is in the analysis of the algorithms. The second difference is in the level of aggressiveness allowed by the protocols. We discuss these two issues in turn.

The analysis presented in this thesis is significantly different from the analysis presented by Lubachevsky in the following way. Lubachevsky derives enough information to show that the aggressive version of his Bounded Lag Algorithm maintains the scalability properties of the non-aggressive version. To accomplish this he derives the conditions under which cascading rollbacks will not develop. Also he

derives an upper bound on the number of events processed in the simulation including events processed more than once due to rollbacks. Our model does not derive the conditions under which cascading rollbacks will not occur. Rather we derive explicit estimates as to the number of first generation rollbacks, the number of second generation rollbacks, etc. Using these derived probabilities we show that the probability of cascading rollbacks is negligible. Most importantly, we derive the expected improvement in performance (due to aggressive processing) *as a function of the level of aggressiveness including the costs of state saving and rollbacks*. Lubachevsky does not address this issue. (In fact, this is the first time this important issue has been addressed.)

The second difference is in the level of aggressiveness allowed by the two protocols. Lubachevsky allows an arbitrary bound on the amount of aggressiveness. Thus at one extreme it can be a fully aggressive algorithm. As discussed in the previous chapter we allow only a limited amount of aggressiveness and our analysis is not applicable to a fully aggressive system.

Also we have derived many of the same measurements as Gupta *et al.* (1991) in their investigation of Time Warp. However they are studying Time Warp which is a fully aggressive protocol. Also we are interested in the expected improvement in the number of messages processed between global synchronization points due to our modified algorithm. This is not the same issue as investigated by Gupta *et al.* Further, in terms of a queueing model they assume infinite servers while our model assumes one server per LP and therefore imposes some kind of queueing. Finally they assume state saving costs are negligible and we include these costs in our model.

Now that we have described our mechanism for correcting causality errors we can develop our model to account for the costs of this aggressiveness. The model presented in this chapter builds on the model developed in the Chapter 3, and uses the same assumptions. Recall we assume a closed system that is heavily loaded, an exponential service time with mean $1/\lambda$, each LP is equally likely to receive a given "pre-sent" completion message and all processing is performed in the aggressive window before the LP receives any arrival messages. In subsequent chapters we relax the assumption of a closed system that is heavily loaded and model an open system with external Poisson arrival streams.

4.2. Costs of Aggressive Processing

There are two primary costs associated with aggressive processing: *state saving costs* and the potential for *cascading rollbacks and echoing*. As discussed by Fujimoto (1990) large state saving costs can significantly impact the performance of aggressive processing. However if the granularity of event processing is significantly larger than state saving overhead, or if hardware support is used (Buzzel *et al.* 1990), then good performance can be achieved with aggressive processing (Fujimoto 1990). We discuss this issue in detail in the following chapter.

The second primary cost associated with aggressive processing is the possibility of echoing or cascading rollbacks. Echoing occurs when one rollback causes a chain of rollbacks and the amplitude of the rollbacks increases without bound (Lubachevsky *et al.* 1989). By amplitude we mean the amount of logical time that must be rolled back. Cascading occurs when a rollback chain develops and the number of participants increases without bound (Lubachevsky *et al.* 1989). Echoing is not a concern in our protocol as an LP can never roll back beyond the ceiling of the lookahead window. In this section we derive the probability of one rollback causing another rollback and show that the probability of cascading is negligible.

4.2.1. Probability of Generating an Anti-Message

In this section we compute the probability that an LP generates an anti-message. In order to discuss this probability we briefly review one aspect of Nicol's (1993) windowing algorithm that is critical to our analysis.

As discussed in section 3.2.1, in Nicol's algorithm each LP "pre-sends" its completion message. That is, the completion time of an activity, and the LP to receive this activity upon its completion, are both calculated at the time the activity begins. The LP to subsequently receive the activity is notified of this reception *at the time the activity enters into service.* Recall that we term a message sent to inform an LP of a future arrival a *"pre-sent" completion message.* Note that these "pre-sent" completion messages are the only messages exchanged by the LPs.

When a server places an activity into service it also schedules a message for itself at the service completion time. Recall that we term this message the *"complete_service"* message. Processing of the "complete_service" message consists of giving service to the next scheduled activity, sending the "pre-sent" completion message to the LP to receive the activity upon its completion, as well as any statistics gathering required by the simulation. Thus for every server that is busy there is one "pre-sent" completion message, sent to the receiving LP, and one "complete_service" message scheduled on its own event list. In section 3.2.4 we discussed the "complete_service" message in detail and discussed its various timestamp distributions in detail. In particular we defined a *first generation* "complete_service" message as one with a timestamp that is a conditional exponential, conditioned on being within the aggressive window.

Recall our assumption that the system is heavily loaded and that the probability of an idle server is very low. This implies that with high probability each LP will have a "complete_service" message on its event list scheduled for some time in the future. It also implies that with high probability any arrival message received will not go into service immediately (recall that an arrival message is a message received by an LP with a timestamp that falls within the aggressive window). Therefore it is with high probability that it is only the processing of a "complete_service" message that will cause another activity to be placed into service. Given that it is only the placing of an activity into service that results in a message being sent to another LP (the "pre-sent" completion message), and given our assumption of a heavily loaded system, we conclude that it is only the processing of a "complete_service" message that will result in a message being sent to another LP. For the purposes of our analysis we assume this is always the case.

In fact it is not the processing of the "complete_service" message that causes the "pre-sent" completion message to be sent. Rather, it is the placing of the next job into service that causes this message to be generated. Since the next job will not enter into service until the "complete_service" message is processed (because we assume the server is always busy), we state it as if it is the processing of the "complete_service" message that causes the LP to send a "pre-sent" completion message.

If a "complete_service" message is processed aggressively, and the LP receives an arrival message with a timestamp less than the timestamp of the "complete_service" message, the "pre-sent" completion message generated as a result of this processing *may* be invalid. It is not necessarily the case that it will

be invalidated but for the purposes of this analysis we assume it always will. Thus if a "complete_service" message is processed aggressively, and the LP receives an arrival message with a timestamp less than the timestamp of the "complete_service" message, we assume the "pre-sent" completion message generated as a result of processing the "complete_service" message is invalid. The LP must therefore send an anti-message to cancel this message.

By definition the timestamp of the anti-message will have the same timestamp as the message being cancelled. If this timestamp falls outside of the aggressive window it will do no serious damage as the message it will cancel has not yet been processed. If however it falls within the aggressive window it can cause other aggressive messages to be rolled back and possibly cause another (second generation) anti-message to be generated. We now determine the probability that an LP generates an anti-message with a timestamp that falls within the aggressive window.

There are three conditions required for an LP to generate an anti-message with a timestamp within the aggressive window. First, the LP must process the "complete_service" message aggressively. Second, the processing of the "complete_service" message must be invalidated. Third, the message generated as a result of processing the "complete_service" message must have a timestamp that falls within the aggressive window. We calculate the probability of each of these events.

The first condition is that the LP must process the "complete_service" message aggressively. In order for this to occur the timestamp of the "complete_service" message must fall within the aggressive window. We give an upper bound on the probability of processing the "complete_service" message aggressively by assuming it is processed aggressively when it falls within either the lookahead or the aggressive window. Let (C) be the event that the "complete_service" message has a timestamp that falls within the simulation window (i.e. either the lookahead or the aggressive window). The probability of this event is

$$P(C) = 1 - e^{-\lambda(L+A)} \tag{4.1}$$

The second condition required to generate an anti-message is that the LP receive at least one arrival message with a timestamp less than the timestamp of the "complete_service" message. Remember an anti-message can also invalidate the processing of the "complete_service" message. We ignore this issue

for the moment and discuss it in detail below. To determine the probability that an arrival message invalidates the "complete_service" message we first need the distribution for the number of arrival messages that fall within the aggressive window. As shown in section 3.2.3, the probability that an LP receives K arrival messages (with timestamps that fall within the aggressive window) is Poisson distributed with rate $\lambda A - (e^{-\lambda L} - e^{-\lambda(L+A)})$.

In order to predict the probability that an arrival message invalidates a "complete_service" message (i.e. has a timestamp less than that of the "complete_service" message) we need to know the timestamp distribution of both types of messages. In section 3.2.4 we discussed the timestamp distributions of both arrival messages and "complete_service" messages in detail. For the purposes of this discussion we define the event *Inval* as the event that an arrival message invalidates a "complete_service" message. We derived the probability of this event in Equation (3.27).

The final event that must occur in order to produce an anti-message is that the "pre-sent" completion message generated as a result of processing the (now invalidated) "complete_service" message must fall within the aggressive window. Recall that the service time is drawn from independent, identically distributed (*iid*) exponential random variables with mean $1/\lambda$. Consider a "complete_service" message with timestamp $T_{cs} = t$ which is a conditional exponential, conditioned on being within the aggressive window. Assuming there is some activity A_1 to enter into service upon the processing of this "complete_service" message, the completion time of activity A_1 will be $t_2 = t + \xi$ where ξ is an exponential random variable with mean $1/\lambda$. We seek the probability that t_2 falls within the aggressive window. Recall that the aggressive window has a width of A logical time units. For a particular timestamp $T_{CS} = t$ of the "complete_service" message this probability is

$$P(t_2 < A \mid T = t) = 1 - e^{\lambda(A-t)}.$$

We compute the unconditional probability by integrating the above equation over all possible values of $T_{CS} = t$ times the probability of $T_{CS} = t$. Remember the distribution for T_{CS} is a conditional exponential, conditioned on being within the aggressive window (see Equation 3.19). Let (*2nd*) be the event that the completion time of activity A_1 falls within the aggressive window. The probability of this event is

$$P(t_2 < A) = P(2nd) = \int_0^A (1-e^{-\lambda(A-t)}) \frac{\lambda e^{-\lambda t}}{(1-e^{-\lambda A})} dt = \frac{1-(\lambda A e^{-\lambda A} + e^{-\lambda A})}{1-e^{-\lambda A}}. \quad (4.2)$$

We now have all of the information necessary to determine the probability of generating an anti-message with a timestamp that falls within the aggressive window. Let (*Anti*) be the event that an LP generates an anti-message with a timestamp within the aggressive window. The probability of this event occurring is the probability that an LP processes the "complete_service" message aggressively, receives an arrival message with a timestamp less than the timestamp of the "complete_service" message, and the next activity to enter into service completes within the aggressive window. Noting the independence of these events this probability is

$$\begin{aligned} P(Anti) \approx\ & P(C)\, P(Ar{=}1)\, P(2nd)\, P(Inval) \\ & + P(C)\, P(Ar{=}2)\, P(2nd)\, (1-(1-P(Inval))^2) \\ & + P(C)\, P(Ar{=}3)\, P(2nd)\ \ (1-(1-P(Inval))^3). \end{aligned} \quad (4.3)$$

The first term in Equation (4.3) ($P(C)$)is the probability of processing the "complete_service" message aggressively. The second term ($P(Ar{=}1)$) is the probability of receiving one arrival message. The third term ($P(2nd)$) is the probability of generating a message within the aggressive window. The fourth term ($P(Inval)$) is the probability that the arrival message has a timestamp less than the timestamp of the "complete_service" message. The other terms are similarly defined. The probability of receiving more than three arrival messages is negligible and is not included in the equation. For this reason Equation (4.3) is an approximation. Finally, we define a *first generation anti-message* as an anti-message which is produced when a "complete_service" message is invalidated by the receipt of an arrival message.

4.2.2. Higher Order Generation Anti-Messages

We define a *second generation anti-message* as an anti-message which is caused by the receipt of a first generation anti-message rather than the receipt of an arrival message. We compute the probability of a second generation anti-message in a manner similar to the derivation of the probability of a first generation anti-message. There are four conditions necessary for a second generation anti-message to be produced. First, the LP must receive an anti-message. Second, the LP must process the "complete_service"

message aggressively. Third, the timestamp of the anti-message must be less than the timestamp of the "complete_service" message. Fourth, the "complete_service" message must have generated a message with a timestamp within the aggressive window. We calculated the probability of each of these events above except for the probability that the timestamp of an anti-message is less than the timestamp of a "complete_service" message. We now derive this probability.

Consider the timestamp distribution of a first generation anti-message. A first generation anti-message is produced when a "complete_service" is processed aggressively, the next activity to enter into service completes within the aggressive window, and the LP receives an arrival message with a timestamp less than that of the "complete_service" message. The anti-message will have the same timestamp as the "pre-sent" completion message it is sent to cancel, and this timestamp will be the sum of the logical time of the "complete_service" message and an exponential random variable with mean $1/\lambda$.

The timestamp of a "complete_service" message is a conditional exponential, conditioned on being within the aggressive window. Assume LP_k has a "complete_service" message CS_1 with timestamp $T_{CS}=t$ within the aggressive window. Further assume the processing of CS_1 causes activity A_1 to be placed into service, and that A_1 completes service at time $x = t + \xi$ where ξ is an exponential random variable with mean $1/\lambda$. Further assume x falls within the aggressive window. Then LP_k will have sent some LP in the system a "pre-sent" completion message with timestamp $X=x$. Assume LP_i is the chosen LP. Now assume that "complete_service" message CS_1 is invalidated by the receipt of an arrival message. Then the "pre-sent" completion message with timestamp $X=x$ may be invalid and is cancelled by an anti-message sent from LP_k to LP_i (with timestamp $X=x$).

Now consider LP_i which receives the anti-message from LP_k. Assume LP_i has aggressively processed a "complete_service" message CS_2 with timestamp $S=s$. Further assume that the processing of CS_2 caused another activity A_2 to be placed into service, and that the completion time of activity A_2 is $y = s + \xi$. Assume some LP_j receives activity A_2 when the activity completes its execution, and that the completion time of activity A_2 falls within the aggressive window. Thus LP_i will send LP_j a "pre-sent" completion message with timestamp $Y=y$. Assume that some time after LP_i sends this "pre-sent" completion message to LP_j, it receives the anti-message (with timestamp $X=x$) from LP_k. If $X=x$ is less than $S=s$

then the "pre-sent" completion message for activity A_2 may have been sent in error. This "pre-sent" completion message is thus cancelled by an anti-message. This will be a second generation anti-message since the "complete_service" message was invalidated by the receipt of a first generation anti-message rather than the receipt of an arrival message.

To determine the probability of a second generation anti-message we must compute the probability that timestamp X is less than timestamp S. In section 3.2.4 we defined a first generation "complete_service" message as one which has a timestamp that is a conditional exponential, conditioned on being in the aggressive window. Both "complete_service" message CS_1 and CS_2 are first generation messages (the discussion of this is given in section 3.2.4). We give the timestamp distribution of a first generation "complete_service" message below and note that it is derived in section 3.2.4.

$$pdf\,(x) = \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda A}}.$$

We define *Anti_Inval* as the event that an anti-message invalidates a "complete_service" message processed aggressively. To compute the probability of this event we seek the probability that $x = t + \xi < s$ or alternatively $1 - P\,(s < x)$. Noting that this second form is somewhat easier to work with, we compute this probability below.

$$P\,(Anti\_Inval) = 1 - P\,(s < x) = 1 - \int_0^A \int_0^t \int_0^x \frac{\lambda e^{-\lambda s}}{(1 - e^{-\lambda A})} \; \frac{\lambda e^{-\lambda x}}{e^{-\lambda t} - e^{-\lambda A}} \; \frac{\lambda e^{-\lambda t}}{(1-e^{-\lambda A})} ds\, dx\, dt = .25 \qquad (4.4)$$

In Equation (4.4) the inner most integral is the probability that s is less than x given a particular value $X=x$. Then we integrate over all possible values of $X=x$ times the probability of X. Note that X is a conditional exponential, conditioned on being between $T=t$ and A. In order to uncondition we integrate this over all possible values of $T=t$ times the probability of $T=t$. Since T is a conditional exponential, conditioned on being within the aggressive window, this is done in the outer most integral.

We now have all of the information necessary to calculate the probability of producing a second generation anti-message. Let $Anti_2$ be the event that an LP produces a second generation anti-message. The probability of this event is

$$P\,(Anti_2) = P\,(Anti)\,P\,(C)\,P\,(2nd)\,P\,(Anti\_Inval). \qquad (4.5)$$

The probability that a given LP produces higher order generation anti-messages is computed in a similar manner. We can develop a simple recurrence relation to denote the probability that a given LP produces an *Nth* generation anti-message. In particular, an *Nth* generation anti-message is produced when an LP receives an $N-1$ generation anti-message, it has processed the "complete_service" message aggressively, the next activity to enter into service completes within the aggressive window and the timestamp of the anti-message is less than that of the "complete_service" message. We give this probability below.

$$P(Anti_N) = P(Anti_{N-1})\,P(C)\,P(2nd)\,P(Anti\_Inval) \tag{4.6}$$

4.3. Simulation Results

In order to test the accuracy of our predictions we simulated a simple FCFS queueing network with N=1500 LPs. We built the state saving and rollback mechanism into our simulation in order to test the impact of the correction mechanism. The system meets some of the assumptions of our model. As we discuss below however many of the major assumptions are not met. The system does meet the assumptions of an exponential service distribution, a uniform distribution of messages to the other LPs and that all messages in the aggressive window are processed before the first arrival message is received. Similarly, the first arrival message is processed before the second arrival message is received and so forth. This order of processing represents the worst case assumption as we discussed in section 3.2.1.

Probably the most important assumption of our model that is *not* met by the simulation is the assumption that all of the activity in the window can be captured by considering only those messages in the aggressive window at the synchronization point. This is a significant divergence from the assumptions of our model and is discussed more fully below. Also the simulation does not force arrival messages to be either first or second generation messages as is assumed in our model. Finally the predicted results use the expected value of L (the width of the lookahead window), and as we discussed in section 3.3.5 the width of the lookahead window is actually a random variable rather than a constant.

We developed two sets of predicted results. In one set (shown in Figure 4.1) we account for first and second generation arrival messages. In the second set (shown in Figure 4.2) we assume all arrival

messages are first generation messages (recall we discussed these terms in section 3.2.4). We did this in order to show the impact of the timestamp distribution approximation on the predicted results.

In Figure 4.1 we plot the predicted versus observed probability of a first generation anti-message for a system with N=1500 LPs, a mean service time of $1/\lambda = 1$ and aggressive window sizes from 10% to 100% of the mean service time. This figure shows our predictions given the assumption that all arrival messages are either first or second generation messages. Each observed data point represents the average of sixteen simulation runs, where each run represents one thousand iterations of the algorithm. There was very little variation between the sixteen runs.

As can be seen our predictions are quite accurate for a very reasonable range of aggressive window sizes. Also, the probability that an LP produces an anti-message is quite small even for large aggressive window sizes. This is very encouraging. Note however that as the aggressive window size approaches the mean service time our model begins to under-predict the probability of a first generation anti-message. We now discuss the reason for this under-prediction.

Our model focuses only on the messages in the aggressive window at the synchronization point. This presents an accurate picture of the state of an LP before it begins to process in the new simulation window. It is a static picture however and does not capture the effects of processing within the aggressive window. This is because it does not account for the probability that a "complete_service" message that is processed within the aggressive window may place an activity into service which also completes within the aggressive window. This other completion within the aggressive window implies that a second "complete_service" message will be placed on the event list and (since its timestamp falls within the aggressive window) will also be processed aggressively (see section 3.2.4 for a further discussion of this issue). This second "complete_service" message (of course there may be other such messages as well) may also be invalidated through the receipt of an arrival message and thus may also produce a first generation anti-message. Clearly the size of the aggressive window determines the probability of more than one "complete_service" message being generated within the aggressive window. As can be seen in Figure 4.1, as the aggressive window size approaches the mean service time this happens enough such that our model under-predicts the probability of an anti-message.

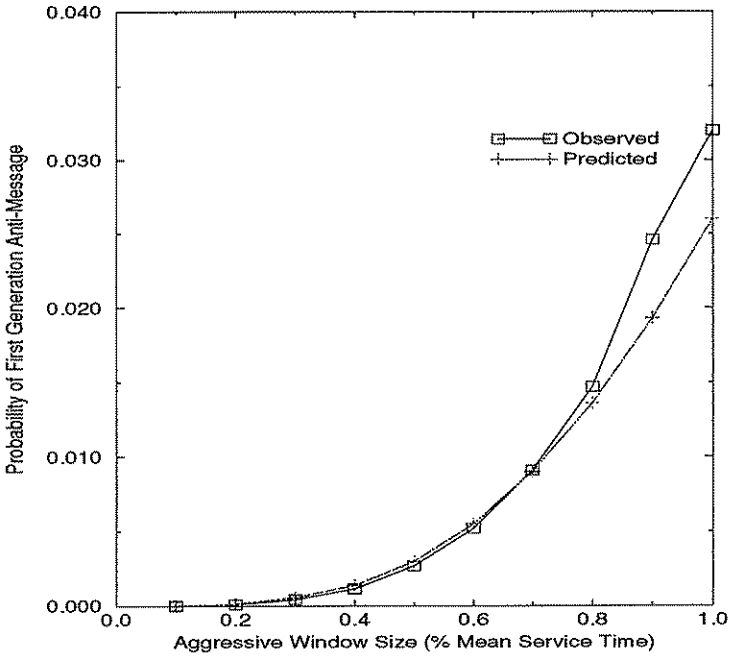

Figure 4.1 - Probability of Producing a First Generation Anti-Message

This is a good example of the trade-off between precise predictions and the tractability of our model. Our model for predicting the probability of an anti-message is reasonably simple, and yet gives very reasonable results for a wide range of aggressive window sizes. We could have developed our model such that we consider more than one "complete_service" message produced as a result of processing within the aggressive window. The model would however become intractable very quickly. We feel we have reached a reasonable compromise between the complexity of the model and the accuracy of our predictions at the upper bound of the aggressive window size.

As discussed in section 3.2.4.2, a much simpler approach is to assume all arrival messages are first generation messages. It is interesting to show the differences in the predictions of our model depending on the timestamp approximation for arrival messages. In Figure 4.2 we plot the predicted and observed probabilities of an anti-message assuming all arrival messages are first generation messages. Also, we show the predicted results when we account for second generation arrival messages. As can be seen, assuming all arrival messages are first generation messages bounds from above the probability of an anti-

message. It is interesting to note that the observed probability falls between our two predicted values.

As can be seen the predicted and observed values of the probability of a first generation anti-message are quite small. Given this very low probability of a first generation anti-message the probability of cascading rollbacks developing is quite small. This is because in order to produce a second generation anti-message all of the conditions required for a first generation anti-message are required (and obviously the probability of these conditions occurring is quite small) *and* the LP must receive a first generation anti-message. To see this, consider our predictions for higher order generation anti-messages given an aggressive window size set to its maximum value of the mean service time. Our model predicts that the probability of a second generation anti-message is .001, the probability of a third generation anti-message is .0001 and the probability of a fourth generation anti-message is .000006. In order to test our predictions we tracked the maximum generation anti-message observed *out of all of our simulation runs.* In Table 4.1 we show the highest order generation anti-message observed for this system given aggres-

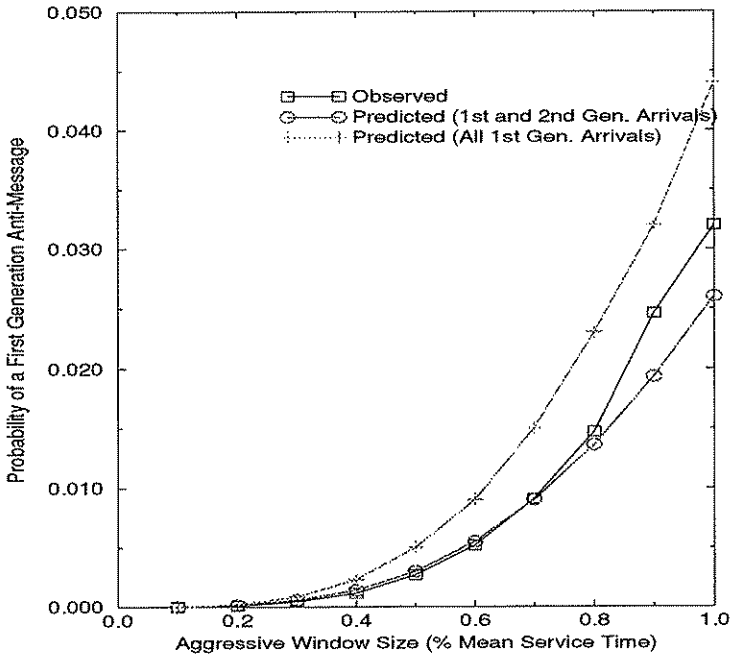

Figure 4.2 - Probability of Producing a First Generation Anti-Message

sive window sizes from 10% to 100% of the mean service time. As can be seen there is no observed problem with cascading rollbacks developing.

4.4. Scalability Issues

As discussed in Chapter 3, we have not yet developed scalability results for a system such as ours. We have however developed a set of results which we feel represent a significant step towards proving system-level scalability. In this section we investigate the probability that a given LP produces an anti-message as the number of LPs in the system approaches infinity. We feel that we can use these results as the foundation for system-level scalability results.

Aggressive Window Size (% MST)	Maximum Observed Anti-Mess.
.1	1st
.2	2nd
.3	2nd
.4	2nd
.5	3rd
.6	3rd
.7	3rd
.8	3rd
.9	4th
1.0	4th

Table 4.1 - Highest Order Generation Anti-Message Observed in the System

We are interested in the probability that an LP produces a first generation anti-message as the number of LPs and number of processors approaches infinity. The approach we use is the same as outlined in section 3.5, and we do not reiterate the arguments in this section. As discussed in section 3.5, the value of N (the number of LPs in the system), is not directly reflected in our equations. Rather, this value is contained in the value of L, the width of the lookahead window. In order to show the dependence of L on N we use the notation $L(N)$ for the remainder of this section. As discussed in section 3.5, as $N \rightarrow \infty$ $\lambda L(N) \rightarrow 0$. Thus to investigate the behavior of our protocol as N approaches infinity, we investigate its behavior as $\lambda L(N) \rightarrow 0$.

As discussed in section 3.5, we need to account for the fact that the synchronization costs increase as the size of the architecture increases. In order to keep the workload per processor constant, we assume (in a system with P processors) there are $K = P\ Log_2\ P$ LPs per processor. Thus we assume there will be $N = K\ P$ LPs in the system. We investigate the probability an LP generates an anti-message as N approaches infinity.

In Figure 4.3 we show the (predicted) probability of producing an anti-message (at a given LP) as $\lambda L(N) \rightarrow 0$. In this graph the aggressive window size is set to the maximum value we consider (i.e. $A = 100\%$ of the mean of the service time distribution). Our model predicts that as more LPs are added to the system the probability of a given LP producing a first generation anti-message declines. Note that it reaches the lowest probability (approximately .023) at $\lambda L(N) = 0$ (i.e. the limit as $N \rightarrow \infty$). This is very encouraging since it implies that the probability a given LP initiates a rollback chain does not increase *as the number of LPs approaches infinity.*

As can be seen from Figure 4.3, the probability of a given LP producing an anti-message is approximately .12 at $\lambda L(N) = 2$, and declines as $L(N)$ decreases. It is important to note that in a system with approximately 50 LPs, the value of $\lambda L(N)$ will be approximately .17 (see section 3.3 for a discussion of determining the number of LPs in a system for a given value of λL). In Figure 4.3 we have drawn a vertical line at $\lambda L = .17$. This is important because it shows that in a system with 50 or more LPs the probability of producing an anti-message (at a given LP) is reasonably low ($\leq .05$). Thus it is only in a relatively small system that we would expect to observe a probability of producing an anti-message (at a given

LP) that is higher than .05.

Again we note this probability of an anti-message is the probability at a given LP, not in the system. We discuss system level performance in the following chapters. In order to test our predictions that the probability of producing an anti-message (at a given LP) declines as more LPs are added to the system we increased the number of LPs in our simulated system from 1500 to 3000. We then re-ran our simulation for an aggressive window size that is 100% of the mean service time, and 50% of this mean. For an aggressive window size set to 100% of the mean service time, the probability of a first generation anti-message given 3000 LPs is 0.0319 compared with 0.032 for 1500 LPs. For an aggressive window size set to 50% of the mean service time, the probability of a first generation anti-message is 0.0025 for a system with 3000 LPs versus 0.027 for a system with 1500 LPs. While these reductions in the probability of an anti-message are minimal, it is encouraging to note that this probability does not increase as the number of LPs increases. As discussed, we believe these results will lay the foundation for proving

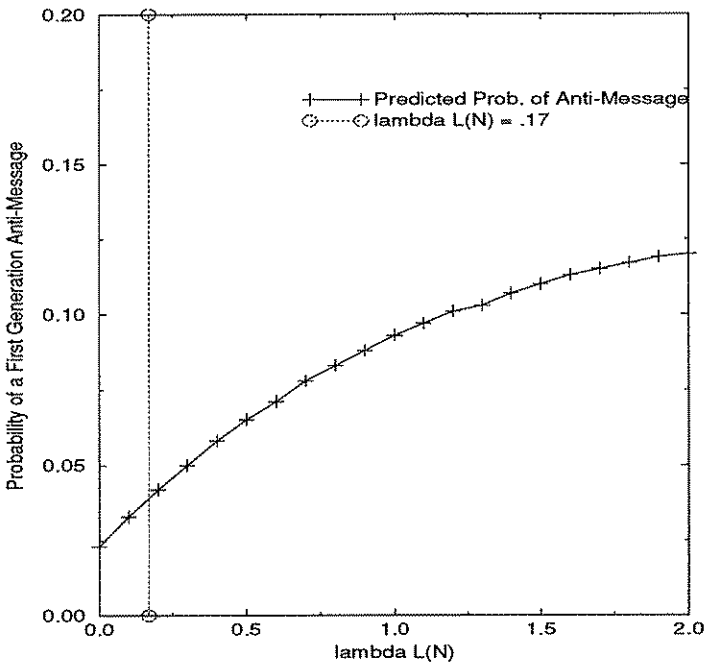

Figure 4.3 - Probability of Producing an Anti-Message as $N \rightarrow \infty$

system-level scalability.

4.5. Conclusions

In this chapter we have defined our correction mechanism and derived the probability that an LP produces a given generation anti-message *as a function of the level of aggressiveness*. We stress that this is the first time this relationship has been established. Also we have shown theoretically and through simulation studies that the probability of an anti-message at a given LP actually decreases slightly *as the number of LPs approaches infinity*. This is a very encouraging result.

What we have not yet addressed is the issue of *system level performance*. Until now we have not laid the theoretical foundation necessary to discuss this issue. In the next chapter we relax our assumption of the closed system and model an open system with external Poisson arrival streams. Then we use the theoretical base we have built to investigate system level performance.

CHAPTER 5

Analysis of an Open System

The model developed in previous chapters assumed a closed system that is heavily loaded. As we discuss below, the assumption of a heavily loaded system simplifies our analysis significantly. The goal of this chapter is to show that we can relax our assumption of a closed system that is heavily loaded, and extend our analysis to include a lightly loaded system provided it is open. The model developed in this chapter assumes an open system where each LP receives messages from an external source as well as from other LPs in the system. We discuss the details of this new model below.

The purpose of this chapter is to demonstrate that our analysis can be extended to an open system rather than to give a complete analysis of such a system. As will be seen, our approach to the equations for an open system are more tedious and complex than those for a closed system. Thus while we show how to derive the required distributions and desired equations, we do not give the full range of theoretical and empirical results that we have for a closed system. Our goal is to demonstrate that our analysis can be extended to an open system, and to show what is required in order to make this extension.

The remainder of this chapter is organized as follows. In section 5.1 we develop our model for the open system. In section 5.2 we derive the distributions of the number of aggressive messages at the synchronization point, the number of arrival messages, the timestamp of aggressive messages and the timestamp of arrival messages for an open system. In section 5.3 we show how to use these distributions to compute the probability of a fault and the expected number of messages processed successfully at a given LP. Also, we give simulation results to check the accuracy of our model. In section 5.4 we discuss the calculations involved in computing the probability that an LP produces an *Nth* generation anti-message. Also in section 5.4 we give simulation results which support the predictions of our model. We give our conclusions in section 5.5.

5.1. Model of Open System

The model of the open system is very similar to the model of a closed system developed in previous chapters. We model the system as a collection of servers where activities occur. When an activity completes service at a given server it causes other activities to occur. In our model we assume each completion causes exactly one other activity. The delay in simulation time between when an activity begins and ends is called the *duration* of the activity. We assume each server chooses the duration of an activity from independent, identically distributed exponential random variable with mean $1/\lambda$. We assume there is one server per LP and N LPs in the system.

As in previous chapters, we treat the width of the lookahead window (L) as a constant equal to its expected value rather than a random variable. This expected value can be obtained analytically (see Nicol 1993) or through sample simulation runs.

We assume a server is never idle if there is an activity available to receive service. As in previous chapters we assume an LP processes all of its messages in the aggressive window before receiving any arrival messages. In section 3.2.1 we discussed how this is a worst case assumption in that an arrival message will invalidate the maximum number of messages possible.

Each LP receives messages from both an external source and from other LPs in the system. The external arrival messages are in the form of a Poisson process with rate λ_E. As discussed in section 3.2.1, the messages received from the other LPs in the system are the "pre-sent" completion messages.

Once an activity completes service it departs the system with probability p_0 and stays in the system with probability p_1. If an activity stays in the system the next LP to receive it is chosen at random, where each LP is equally likely to be chosen.

In section 3.2.4 we discuss the timestamp distribution of arrival messages. As discussed, this timestamp distribution can be quite complicated, depending on the circumstances under which the arrival message is produced. In section 3.2.4, we showed if we restrict the size of the aggressive window, we can approximate the timestamp distribution by assuming all arrival messages are first generation messages. With this assumption our analysis is greatly simplified. However, this assumption gives a pessimistic

view of the performance of our protocol (i.e. it over-estimates the probability of a fault and under-estimates the expected improvement in performance). As discussed, we can make the timestamp distribution more exact and assume all arrival messages are either first, or first and second generation messages. In this case the results are more accurate, but the analysis is much more complicated. Thus there is a trade-off between the complexity of the analysis and the exactness of the results.

As in Chapter 3, we restrict the size of the aggressive window to be no greater than the mean of the service time distribution. There is no special significance in this choice. It is chosen because it is large enough to demonstrate the error of the timestamp distribution approximation as the size of the aggressive window increases, and small enough such that we feel the predictions of our model are still reasonable at the upper end of the aggressive window size.

As will be seen, the analysis in this chapter is substantially more complicated than that presented in previous chapters. Due to the complexity of the analysis, we (except where explicitly noted) make the simplifying assumption that all arrival messages are first generation messages. As discussed, this approximation tends to give a more pessimistic view of the performance of our aggressive protocol than when we consider second or higher order generation messages. Without this assumption however the analysis becomes intractable. We discuss the effects of this simplifying assumption during the course of our analysis.

Now that we have defined our model we derive the distributions needed to determine the probability of a fault and the expected improvement due to aggressive processing. We begin by deriving the input rate to a given LP.

5.2. Distributions

In this section we derive the distribution of the number of messages in the aggressive window at the synchronization point, the number of arrival messages, the probability that an arrival message invalidates k messages and so forth. In order to derive these distributions we first must determine the total input rate to a given LP. We do this below.

5.2.1. Total Input Rate

The total input rate (λ_T) at each LP is the sum of the external arrival rate plus the arrival rate from the other LPs in the system (internal arrival rate).

$$\lambda_T = \lambda_E + \lambda_I \tag{5.1}$$

As noted by Kleinrock (1975), Jackson (1957) showed that each LP in this system will *behave* as an independent M/M/1 system with Poisson input stream with rate λ_T. This is true even though the total input rate will not in general be a Poisson process.

The external arrival stream is, by definition, Poisson with rate λ_E. In order to determine the rate of the internal arrival stream, consider a given LP_i. LP_i will have a total input stream with rate λ_T. After each activity is processed the activity will stay in the system with probability p_1. The output rate of LP_i (that stays in the system) will therefore be $\lambda_T p_1$. This total output rate from LP_i will fork into N output streams since each LP is equally likely to receive a given activity. Therefore the input stream to a given LP_j from LP_i will have rate $\lambda_T p_1 1/N$. The total input stream to a given LP_j (from internal sources) will be the sum of all such input streams.

$$\lambda_T = \lambda_E + \lambda_I = \lambda_E + \sum_{i=1}^{N} \lambda_T p_1 1/N = \lambda_E + p_1 \lambda_T. \tag{5.2}$$

We now solve for the total input rate λ_T.

$$\lambda_T(1 - p_1) = \lambda_E \tag{5.3}$$

$$\lambda_T = \frac{\lambda_E}{p_0}.$$

The utilization of the server at a given LP (denoted by the symbol ρ) is defined to be $\rho = \lambda_T/\lambda$. The condition for stability in the system is $0 \leq \rho < 1$. We assume this condition is always true (that is, we assume that the parameters of the system are always chosen such that this condition will hold).

Now that we have determined the total input rate to a given LP we can derive the distribution for the number of messages in the aggressive window at the synchronization point. We do this in the following section.

5.2.2. Number of Messages in the Aggressive Window at the Synchronization Point

After the LPs have completed processing within the simulation window they synchronize globally to define the new simulation window. In the non-aggressive version of the algorithm the LPs define the lookahead window. Recall this is defined such that all events with timestamps falling within the window can be executed concurrently without any possibility of a causality error. In the aggressive version of the algorithm the LPs define the aggressive window as an extension to the lookahead window. Events within the lookahead window are the only events executed between global synchronization points in the non-aggressive algorithm. If we temporarily ignore the arrival messages, then the events processed within the aggressive window represent the extra processing allowed by the aggressive algorithm between global synchronization points. We are interested in how much extra processing is performed between global synchronization points due to aggressive processing. As in Chapter 3, we compute the extra parallelism available due to aggressive processing by determining the number of events with timestamps falling within the lookahead window, and the number of events with timestamps falling within the aggressive window.

We assume all of the external arrivals with timestamps falling within the lookahead window have arrived by the time the LPs synchronize to determine the next lookahead window. This assumption is required for external arrival messages with timestamps that fall within the lookahead window since the LPs cannot determine a lookahead window if it is possible to receive external arrival messages with unknown timestamps. While external messages could fall within the aggressive window without affecting the correctness of the aggressive version of the algorithm, the analysis is simplified by assuming they are present at the synchronization point. Therefore we make this assumption and note that one way to ensure this requirement is to generate all of the external arrivals at the beginning of the simulation.

At the synchronization point, the number of messages in the aggressive window is the sum of the external arrivals and internal arrivals with timestamps that fall within the aggressive window. The number of messages due to external arrival messages is Poisson distributed with rate $\lambda_E A$, where A is the size of the aggressive window. The number of messages due to internal arrivals is more difficult to determine. We determine this distribution in the remainder of this section.

Recall when an activity enters into service at a given LP, the LP determines both the completion time of the activity and the next LP to receive this activity upon its completion. The LP then sends a message to the next LP to receive this activity, informing the LP of this future arrival. Recall we term these messages *"pre-sent" completion messages* because they are used to inform the next LP of this future arrival. An LP is informed of this future arrival at the wall-clock time the activity enters into service. As noted, these are the only messages exchanged by the LPs.

For the purposes of this analysis there are two important points regarding the "pre-sent" completion messages. First, the timestamp of the "pre-sent" completion message is the logical time the activity completes service, which is also the logical time the activity arrives at the next LP. Second, assume the LPs synchronize at logical time T to define the new simulation window, and consider the servers that are busy at the synchronization point (i.e. logical time T). Each such server will be busy with an activity that began service at logical time $s < T$, completes at time $T' > T$, and whose completion was pre-sent. Thus there will be one "pre-sent" completion message for each server that is busy. We seek to determine the distribution of the number of such "pre-sent" completion messages in the aggressive window of a given LP at the synchronization point. As noted, these messages are processed in the aggressive version of the algorithm and are not processed in the non-aggressive algorithm.

As discussed by Kleinrock (1975, page 18), the utilization factor (ρ) can be interpreted as the expected fraction of servers that are busy in the steady state (assuming $0 \leq \rho < 1$).

$$\rho = E[\textit{fraction of busy servers}] = \frac{\lambda_T}{\lambda} \tag{5.4}$$

Given that the expected fraction of busy servers is known, we can determine the expected number of "pre-sent" completion messages that will be in the system at the synchronization point (recall there is one such message for every busy server). It is expected there will be ρN servers busy at the synchronization point in a system with N servers. Recall the probability that a "pre-sent" completion message stays in the system is p_1. Thus there will be αN "pre-sent" completion messages in the system at the synchronization point, where α is defined as the expected fraction of busy servers times the probability that a given "pre-sent" completion message stays in the system.

$$\alpha = \rho\, p_1 \tag{5.5}$$

The probability of a given LP receiving one of these "pre-sent" completion messages is $1/N$ (due to the assumption that each LP is equally likely to receive a given message). Let K be the random variable denoting the number of "pre-sent" completion messages received by a given LP. The probability $K=k$ is

$$P(K=k) = (\frac{1}{N})^k\ (\frac{N-1}{N})^{\alpha N-k}\ \ \frac{\alpha N!}{k!(\alpha N-k)!} \tag{5.6}$$

Note in Equation (5.6) there is a large number of trials (αN) and the probability of success at any trial is small ($1/N$). As shown by Brieman (1986), in this situation the binomial is closely approximated by the Poisson distribution with rate equal to the number of trials times the probability of success at any trial.

$$P(K=k) \simeq \alpha^k/k! \tag{5.7}$$

Using the same approach as in Chapter 3 (Equations (3.4) through (3.11)), it can be shown that the number of "pre-sent" completion messages in the aggressive window at the synchronization point is closely approximated by the Poisson distribution with rate $\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)}$.

$$P(K=k) \simeq \frac{(\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)})^k}{k!}\ \ e^{-(\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)})} \tag{5.8}$$

As noted, we use the Poisson distribution as an approximation to the binomial distribution. It is convenient to assume this is the true distribution. We do so for the rest of this analysis, but note our analysis is built on this approximation.

Equation (5.8) gives the number of messages at the synchronization point from internal sources. Recall an LP receives messages from both internal and external sources. Further recall (by assumption) the external messages that will arrive in either the lookahead or the aggressive window will be present when the LPs reach the synchronization point. Given that the number of external messages in the aggressive window at the synchronization point is Poisson distributed with rate $\lambda_E A$, and the number of internal messages is Poisson with rate $\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)}$, the total number of messages in the aggressive window at the synchronization point will be the sum of these two Poisson streams. Thus the number of messages will be Poisson distributed with rate $\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)} + \lambda_E A$. Let Ψ be the rate of the merged Poisson streams.

$$\Psi = \alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)} + \lambda_E A \;=\; \alpha e^{-\lambda L}\,(1 - \alpha e^{-\lambda A}) \tag{5.9}$$

Then the number of messages in the aggressive window at the synchronization point is

$$P(K=k) = \frac{\Psi^k}{k!} e^{-\Psi} \tag{5.10}$$

Also, we are interested in the number of messages in the lookahead window at the synchronization point. As discussed in section 3.2.4, the system can be viewed as probabilistically restarting at the synchronization point* (further recall we refer to this as logical time T). That is, if the current floor of the simulation window is logical time T, we can view the lookahead window as extending from logical time 0 to L, rather than logical time T to $T+L$. Similarly, we can view the aggressive window as extending from logical time L to $L+A$ rather than logical time $T+L$ to $T+L+A$. This is what we have done in Equation (5.9) where we compute the distribution of the number of messages in a window extending from logical time L to $L+A$. Similarly, we can compute the number of events in a window extending from logical time 0 to L by substituting 0 for L, and substituting L for A in Equation (5.9). Making this substitution, we see the number of messages in a window from logical time 0 to L (i.e. the lookahead window) is Poisson distributed with rate $\alpha - \alpha e^{-\lambda L} + \lambda_E L$. Let θ be the parameter of the Poisson distribution in the lookahead window at the synchronization point.

$$\theta = \alpha - \alpha e^{-\lambda L} + \lambda_E L. \tag{5.11}$$

Let J be the random variable denoting the number of messages in the lookahead window at the synchronization point. Then

$$P(J=j) = \frac{\theta^j}{j!} e^{-\theta} \tag{5.12}$$

5.2.3. Complete_Service Message

In the above equations we calculate the probability that an LP has K "pre-sent" completion messages (in the aggressive or lookahead windows) from both internal and external sources. As discussed in section 3.2.2.1, every LP with a server that is busy will have a "complete_service" message scheduled on its event list. We need to account for this message in our equations.

*We again note that this is an approximation since the protocol chooses the value of T. As we have shown however this approximation is quite good.

In section 3.2.2.1, we showed that (given the server is busy) the probability the "complete_service" message has a timestamp which falls within the lookahead window is $1 - e^{-\lambda L}$. Similarly, we showed the probability the "complete_service" message falls within the aggressive window is $e^{-\lambda L} - e^{-\lambda(L+A)}$. These probabilities are conditioned on the server being busy.

In order to uncondition we multiply the probability that the "complete_service" message will fall within the lookahead or aggressive windows by the probability that the LP has a "complete_service" message scheduled. Recall ρ ($0 \leq \rho < 1$) is the server utilization, and therefore represents the probability the server will be busy at any given time. For our purposes it represents the probability that an LP will have a "complete_service" message scheduled on its own event list. The unconditioned probability that an LP has a "complete_service" message in the lookahead window is $\rho(1-e^{-\lambda L})$ and the probability the "complete_service" message exists and falls within the aggressive window is $\rho(e^{-\lambda L}-e^{-\lambda(L+A)})$. Let σ be the probability an LP has a "complete_service" message in the lookahead window and κ be the probability that an LP has a "complete_service" message in the aggressive window.

$$\sigma = \rho(1-e^{-\lambda L}) \tag{5.13}$$

$$\kappa = \rho(e^{-\lambda L}-e^{-\lambda(L+A)}) \tag{5.14}$$

Let X be the random variable denoting the total number of messages (including "pre-sent" completion messages and a "complete_service" message) in the aggressive window, and let Y be the random variable representing the number of such messages in the lookahead window.

In order to have $X=0$ messages in the aggressive window an LP must have no "pre-sent" completion messages (in the aggressive window) and the "complete_service" message (if it exists) must fall outside of the aggressive window. Let ($M=0$) be the event that an LP has no messages in the aggressive window. Let $(\overline{C})$ be the event that an LP does not have the "complete_service" message in the aggressive window. The probability that an LP has zero messages in the aggressive window is

$$P(X=0) = P(M=0,\overline{C}) = e^{-\Psi}\ (1-\kappa). \tag{5.15}$$

Let $M'=0$ be the event that an LP has no "pre-sent" completion messages in the lookahead window. Let $\overline{C'}$ be the event that an LP does not have the "complete_service" message in the lookahead window. The probability of zero messages in the lookahead window is

$$P(Y=0) = P(M'=0, \overline{C'}) = e^{-\theta}(1-\sigma). \tag{5.16}$$

Now consider the probability of $(X=x) \geq 1$ messages in the aggressive window. One way for this to occur is for the LP to have x "pre-sent" completion messages and not have the "complete_service" message in the aggressive window. Alternatively, the LP could have $x-1$ "pre-sent" completion messages and have the "complete_service" message in the aggressive window.

$$P(X=x, x \geq 1) = P(M = x-1,\ C) + P(M = x, \overline{C}) = \tag{5.17}$$

$$\frac{(\Psi)^{x-1}}{(x-1)!} e^{-\Psi}(1-\beta) + \frac{(\Psi)^{x}}{x!} e^{-\Psi}\,\kappa$$

Similarly, the probability that an LP has $(Y=y) \geq 1$ messages in the lookahead window is

$$P(Y=y, y \geq 1) = P(M' = y-1, C') + P(M' = y, \overline{C'}) = \tag{5.18}$$

$$\frac{(\theta)^{y-1}}{(y-1)!}\,\sigma + \frac{(\theta)^{y}}{y!}(1-\sigma)$$

5.2.4. Distribution of Arrival Messages

In this section we derive the probability distribution for the number of messages that arrive in the aggressive window. Recall we term a message received by an LP, such that the timestamp of the message falls within the aggressive window, an *arrival message*. As discussed in section 3.2.1, we assume an LP processes all of its messages within the lookahead and aggressive windows before it receives any arrival messages. As noted, this may not be strictly true, but is reasonable given the relatively small aggressive window sizes we consider. Further, as shown in section 3.2.1, this represents the worst case assumption. Given that all messages within the aggressive window have been processed by the time an LP receives an arrival message, if the timestamp of this message is less than that of any message processed aggressively, this results in a causality error. Since arrival messages can lead to causality errors it is important to determine the distribution of the number of such messages received by a given LP.

Recall our assumption that all external messages will have arrived by the time the LPs reach the synchronization point. Thus the only additional arrival messages will be those generated by the other LPs in the system.

In Equation (5.2) we showed

$$\lambda_E + \lambda_I = \lambda_E + \lambda_T p_1.$$

Thus

$$\lambda_I = \lambda_T p_1.$$

That is, the internal arrival rate into a given LP is equal to the total input rate times the probability that a message stays in the system.

In Equation (5.8) we showed that the number of messages in the aggressive window (at the synchronization point) due to internal sources is Poisson distributed with rate $\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)}$. As shown above, the total internal arrival rate to a given LP is Poisson with rate $\lambda_T p_1$. The total arrival stream in the aggressive window (which has a width of A logical units) from internal sources will therefore be Poisson distributed with rate $\lambda_T p_1 A$. We break this total internal arrival rate into two component rates. First, the rate for the messages in the aggressive window at the synchronization point, and second, the rate of the internal arrival messages.

$$\lambda_T p_1 A = \lambda_{arrival} + (\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)}) \tag{5.19}$$

$$\lambda_{arrival} = \lambda_T p_1 A - (\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)})$$

We conclude that the number of messages that arrive in the aggressive window, and thus the distribution for the number of arrival messages, is Poisson distributed with rate $\lambda_T p_1 A - (\alpha e^{-\lambda L} - \alpha e^{-\lambda(L+A)})$.

5.3. Probability of a Fault at a Given LP

In Chapter 3, we showed (assuming a closed system) the timestamps of the messages in the aggressive window at the synchronization point are conditional exponentials, conditioned on falling within the aggressive window. Also we argued that a reasonable approximation for the timestamp distribution of arrival messages is to assume they are also conditional exponentials, conditioned on falling within the aggressive window. Given these distributions, we showed how to compute the probability of a fault and the probability of successfully processing K messages in the aggressive window.

In an open system it is somewhat more difficult to determine these values. This is because there are two different sources of messages that fall within the aggressive window, and two different timestamp

distributions we must consider. First, some of the messages in the aggressive window (at the synchronization point) are due to internal sources. That is, these are the messages exchanged by the LPs. Note that we have discussed these messages at length in sections 3.2.2 - 3.5. As discussed, the timestamps of these internal messages are conditional exponentials, conditioned on falling within the aggressive window. The second type of messages are those received from the external Poisson process. By definition of the Poisson process, we know that the distance between the timestamps of these external messages is exponentially distributed.

Consider the calculation of the probability of a fault at a given LP. Recall our assumption that all of the external messages have arrived in the aggressive window by the time the LPs synchronize to define a new simulation window. Thus it is only the receipt of an arrival message (from an internal source) that can result in a causality error. Recall we derived the probability distribution of the number of arrival messages received by a given LP in Equation (5.19). An arrival message has the potential to invalidate either of the two types of messages (i.e. the messages from external sources or the messages from internal sources) in the aggressive window. First consider the messages due to internal sources.

In Equation (5.8), we derive the distribution of the number of messages in the aggressive window at the synchronization point due to internal sources. Given that we know the distribution of the number of such messages in the aggressive window, the timestamp distribution of these messages, the timestamp distribution of the arrival messages, and the distribution of the number of arrival messages received, we can determine the probability of a fault and the expected number of messages successfully processed. This is exactly what we discussed in detail in section 3.4, and do not reiterate. We term a fault caused by an arrival message invalidating an internal message an *internal fault*. We denote the probability of such a fault as P (*Fault Internal*).

Now consider the messages in the aggressive window of a given LP due to the external Poisson arrival process with rate λ_E. We seek the probability that an arrival message invalidates one of these messages. Let P (*Fault External*) be the probability that an arrival message invalidates an external message in the aggressive window. Consider an aggressive window extending from logical time $0 .. A$, and assume the LP receives an arrival message with timestamp t such that $0 < t < A$. Consider the number of external

messages in the interval from t .. A. Note this is exactly the number of messages in the aggressive window with a timestamp greater than that of the arrival message, and if this number is greater than zero then the LP has faulted.

By definition of the Poisson process, the probability of $K=k$ messages in an interval from t .. A is

$$P\,(K=k \text{ Messages in interval } t \,..\, A) = \frac{(\lambda_E\,(A-t)\,)^k}{k!}\; e^{-\lambda_E(A-t)}.$$

Thus we know the probability of $K=k$ messages (in the aggressive window) with timestamps greater than that of an arrival message with timestamp $T_A = t$. In order to uncondition this expression we integrate over all possible values of $T_A = t$, times the probability of $T_A = t$. Recall our assumption that the timestamp of an arrival message is a conditional exponential, conditioned on falling within the aggressive window. The unconditioned probability of $K=k$ messages in the aggressive window with a timestamp greater than that of an arrival message is thus

$$P\,(K=k \text{ Messages with timestamps} > T_A = t) = \int_0^A \frac{(\lambda_E\,(A-t)\,)^k}{k!}\; e^{-\lambda_E(A-t)}\; \frac{\lambda\, t\, e^{-\lambda t}}{(1-e^{-\lambda A})}\, dt.$$

The probability of $K = 0$, and thus the probability that an arrival message does *not* invalidate an external message is

$$P\,(\overline{\text{Fault External}}) = \int_0^A e^{-\lambda_E\,(A-t)}\; \frac{\lambda\, t\, e^{-\lambda t}}{(1-e^{-\lambda A})}\, dt.$$

The probability of *not* faulting given one arrival message is

$$P\,(\text{No Fault}) = P\,(\overline{\text{Fault Internal}})\;\; P\,(\overline{\text{Fault External}}).$$

That is, the arrival message must not invalidate an external message, and it must not invalidate an internal message. The probability that an arrival message *does* cause a fault is one minus this probability.

$$P\,(\text{Fault}) = 1 - P\,(\text{No Fault})$$

The probability of faulting given more than one arrival message is similar to other such derivations given in previous chapters and is not reiterated.

Now we have derived all of the equations necessary to calculate the probability of a fault (at a given LP) for an open system. Clearly these same equations and distributions are sufficient to compute the expected number of messages processed successfully at a given LP. As discussed, the number of

internal messages processed successfully is computed exactly as shown in section 3.4. The number of external messages processed successfully is computed in a manner similar to the probability of invalidating K messages shown above. In particular, the probability of processing $K=k$ messages successfully given an arrival message with timestamp $T_A = t$, is the probability that there are $K=k$ messages in the interval from $0 .. t$. As above, we need to consider all possible timestamps $T_A = t$, times the probability of $T_A = t$. We give the unconditioned expression below.

$$P(K=k \text{ Messages with timestamps} < T_A = t) = \int_0^A \frac{(\lambda_E t)^k}{k!} e^{-\lambda_E t} \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda A}} dt$$

As can be seen, our analysis of a closed system can be extended to model an open system with an external Poisson arrival process. Below we give simulation results to demonstrate the correctness of our analysis.

5.3.1. Simulation Results

In this section we give a set of simulation results to examine the accuracy of our equations. We simulated an open queueing system where an LP received messages from both internal sources and from an external Poisson arrival process with rate λ_E. As discussed, we generated all of our external arrival messages at the beginning of the simulation. In our analysis, each data point represents the average taken over sixteen trials, where each trial consisted of 1000 iterations of the algorithm. There was little variance between the sixteen trials.

Recall in Chapter 3 we simulated a large system (1500 *LPs*) which was heavily loaded. In this section we simulate a small system (250 *LPs*) that is lightly loaded. The particular system we simulated had an external arrival rate of $\lambda_E = 0.5$, and an exponential service time distribution with mean $1/\lambda = 1$. The probability that a message stayed in the system was $p_1 = 0.2$, and the probability a message exited the system was $p_0 = 0.8$. Thus the server utilization of our system was $\rho = 0.625$.

In Figure 5.1 we show the predicted and observed probability of a fault given an aggressive window size from $A = 10\%$ of the mean of the service time distribution to $A = 100\%$ of the mean of the service time distribution. As can be seen, our predictions are quite close for the smaller aggressive window

sizes. As the size of the aggressive window increases however our equations begin to over-predict the probability of a fault. This was exactly the pattern we observed in our predictions for the closed system. As discussed in detail in section 3.2.4.2, the reason for this over-prediction is the error of our timestamp approximation for the arrival messages. As discussed, the larger the size of the aggressive window, the larger the error of this approximation. As can be seen, given a lightly loaded system the probability of a fault at a given LP is quite small.

The final parameter of interest in an open system is the probability of producing an *Nth* generation anti-message. We derive this probability in the following sections.

5.4. Anti-Messages

In this section we calculate the probability an LP produces an anti-message with a timestamp falling within the aggressive window. Recall we are not concerned with anti-messages whose timestamps fall outside of the aggressive window because the message being cancelled will not yet have been processed. As the message will not yet have been processed there is no way the error can propagate through

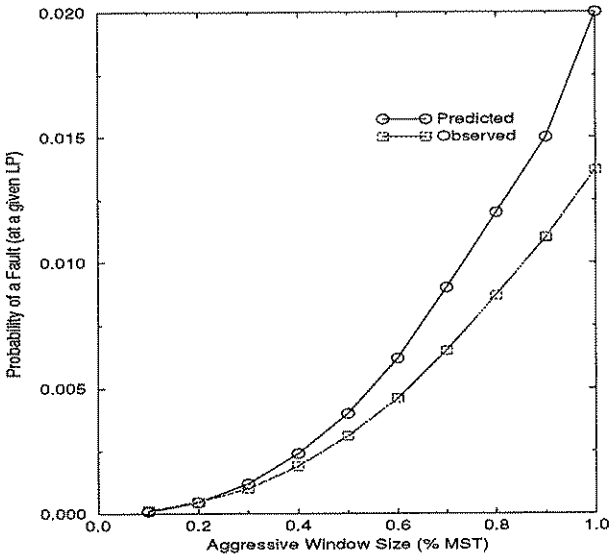

Figure 5.1 - Probability of a Fault in an Open System

the system. As we will show, the computation of the probabilities associated with an anti-message in an open system are much more complicated than in the closed system assumed in previous chapters. Before deriving these probabilities we describe briefly the characteristics of the two systems that affect this computation significantly.

In previous chapters we have assumed a closed system that is heavily loaded. As discussed, this assumption simplifies the derivation of the various probability distributions we have needed. This is particularly true in the calculation of the probability of producing an *Nth* generation anti-message. Consider the impact on our analysis of assuming a closed, heavily loaded system.

Perhaps the most important implication of a closed, heavily loaded system is that with high probability each server will always be busy. While this probability is not necessarily one, we assume this for the purposes of our analysis. This has two important implications with regard to the probability of generating an anti-message. First, it implies that (with high probability) it is only the processing of the "complete_service" message that will cause a new activity to be placed into service. Thus the processing of the "complete_service" message is a necessary, but not sufficient condition to produce an anti-message. Second, each time a "complete_service" message is processed there will (with high probability) be an activity ready to be placed into service. This implies that with high probability the completion time of the next activity to receive service will be the sum of the timestamp of the "complete_service" message and an exponential random variable with mean $1/\lambda$.

These characteristics of a closed, heavily loaded system significantly reduce the number of cases we must consider in determining the probability of an *Nth* generation anti-message. As shown in section 4.3.2, there are three events we must consider. First, the LP must process the "complete_service" message aggressively. This is because it is only the processing of the "complete_service" message which causes a new activity to be placed into service. Thus it is only the processing of the "complete_service" message (aggressively) that can cause the wrong activity to be placed into service. Second, this new activity must complete within the aggressive window. As mentioned above, the calculation of the completion time of this activity is greatly simplified because we know that it will be the sum of the logical time of the "complete_service" message and an exponential random variable. Given this, it is not difficult to

compute the probability that the activity completes within the aggressive window. Third, the processing of the "complete_service" message must be invalidated by the receipt of an arrival message. Thus the number of events to consider in a closed system is quite small.

As can be seen, the computation of the probability of an anti-message is simplified by the characteristics of the closed system we assumed. Note this is also true for higher order generation anti-messages. In particular, the timestamp distribution of an anti-message will not change from generation to generation. This is because it is only the processing of the "complete_service" message that can cause an incorrect activity to be placed into service, and therefore *any* anti-message will have a timestamp that is the sum of the timestamp of the "complete_service" message and an exponential random variable. As shown in Equation (4.6), this allows us to define a simple recurrence relation to determine the probability of an *Nth* generation anti-message.

When we move to an open system that is not necessarily heavily loaded, we lose the simplifying aspects of the closed system. We can no longer assume it is only the aggressive processing of the "complete_service" message that can cause an incorrect activity to be placed into service. As will be seen, there are a multiplicity of ways an incorrect activity can be placed into service. Further, for each set of circumstances under which an anti-message can be generated, there are different timestamp distributions that must be considered. Many of these timestamp distributions are much more complex than the ones dealt with in the closed system. Therefore this section is not only beneficial in that we expand the scope of our analysis, it also serves to show how quickly the level of complexity increases as we begin to move away from a system that allowed us to make many simplifying assumptions.

The bulk of this analysis has been placed into an appendix because the importance of the material does not justify the length and complexity of the analysis here. In this section we give an overview of the cases that must be considered to determine the probability of an anti-message. Also we give simulation results which support our model.

5.5. Overview of the Probability of Producing an Anti-Message

Recall an anti-message is produced when an LP discovers it has sent a message to another LP in error. Further recall the only time an LP sends a message is when it places an activity into service, and the message is sent to the LP which receives this activity upon its completion. As discussed in section 4.3.1, we assume such a message is sent incorrectly if the event that caused the activity to be placed into service is invalidated. This corresponds to the aggressive cancellation strategy in Time Warp (Reiher *et al.* 1990).

We divide our analysis into two sections. In the first section we define and describe the events necessary to create an anti-message. In the appendix we compute the probability of these events.

Recall our model of a closed, heavily loaded system assumed a server is always busy. Clearly, in an open system which can be *lightly* loaded we cannot make this assumption. Given that a server can become idle in a lightly loaded system, there are two basic cases we must consider in our analysis: either the server is busy when the LP begins processing aggressively or it is free when it begins to process aggressively. Note by "busy when the LP begins processing aggressively" we mean the server is busy, and the completion time of the activity currently receiving service is at least as great as the lower bound of the aggressive window. Sometimes we use the term "busy at the beginning of the aggressive window" to signify this same event. We first consider the events that must occur in order for a given LP to produce an anti-message given the server is busy when the LP begins to process aggressively.

We define *busy* as the event that the server is busy when the LP begins processing in the aggressive window. Given that the server is busy, the activity currently receiving service must complete before a new activity can begin service (recall our assumption of non-preemptive service). Thus it is only the processing of the "complete_service" message which will cause a new activity to be placed into service, and consequently cause a "pre-sent" completion message. As noted, the "complete_service" message must be processed aggressively in order to place an incorrect activity into service, and thus the timestamp of the "complete_service" message must fall within the aggressive window. Let C represent the event that the "complete_service" message falls within the aggressive window, and let T_{cs} represent the timestamp of the "complete_service" message (note all of the timestamps we discuss are assumed to be within the ag-

gressive window unless we explicitly state otherwise).

Given that the "complete_service" message is processed aggressively, there must be an activity to enter into service aggressively (and thus cause a "pre-sent" completion message to be sent aggressively) in order to produce an anti-message. As noted, in a heavily loaded system it is reasonable to assume there will always be an activity ready to enter into service upon the processing of a "complete_service" message. In a system that may be lightly loaded however we cannot make this same assumption.

There are two ways an activity can enter into service given that the "complete_service" message falls within the aggressive window. First, there can be an activity on the server queue (i.e. an activity waiting to receive service). In this case we assume the activity will enter into service immediately upon the processing of the "complete_service" message. If there is no activity on the server queue, there may be an activity on the LP's event list with timestamp $X > T_{CS}$. In this case, the activity will enter into service when the LP has simulated up to logical time X. Thus the server will become idle between logical time T_{CS} and logical time X, and again become busy when it enters this next activity into service.

Recall that in order to produce an anti-message the next activity to enter into service must complete within the aggressive window. As can be seen, it is important to consider whether the activity enters into service at logical time T_{CS} or at logical time X in order to determine the probability that it completes within the aggressive window.

We define the events that must occur in order to produce an anti-message given that the server is busy at the beginning of the aggressive window. Let Act_1 be the event there is an activity to place into service immediately upon the completion of the activity currently receiving service (and thus entering into service at logical time T_{CS}). Let Act_2 be the event that there is no activity waiting to enter into service, but there is an activity with timestamp $X > T_{cs}$ within the aggressive window (that enters into service at logical time X). We define $2nd$ as the event that an activity placed into service at logical time T_{CS} completes within the aggressive window. We define C_2nd as the event an activity which enters into service at logical time $X > T_{CS}$ completes within the aggressive window.

The events defined thus far are necessary, but not sufficient to cause an anti-message. Also, the activity must remain in the system upon its completion with probability p_1. Given the activity completes within the aggressive window and stays in the system upon its completion, the activity chosen for service must be invalidated in order to produce an anti-message. Thus if the activity enters into service at logical time T_{CS} the LP must receive an arrival message with a timestamp less than T_{CS}. Similarly, if the activity enters into service at logical time $X > T_{CS}$, the LP must receive an arrival message with a timestamp less than X.

As in previous chapters, we let $Arr{=}k$ be the event that an LP receives k arrival messages. We define *inval_1* as the event that an arrival message invalidates an activity which enters into service at logical time T_{cs}, and we define *inval_2* as the event that an arrival message invalidates an activity which begins service at logical time $X > T_{cs}$.

This defines all of the events we must consider in order to determine the probability that an anti-message is produced given that the server is busy as the LP begins to process within the aggressive window. This probability is

$$P(Anti \mid busy) = P(C)\,P(Act\,1)\,P(2nd)\,P(Arr{=}1)\,P(inval\_1)\,p_1 + \qquad (5.20)$$
$$P(C)\,P(Act\,2)\,P(C\_2nd)\,P(Arr{=}1)\,P(inval\_2)\,p_1.$$

Note this does not account for the receipt of more than one arrival message. We consider that possibility below. Further, this probability is conditioned on the event that the server is busy at the beginning of the aggressive window.

Now we compute the probability of producing an anti-message given that the server is free as the LP begins to process aggressively. We define *s_free* as the event that the server is idle as the LP begins to process in the aggressive window. If the server is idle at logical time L (the lower bound of the aggressive window), the next activity to enter into service will be the one with the minimum timestamp among the M activities in the aggressive window, assuming such an activity exists. Call this activity A_1, and define *ActW* as the event that such an activity exists. We denote the timestamp of activity A_1 as ϕ.

As discussed, if activity A_1 exists, it must complete within the aggressive window in order to generate an anti-message (with which we are concerned). We define *S_2nd* as the event that activity A_1

completes service within the aggressive window. Also, the LP must receive an event with a timestamp less than ϕ in order to produce an anti-message. We define *inval_3* as the event that activity A_1 is invalidated by the receipt of an arrival message. Finally, the activity must remain in the system upon its completion in order to produce an anti-message (with probability p_1). Let *Anti* be the event that an LP produces a first generation anti-message. As can be seen, the probability of evnt *Anti* given that the server is free at logical time L, and given that activity A_1 exists, is

$$P\ (Anti \mid s\_free \text{ and } ActW) = P\,(S\_2nd)\,P\,(inval\_3)\,P\,(Arr{=}1)\,p_1. \tag{5.21}$$

We consider the probability of more than one arrival message below.

Finally, there is also the probability that an LP will produce an anti-message given that the server is idle at logical time L, and there is no activity within the aggressive window to enter into service (i.e. activity A_1 does not exist). This occurs when an LP receives an arrival message, places this activity into service aggressively, the activity completes within the aggressive window, stays in the system upon its completion, and the activity is invalidated by the receipt of a second (or third) arrival message. As discussed in the appendix, if an arrival message is placed into service aggressively, it will enter into service at logical time T_{CS} (the same logical time as the "complete_service" message). Also, we show the probability this activity completes within the aggressive window is $P\,(2nd)$. Noting the probability one arrival message invalidates another arrival message is 0.5 (since they have, by assumption, the same timestamp distribution), the probability of producing an anti-message given that the server is idle at logical time L, and activity A_1 does not exist is

$$P\,(Anti \mid s\_free \text{ and } \overline{ActW}) = P\,(Arr{\geq}2)\,P\,(2nd)\,p_1\ 0.5. \tag{5.22}$$

We consider the case of more than two arrival messages below.

We now define the events associated with invalidating an activity given more than one arrival message. As discussed, there are three logical times at which an activity can enter into service within the aggressive window. If the server is busy as the LP begins to process aggressively, the next activity to enter into service will begin service at either logical time T_{CS} (the time of the "complete_service" message), or at logical time X such that $X > T_{CS}$. If the server is free as the LP begins to process aggressively, then the next activity will enter into service at logical time ϕ, where ϕ is the minimum of the N activities within

the aggressive window, or at logical time T_{CS} (if an arrival message is placed into service aggressively).

We define *invalidateA* as the event an activity which enters into service at logical time T_{CS} is invalidated by the receipt of an arrival message. We define *invalidateB* as the event that an activity which enters into service at logical time $X > T_{CS}$ is invalidated by the receipt of an arrival message. We define *invalidateC* as the event that an activity which enters into service at logical time ϕ is invalidated by an arrival message. Finally, we define *invalidateD* as the event that an arrival message placed into service aggressively is invalidated by the receipt of another arrival message. The probability that a given LP produces a first generation anti-message is

$$\begin{aligned} P(Anti) = {} & \rho P(C)\, P(Act\,1)\, P(2nd)\, P(invalidateA)\, p_1 \\ & + \rho\, P(C)\, P(Act\,2)\, P(C\_2nd)\, P(invalidateB)\, p_1 \\ & + (1-\rho)\, P(Actw)\, P(S\_2nd)\, P(invalidateC)\, p_1 \\ & + (1-\rho)\,(1 - P(Actw))\, P(2nd)\, P(invalidateD)\, p_1 \end{aligned} \tag{5.24}$$

As can be seen, there are many cases that must be considered in order to compute the probability that an LP produces an anti-message in an open, lightly loaded system. In the appendix we derive the probability of each of these events. Here we give simulation results which support our analysis.

5.6. Simulation Results

In order to test our equations we simulated an open queueing system where an LP receives messages from both internal sources, and from an external Poisson arrival process with rate λ_E. As discussed, we generated all of our external arrival messages at the beginning of the simulation. Each data point represents the average taken over 32 trials, where each trial consisted of 1000 iterations of the algorithm.

Recall in Chapter 3 we simulated a system (1500 *LPs*) which was heavily loaded. In this experiment we simulated a system (250 *LPs*) that was lightly loaded. The particular system we simulated had an external arrival rate of $\lambda_E = 0.5$, and an exponential service time distribution with mean $1/\lambda = 1$. The probability a message stayed in the system was $p_1 = 0.4$, and the probability a message exited the system was $p_0 = 0.6$. Thus the server utilization of our system was $\rho = 0.833$. Note this system was more heavi-

ly loaded than the system we simulated to investigate the probability of a fault (Figure 3.8). This is because there was almost no probability of an anti-message in the more lightly loaded system.

In Figure 5.2 we show the predicted and observed probability of a first generation anti-message given an aggressive window size from $A = 10\%$ of the mean of the service time distribution to $A = 100\%$ of the mean of the service time distribution. As can be seen, our predictions are quite close for the smaller aggressive window sizes. As the size of the aggressive window increases however our equations begin to overpredict the probability of an anti-message. This was exactly the pattern we observed in our predictions for the closed system, and our predictions of the probability of a fault in an open system. As discussed in detail in section 3.2.4.2, the reason for this overprediction is the error of our timestamp approximation for the arrival messages. As we discussed, the larger the size of the aggressive window, the larger the error of this approximation. It is important to note however our predictions bound from above the probability of a first generation anti-message. For this reason our predictions are quite useful even if they are not exact at the larger aggressive window sizes. As can be seen, given a lightly loaded system the probability of an anti-message at a given LP is quite small.

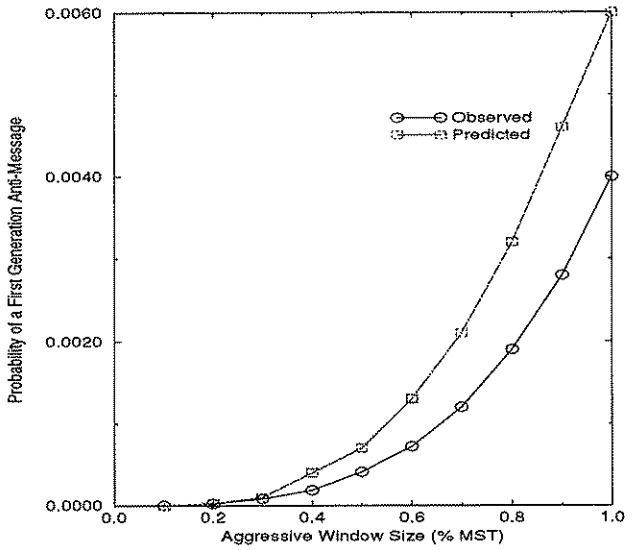

Figure 5.2 - Probability of an Anti-Message in an Open System

As we discuss in the appendix, we are forced to develop an approximation for the timestamp distribution of an *Nth* generation anti-message. Even given this approximation, the computation of the probability of producing an *Nth* generation anti-message is quite time consuming. We computed the probability of a second generation anti-message given an aggressive window size of 50% of the mean of the service time distribution, and 100% of the mean of the service time distribution. Our prediction for the probability of a second generation anti-message given an aggressive window size of 50% of the mean of the service time distribution was 0.000007, and we observed a probability of 0.000005. For an aggressive window size equal to 100% of the mean of the service time distribution, we predicted 0.00019 and observed 0.000124.

As can be seen, our predictions are quite accurate. We conjecture that most of the error in our predictions is caused by the over estimation of the probability of a first generation anti-message, which we use in the equation to predict the probability of a second generation anti-message. When we use the observed probability of a first generation anti-message in our equations, rather than the predicted, we get the following results. For $A = 50\%$ of the mean of the service time distribution we predict 0.0000046 compared with an observed probability of 0.000005. For $A = 100\%$ of the mean of the service time distribution, we predict the probability of a second generation anti-message is 0.000127. This is compared with the observed probability of 0.000124. As can be seen these probabilities are quite small, and the predictive power of our model is quite good.

5.7. Discussion

This chapter is important for three reasons. First, it allows us to extend our analysis to an open system that can be lightly loaded. Thus we are able to relax some our assumptions from previous chapters. Second, it is important in that it demonstrates how quickly the analysis becomes very complex as we move away from a system that allowed us to make many simplifying assumptions. Third, the analysis in this chapter showed the probability of a fault and the probability of an anti-message *at a given LP* in a lightly loaded system is quite small. It seems clear from our predicted and observed results that the more lightly loaded the system, the lower the probability of a fault, and the lower the probability of producing an anti-message.

In Chapters 3 and 4 where we discussed a closed system, and in our current discussion of an open system, we have computed the probability of a fault and an anti-message *at a given LP*. This is certainly useful information, but does not give a complete description of the behavior of a system such as ours. This is because we need to examine the behavior of the *system*, rather than the behavior of a given LP. We address this very important issue in the next chapter.

CHAPTER 6

System Level Performance

In previous chapters we have demonstrated both theoretically and with simulation studies that the aggressive global windowing algorithm offers the potential for significant performance gains over the non-aggressive windowing protocol. We have shown the probability of a causality error occurring *at a given LP* is small. When causality errors do occur and the correction mechanism must correct the out of order processing, we have shown that the probability of a second or higher order generation anti-message is quite small *at a given LP*. Further, we have shown that *on average* our aggressive windowing algorithm processes significantly more messages (that are not later invalidated) between global synchronization points than does the non-aggressive protocol.

While these results are encouraging they are incomplete because they consider only the behavior of a "typical" LP. In a windowing protocol (whether aggressive or non-aggressive), each phase of the algorithm is separated by a barrier synchronization. Thus *every* LP in the system remains blocked until the *slowest* LP completes its processing. Thus it is the *worst case behavior*, rather than average case behavior, that dominates system performance.

In our discussion of the error correction mechanism we showed that the probability of a *given* LP receiving a second generation anti-message is quite low. However, in a system with a large number of LPs the probability that *some* LP in the system receives a second generation anti-message is quite high. Thus the whole system will (at a minimum) have to block until this second generation anti-message is processed. In order to make statements about the performance of the system we must consider the high probability that *some* LP will receive a second generation anti-message rather than the low probability that a given LP receives such a message.

In this chapter we extend our model in order to investigate system-level performance. Also, we extend our model to include the costs of aggressive processing such as state saving and the necessity to reprocess messages due to rollbacks. Given these extensions we are able to predict the costs (in terms of

the number of messages that must be processed) of processing one unit of logical time for both approaches. Finally, we give simulation results which support our model.

The rest of this chapter is organized as follows. In section 6.1 we discuss the assumptions of our model and describe our technique for capturing the cost of each approach. In section 6.2 we model the costs of the non-aggressive approach. In section 6.3 we model the costs of the aggressive approach. In section 6.4, we give our theoretical and simulation results. We give our conclusions in section 6.5.

6.1. Model

In the non-aggressive version of the algorithm the time required to complete the processing within the lookahead window is a function of two costs: the workload within the window and the costs of global synchronization. In the aggressive algorithm we must also consider the costs of state saving and reprocessing messages due to rollbacks. In either approach the time required to complete processing within the simulation window is dominated by the slowest LP in the system. As discussed above, it is for this reason that it is inadequate to consider only average case behavior. We begin by discussing the assumptions of our model and our technique for modelling the costs of each approach.

We briefly review the assumptions given in section 3.2, all of which we continue to assume. We assume there are N LPs in the system with one server per LP. We assume the duration of an activity is drawn from independent, identically distributed exponential random variables with mean $1/\lambda$. We assume that when an activity completes it causes exactly one other activity at some LP in the system. The next LP to receive this activity upon its completion is chosen at random where each LP is equally likely to be picked.

Also, we make assumptions regarding the processing order of events for the aggressive algorithm. We assume that all of the events within the aggressive window are processed before an LP receives its first arrival message. This is a reasonable assumption given the relatively small aggressive window sizes we consider. Also, as discussed in section 3.2.1, this represents a worst case assumption. Similarly, we assume an LP completes all of the processing required by the first arrival message before the second arrival message is received and so forth. The arguments given in section 3.2.1 can be extended to show that

this is also the worst case assumption.

We assume all arrival messages are first generation arrival messages. As discuss in section 3.2.4.2, this implies that arrival messages have the same timestamp distribution as the aggressive messages (recall that an aggressive message is an event with a timestamp that falls within the aggressive window, and the event is in the aggressive window at the time the LPs synchronize to define a new simulation window). As discussed in section 3.2.4.2, this represents a worst case assumption.

We now describe our technique for modelling the costs of each approach. We define the *processing cost* as the expected cost of processing one unit of logical time. Note this cost includes the messages that must be processed (i.e. the real work of the simulation), as well as the over-head costs associated with the particular approach. In the case of the non-aggressive approach one such over-head is the cost of gloabl synchronization. In the aggressive approach one such over-head cost is saving state. In our analysis, we model the over-head costs of each approach relative to the cost of processing a single message.

For example, consider the processing cost for the non-aggressive approach. Assume a given LP has four messages to process in a particular unit of logical time. Further, assume the LPs synchronized three times during the course of processing through the unit of logical time. If the cost of gloabl synchronization was equal to the cost of processing a message, then the processing cost would be seven.

We assume each message takes approximately the same amount of real time to process. Our model does not consider the cost of message passing and thus assumes this cost is zero. Note that the message passing costs of the aggressive version will be somewhat higher than the non-aggressive version, but we do not think this is significant. Recall that we have shown the probability of sending a message in error is quite small (i.e. the probability of an anti-message is quite small). Thus most of the messages sent in the aggressive version of the algorithm will also be sent in the non-aggressive version of the algorithm. Also, while we do consider the cost of state saving and reprocessing messages due to rollback, we assume the cost of rolling back is zero. This is reasonable given that the cost of aggressive processing is dominated by state saving and reprocessing messages.

The equations, examples and empirical results developed in this chapter assume a closed system that is heavily loaded. This is because the equations for a closed system are easier to work with than those of an open system. However, these same equations could have been derived for an open system as well.

There are other assumptions we need to make as the chapter progresses, but we have not yet developed the context for these assumptions. We begin our discussion with the costs of processing one unit of logical time for the non-aggressive windowing protocol.

6.2. Costs of Non-Aggressive Processing

In the non-aggressive algorithm the LPs concurrently process their events with timestamps falling within the lookahead window. When an LP completes this processing it enters into a barrier synchronization waiting for the other LPs to similarly complete. Consider the cost of processing within the lookahead window. Given our assumption that each message takes approximately the same amount of real time to compute, processing within the lookahead window will be dominated by the LP with the most messages to process. Thus a reasonable approach to modeling this cost would be to determine the number of messages we expect the maximally loaded LP to process. We do not use this approach however since we seek to compare the two approaches given the most optimistic set of assumptions regarding the performance of the non-aggressive algorithm.

The best case (non-trivial) assumption is that the maximally loaded LP has one message to process. We add the cost of the global synchronization (performed at the upper bound of the lookahead window) to the cost of processing one message. For the moment assume the system uses a traditional barrier synchronization such as the **gsync()** routine provided on an Intel iPSC2 hypercube. We discuss other approaches below.

The cost of a barrier synchronization is $O(Log_2 P)$ given a system with P processors. Let C_{MP} represent the cost of processing one message and C_{LA} represent the total cost of processing within the lookahead window. Then

$$C_{LA} = C_{MP} + c\ Log_2 P. \quad (6.1)$$

The c term in Equation (6.1) is a factor used to express the cost of a global synchronization relative to the cost of processing a single message.

6.3. Costs of Aggressive Processing

The costs of aggressive processing are more difficult to capture than the costs of non-aggressive processing. This is because it is necessary to consider the costs of state saving, reprocessing messages and the effects of anti-messages.

The total time required to complete processing within the aggressive window is the sum of two components. First is the cost of processing the events within the aggressive window. This cost will be incurred by either approach (although the non-aggressive algorithm would define one or more lookahead windows to process all such messages), and represents the work that must be completed within the window. The second component is the cost incurred due to aggressive processing. This represents the extra work required by the correction mechanism that would not be incurred if the events were processed non-aggressively. This cost includes the cost of state saving and reprocessing events due to rollbacks and anti-messages.

Before developing our model to capture these costs we briefly review the aspects of the correction mechanism that are critical to our analysis. Recall we discussed the correction mechanism in detail in Chapter 4.

Our aggressive algorithm consists of two phases. In the first phase the LPs synchronize to determine the new lookahead and aggressive windows. In the second phase of the algorithm the LPs concurrently process their events with timestamps falling within the newly defined simulation window. Any new events generated as a result of this processing are immediately sent to the receiving LP.

We assume state is saved after each event that is processed aggressively. Thus state information must be saved as the LP processes within the aggressive window. However, an LP does not need to save state as it processes within the lookahead window since this processing is guaranteed to be correct.

A causality error occurs when an LP receives an arrival message with a timestamp less than that of some event(s) processed aggressively. When this occurs we assume that any event with a timestamp greater than that of the offending arrival message is in error. It is not necessarily the case that event dependencies have been violated, but for the purposes of this analysis we assume it always is. To correct a causality error two steps are required. First, the LP rolls back to the logical time immediately prior to the timestamp of the offending arrival message. Given our assumption that all processing with a logical time greater than that of the offending arrival message is in error, any events generated within this time interval are considered invalid. Thus part of the rollback procedure is to cancel any such events sent to other LPs. Recall we cancel events (assumed) to be sent in error through the use of an anti-message (see Chapters 4 and 5 for a discussion of anti-messages). This corresponds to the aggressive cancellation policy in Time Warp (Reiher *et al.* 1990).

The second phase required to correct a causality error is to reprocess all of the events with timestamps greater than that of the offending arrival message. Since we limit the amount of aggressive processing, the amount of reprocessing required is limited by the upper bound of the aggressive window.

Also it is possible that the receipt of an anti-message will lead to a causality error. We assume when an arrival message is cancelled through the receipt of an anti-message, any events processed with timestamps greater than that of the cancelled arrival message are invalid. Again note this is not necessarily the case, but for the purposes of this analysis we pessimistically assume it always is.

As discussed, both the cost of saving state and the cost of reprocessing messages must be be considered in our model. Clearly the cost of these two items is a function of the number of events processed aggressively and the number of arrival messages received. In previous chapters we have computed the expected number of messages processed aggressively and the expected number of arrival messages received by a "typical" LP in the system. As discussed however it is not the behavior of the "typical" LP that dominates system level performance. Rather it is the behavior of the slowest LP in the system. We now give an overview of our approach to capture the behavior of the slowest LP in the system.

6.3.1. Overview of System Level Performance

In previous chapters we developed a model to give precise estimates for the expected number of messages processed aggressively, the expected number of arrival messages, the probability of a first generation anti-message and so forth. We could to do this because we were able to derive the necessary probability distributions, and then apply mathematical principles to derive the desired equations. Unfortunately there is no precise mathematical formulation that defines the behavior of the LP which will dominate system performance. Similarly, there is no precise definition of the "worst case" behavior of a system such as ours.

Given that there is no precise formulation of the "worst case" behavior of our system, we develop our model such that it is reasonable to believe that we have captured the most important activities in the system which will dominate system performance. This is what we seek to accomplish. In order to do this we make a set of very pessimistic assumptions regarding the behavior of our system. We do this so that (given a system that performs under the assumptions of our model) the system will perform better than our model suggests. At the end of the chapter we give empirical results which show that our model does indeed capture a very pessimistic view of system performance.

As discussed, it is the slowest LP in the system which dominates system performance. Our model assumes that the LP with the most processing to perform is the slowest LP in the system. Therefore we seek to determine the maximum number of messages processed by any LP in the system. Our approach to this problem is as follows.

We define a special LP which we use to track the occurrences in the system which will dominate system performance. That is, we pick one LP out of the system, and define the workload of this chosen LP such that we expect its workload to be greater than that of any other LP in the system. We term this chosen LP the *dominant LP* of the system. As discussed, we make a set of very pessimistic assumptions regarding the amount of processing required by the dominant LP. Thus we use these pessimistic assumptions to compute the workload of the dominant LP, and then use this computed workload in our calculation of the processing cost of the aggressive approach.

6.3.2. Workload of the Dominant LP

In this section we discuss our basic approach to capturing the workload of the dominant LP. In subsequent sections we derive the equations needed to compute this value.

In order to derive the workload of the dominant LP we consider the significant occurrences in the system which will affect its performance. The receipt of arrival messages is one important occurrence. Recall that arrival messages may cause a rollback, and that a rollback may require the reprocessing of events. Another significant event is the receipt of anti-messages which may require a rollback, and may result in the generation of other anti-messages. The number of messages processed in the lookahead window and the number of messages processed in the aggressive window must also be considered. Finally, the cost of saving state must be considered. Given our assumption that state is saved after every message processed aggressively, it is the LP which processes the most aggressive messages that will incur the most state saving costs. Thus we expect the dominant LP will have the highest state saving cost in the system.

First, we assume the dominant LP has the maximum number of events in the lookahead window taken over all of the LPs in the system. The first step in our analysis therefore is to compute the maximum number of events in the lookahead window taken over all of the LPs in the system.

Next, we consider the effects of arrival messages. As discussed, each arrival message has the potential to invalidate events within the aggressive window as well as other arrival messages. Additionally, each time a message is reprocessed the state of the LP must be saved again. Thus the receipt of arrival messages can cause a great deal of processing. In our model we assume the dominant LP receives the most arrival messages of all of the LPs in the system. Our second task therefore is to derive the expected number of arrival messages received by the dominant LP. Additionally, we must determine the amount of reprocessing and state saving required as a result of each arrival message.

The other major factor we must consider is the effect of anti-messages. Certainly one of the primary costs associated with any aggressive protocol (that uses state saving and rollback) is the effect of anti-messages propagating through the system. Before we discuss how we account for the effect of anti-messages in our model we briefly describe their impact upon system performance.

As has been shown, the probability of cascading rollbacks developing in our system is quite small due to limited aggressive processing. Nevertheless, the propagation of anti-messages can significantly impact system performance even given the limited aggressive processing we allow.

We define a *rollback chain* as a sequence of anti-messages where an $Nth-1$ generation anti-message causes an Nth generation anti-message to be produced. The *depth* of a rollback chain is the maximum generation anti-message produced in the chain. The processing cost to a *given* LP that receives one of the anti-messages in a rollback chain may not be large, but the cost to the *system* is cumulative. That is, the system must remain blocked until *every* anti-message in a given rollback chain is processed. We account for the cumulative effects of a rollback chain in the following way.

We define the *maximum rollback chain* as the rollback chain which produces the highest order generation anti-message observed in the system in a given simulation window. We develop our model to compute the probability that this maximum rollback chain reaches depth $D=d$. In order to account for the cumulative effects of the maximum rollback chain, we make the pessimistic assumption that the dominant LP receives an anti-message for *each generation anti-message* in the maximum rollback chain. Clearly the more aggressive processing an LP has performed, the more processing we would expect to be invalidated by the receipt of an anti-message. Recall the dominant LP receives the maximum number of arrival messages of any LP in the system. Thus it is reasonable to believe an anti-message received by the dominant LP will require more reprocessing than an anti-message received by some other LP in the system. For this reason we expect our assumption that the dominant LP receives *each* anti-message in the maximum rollback chain will over-estimate the damage caused by such a chain developing in our system.

To summarize our approach, we pick one LP in the system and define it as the dominant LP. We assume the dominant LP begins by processing the most events within the lookahead window. Then we assume it receives the most arrival messages of any LP in the system. We assume these messages are received one at a time, and that all of the reprocessing required by a given arrival message is completed by the time the next arrival message is received. Then we calculate the costs involved in processing each arrival message (including state saving and reprocessing messages). After the dominant LP has received all of its arrival messages, we assume it begins receiving anti-messages one at a time. We assume the

dominant LP receives an anti-message for each generation anti-message in the maximum rollback chain. Again we make the worst case assumption that all of the reprocessing required by the receipt of the *Nth* generation anti-message is completed before the dominant LP receives the $Nth + 1$ generation anti-message. Then we calculate the total amount of processing required to process all of the anti-messages. The total cost of processing for the dominant LP is the sum of all of this processing. Then we use the total cost of processing obtained for the dominant LP in our calculation of the processing cost of the aggressive version of the algorithm.

Now that we have outlined our basic approach we compute the probabilities required to model the costs of aggressive processing. We begin with the events with timestamps that fall within the lookahead window.

6.3.3. Events Within the lookahead Window

We assume the dominant LP has the most events within the lookahead window of any LP in the system. In this section we derive the expected number of such events for the dominant LP.

Let M_{LA} be the number of events with timestamps that fall within the lookahead window of the dominant LP. We compute the expected value of M_{LA} in the following way. Recall in Equation (3.15) we derived the probability distribution for the number of events within the lookahead window for a given LP. We seek the distribution for the maximum number of events within the lookahead window taken over the N LPs in the system. Let $X = x_i$ be the random variable denoting the number of events in the lookahead window of LP_i, and let $Y = y$ be the random variable denoting the maximum number of events in the lookahead window taken over *all* N LPs in the system.

$$Y = \max \{x_1, x_2, x_3 \cdots x_N\}$$

In order for the maximum value to be less than or equal to y, each of the x_i must be less than or equal to y. Assuming independence between the N LPs, the probability that all N x_i are less than or equal to y is

$$P(Y \leq y) = P(x_i \leq y)^N.$$

This equation gives the probability that $(Y \leq y)$ and is thus the cumulative distribution function (CDF) of

Y.

$$F(Y) = P(Y \leq y) = P(x_i \leq y)^N.$$

Recall that the probability distribution for the number of events in the lookahead window of a given LP is a discrete distribution. Then by the additive property of probability distributions

$$P(Y = y) = P(Y \leq y) - P(Y \leq y-1) = F(y) - F(y-1).$$

Thus the probability that $Y = y$ is

$$P(Y = y) = P(x_i \leq y)^N - P(x_i \leq y-1)^N.$$

As noted above, we derive the probability that $X = x_i$ in Equation (3.15). The probability that $x_i \leq y$ is

$$P(x_i \leq y) = \sum_{j=0}^{y} P(x_i = j).$$

Let M_{LA} be the number of events within the lookahead window of the dominant LP. The expected value of the maximum number of events in any lookahead window, and thus the expected value of the number of events in the lookahead window of the dominant LP is

$$E[M_{LA}] = E[Y] = \sum_{y=1}^{\infty} y \left(\sum_{j=0}^{y} P(x_i = j) \right)^N - \left(\sum_{k=0}^{y-1} P(x_i = k) \right)^N. \tag{6.2}$$

6.3.4. Arrival Messages

We seek the maximum number of arrival messages received by any LP in the system. We derive this maximum value in the same manner as the derivation for the maximum number of events within the lookahead window given above. Recall we derived the probability that a given LP receives K arrival messages in Equation 3.18. We seek the maximum number of arrival messages received over all of the N LPs in the system. Let $X = x_i$ be the random variable denoting the number of arrival messages received by LP_i, and let Y be the maximum number of arrival messages received by any LP in the system.

$$Y = \max \{x_1, x_2, x_3 \cdots x_N\}$$

In order for $Y \leq y$, all N x_i must be less than or equal to y. As we did for Equation (6.2), we define the cumulative distribution function for the maximum number of arrival messages, and use the CDF to determine the probability that the maximum number of arrival messages is $Y = y$. The probability that $Y \leq y$ is the probability that all N x_i are less than y. Assuming independence among the LPs, this probability is

$$F(Y) = P(Y \leq y) = P(x_i \leq y)^N.$$

This equation gives the probability that $Y \leq y$ and is thus the CDF of the maximum number of arrival messages received by any LP in the system. Let M_{Ar} be the number of arrival messages received by the dominant LP. Following the same approach taken for Equation (6.2), we note the expected value for the maximum number of arrival messages received by any LP in the system is

$$E[M_{Ar}] = E[Y] = \sum_{y=1}^{\infty} y \left(\left(\sum_{j=0}^{y} P(x_i = j) \right)^N - \left(\sum_{k=0}^{y-1} P(x_i = k) \right)^N \right). \tag{6.3}$$

Again recall the probability that $x_i = k$ is given in Equation (3.18).

Our next task is to calculate the costs associated with processing the arrival messages. As discussed, one cost associated with processing an arrival message is the potential to invalidate some of the aggressive messages (aggressive messages are defined in section 3.2.1).. Recall if the first arrival message invalidates some of the aggressive messages, all of the reprocessing is completed by the time the dominant LP receives another arrival message. Thus this second arrival message may *again* invalidate some of the aggressive messages, *and* it may also invalidate the first arrival message. This is also true for each subsequent arrival message received by the dominant LP.

In order for the dominant LP to be able to perform a rollback we assume it saves its state after every event processed aggressively. As discussed, arrival messages may also be invalidated requiring that the state of the dominant LP be saved after every arrival message processed. We compute the total amount of processing performed by the dominant LP as a result of all of its arrival messages. Also we compute the total number of times the dominant LP must save its state.

We begin with the assumption that the dominant LP has the same probability distribution for the number of aggressive messages as do the other LPs in the system. Recall we derived this distribution in Equation 3.13. Let $P(M_{Ag} = m)$ represent the probability that the dominant LP has m aggressive messages. Then the expected value of M_{Ag} is

$$E[M_{Ag}] = \sum_{m=1}^{\infty} m \, P(M_{Ag} = m). \tag{6.4}$$

Next we determine the number of aggressive messages invalidated by the receipt of the first arrival message. These are the events that will have to be reprocessed. Recall our (worst case) assumption that

all arrival messages are first generation arrival messages (i.e. are conditional exponentials, conditioned on being within the aggressive window). Given this assumption both the aggressive messages and the arrival messages have the same timestamp distribution. Thus we expect that the first arrival message will invalidate half of the aggressive messages. Let NR_i represent the expected number of aggressive messages that must be reprocessed due to the receipt of the *ith* arrival message. Then the expected amount of reprocessing caused by the first arrival message is

$$NR_1 = \sum_{m=1}^{\infty} \frac{m}{2} P(M_{Ag} = m) = \frac{1}{2} E[M_{Ag}]. \tag{6.5}$$

After receiving its first arrival message the dominant LP must reprocess any events invalidated by this message. After this reprocessing is completed, the dominant LP receives its second arrival message. This second arrival message is then processed, and any reprocessing required by this message is performed. This continues until the dominant LP has received its last arrival message.

Note the expected number of aggressive messages invalidated by the receipt of an arrival message is the same for each arrival message, and this expected value is given in Equation (6.5). However, after the first arrival message is received we must also consider the probability one arrival message invalidates another arrival message already processed. For the moment we ignore the possibility of one arrival message invalidating another arrival message, and determine the expected number of aggressive messages that must be reprocessed.

As discussed, the expected amount of reprocessing caused by *each* arrival message is given in Equation (6.5). Our next task is to determine the amount of reprocessing caused by *all* of the arrival messages received by the dominant LP. To compute this, we calculate the expected amount of reprocessing given that the dominant LP receives K arrival messages, times the probability that the dominant LP receives K arrival messages, over all possible values of K. For example, if it is known that the dominant LP receives K arrival messages, then the expected amount of reprocessing caused by all K arrival messages would be

$$E[Reprocessing \mid M_{Ar}=k] = \sum_{i=1}^{k} \sum_{m=1}^{\infty} \frac{m}{2} P(M_{Ag} = m) = k \frac{E[M_{Ag}]}{2}.$$

We uncondition this expression by summing over all possible values of $M_{Ar} = k$ times the probability that $M_{Ar} = k$. Let NR_T be the random variable denoting the total amount of reprocessing (of the aggressive messages) caused by all of the arrival messages received by the dominant LP. Then

$$E[NR_T] = \sum_{k=1}^{\infty} k\, E[M_{Ag}]\, P(M_{Ar=k}) = \frac{E[M_{Ar}]\, E[M_{Ag}]}{2}. \tag{6.6}$$

Next we consider the expected amount of reprocessing caused by one arrival message invalidating other arrival messages. Note all the arrival messages have the same timestamp distribution. Thus if K arrival messages have been received and processed, then it is expected the next arrival message will have a timestamp less than half of the timestamps for arrival messages already processed. Let Ar_{RP} denote the expected number of arrival messages reprocessed by the dominant LP due to receipt of all of its arrival messages. If we know the dominant LP receives $M_{Ar} = k$ arrival messages, then the expected number of arrival messages reprocessed is

$$E[Ar_{RP} \mid M_{Ar} = k] = \sum_{i=1}^{k} \frac{i-1}{2} = \frac{1}{4} k^2 - k.$$

In order to uncondition this expression we sum over all possible values of $M_{Ar} = k$ times the probability of $M_{Ar} = k$. The expected value of Ar_{RP} is

$$E[Arr\_RP] = \frac{1}{4} \sum_{k=1}^{\infty} k^2 - k\, P(M_{Ar} = k) = \frac{E[k^2] - E[k]}{4}. \tag{6.7}$$

We have now derived all of the terms necessary to compute the expected amount of processing required by the dominant LP due to the receipt of arrival messages. This number is the sum of a) the expected number of the aggressive messages reprocessed due the receipt of arrival messages, b) the expected number of arrival messages received by the dominant LP and c) the expected number of arrival messages that must be reprocessed. Let Ar_T be the total processing required by the dominant LP due to the arrival messages. The expected value of Ar_T is

$$E[Ar_T] = E[M_{Ar}] + E[NR_T] + E[AR_{RP}]. \tag{6.8}$$

As discussed, we assume the dominant LP saves its state after every message it processes aggressively. Thus the total number of times it must save its state due to arrival messages is also given in Equation (6.8). Let SS_{Ar} be the number of times the dominant LP saves its state due to arrival messages. Then

the expected value of SS_{Ar} is

$$E[\,SS_{Ar}\,] = E[\,M_{Ar}\,] + E[\,NR_T\,] + E[\,AR_{RP}\,]. \tag{6.9}$$

Equation (6.8) gives the expected amount of processing required due to the receipt of arrival messages. Recall the dominant LP processes its aggressive messages (once) before the first arrival message is received. Similarly, the dominant LP saves its state after each aggressive message processed. Thus we must include these costs in our computation. Let AM_T be the total number of events processed before the dominant LP receives its first anti-message. Then the expected value of AM_T is

$$E[\,AM_T\,] = E[\,M_{LA}\,] + E[\,M_{Ar}\,] + E[\,NR_T\,] + E[\,AR_{RP}\,] + E[\,M_{Ag}\,]. \tag{6.10}$$

Similarly, let SS_{AMT} be the total number of times the dominant LP must save state before receiving the first anti-message. Since the dominant LP must save its state after every message processed aggressively (note it does not save state as it processes within the lookahead window) the expected value of SS_{AMT} is

$$E[\,SS_{AMT}\,] = E[\,M_{Ar}\,] + E[\,NR_T\,] + E[\,AR_{RP}] + E[\,M_{Ag}\,]. \tag{6.11}$$

6.3.5. Maximum Rollback Chain

The other set of events in the system that we expect to dominate system performance is the development of rollback chains. Recall that we model the cumulative effect of rollback chains on system performance by assuming the dominant LP receives *each* anti-message of the dominant rollback chain. Let D be the random variable representing the depth of the maximum rollback chain. In this section we derive the probability that the maximum rollback chain reaches depth $D = d$, and determine the amount of reprocessing required by each anti-message.

We have shown in section 5.3.2 the timestamp distribution of an anti-message (in a closed system that is heavily loaded) does not change from generation to generation. Thus it is not necessary to consider whether an anti-message is a first generation or higher order generation anti-message in our analysis. Therefore we model the costs of anti-messages to the system in the following way.

First, we compute the probability that the depth of the maximum rollback chain reaches $D = d$. Then we assume that if the depth of the maximum rollback chain is $D = d$ with probability P, the dominant LP receives $D = d$ anti-messages with probability P. Then we compute the expected amount of

reprocessing caused by each anti-message.

Recall we make the worst case assumption that all arrival messages are first generation arrival messages (i.e. conditional exponentials, conditioned on being within the aggressive window). This is the worst case assumption regarding the timestamp distribution of an arrival message. Given our assumption that all arrival messages are conditional exponentials, it follows that an anti-message sent to cancel one of the arrival messages will also have this same timestamp distribution. Since this represents the worst case assumption, and since this assumption significantly simplifies our analysis, we assume this to be the case.

We now determine the probability that the maximum rollback chain reaches depth $D = d$. In section 5.3.1 we showed the probability that a given LP produces a first generation anti-message is

$$\begin{aligned} P(Anti_1) = {} & P(C)\,P(2nd)\,P(Arr = 1)\,P(Inval) \\ & + P(C)\,P(2nd)\,P(Arr{=}2)\,(1 - (1 - P(Inval)^2) \\ & + P(C)\,P(2nd)\,P(Arr{=}3)\,(1 - (1 - P(Inval)^3)... \end{aligned} \tag{6.12}$$

In Equation (6.12), the $P(C)$ term is the probability that an LP processes the "complete_service" message aggressively. The $P(2nd)$ term is the probability that the next activity to enter into service completes within the aggressive window. The $P(Arr = k)$ term is the probability that the LP receives k arrival messages and the $P(Inval)$ term is the probability that an arrival message invalidates the "complete_service" message (this is given in Equation (3.27)).

Let $Anti_d$ be the event that a given LP produces an anti-message of generation d. In section 5.3.2 we showed that

$$P(Anti_d) = P(Anti_{d-1})\,P(C)\,P(2nd)\,P(Anti\_Inval). \tag{6.13)}$$

In Equation (6.13) the $P(Anti_{d-1})$ is the probability of receiving a $dth-1$ generation anti-message, and $P(Anti\_Inval)$ (derived in Equation 4.4) is the probability that this anti-message invalidates the "complete_service" message. The other terms are the same as in Equation (6.12).

We compute the depth of the maximum rollback chain in the following way. The probability that a given LP produces a dth generation anti-message is given in Equation (6.13). The probability that a given LP does not produce a dth generation anti-message is $(1 - P(Anti_d))$. In a system with N LPs, the probability that *no* LP in the system produces a dth generation anti-message is $(1 - P(Anti_d))^N$. The

probability that *some* LP in the system produces a *dth* generation anti-message, and thus the probability that the maximum rollback chain reaches depth $D = d$ is

$$P(D = d) = 1 - (1 - P(Anti_d))^N. \tag{6.14}$$

We calculate the expected amount of reprocessing required by the dominant LP due to the receipt of anti-messages in the following way. First, we make the assumption that each arrival message is equally likely to be invalidated by a given anti-message. Assume the dominant LP has received $M_{Ar} = k$ arrival messages. We can number the arrival messages from 1 .. k, where 1 is the arrival message with the smallest timestamp and k is the arrival message with the largest timestamp. Assume an anti-message is received which invalidates the arrival message with the minimum timestamp. In this case all of the other $k - 1$ arrival messages will have timestamps greater than that of the invalidated arrival message, and therefore will have to be reprocessed. Assuming each arrival message is equally likely to be invalidated, the probability of invalidating the arrival message with the minimum timestamp is $1/k$. Similarly, if the anti-message cancels the arrival message with the second smallest timestamp, then the other $k - 2$ arrival messages must be reprocessed. This also occurs with probability $1/k$. Let RP_Arr_1 be the number of arrival messages that must be reprocessed due to the receipt of the first anti-message. As can be seen, the expected value of RP_Arr_1 *given* k arrival messages is

$$E[RP\_Arr_1 \mid M_{Ar} = k] = \sum_{i=1}^{K} (K - i)\ \frac{1}{K} \ = \ \frac{1}{K}\sum_{i=1}^{k-1} i \ = \ \frac{k-1}{2}. \tag{6.15}$$

Before continuing with our analysis there is one issue that must be addressed. In our model we define the dominant LP, and define the workload of this LP such that we expect it to be higher than the workload of any other LP in the system. It is for this reason that we assume the dominant LP has the most events in the lookahead window, receives the most arrival messages and receives each anti-message in the maximum rollback chain. Note if we consider a "real" LP in the system (rather than the dominant LP which we define for the purposes of this analysis) the receipt of a given anti-message cancels a given arrival message. Therefore since each anti-message is sent to cancel a *particular* arrival message, an LP will never receive more anti-messages than the number of arrival messages it has received.

As discussed, our model is not considering a "real" LP in the system but rather we define the dominant LP to track the maximum amount of processing required in the system. If we decrement the arrival message count after each anti-message received, this count may become negative *unless* it can be shown that the maximum number of arrival messages received by the dominant LP is greater than the depth of the maximum rollback chain. This seems like a difficult proof, and one that is not really very useful except in this particular context. Given that our analysis is greatly simplified if we do not decrement the arrival message count, and given that the effect of this action is to over-estimate the amount of reprocessing required by the dominant LP, we proceed with our analysis assuming that the arrival message count is not decremented after the receipt of an anti-message.

As discussed, we make the worst case assumption that all of the reprocessing required by the receipt of the first anti-message is completed by the time the dominant LP receives the second arrival message. The probability that the depth of the maximum rollback chain reaches level $D = d$, and thus the probability that the dominant LP receives $D = d$ anti-messages, is given in Equation 6.14. Given that the message arrival count is not decremented after the receipt of an anti-message, the expected amount of reprocessing (given $M_{Ar} = k$ arrival messages) required by each of the anti-messages is given in Equation 6.15 (i.e., each subsequent anti-message causes the same amount of reprocessing as the first anti-message).

Let AR_{RPA} be the total number of arrival messages that must be reprocessed due to the receipt of the $D = d$ anti-messages. The expected value of AR_{RPA} given $M_{Ar} = k$ and given $D = d$ is

$$E[Ar_{RPA} \mid M_{Ar} = k \text{ and } D = d] \;=\; d\,\frac{k-1}{2}.$$

In order to uncondition this expression we sum over all possible values of $M_{Ar} = k$ and all possible values of $D = d$.

$$E[AR_{RPA}] = \sum_{d=1}^{\infty}\sum_{k=1}^{\infty} d\,\frac{k-1}{2}\,P(M_{Ar} = k)\,P(D = d) \;=\; \frac{(E[M_{Ar}] - 1)\,E[D]}{2}. \tag{6.16}$$

Now consider the aggressive messages (i.e. events within the aggressive window before the first arrival message was received) reprocessed due to the anti-messages. Recall we make the worst case assumption that the arrival messages, the anti-messages and the aggressive messages all have the same

timestamp distribution (conditional exponentials, conditioned on being within the aggressive window). Thus we expect that 50% of the aggressive messages are invalidated, and therefore must be reprocessed, due to the receipt of an anti-message. Let Ag_{RP} be the number of aggressive messages that must be reprocessed by a given anti-message. Then the expected value of Ag_{RP} is

$$E[Ag_{RP}] = \sum_{m=1}^{\infty} \frac{m}{2} P(M_{AG} = m) = \frac{E[M_{Ag}]}{2}. \tag{6.17}$$

In this equation the $m/2$ term is the expected number of aggressive messages invalidated given $M_{Ag} = m$ aggressive messages. We sum this over all possible values of $M_{Ag} = m$ times the probability of $M_{Ag} = m$. Equation (6.17) gives the expected number of aggressive messages invalidated (and thus reprocessed) due to the receipt of one anti-message. Let Ag_T be the total number of aggressive messages that must be reprocessed due to the receipt of *all* of the anti-messages received by the dominant LP. The expected value of Ag_T is

$$E[Ag_T] = \sum_{d=1}^{\infty} \sum_{m=1}^{\infty} \frac{m}{2} P(M_{Ag} = m) P(D = d) = \frac{E[M_{Ag}] E[D]}{2}. \tag{6.18}$$

In Equation (6.18) we sum the expected number of messages reprocessed per anti-message, over all possible depths $D = d$ of the maximum rollback chain, times the probability that the maximum rollback chain reaches depth $D = d$ (and thus the dominant LP receives $D = d$ anti-messages).

Now we have computed all of the terms necessary to determine the expected number of events that must be reprocessed by the dominant LP due to anti-messages. As discussed, these anti-messages may cause the dominant LP to reprocess both arrival messages and aggressive messages. Recall that $E[AR_{RPA}]$ is the expected number of arrival messages reprocessed due to anti-messages, and that $E[Ag_T]$ is the expected number of events within the aggressive window that must be reprocessed due to anti-messages. Let TRP_{anti} be the total number of messages that must be reprocessed due to the receipt of anti-messages. Then the expected value of TRP_{anti} is

$$E[TRP_{anti}] = E[AR_{RPA}] + E[Ag_T]. \tag{6.19}$$

Again note we assume state is saved after each message that is reprocessed due to the receipt of an anti-message. Let SS_Anti_T represent the total number of times the dominant LP must save state due to the receipt of anti-messages. The expected value of SS_Anti_T is thus

$$E[\,SS\_Anti_T\,] = E[\,AR_{RPA}\,] + E[\,Ag_T\,]. \tag{6.20}$$

Now we have computed the workload of the dominant LP except for the cost of global synchronization which we discuss below. To summarize, the total number of messages processed by the dominant LP is the sum of a) the messages in the lookahead window, b) the aggressive messages, c) the arrival messages, d) the aggressive messages and the arrival messages reprocessed due to the receipt of *arrival* messages and e) the aggressive messages and the arrival messages reprocessed due to the receipt of *anti-messages*. Recall that AM_T is the total number of messages processed by the dominant LP before it receives the first anti-messages. Also, TRP_{anti} is the number of messages that be reprocessed due to the receipt of anti-messages. If we let MP_T represent the total number of messages processed by the dominant LP, then the expected value of MP_T is

$$E[\,MP_T\,] = E[\,AM_T\,] + E[\,T_{RP}\,]. \tag{6.21}$$

The dominant LP must save its state after every message that it processes aggressively. Recall SS_{AM} is the total number of times the dominant LP saves state before it receives the first anti-message. Also, SS_Anti_T is the number of times the dominant LP saves state due to the receipt of anti-messages. Let SS_T be the total number of times the dominant LP saves its state. Then the expected value of SS_T is

$$E[\,SS_T\,] = E[\,SS_{AM}\,] + E[\,SS\_Anti_T\,] \tag{6.22}$$

Now we have captured the costs of processing through the aggressive window. The final cost of aggressive processing that must be considered is the cost of global synchronization. As discussed in section 4.1, the traditional barrier synchronization routine is not powerful enough for an aggressive windowing algorithm. In this analysis we model the costs of global synchronization assuming a software solution such as that provided by Nicol (1992a). We do this so we can discuss the protocol without assuming any particular hardware support.

Now we are ready to compare the processing cost of the two approaches. Recall we assume (section 6.2.2) the cost of global synchronization for the non-aggressive algorithm is $c\ Log_2\ P$ given P processors. As discussed, the c term is a factor used to express the cost of global synchronization in relative to the cost of processing a single message. As discussed in section 6.2.2, we assume the total cost of processing within the lookahead window is the cost of processing one message plus the cost of a global syn-

chronization. Thus the cost of processing one unit of logical time, and therefore the processing cost of the non-aggressive approach, is

$$CostNA = \frac{C_{MP} + c\ Log_2\ P}{L}. \tag{6.23}$$

The P term in Equation (6.23) is the number of processors in the system. Recall C_{MP} is the cost of processing one message. As noted, this represents the best case for the costs of non-aggressive processing.

The cost of processing for the aggressive approach is computed in a similar manner. Recall that the aggressive barrier synchronization required in the aggressive approach is approximately twice as slow as the non-aggressive barrier synchronization. Thus cost of global synchronization for the aggressive approach is $2\ c\ Log_2\ P$, i.e. twice the cost of the non-aggressive approach. The number of messages processed in the aggressive approach is the sum of a) the messages with timestamps that fall within the lookahead window, b) the aggressive messages, c) the arrival messages d) the aggressive messages reprocessed due to arrival messages, e) the arrival messages reprocessed due to arrival messages, f) the aggressive messages reprocessed due to anti-messages and g) the arrival messages reprocessed due to anti-messages. We must also consider the cost of saving state. The expected processing cost for the aggressive approach is thus

$$CostA = \frac{E[\ MP_T\ ] + SC\ E[\ SS_T\ ] + 2c\ Log_2\ P}{L + A}. \tag{6.24}$$

The SC term in Equation (6.24) is the cost of saving state relative to the cost of processing a single message. We are now ready to present our theoretical results.

6.4. Theoretical Results

In this section we use the equations developed in the previous sections to compare the processing cost of the two approaches. All of our results assume a system with $N = 1024$ LPs and a mean service time of $1/\lambda = 1$. Given these parameters, the expected width of the lookahead window is $E[\ L\ ] = .0386$. In order to compute the cost of global synchronization we must assume some hardware configuration. In order to be consistent with our analytic model we assume there is one LP per processor, and thus we assume $P = 1024$. Our predicted results would be slightly worse if we assumed a smaller configuration (due to the reduced cost of global synchronization of the non-aggressive approach). We derive results for

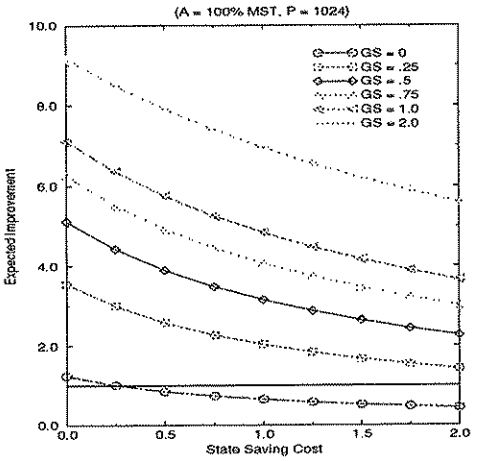

Figure 6.1 - Theoretical Improvement, A = 100% MST

aggressive window sizes of $A = 100\%$, $A = 50\%$ and $A = 10\%$ of the mean service time.

There are two independent variables in our model which must be considered: the cost of global synchronization relative to the cost of processing a single message, and the cost of saving state relative to the cost of processing a single message. One way to present the results would be to present a three dimensional plot which gives speed up as a function of these two variables. We did not choose this ap-

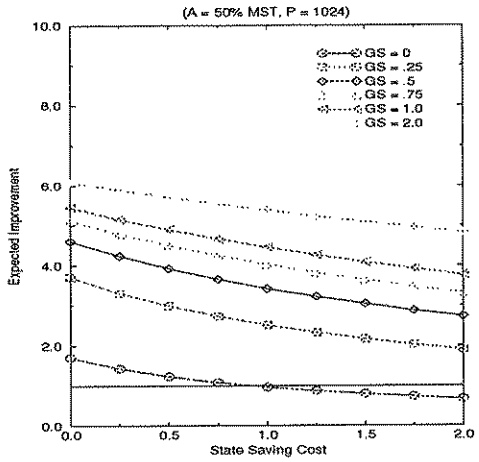

Figure 6.2 - Theoretical Improvement, A = 50% MST

proach however because we found the three dimensional plots to be somewhat difficult to read. For this reason we present a family of curves, where the state saving cost is varied on the x axis, and the corresponding expected improvement in the processing cost is shown on the y axis. We vary the state saving cost from zero to twice the cost of processing a single message. Each curve represents the improvement in processing cost given a particular value for the cost of global synchronization. We compute the results assuming the cost of global synchronization is 0, .25, .5, .75, 1.0 and 2.0 times the cost of processing a single message.

The formula we use to compute the expected improvement in processing cost is

$$EI = \frac{CostNA}{CostA}. \tag{6.25}$$

Thus ratios greater than one indicate the superiority of the aggressive approach.

In Figure 6.1 we plot the expected improvement given an aggressive window size of $A = 100\%$ of the mean service time. When the cost of global synchronization is zero, we predict the non-aggressive approach will offer better performance unless the cost of state saving is close to zero. This is not surprising given that the cost of global synchronization is the dominant cost for the non-aggressive approach in our model. Since our model assumes the best case performance of the non-aggressive approach (i.e. we

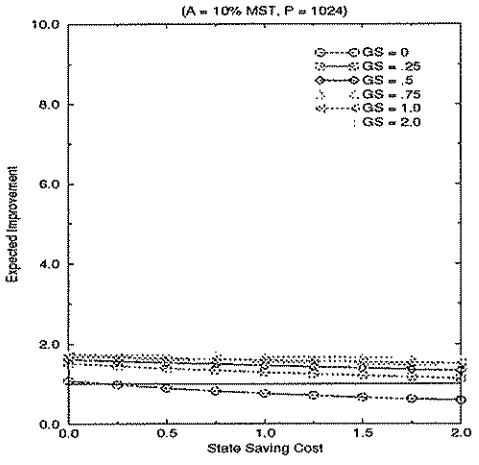

Figure 6.3 - Theoretical Improvement, A = 10% MST

assume there is no idleness in this approach), it is expected that our model would predict the non-aggressive approach to offer better performance if the cost of global synchronization is free. As we increase the cost of global synchronization our model predicts the aggressive approach will offer better performance. This is true even if the cost of saving state is twice the cost of of processing a message. We find these results very encouraging, particularly given our very pessimistic assumptions regarding the cost of aggressive processing.

In Figure 6.2 we plot the expected improvement given an aggressive window that is 50% of the mean service time. The maximum expected improvement given this smaller aggressive window size is less than the maximum expected improvement for an aggressive window size of 100% of the mean service time. The reason for this difference is the high over-head cost of the aggressive approach assumed in our model. Recall our assumption that the dominant LP has the *highest number* of messages to process within the lookahead window of any LP in the system. This is compared to our assumption that the cost of processing within the lookahead window for the non-aggressive approach is the cost of processing a single message. Further recall that the cost of global synchronization for the aggressive approach is twice that of the non-aggressive approach. Thus the smaller the size of the aggressive window, the less opportunity there is to amortize the high overhead costs assumed in our model.

In Figure 6.3 we show the expected improvement given an aggressive window size set to 10% of the mean service time. The high (assumed) cost of processing within the lookahead window, and the high synchronization cost, dominates the performance of the aggressive algorithm given an aggressive window of this size. Thus the high over-head cost cannot be amortized over the aggressive window, and our model predicts limited improvement in performance.

6.5. Simulation Results

In this section we give a set of simulation results in order to test the predictions of our analytic model. All of these results (except where noted) assume $P = 1024$ processors, $N = 1024$ LPs and a mean service time of $1/\lambda = 1$. All of the graphs (except for Figure 6.4) give the results for an aggressive window size set to 100% of the mean service time. The system we simulated is a simple FCFS queueing net-

work with one server per LP. Each data point represents the mean observed value taken over sixteen trials, where each trial consisted of one thousand iterations of the algorithm. Note that there was very little observed difference between the trials.

Many of the assumptions of our model were met in these simulations. Except where stated, the communication pattern was random as assumed in our model (i.e. each LP was equally likely to receive a given message). We implemented our state saving and rollback algorithm as described in Chapter 4, and thus errors which did occur were corrected through the rollback procedure. As discussed, one aspect of the rollback procedure is to cancel messages sent incorrectly through the use of anti-messages. Thus rollback chains could (and did) develop in our simulation.

Some of the assumptions of our model were *not* met in our simulations. First, not all arrival messages were first (or first and second) generation messages as assumed in our model. Second, recall our model only considers the effects of one "complete_service" message within the aggressive window. In the course of computing within the aggressive window however it is possible that more than one such message is processed. Finally, our model assumes the width of the lookahead window is a constant, and the expected value of this window is used in our calculations. As discussed however, the width of the loo-

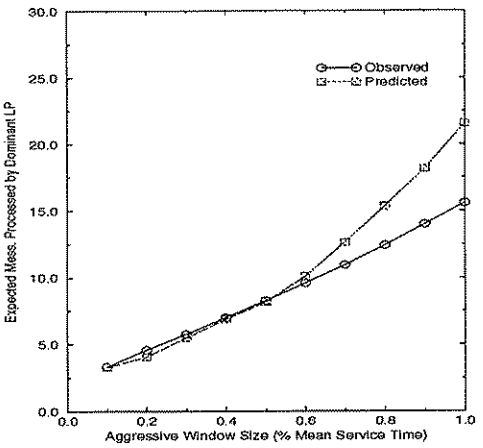

Figure 6.4 - Number of Messages Processed by Dominant LP

kahead window is in fact a random variable.

In Figure 6.4 we show the "raw" count of the number of messages processed by the dominant LP (i.e. the LP which processed the maximum number of messages in the system) compared with the costs of processing predicted by our model. It is important to stress that these observed values represent the *maximum* number of messages processed by *any* LP in the system. We say "raw" because Figure 6.4 does not account for state saving costs or the cost of global synchronization. Rather, it is just a count of the number of messages processed by the maximally loaded LP in the system. Note that for the smaller aggressive window sizes the predictions of the model are fairly close to the observed values, and in some cases very slightly under-predicts the observed values. The reason for this under-prediction is similar to the reasons given in section 4.3. Our model assumes a static picture of the aggressive window and thus does not consider events such as more than one "complete_service" message being produced at a given LP. For small values of A (the size of the aggressive window) our equations do not compensate for the events not considered in the model. As A increases however our equations begin to over-predict the amount of processing required by the dominant LP due to the very pessimistic assumptions of our model.

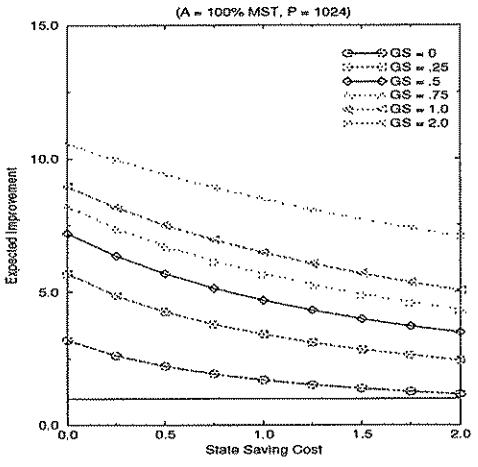

Figure 6.5 - Expected Improvement, Random Communication Pattern

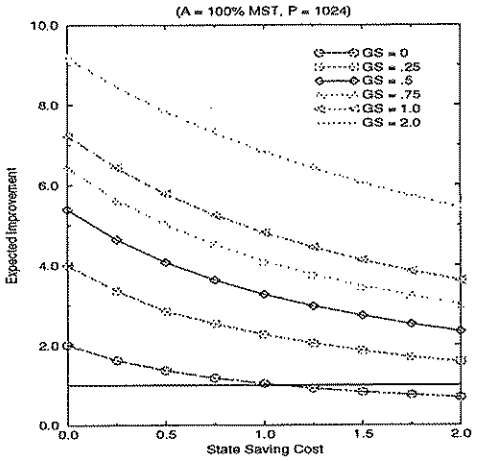

Figure 6.6 - Expected Improvement, Nearest Neighbor Communication Pattern

In Figure 6.5 we use this "raw" message count (for the dominant LP) and plot the expected improvement in performance given various state saving and global synchronization costs. Again we present a family of curves rather than a three dimensional plot. Each such curve represents the expected improvement given global synchronization costs of 0, .25, .5, .75, 1.0 and 2.0 times the cost of processing a single message. We vary the state saving cost between zero and two times the cost of processing a single message.

Figure 6.5 shows our aggressive algorithm can offer significant improvement in performance. Note some improvement is expected *even when the cost of global synchronization is free.* This is very encouraging. As the cost of global synchronization increases the expected improvement due to aggressive processing increases. This is expected since (given a lookahead window of .0386) the non-aggressive algorithm must synchronize approximately twenty six times to process one unit of logical time. Even though the global synchronization cost for the aggressive algorithm is twice that of the non-aggressive algorithm, this cost is incurred only once (at the end of the simulation window).

Our model assumes each LP is equally likely to receive a given message. We make this assumption in order to make the analysis tractable. However, we are interested in the performance of our algorithm given different communication topologies. In section 4.7.1 we investigated the effects of the com-

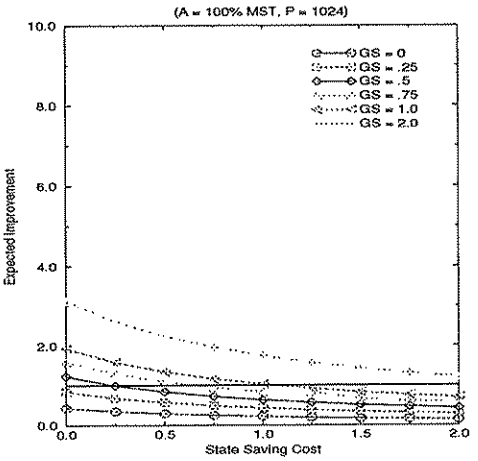

Figure 6.7 - Expected Improvement, Hot Spot (1%) Communication Pattern

munication topology under the assumption that the overhead of the correction mechanism is zero. Now we study this issue given that we have defined and implemented a correction mechanism.

In Figure 6.6 we show the expected improvement given a nearest neighbor communication topology. We assume a 2D toroidal mesh where an LP communicates only with its immediate neighbors. We assume each neighbor is equally likely to receive a given message. As can be seen, our aggressive algo-

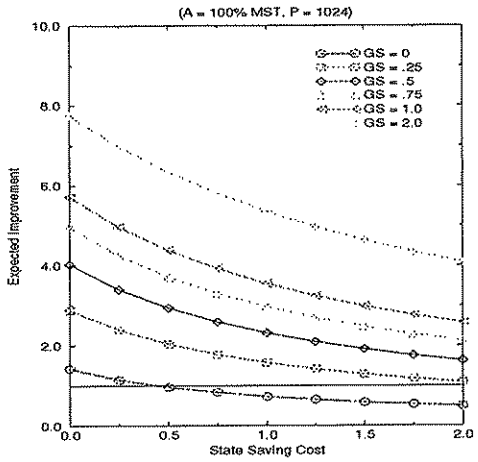

Figure 6.8 - Expected Improvement, Hot Spot (5%) Communication Pattern

rithm offers an improvement in performance except under the condition that the cost of global synchronization is zero and the cost of saving state is relatively high. These results are very encouraging since the nearest neighbor communication topology is an important configuration in parallel simulation.

The next issue we investigated is the behavior of our algorithm given a "hot spot" in the communication topology. In one experiment we assume a nearest neighbor communication pattern, where an LP sends a message to one of its immediate neighbors with an 80% probability. Note each neighbor is equally likely to receive the given message. There is a 20% probability that the LP sends the message to one of ten LPs in the system. Thus 1% of the LPs received 20% of the message traffic. The results of this experiment are shown in Figure 6.7.

As can be seen, this highly unbalanced communication pattern adversely affected our aggressive algorithm. The reason for this is that the LPs receiving the heavy message traffic were frequently rolling back to reprocess messages. Also, the performance was hurt by the requirement that all reprocessing caused by one arrival message be completed before the next message is received. It is expected that in a real system some messages may be received before all of the reprocessing associated with a preceding message is completed. In this case the LP could order the messages, and potentially avoid reprocessing

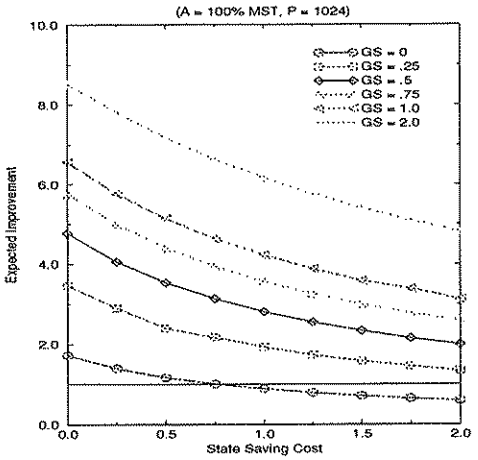

Figure 6.9 - Expected Improvement, Hot Spot (8%) Communication Pattern

the same message more than once.

Another problem may be our use of the aggressive cancellation strategy. Recall in this strategy an anti-message is sent as soon as an error becomes apparent. Thus whenever one of the LPs was rolled back, if a message was *potentially* sent in error it immediately sent an anti-message to cancel this message. Our simulation employs the aggressive cancellation strategy because of our worst case assumption that *any* out of order processing results in a violation of event dependencies. As discussed however, this is not necessarily the case. It is certainly possible that out of order processing does not violate event dependencies because no such dependencies exist between messages processed in the wrong order.

This fact has been noted and studied in connection with Time Warp (Jefferson 1985). For this reason the *lazy cancellation policy* (Reiher *et al 1990*)) was developed. In this strategy an anti-message is not sent because a message *may* have been sent in error. Rather, an anti-message is sent only when it is *known* that a message was sent in error. Thus if no event dependencies are violated by out of order processing, no anti-message is sent.

Another reason an actual system may show better results than our simulation is because it is likely that an LP will have more than one message in its queue at a time. Thus an LP can order the messages before processing (recall our simulation assumes all messages are received one at a time and therefore no ordering of messages occurs). Also, it is possible that an LP will receive an anti-message sent to cancel a particular arrival message before this arrival message has been processed. In this case the arrival message would not be processed, and the LP would not be forced to roll back due to the receipt of the arrival message.

In Figure 6.8 we show the improvement in processing cost given a "hot spot" where 5% of the LPs receive 20% of the messages. As can be seen, the performance of our algorithm is greatly improved when the communication pattern is slightly more balanced. Given this communication pattern the aggressive approach again offers significant improvement in the processing cost compared to that of the non-aggressive approach. In Figure 6.9 we show the improvement in processing cost given a "hot spot" where 8% of the LPs receive 20% of the messages. As can be seen, our approach offers excellent performance in this situation. Thus our system can perform quite well even when the message traffic is unbal-

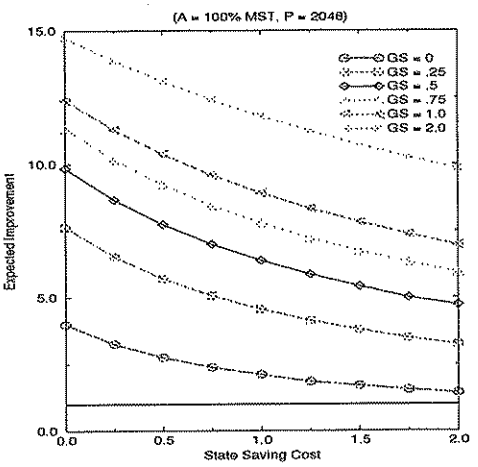

Figure 6.10 - Expected Improvement, 2048 LPs (Random Communication Pattern)

anced. When the communication pattern is highly unbalanced however our research indicates that the aggressive algorithm (without some modification) is not a good choice unless the cost of global synchronization is prohibitive.

Synch Cost	Equal Likely	Nearest Neigh.	HS 1%	HS 5%	HS 8%	2048 LPs
0	2.45	1.07	-	0.43	0.77	3.35
.25	7.00	3.90	-	2.25	3.10	9.80
.5	11.50	6.70	0.24	4.05	5.40	16.24
.75	16.00	9.50	0.65	5.90	7.70	22.75
1.0	20.55	12.40	1.07	7.70	10.00	29.00
2.0	38.40	23.50	2.70	15.00	19.30	55.00

Table 6.1 - Critical State Saving Values for Various Communication Patterns

In order to give a concise picture of the costs/benefits of aggressive processing we define what we term the *critical state saving cost.* A critical state saving cost of $SS = c$ implies that if the cost of saving state is less than c times the cost of processing a single message, the processing cost of the aggressive algorithm offers an improvement over the processing cost of the non-aggressive algorithm. Conversely, a state saving cost greater than c times the cost of processing one message implies that the processing cost of the non-aggressive algorithm is better than the processing cost of the aggressive algorithm.

In Table 6.1 we show the critical state saving cost for various communication patterns as a function of the cost of global synchronization. We assume an aggressive window size equal to 100% of the mean of the service time distribution. We give the critical state saving value for a system where 5% of the LPs receive 20% of the messages ("HS (5%)"), 8% of the LPs receive 20% of the messages ("HS (8%)"), as well as a random communication pattern where each LP is equally likely to receive a given message, and the nearest neighbor communication pattern.

Consider determining the critical state saving value for a system with a nearest neighbor communication pattern, and a global synchronization cost of .5 (i.e. 50% of the cost of processing a message). In this case we would look at the column labeled "Synch Cost" and find .5 (third row), then we would find the third row of the column labeled "Nearest Neighbor" and determine that the critical state saving value for this set of parameters is 6.70. Thus in a system with a nearest neighbor communication topology, if the cost of global synchronization is .5, then for any state saving cost less than 6.70 times the cost of processing a single message, the aggressive approach would offer an improvement in performance.

6.5.1. Behavior of Dominant LP as Workload Increases

In this section we investigate the behavior of the dominant LP as we increase the workload relative to a fixed architecture. What we would like to show is that the processing cost of the dominant LP increases only slightly as the size of the simulation problem grows. To investigate this issue we increased the number of LPs in the system from $N = 1024$ LPs to $N = 2048$ LPs.

The results of this experiment showed that the number of messages processed by the dominant LP increased very little as a result of doubling the number of LPs in the system. In particular, the mean

number of messages processed by the dominant LP in a system with $N = 1024$ LPs was 15.57. When the system was increased to $N = 2048$ LPs, the mean number of messages processed by the dominant LP increased only to 16.8 messages. Thus the processing cost was changed very little *when the number of LPs in the system doubled.* This is a very encouraging result.

We show the speed up in processing cost for a system with $N = 2048$ LPs and $P = 2048$ processors in Figure 6.10. As can be seen there is a significant improvement. It is important to note however that this improvement is due to the fact that the size of the lookahead window decreased from $L = .0386$ to $L = .0272$ as a result of doubling the number of LPs in the system. Thus the LPs (in the non-aggressive approach) are forced to synchronize thirty seven times to compute one unit of logical time rather than the twenty six times required with the smaller system. If the cost of global synchronization is greater than zero, the processing cost of the non-aggressive approach will be severely affected.

6.6. Discussion

In this chapter we have enhanced our analytic model such that we can examine the behavior of the *system* rather than the behavior of a "typical" LP. We developed our model assuming the best case behavior of the non-aggressive protocol and a set of very pessimistic assumptions regarding the behavior of the aggressive algorithm. We have shown that even under these conditions our approach can significantly improve performance.

Also we have presented a set of simulation results which support the predictions of our model. In particular, the simulation results show that our model tends to *underestimate* the potential for improvement. We further tested our analytic results by investigating the impact of various communication topologies on the behavior of our system. These results indicate that except in the case of a highly unbalanced communication pattern, aggressive processing offers the potential for a significant increase in performance. Even given a highly unbalanced communication pattern, the aggressive approach offers an improvement in the processing cost if the cost of global synchronization is high. Finally, simulation studies show that the behavior of the dominant LP is only slightly affected as the number of LPs in the system is doubled.

CHAPTER 7

Conclusions

Parallel discrete event simulation is an important and interesting application of parallel processing. As we have shown, parallel simulation presents very difficult synchronization issues due to the underlying sense of logical time. Many protocols have been introduced to solve these difficult issues, but only recently have we begun to develop a body of theoretical results to help predict, explain or bound the performance of any given synchronization mechanism. The research presented in this dissertation makes a significant contribution to the small but growing set of analytic results for parallel simulation.

In this research we have argued that we need to define synchronization mechanisms that blend aspects of aggressive and non-aggressive protocols in order to maximize the advantages, and minimize the disadvantages of each approach. We have defined one such protocol and developed a set of analytic results to study the improvement in performance made possible by this approach. This work represents the first time the relationship between the level of aggressiveness and the expected improvement in parallelism has been established. It is also the first time the relationship between the level of aggressiveness and the costs of aggressive processing has been established. Further, we have made significant progress towards demonstrating that our aggressive windowing algorithm maintains the important scalability features of the non-aggressive algorithm. Finally, we have derived the probability of a causality error at a given LP, and the probability that an LP initiates a rollback chain by issuing an anti-message, as the number of LPs approaches infinity. These are all significant contributions.

Below we give a summary of the contributions of each chapter, and where appropriate discuss the weaknesses of our approach. We conclude this chapter with our thoughts for future research.

7.1. Summary of Results

In Chapter 3, we define the *aggressive window* as a mechanism for adding aggressiveness to an existing non-aggressive protocol. We derive the probability distributions required for our analysis. Given

these distributions, we develop a model to investigate the probability of a causality error, and the expected improvement due to aggressive processing, as a function of the level of aggressiveness. This model is developed under the assumption of a closed queueing system that is heavily loaded, has exponential service time distributions and a random routing of messages where each LP is equally likely to receive a given message. We do not develop a correction mechanism in that chapter, and thus do not include the costs of this mechanism in the model. Further, the analysis developed in Chapter 3 investigates the behavior of a "typical" LP rather system level performance. However, the results derived in this chapter lay the foundation for our analyses where we do include the cost of the synchronization mechanism and investigate system-level performance.

The primary weakness of this chapter is in our approach to computing the probability of a causality error and the expected improvement in parallelism due to aggressive processing. We derive these results by enumerating all of the possible ways a causality error can occur, and all of the possible ways an LP can process K aggressive messages successfully. This computation is very tedious. The first extension to this work will be the use of order statistics to make this analysis less cumbersome.

In Chapter 4 we define a mechanism to correct the causality errors that occur as a result of aggressive processing. Also, we extend our earlier analysis to investigate the probability of a causality error that requires an LP to produce an anti-message. The main result of this chapter is the establishment of the relationship between the level of aggressiveness and the probability that an LP produces an *Nth* generation anti-message.

In Chapter 5 we extend our analysis to an open queueing network, where the system can be lightly loaded. This analysis is important in that it allows us to analyze lightly loaded systems. Also it is important in that it demonstrates how quickly the analysis becomes intractable as we move away from a system that allowed us to make many simplifying assumptions. Again, the analysis is tedious. For this reason most of the analysis appears in the appendix.

In Chapter 6 we synthesize the results of the previous chapters. The main contribution of this chapter is a model that describes *system level performance* rather than the behavior of a "typical" LP. We develop our model to compare the *processing cost* of the two approaches, where we define the processing

cost as the cost of processing one unit of logical time. In this model we include the synchronization costs of the non-aggressive protocol, as well as the cost of aggressive processing. Our model makes the best case assumptions for the non-aggressive approach, and a set of very pessimistic assumptions for our aggressive algorithm. Even under this very unbalanced set of assumptions, our model predicts the potential for significant performance gains due to aggressive processing. Also in Chapter 6 we investigate the impact of various communication topologies on our system. It is shown that aggressiveness can improve performance *except* under the conditions of a very highly unbalanced communication pattern with a low global synchronization cost. Finally, we demonstrate that the costs of aggressive processing increases very slowly as the number of LPs in the system increases.

7.2. Future Research

The results developed in this research are primarily theoretical in nature. In order to make the analysis tractable, we have been forced to make a number of simplifying assumptions. These assumptions include that each LP is equally likely to receive a given message, that the completion of one activity causes exactly one activity to be scheduled somewhere in the system, and that service times are exponential. The assumptions we have made are common. However, we would however like to augment our analytic results with a set of empirical results derived from a multiprocessor implementation of our algorithm. While our simulation studies suggest that our analytic results are applicable beyond the simplifying assumptions of our model, we would like to validate this through an implementation of our protocol.

Perhaps the most striking unanswered question of this research is whether there is some optimal level of aggressiveness with a protocol such as ours. We discussed the increase in parallelism, and the costs of aggressive processing as a function of the level of aggressiveness. We did not focus on an optimal tradeoff between these competing costs. This would be a very interesting area of future research.

Our comparison of the performance of the aggressive and non-aggressive algorithms assumes a system with some lookahead capabilities. We would like to extend the analysis of our protocol to systems with no lookahead. It would be a significant contribution if we could determine the costs of aggressive processing, as a function of the level of aggressiveness, for a system without lookahead.

Another area of future research is to theoretically demonstrate the scalability of our approach. Our empirical results demonstrate that the cost of aggressive processing increases very slowly as the size of the system increases. We have not yet established this result analytically.

7.3. Concluding Remarks

In this thesis we have studied the difficult and interesting synchronization issues encountered in parallel discrete event simulation. It is our belief that the future direction of synchronization mechanisms for parallel simulation should be toward the blending of non-aggressive and aggressive protocols; we feel the work in this thesis demonstrates the advantages of this approach. The results presented in this thesis have been primarily theoretical, and we feel we have made a contribution to the growing body of theoretical results for parallel simulation. Our hope is that these theoretical results will be used to help guide the development of powerful mechanisms for parallel discrete event simulation.

CHAPTER 8

Bibliography

BIBLIOGRAPHY

Aahlad, A. and J. Browne 1988. The Persistent Echo Problem and a Solution. *Unpublished Manuscript.*

Akyldiz, I.F., L. Chen, S.R. Das, R.M. Fujimoto and R.F. Serfozo 1992. Performance Analysis of Time Warp With Limited Memory. *1992 ACM Sigmetrics Conference.*

Ayani, Rassul 1989. A Parallel Simulation Scheme Based on Distances Between Objects. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, Volume 21 Number 2, 113-118. Society for Computer Simulation, March 1989.

Breiman, Leo 1986. Probability and Stochastic Processes With a View Toward Applications. *The Scientific Press.* Second Edition.

Bryant, R.E. Simulation of packet communication architecture computer systems. *MIT-LCS-TR-188*, Massachusetts Institute of Technology, 1977.

Buzzle, C., M. Robb and R. Fujimoto 1990. Modular VME Rollback Hardware for Time Warp. *Proceedings of the SCS 1990 Multiconference on Distributed Simulation.* January, 1990.

Chandy, K.M. and J. Misra 1979. A Case Study in the Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* SE-5,5 May 1979, 440-452.

Chandy, K. and J. Misra 1987. Conditional Knowledge as a Basis for Distributed Simulation. Technical Report 5251:TR:87, California Institute of Technology, 1987.

Chandy, K., and R. Sherman 1989. The Conditional Event Approach to Distributed Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation,* Pgs. 93-99, January, 1989.

Dickens, P. and P. Reynolds 1990. SRADS with Local Rollback. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation,* January, 1990, 161-164.

Dickens, P. and P. Reynolds 1991. A Performance Model for Parallel Discrete Event Simulation. *1991 Winter Simulation Conference.* December, 1991, 618-626.

Dickens, P., P. Reynolds and J.M. Duva 1992. Performance and Scalability Results for an Aggressive Global Windowing Algorithm. TR-92-10, Department of Computer Science, University of Virginia, April, 1992.

Ebling, M. *et al,* 1989. An Ant Foraging Model Implemented on the Time Warp Operating System *Distributed Simulation,* V 21, Num. 2, Pgs. 21-26, March, 1989.

Felderman, R.E. and L. Kleinrock 1990. An Upper Bound on the Improvement of Asynchronous vs. Synchronous Distributed Processing. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation,* January, 1990, 131-136.

Felderman, R. and L. Kleinrock 1991. Two Processor Time Warp Analysis: Some Results on a Unifying Approach. *Distrubted Simulation 1991,* SCS Simulation Series, Vol.23, pgs. 3-10, 1991.

Felderman, Robert and Leonard Kleinrock 1991a. "Bounds and Approximations for Self-Initiating Distributed Simulation Without Lookahead." Submitted (September 1991): *ACM Transactions*

on Modeling and Computer Simulation.

Felderman, R. and L. Kleinrock 1992a. Two Processor Time Warp Analysis: Capturing the Effects of Message Queueing and Rollback/State Saving Costs. Unpublished Manuscript.

Felderman, R. and L. Kleinrock 1992. Two Processor Conservative Simulation Analysis. *Distributed Simulation 1992.* SCS Simulation Series, Vol. 24 Num. 3, pgs. 169-177, 1992.

Fox, G., A. Johnson, A. Lyzenga, A. Otto, J. Salamon and D. Walker, 1988. Solving Problems on Concurrent processors *Prentice-Hall*, 1988.

Fujimoto, 1988. Lookahead in Parallel Discrete Event Simulation. *Proc. 1988 International Conference on Parallel Processing*, Volume 3, Pgs. 34-41, August, 1988.

Fujimoto, R., J. Tsai and A. Gopalakrishnan 1988a. The Rollback Chip: Hardware Support for Distributed Simulation Using Time Warp. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, Pgs. 81-86, January, 1988.

Fujimoto, 1989. Performance Measurements of Distributed Simulation Strategies. *Transactions of the SCS*, Society for Computer Simulation, April, 1989.

Fujimoto, R. 1989a. Time Warp on a Shared Memory Multiprocessor. *Proceedings of the 1989 International Conference on Parallel Processing*, 1989.

Fujimoto, R. 1990. Parallel Discrete Event Simulation. *Communications of the ACM*, Volume 33, Number 10, October 1990, 30-53.

Fujimoto, R. 1990a. Performance of Time Warp Under Synthetic workloads. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation,* Pgs. 23-28, January, 1990.

Gafni, A. Rollback Mechanisms for Optimistic Distributed Simulation Strategies. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation,* Pgs. 61-67, January, 1988.

Gupta, A., I. Akyldiz and R. Fujimoto 1991. Performance Analysis of Time Warp With Multiple Homogeneuos Processors. *IEEE Transactions on Software Engineering.* Volume 17 (No. 10):1013-1027, Oct. 1991.

Groselj, A.B. and C. Tropper 1988. The Time of Next Event Algorithm. *Proc. of the 1988 SCS Multiconference on Distributed Simulation,* pgs. 25-29, January 1988.

Groselj, A.B. and C. Tropper 1989. A Deadlock Resolution Scheme for Distributed Simulation. *Proc. of the 1989 SCS Multiconference on Distributed Simulation,* Pgs. 108-112, January, 1989.

Jefferson, D.R. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems,* 7,3 (1985), 404-425.

Kumar, D. 1986. Simulating Feedforward Systems Using a Network of Processors. *The 19th Annual Simulation Symposium,* pgs. 127-144, March 1986.

Lavenburg, S., R. Muntz and B. Samadi 1983. Performance Analysis of a rollback method for distributed simulation. *PERFORMANCE 83,* Elsevier Science Pub (North Holland) pgs. 117-132, 1983.

Lin, Y., J. Baer and E. Lazowska 1988. Tailoring a Parallel Trace-Driven Simulation Technique to Specific Multiprocessor Cache Coherence Protocols. Technical Report 88-03-02, University of Washington March, 1988.

Lin, J. and E. Lazowska 1989. Exploiting Lookahead in Parallel Simulation. Technical Report 89-10-06, University of Washington, October, 1989.

Lin, J and E. Lazowska 1989a. A Study of Time Warp Rollback Mechanisms. Technical Report 89-09-07, University of Washington, 1989.

Lin Y.B., and E.D. Lazowska 1990. Optimality Considerations for Time Warp Parallel Simulation. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, January, 1990,29-34.

Lin Y.B., E. Lazowska and J.L. Baer 1990a. Conservative Parallel Simulation for Systems with no Lookahead Prediction *Proc. 1990 SCS Multiconference on Distributed Simulation,* pgs. 144-149, January, 1990.

Liu, L.Z. and C. Tropper 1990. Local Deadlock Detection in Distributed Simulations. *Proceedings of the 1990 Multiconference on Distributed Simulation*, pgs. 64-69, January 1990.

Lubachevsky B. 1988. Bounded Lag Distributed Discrete Event Simulation. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, January, 1988,183-191.

Lubachevsky B., A. Shwartz and A. Weiss 1989. Rollback Sometimes Works... If Filtered. *Proceedings of the 1989 Winter Simulation Conference.* December, 1989, 630-639.

Lubachevsky, B. 1989a. Scalability of the Bounded Lag Distributed Event Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, January, 1989, 100-105.

Madisetti V., J. Walrand and D. Messerschmitt 1990. Synchronization in Message-Passing Computers. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, January, 1990, 35-48.

Madisetti, V., D. Hardaker and R. Fujimoto 1992. The MINDIX Operating System for Parallel Simulation. *Distributed Simulation*, SCS Simulation Series, Vol. 24, Num. 3, pgs. 65-74, Jan. 1992.

Misra, J. 1986. Distributed Discrete-Event Simulation. *Computing Surveys*, Vol. 18, pgs. 39 - 64, March 1986.

Mitra, D. and I. Mitrani 1984. Analysis and Optimum Performance of Two Message-Passing Parallel Processors Synchronized by Rollback. *PERFORMANCE 84*, Elsevier Science Pub (North Holland) pgs. 35-51, 1984.

Mizel D. and R. Lipton 1990. Time Warp vs. Chandy-Misra: A Worst Case Comparison. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, January, 1990, 137-143.

Nicol, D. 1984. Synchronzng Network Performance. *Master's Thesis*, University of Virginia, 1984.

Nicol, D. 1988. High Performance Parallelized DES of Stochastic Queueing Networks. *Proceedings of the 1988 Winter Simulation Conference*, pgs. 306-313, San Diego, Ca., December

1988.

Nicol, D. and Reynolds, P. 1984. Problem Oriented Protocol Design. *Proceedings 1984 Winter Simulation Conference*, pgs. 471-474, Dallas, Texas, November, 1984.

Nicol, David M. 1991. "Performance Bounds on Parallel Self-Initiating Discrete Event Simulations". *ACM Transactions on Modelling and Computer Simulation.* Volume 1, No. 1, 1991

Nicol, D. 1992. Conservative Parallel Simulation of Priority Class Queueing Networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, Num. 3, Pgs. 294-303, May, 1992.

Nicol, D. 1992a. Optimistic Barrier Synchronization. Submitted: *Parallel Computing.*

Nicol, D. 1992b. Personal Communication.

Nicol, D. 1993. The Cost of Conservative Synchronization in Parallel Discrete Event Simulation. *JACM*, to appear, April 1993.

Peacock, J.K, J.W. Wong and E.G. Manning 1979. Distributed Simulation Using a Network of Processors. *Computer Networks* 3 (1979), 44-56, North-Holland Publishing.

Peacock, J.K, J.W. Wong and E.G. Manning 1979a. A Distributed Approach to Queueing Network Simulation *Proceedings of the 1979 Winter Simulation Conference,* pgs. 399-406 December, 1979.

Presley, M., M. Eblin, F. Wieland, D. Jefferson, 1989. Benchmarking the Time Warp Operating System with a Computer Network Simulation *Distributed Simulation*, V 21, Num. 2, Pgs. 8-13,

March, 1989.

Reiher, P., R. Fujimoto, S. Bellenot, D. Jefferson 1990. Cancellation Strategies in Optimistic Execution Systems. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation.*, January, 1990.

Reynolds, P. 1982. A Shared Resource Algorithm for Distributed Simulation. *Proceedings of the Ninth Annual International Computer Architecture Conference*, pgs. 259-266, Austin, Texas, April, 1982.

Reynolds, P. 1988. A Spectrum of Options for Parallel Simulation. *Proceedings of the 1988 Winter Simulation Conference*, December 12-14, San Diego, California, 325-332.

Reynolds, P., C. Pancerella and S. Srinivasan 1993. Design and Performance Analysis of Hardware Support for Parallel Simulations, to appear in a special issue of *Journal of Parallel and Distributed Computing* on Parallel and Distributed Simulation, August 1993.

Sokol, L., D.Briscoe and A. Wieland 1988. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. *Proc. 1988 SCS Multiconference on Distributed Simulation*, Pgs. 34-42, January 1988.

Sokol, L.and B. Stucky 1990. MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, Pgs. 169-173, January, 1990.

Steinman, J. 1992. SPEEDES: A Unified Approach to Parallel Simulation. *Distributed Simulation*, SCS Simulation Series, Vol. 24, Num. 3, pgs. 75-84, Jan. 1992.

Theofanos, M. 1984. Distributed Simulations of Queueing Networks. Master's Thesis, University of Virginia, 1984.

Turner, S. and M. Xu 1992. Performance Evaluation of the Bounded Time Warp Algorithm. *Distributed Simulation*, SCS Simulation Series, Vol. 24, Num. 3, pgs. 117-126, Jan. 1992.

Wagner, D. and E. Lazowska 1988. Parallel Simulation of Queueing Networks: Limitations and Potentials. TR88-09-05, University of Washington, Seattle, Washington, September 1988.

Wieland, F. *et al*, 1989. Distributed Combat Simulation and Time Warp: The Model and its Performance *Distributed Simulation*, March 1989.

APPENDIX

A.1. Introduction

In Chapter 5 we defined and described all of the events which must occur in order for an LP to produce an anti-message. We showed the probability of a first generation anti-message (at a given LP) is

$$\begin{aligned} P(Anti) = {} & \rho P(C)\, P(Act\,1)\, P(2nd)\, P(invalidateA)\, p_1 \\ & + \rho\, P(C)\, P(Act\,2)\, P(C\_2nd)\, P(invalidateB)\, p_1 \\ & + (1-\rho)\, P(Actw)\, P(S\_2nd)\, P(invalidateC)\, p_1 \\ & + (1-\rho)\,(1 - P(Actw))\, P(2nd)\, P(invalidateD)\, p_1 \end{aligned} \tag{A.1}$$

In this appendix we compute the probability of each event in this equation.

We organize the appendix as follows. First we compute the probability of an anti-message given that the server is busy with an activity which began service at logical time $t < L$, and which completes service at logical time $t' > L$. That is, the server is busy with an activity which completes at a logical time greater than the lower bound of the aggressive window. (Throughout this appendix we use the term "busy as the LP begins to process in the aggressive window", or "busy as the LP begins to process aggressively" to denote this same event.) Then we compute the probability of an anti-message given that the server is idle at logical time L, the lower bound of the aggressive window. (Throughout the appendix we use the term "free at the beginning of the aggressive window", or "free as the LP begins to process aggressively" to denote this same event.) Our last step is to compute the probability that a given LP produces an *Nth* generation anti-message.

A.2. Probability of an Anti-Message Given a Busy Server

As discussed in Chapter 5, there are two basic cases which must be considered: the server is busy as the LP begins to process in the aggressive window or the server is free as the LP begins to process in the aggressive window. In this section we investigate the probability of an anti-message given that the server is busy as the LP begins to process in the aggressive window.

The first two lines of Equation (A.1) give the probability of an anti-message given that the server is busy as the LP begins to process aggressively.

$$P(Anti \mid Server\ Busy) = \rho\, P(C)\, P(Act\_1)\, P(2nd)\, P(invalidateA)\, p_1 + \rho\, P(C)\, P(Act2)\, P(C\_2nd)\, P(invalidate\_2)\, p_1 \quad \text{(A.1.A)}$$

We begin by deriving the probability of each of these events.

Recall that ρ is the probability that a server is busy at any given time, and thus represents the probability that the server is busy as the LP begins to process aggressively. In Chapter 5 we showed that $\rho = \lambda_T/\lambda$.

We next term in this equation is $P(C)$, the probability that the activity currently receiving service completes within the aggressive window. Recall the duration of an activity is drawn from independent, identically distributed exponential random variables with mean $1/\lambda$. The probability that the completion time of the activity currently receiving service falls within the aggressive window (given that the completion time is greater than L, the lower bound of the aggressive window) is

$$P(C) = 1 - e^{-\lambda A}. \quad \text{(A.2)}$$

We now calculate the probability of event Act_1, the event that there is an activity to begin service immediately after the server becomes idle. This will occur if a) there is an activity on the server queue, or b) there is an activity in the aggressive window with a timestamp less than that of the "complete_service" message. Consider the probability there are *no* activities on the server queue. As shown by Trevedi (1982, page 420), the probability of zero activities on the server queue in the steady state is $1-\rho$.

Now consider the probability of zero messages in the aggressive window with a timestamp less than that of the "complete_service" message. As discussed in Chapter 5, there are two types of messages in the aggressive window which must be considered: messages from internal sources and messages from external sources. Let *NoInternal* be the event that there are no messages from internal sources with a timestamp less than T_{CS} (the timestamp of the "complete_service" message). Recall we derived the probability of K messages in the aggressive window (due to internal sources) in Equation (5.8). The probability of *NoInternal* is the probability that there are zero messages in the aggressive window (from

internal sources), or there is one message from an internal source with a timestamp greater than T_{CS}, or two messages from internal sources, both of which have timestamps greater than T_{CS}, and so forth. The computation of this probability is essentially the same as in Equation (3.28) and is not reiterated in this section.

We define *NoExternal* as the event that there are no external messages in the aggressive window with a timestamp less than T_{CS}. Consider an interval from logical time 0 .. $T_{CS} = t$. Given that the external messages are from a Poisson process with rate λ_E, the probability of zero messages in this interval is

$$P\,(NoExternal \mid T_{CS} = t) = e^{-\lambda_E t}.$$

In order to uncondition this expression, we integrate over all possible values of $T_{CS} = t$ times the probability of $T_{CS} = t$. Recall the timestamp of the "complete_service" message is a conditional exponential, conditioned on being within the aggressive window. The unconditioned probability of *NoExternal* is

$$P\,(NoExternal) = \int_0^A e^{-\lambda_E t} \frac{\lambda e^{-\lambda t}}{(1 - e^{-\lambda A})} dt. \tag{A.3}$$

This is a complicated integral with no closed form solution of which we are aware. We leave it in integral form and note that it must be solved numerically. The probability of the event *Act_1* (the event that there exists an activity ready to receive service upon the processing of the "complete_service" message) is one minus the probability that no such activity exists. Thus it is one minus the probability of no activities on the server queue, no internal messages with a timestamp less than T_{CS} and no external messages with a timestamp less than T_{CS}. This probability is

$$P\,(Act\,1) = 1 - ((1 - \rho)\,P\,(NoInternal)\,P\,(NoExternal)). \tag{A.4}$$

Next we determine the probability of the event *2nd*, the event that an activity that begins service immediately after the processing of the "complete_service" message has a completion time within the aggressive window. The completion time of the next activity to enter into service is $T_{CS} + \xi$, where ξ is an exponential random variable with mean $1/\lambda$. Given a particular timestamp $T_{CS} = t$, this probability is $1 - e^{-\lambda(A - t)}$. We uncondition this expression below.

$$P\,(2nd) = \int_0^A (1 - e^{-\lambda(A-t)}) \frac{\lambda e^{-\lambda t}}{(1 - e^{-\lambda A})} dt = \frac{1 - (\lambda A e^{-\lambda A} + e^{-\lambda A})}{1 - e^{-\lambda A}} \tag{A.5}$$

For the remainder of this appendix we define ξ to be an exponential random variable with mean $1/\lambda$.

Also, we define T_{CS} to be the timestamp of the "complete_service" message (given that it falls within the aggressive window), and thus it is a conditional exponential, conditioned on being within the aggressive window.

The next term on the first line of Equation (A.1.A) is $P(InvalidateA)$, the probability that the LP receives an arrival message with a timestamp less than T_{CS}. We discuss this probability below. The final term on this line is p_1, which we have previously discussed.

Now consider the second line of Equation (A.1.A). Act_2 is the event that there is no activity to enter into service at logical time T_{cs}, but there is an activity in the aggressive window with a timestamp X such that $X > T_{CS}$. The probability of Act_2 is one minus the probability that there are no internal messages in the interval T_{CS} .. A and there are no external messages in this interval. As noted, the timestamp distribution of the "complete_service" message is a conditional exponential, conditioned on being within the aggressive window. Thus we seek the probability that there is at least one message in the aggressive window with a timestamp that is greater than T_{CS}, which is a conditional exponential. This is exactly the probability of a fault computed in section 5.3. We do not recompute this probability in this appendix.

We now calculate the probability of the event Act_2. The probability of no activities on either the server queue or in the aggressive window with a timestamp less than that of the "complete_service" message is one minus the probability of Act_1 (given in Equation (A.4)). The probability of an activity in the aggressive window with a timestamp greater than that of the "complete_service" message is the probability of a fault given in section 5.3. Due to the independence of these events, the probability of Act_2 is

$$P(Act\_2) = (1 - P(Act\_1))\ P(Fault). \tag{A.6}$$

Now assume that event Act_2 occurs, and thus the next activity to enter into service has a timestamp between T_{cs} (the timestamp of the "complete_service" message) and A. We represent this timestamp as logical time X. We need to compute the probability of event C_2nd, the event that an activity which enters into service at logical time X completes within the aggressive window. X is a conditional exponential, conditioned on being between the "complete_service" message and the upper bound of the aggressive window. Note there is a non-zero probability of having no activities to enter service upon the processing of the "complete_service" message, and $N>1$ activities with timestamps between the

"complete_service" message and the upper bound of the aggressive window. In this case the timestamp of the next activity to receive service would have a distribution that is the minimum of N conditional exponentials (conditioned on falling between the "complete_service" message and the upper bound of the aggressive window). However, because of the restrictions that $0 \leq \rho < 1$ and $0 \leq A \leq 1/\lambda$, the probability there are no activities available to enter into service when the "complete_service" message is processed, and $N > 1$ activities with a timestamp between that of the "complete_service" message and A is quite small. For the purposes of our discussion we assume this probability is negligible and do not consider the affects of more than one activity in this interval.

Given an activity with timestamp X (in the interval between T_{cs} and A) is the next activity to receive service, we seek the probability that this activity completes within the aggressive window. Recalling that the service time distribution is exponentially distributed with mean $1/\lambda$, we seek the probability that the new timestamp $X + \xi$ is less than A. Given a particular value of $X=x$ this probability is $1 - e^{-\lambda(A-x)}$. As noted, x can range between $T_{CS} = t$ and A. Thus the probability that $X + \xi < A$ given a particular value of $T_{cs}=t$ is

$$P(x + \xi < A \mid T_{cs}=t) = \int_t^A (1-e^{-\lambda(A-x)}) \frac{\lambda e^{-\lambda x}}{e^{-\lambda t}-e^{-\lambda A}}\, dx = \frac{\lambda A e^{-\lambda A} + e^{-\lambda A} - e^{-\lambda t} - \lambda t e^{-\lambda A}}{e^{-\lambda A}-e^{-\lambda t}}.$$

Note $\frac{\lambda e^{-\lambda x}}{e^{-\lambda t}-e^{-\lambda A}}$ is the conditional distribution of X given that it ranges between $T_{cs}=t$ and A. We uncondition this expression by intergating over all possible values of $T_{cs}=t$ times the probability that $T_{cs}=t$. Noting that T_{cs} is a conditional exponential we do this below.

$$P(C\_2nd) = P(X + \xi < A) = \int_0^A \frac{\lambda A e^{-\lambda A} + e^{-\lambda A} - e^{-\lambda t} - \lambda t e^{-\lambda A}}{e^{-\lambda A}-e^{-\lambda t}} \frac{\lambda e^{-\lambda t}}{1-e^{-\lambda A}} dt \qquad (A.7)$$

This is a difficult integral with no closed form solution of which we are aware. We leave it in integral form and note that it must be solved numerically.

Now that we have computed the probabilities associated with the next activity to receive service, we need to compute the probability that the chosen activity is invalidated by the receipt of an arrival message. Again we note our assumption that the chosen activity is invalidated if the LP receives an arrival message with a timestamp less than the logical time of the activity entered into service aggressively.

Recall *inval_1* is the event that the chosen activity is invalidated given that it enters service at logical time T_{cs}. This probability is 50% since both the arrival message and the "complete_service" message have (by assumption) the same timestamp distribution (conditional exponentials, conditioned on falling within the aggressive window).

$$P(inval\_1) = .5$$

It is somewhat more difficult to calculate the probability of invalidating an activity given that the timestamp of the activity is a conditional exponential, conditioned on being between T_{cs} and A. Noting that this probability is derived in a manner similar to the other such probabilities derived above, the unconditioned probability of this event is

$$P(inval\_2) = \int_0^A \int_t^A \int_0^x \frac{\lambda e^{-\lambda s}}{(1-e^{-\lambda A})} \frac{\lambda e^{-\lambda x}}{e^{-\lambda t}-e^{-\lambda A}} \frac{\lambda e^{-\lambda t}}{1-e^{-\lambda A}} ds\, dx\, dt = 0.75 \qquad \text{(A.8)}$$

The $\frac{\lambda e^{-\lambda s}}{(1-e^{-\lambda A})}$ term is the timestamp distribution of an arrival message. The inside integral represents the probability that an arrival message has a timestamp less than the logical time of the activity that is placed into service aggressively, given that the activity has a particular timestamp $X=x$. The next integral is the probability distribution for $X=x$ given a particular timestamp of the "complete_service" message $T_{cs}=t$. The final integral computes this probability over all possible values of the "complete_service" timestamp times the probability of these values.

Now we account for the fact that an LP may receive more than one arrival message. We seek the probability that the activity chosen for service is invalidated given that the server is busy when the LP begins processing within the aggressive window. First consider the case that the activity enters into service at logical time T_{cs}, the timestamp of the complete service message. Recall we defined *invalidateA* as the event that such an activity is invalidated by the receipt of an arrival message, given that the LP may receive more than one such message. As shown above, the probability of any particular arrival message invalidating such an activity is 50% since they both have the same timestamp distribution. The probability of invalidating this activity given more than one arrival message is thus

$$P(invalidateA) \simeq P(Arr=1)\ 0.5 + P(Arr=2)\ 0.75 + P(Arr=3)\ 0.875 \qquad \text{(A.9)}$$

In Equation (A.9), the second term is the probability of invalidating the activity given two arrival

messages. This probability is 0.75 because the probability that a given arrival message has a timestamp greater than that of the arrival message (and thus not invalidating the activity) is 0.5. The probability that the timestamps of two arrival messages are both greater than that of the activity is $0.5^2 = 0.25$. The probability that at least one of the arrival has a timestamp less than that of the activity, and thus the probability that the activity is invalidated, is $1 - 0.25 = 0.75$. The other term is similarly derived. The probability of receiving more than three arrival messages is negligible and is not considered. Given that it is possible to receive more than three arrival messages however this equation is an approximation.

Recall in Chapter 5 we defined *invalidateB* as the event that an activity is invalidated given that it entered into service at logical time $X \mid T_{cs} < X < A$. As shown in Equation (A.8), the probability of an invalidation given one arrival message is 0.75. We derive the probability of an invalidation given more than one event in a manner similar to this derivation in Equation (A.8). The probability of event *invalidateB* is

$$P(invalidateB) \simeq P(Arr=1)\ 0.75 + P(Arr=2)\ 0.93\ P(Arr=3)\ 0.98 \qquad \text{(A.10)}$$

Again we note that the probability of receiving more than three arrival messages is negligible and is not considered. Because it is possible to receive more than three events however Equation (A.10) is an approximation.

The final requirement to produce an anti-message (with which we are concerned) is that the activity remain in the system upon its completion. As noted in Chapter 5, the probability of this event is p_1.

We have now computed all of the probabilities associated with producing an anti-message given that the server is busy at the beginning of the aggressive window. Our next task is to compute this probability given that the server is free as the LP begins to process aggressively.

A.3. Probability of an Anti-Message Given Server is Free

In this section we derive the probabilities associated with generating an anti-message given that the server is free at the beginning of the aggressive window. Recall the last two lines of Equation (A.1) give the probability of an anti-message when the server is idle as the LP begins to process aggressively.

$$P(Anti \mid Server\ Idle) = + (1-\rho)\ P(Actw)\ P(S\_2nd)\ P(invalidateC)\ p_1 \qquad \text{(A.1.B)}$$

$$+ (1-\rho)\,(1 - P\,(Actw))\,P\,(2nd)\,P\,(invalidateD)\,p_1$$

There are two basic cases to consider: either the LP has an activity with a timestamp that falls within the aggressive window (which it would place into service aggressively) or it does not. First consider the case that such an activity exists.

The first term in Equation (A.1.B) is $1-\rho$, which is one minus the probability that the server is busy at any given time. It therefore represents the probability that the server is idle at any given time, and is thus the probability that the server is idle as the LP begins to process aggressively.

The next term is *ActW*, the event that there is an activity to be placed into service aggressively given that the server is idle as the LP begins to process aggressively. Recall that the number of activities in the aggressive window is Poisson distributed with rate Ψ. The probability of at least one activity within the aggressive window is thus

$$P\,(ActW) = 1 - e^{-\Psi}. \tag{A.11}$$

The next term in this equation is *S_2nd*, the probability that the next activity to enter into service completes within the aggressive window. In order to compute the probability of this event, we must determine the timestamp distribution of the next activity to enter into service.

The next activity to receive service will be the activity with the smallest timestamp within the aggressive window. Given $N \geq 1$ activities within the aggressive window, we seek the timestamp distribution for the activity with the minimum timestamp. We note that the calculation of the *pdf* for the minimum timestamp among the $N \geq 1$ activities in the aggressive window is performed in a manner analogous to the calculation of the *pdf* of the Clock random variable given in Dickens and Reynolds (1991). For this reason we do not elaborate on the technique in this appendix. Let ϕ be the minimum timestamp among all timestamps within the aggressive window. The CDF of ϕ is

$$CDF\,(\phi) = 1 - \sum_{M=1}^{\infty} \left(\frac{e^{-\lambda\phi} - e^{-\lambda A}}{1-e^{-\lambda A}} \right)^{M} \frac{\frac{\Psi^M}{M!}e^{-\Psi}}{1-e^{-\Psi}}. \tag{A.12}$$

The $\frac{\frac{\Psi^M}{M!}e^{-\Psi}}{1-e^{-\Psi}}$ term is the probability of M activities in the aggressive window given that $M \geq 1$. The *pdf* of ϕ is the derivative of Equation (A.12) with respect to ϕ. For the remainder of this appendix we term

this *pdf* $f(\phi)$ and direct the interested reader to Dickens and Reynolds (1991) for its derivation.

Given $f(\phi)$ we can derive the probability of the event *S_2nd*, the event that the chosen activity completes within the aggressive window. The probability of this event is the probability that the new timestamp $\phi + \xi$ falls within the aggressive window. This probability is

$$P(S\_2nd) = \int_0^A 1 - e^{-\lambda(A-\phi)} f(\phi)\ d\phi. \tag{A.13}$$

The logic of this derivation is similar to the derivation of events *C_2nd* and *2nd* discussed above.

Given that such an activity exists, we seek the probability of event *invalidateC*, the event that the LP receives an arrival message with a timestamp less than ϕ, the logical time at which the activity entered into service. We term this event *inval_3*. This probability is

$$P(inval\_3) = \int_0^A \int_0^\phi \frac{\lambda e^{-\lambda s}}{(1-e^{-\lambda A})}\ f(\phi)\ ds\ d\phi. \tag{A.14}$$

The derivation of this probability is similar to the derivation of the probability of events *inval_1* and *inval_2* discussed above. This is a complex integral with no closed form solution of which we are aware.

Event *inval_3* gives the probability of an invalidation given one arrival message. We now compute the probability of *invalidateC*, the probability that the activity is invalidated given more than one arrival message. The probability of this event is

$$P(invalidateC) \simeq P(Arr{=}1)\ inval\_3 + P(Arr{=}2)\ (1 - (1 - P(inval\_3)^2)) + (1 - (1 - P(inval\_3)^3)) \tag{A.15}$$

This probability is computed in a manner analogous to events *invalidateA* and *invalidateB* discussed above. The probability of receiving more than three arrival messages is negligible and is not considered. For this reason however the above equation is an approximation.

The next term in Equation (A.1.B) is p_1, the probability that the activity stays in the system upon its completion. We have now calculated the probability of each event in the first line of Equation (A.1.B).

Now consider second line of Equation (A.1.B). This line gives the probability of an anti-message given that the server is free at the beginning of the aggressive window and there are no activities within the aggressive window. The first term is $1 - \rho$, the probability that the server is idle. The next term is

$1 - ActW$, the probability that there is no event with a timestamp that falls within the aggressive window. These probabilities have been discussed above.

As discussed, if the server is idle as the LP begins to process aggressively, and there are no activities within the aggressive window, then in order to produce an anti-message the LP must receive an arrival message and place this activity into service aggressively. Also, this activity must complete within the aggressive window. The timestamp of an arrival message is (by assumption) a conditional exponential, conditioned on falling within the aggressive window. This is also the timestamp distribution of a "complete_service" message. Thus the probability that an arrival message (which is placed into service) completes within the aggressive window is the same as the probability an activity placed into service at logical time T_{CS} (timestamp of the "complete_service" message) completes within the aggressive window. This probability is $P(2nd)$ discussed in previous sections.

The arrival message placed into service aggressively must be invalidated by the receipt another arrival message in order to produce an anti-message. Recall we defined the event *invalidateD* as the event that this first arrival message is invalidated by the receipt of a second (or third) arrival message. The probability of this event is

$$P(invalidateD) \approx P(Arr=2)\ 0.5 + P(Arr=3)\ 0.75. \tag{A.16}$$

The first term in this expression is the probability of receiving two arrival messages, where the second arrival message invalidates the first. Again we note that this is an approximation since we do not consider the probability of receiving more than three arrival messages.

The final term in Equation (A.1.B) is p_1, the probability that the activity stays in the system upon its completion.

We have now derived the probabilities for each event required to produce a first generation anti-message. In the next section we derive the probability of producing an *Nth* generation anti-message.

A.4. Nth Generation Anti-Messages

As noted in previous chapters, one of the primary problems with a fully aggressive system is the possibility of cascading rollbacks. A cascading rollback develops when one anti-message leads to other

anti-messages, and the error propagates without bound. As we discussed in Chapter 4, the probability of cascading rollbacks developing with the limited aggressiveness we allow is negligible. In this section we describe the events necessary for one anti-message to cause another anti-message, and the probabilities associated with these events.

As will be seen, the computation of the probability of an *Nth* generation anti-message becomes quite complex. The reason for this is as follows. Recall that the final, unconditioned expression for the probability of a first generation anti-message has four terms (Equation (A.1)). For each of these four terms, we calculate the probability that the LP receives an arrival message with a timestamp less than the logical time the activity is placed into service aggressively. As we have seen, this is complicated by the fact that there are three logical times at which an activity can be entered into service aggressively (i.e. logical time T_{CS}, logical time $X \mid X > T_{CS}$, and logical time ϕ).

This process becomes much more complicated as we compute the probability of a *2nd* generation anti-message because the timestamp distribution of a first generation anti-message will be conditioned on the particular set of circumstances under which the anti-message was produced. For example, if the activity enters into service aggressively at logical time T_{cs} (the timestamp of the "complete_service" message), and this activity is later invalidated, the timestamp distribution of the message that must be cancelled (and thus the timestamp distribution of the anti-message sent to cancel this message) will be $Y = T_{cs} + \xi$. If, on the other hand, the activity entered into service at logical time ϕ, then the message that must be cancelled will have timestamp $Z = \phi + \xi$. There is also the third case to consider, where the invalidated activity has a timestamp X that is a conditional exponential, conditioned on being between T_{cs} and A. In this case the anti-message will have timestamp $D = X + \xi$. We discuss this issue more fully below.

Thus the timestamp of a first generation anti-message will have a *pdf* conditioned on the particular set of circumstances under which the anti-message was generated. Given this, we must then compute the probability that the anti-message will have a timestamp less than the logical time an activity was placed into service aggressively. As there are three different logical times an activity can be placed into service aggressively, this requires nine different computations (three possible times an activity can be placed into

service aggressively, for each possible logical time three components of the conditional *pdf* of a first generation anti-message). These computations are complicated by the fact that many of the distributions cannot be expressed in terms of elementary functions. This is further complicated by the fact that we cannot prove that the timestamp distribution of an anti-message does not change from generation to generation, though we believe it does not.

For these reasons we take the following approach to the analysis. First, we discuss the steps required to compute the probability of producing a second generation anti-message. This allows us to develop an exact prediction of this probability, and demonstrates again how quickly the level of complexity grows as we model a more complex system. We then give an approximation that bounds (from above) the probability of producing an anti-message. With this approximation, we are able to develop a recurrence relation describing the probability of an *Nth* generation anti-message. As discussed in section 5.5, this approximation has a minimal impact on the predictive power of the model. We begin by describing the technique to compute the probability of a second generation anti-message.

Recall that a first generation anti-message is produced when an activity placed into service aggressively is invalidated by the receipt of an arrival message. An *Nth* generation anti-message is produced when an activity is invalidated by the receipt of an $Nth - 1$ generation anti-message. Consider the probability of producing a *2nd* generation anti-message. Again the LP must place an activity into service aggressively, the activity must complete within the aggressive window, the activity must stay in the system after its completion and the activity must be invalidated by the receipt of another message. Therefore, all of the events necessary to produce a first generation anti-message must occur, except that the activity must be invalidated by the receipt of a first generation anti-message rather than by the receipt of an arrival message. In the previous section we determined the probabilities associated with each event necessary to produce a second generation anti-message except for the probability that an anti-message invalidates the activity chosen for service. The calculation of this probability requires the timestamp distribution of a first generation anti-message which we now derive.

As mentioned above, the timestamp distribution of a first generation anti-message is dependent upon the particular set of circumstances under which the anti-message was generated. Recall there are

four sets of circumstances under which an anti-message can be produced. For the purposes of our analysis, it is convenient to define four different classes of anti-messages, where the distinction between the classes is the set of circumstances under which an anti-message is produced. More specifically, we classify anti-messages on the basis of the logical time the invalidated activity entered into service aggressively. We now review the four sets of events which can produce an anti-message, and the timestamp distribution of the anti-message produced given each particular set of events.

One set of events which can lead to an anti-message is when the server is busy at logical time L (the lower bound of the aggressive window), the activity currently receiving service completes at logical time T_{CS} (within the aggressive window), there is an activity to place into service at logical time T_{CS}, the activity stays in the system upon its completion and the activity entered into service at logical time T_{CS} is invalidated by an arrival message. We classify an anti-message produced due to this sequence of events a *CSanti*. As shown, the probability of these events occurring, and thus the probability of producing an anti-message that we classify as a *CSanti* is

$$P(CSanti) = \rho\, P(C)\, P(Act\,1)\, P(2nd)\, P(invalidate\,1)\, p_1. \tag{A.17}$$

Consider the timestamp distribution of a *CSanti*. As noted, the invalidated activity entered into service at logical time T_{CS} and completed service within the aggressive window. Thus the completion time of the activity (and therefore the timestamp of the "pre-sent" completion message), is a conditional exponential, conditioned on being between T_{cs} and A. Thus the timestamp of the anti-message sent to cancel this "pre-sent" completion message will (by definition) have this same timestamp distribution. Let $Y = T_{CS} + \xi$ be the timestamp of a *CSanti*. Below we give the *pdf* of Y, given a particular timestamp of the "complete_service" message $T_{CS} = t$.

$$pdf\,(y \mid CSanti, T_{cs}=t) = \frac{e^{-\lambda y}}{e^{-\lambda t} - e^{-\lambda A}} \tag{A.18}$$

Note that by $pdf\,(y \mid CSanti, T_{cs} = t)$ we mean the timestamp distribution of an anti-message, given that the anti-message is classified as a *CSanti*, and given that the invalidated activity entered into service at logical time $T_{CS}=t$. This *pdf* is conditioned on a particular value of $T_{cs} = t$, and the expression therefore needs to be unconditioned. We do this in the usual way by integrating over all values of $T_{cs} = t$ times the probability of $T_{cs} = t$. Recalling that the timestamp of the "complete_service" message is a conditional

exponential, conditioned on being within the aggressive window, we give the unconditioned expression below.

$$pdf\,(y \mid CS\_Anti) = \int_0^A \frac{\lambda e^{-\lambda y}}{e^{-\lambda t} - e^{-\lambda A}} \; \frac{e^{-\lambda t}}{1 - e^{-\lambda A}} dt \tag{A.19}$$

This is a complex integral with no closed form solution of which we are aware. We leave it in integral form and note that it must be solved numerically.

Recall that an anti-message can also be produced under the following circumstances. The server is busy at logical time T, the activity receiving service completes at logical time T_{CS} (within the aggressive window), there is no activity to enter into service immediately, there exists an activity with timestamp $X = x \mid X > T_{CS}$ that is placed into service at logical time x, this activity completes within the aggressive window, stays in the system upon its completion and is invalidated by the receipt of an arrival message. We classify an anti-message produced under this set of circumstances a *Banti*. As shown, the probability of producing an anti-message which we classify as a *Banti* is

$$P\,(Banti) = \rho\, P\,(C)\, P\,(Act2)\, P\,(C\_2nd)\, P\,(invalidate2)\, p_1. \tag{A.20}$$

Now consider the timestamp distribution of a *Banti* anti-message.

The invalidated activity entered into service at logical time $X \mid T_{CS} < X < A$, and completed service at logical time $Z = X + \xi$. By definition, the anti-message sent to cancel the "pre-sent" completion message will have this same timestamp distribution. The *pdf* of Z is

$$pdf\,(z \mid Banti, X=x) = \frac{\lambda e^{-\lambda z}}{e^{-\lambda x} - e^{-\lambda A}} \tag{A.21}$$

Note that this *pdf* is conditioned on $X = x$. In order to uncondition we integrate over all values of $X=x$ times the probability of $X=x$. This is complicated by the fact that X is conditioned on being between $T_{CS} = t$ and A, and T_{CS} is a conditional exponential, conditioned on being within the aggressive window. We uncondition this expression below.

$$pdf\,(y \mid B\_Anti) = \int_0^A \int_t^A \frac{\lambda e^{-\lambda z}}{e^{-\lambda x} - e^{-\lambda A}} \; \frac{\lambda e^{-\lambda x}}{e^{-\lambda t} - e^{-\lambda A}} \; \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda A}} dx\, dt. \tag{A.22}$$

This complicated integral has no closed form solution of which we are aware.

Recall that an anti-message can also be produced when the invalidated activity entered into service at logical time ϕ, where ϕ is the minimum timestamp among the $N \geq 1$ activities within the aggressive window. As shown above, this occurs when the server is free as the LP begins to process aggressively, there are no activities on the server queue, there is an activity within the aggressive window that enters into service at logical time ϕ, the activity completes within the aggressive window, the activity stays in the system upon its completion and the activity is invalidated by the receipt of an arrival message. We classify an anti-message produced under these circumstances an *Nanti*. As we have shown, the probability of generating an anti-message which we classify as an *Nanti* is

$$P(Nanti) = (1-\rho)\, P(Actw)\, P(S\_2nd)\, P(invalidate\,3)\, p_1. \tag{A.23}$$

Note that the completion time of this activity placed into service aggressive, and therefore the timestamp of the anti-message sent to cancel the "pre-sent" completion message, is $D = \phi + \xi$. The *pdf* of D

$$pdf(d \mid Nanti) = \int_{\phi}^{A} \frac{e^{-\lambda d}}{e^{-\lambda\phi} - e^{-\lambda A}} f(\phi)\, d\phi \tag{A.24}$$

Again, this is a complicated integral which must be solved numerically.

The final set of circumstances under which an anti-message can be produced is when the server is idle as the LP begins to process aggressively, there are no activities on the server queue or in the aggressive window, the LP receives an arrival message which it places into service, the activity completes within the aggressive window, the activity remains in the system upon its completion and the activity is invalidated by the receipt of a second (or third) arrival message. We classify an anti-message produced under these circumstances an *Aanti*. As shown, the probability of producing an anti-message under these circumstances is

$$P(Aanti) = (1-\rho)\,(1 - P(Actw))\, P(2nd)\, P(invalidate\,4)\, p_1. \tag{A.25}$$

Recall our assumption that an arrival message and the "complete_service" message have the same timestamp distribution: conditional exponential, conditioned on being within the aggressive window. Thus the timestamp of an anti-message produced under these conditions will have the same distribution as an anti-message produced when the invalidated activity entered into service at logical time T_{cs}. This implies that an anti-message we classify as an *Aanti* has the same timestamp distribution as an anti-message we

classify as a *CSanti*.

We have now discussed the timestamp distribution of each of the four classes of anti-messages. Note that all of the anti-messages discussed thus far are first generation anti-messages. That is, these anti-messages are produced as a result of an activity being invalidated by the receipt of an arrival message (rather than by the receipt of an anti-message). As can be seen, the *pdf* of a first generation anti-message is conditioned on the class to which the anti-message belongs. Let $P(Anti)$ be the probability that an anti-message is produced. As noted, the probability of an anti-message is the probability that an LP produces a *CSanti*, plus the probability that it produces a *Banti*, plus the probability it produces an *Nanti* plus the probability it produces an *Aanti*. The probability that an anti-message belongs to a particular class, and thus the probability that the anti-message has the timestamp distribution of that particular class, is the proportional probability of that class of anti-messages to the total probability of an anti-message. Let S be the timestamp of a first generation anti-message. We give the *pdf* of S below.

$$pdf(s) = \int_0^A \frac{\lambda e^{-\lambda s}}{e^{-\lambda t} - e^{-\lambda A}} \frac{e^{-\lambda t}}{1 - e^{-\lambda A}} dt \; \frac{P(CSanti) + P(Aanti)}{P(Anti)} + \tag{A.26}$$

$$\int_0^A \int_t^A \frac{\lambda e^{-\lambda s}}{e^{-\lambda x} - e^{-\lambda A}} \frac{e^{-\lambda x}}{e^{-\lambda t} - e^{-\lambda A}} \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda A}} dx \, dt \; \frac{P(Banti)}{P(Anti)} +$$

$$\int_\phi^A \frac{\lambda e^{-\lambda s}}{e^{-\lambda \phi} - e^{-\lambda A}} f(\phi) \, d\phi \; \frac{P(Nanti)}{P(Anti)}.$$

Given the *pdf* of a first generation anti-message, we can compute the probability of producing a *2nd* generation anti-message. A second generation anti-message is produced when an activity enters into service aggressively, completes within the aggressive window, stays in the system upon its completion and the activity is invalidated by the receipt of a first generation anti-message. Thus all of the events necessary to produce a first generation anti-message must occur, except that the activity placed into service aggressively is invalidated by the receipt of a first generation anti-message rather than by an arrival message. Consider the final equation for the probability of producing a first generation anti-message.

$$P(Anti) = \rho P(C) P(Act\,1) P(2nd) P(invalidateA) p_1$$

$$+ \rho P(C) P(Act\,2) P(C\_2nd) P(invalidateB) p_1$$

$$+ (1-\rho)\, P(Actw)\, P(S\_2nd)\, P(invalidateC)\, p_1$$

$$+ (1-\rho)\, (1 - P(Actw))\, P(2nd)\, P(invalidateD)\, p_1$$

We need to replace the events *invalidateA* through *invalidate_D* with new events which represent the probability that the activity placed into service aggressively is invalidated by a first generation anti-message.

Consider the first term in the above equation. This term represents the probability of producing a first generation anti-message given that the activity enters into service aggressively at logical time T_{cs}. We need to compute the probability that a first generation anti-message invalidates such an activity. Define *invalA* as the event that a first generation anti-message has a timestamp less than logical time T_{cs}. Let *AM* be the random variable associated with the timestamp of a first generation anti-message. Recall the probability this anti-message has a timestamp distribution $Y = T_{CS} + \xi$ $(P(CSanti) + P(Aanti))/P(Anti)$. Also we have shown that random variable *AM* is $Z = X + \xi$ where $X \mid T_{cs} < X < A$ with probability $\frac{P(Banti)}{P(Anti)}$. Finally, random variable *AM* is $D = \phi + \xi$, where ϕ is the minimum timestamp among $N \geq 1$ activities within the aggressive window, with probability $\frac{P(Nanti)}{P(Anti)}$.

To compute the probability of event *invalA*, we compute the probability that *AM* is less than T_{cs}, given each of the three timestamp distributions for *AM*. This is unconditioned by multiplying the probability of an invalidation given a particular timestamp distribution of *AM*, times the probability that *AM* has that timestamp distribution. Thus three separate calculations are required. We note that this computation is performed in a manner analogous to other such computations given in previous sections.

The probability that a first generation anti-message invalidates an activity given that the activity enters into service at logical time $X \mid T_{cs} < X < A$ is computed in a similar manner. We define event *invalB* as the event that a first generation anti-message invalidates such an activity. We define *invalC* as the event that a first generation anti-message invalidates an activity that enters into service at logical time ϕ. Again we do not elaborate on the techniques to compute these probabilities.

Given the probabilities for events *invalA*, *invalB* and *invalC*, we give the final expression for the probability of producing a second generation anti-message below.

$$P(Anti_2) = P(Anti_1)\,\rho\,P(C)\,P(Act1)\,P(2nd)\,P(invalA)\,p_1 \quad (A.27)$$
$$+ P(Anti_1)\,\rho\,P(C)\,P(Act2)\,P(C\_2nd)\,P(invalB)\,p_1$$
$$+ P(Anti_1)\,(1-\rho)\,P(Actw)\,P(S\_2nd)\,P(invalC)\,p_1$$
$$+ P(Anti_1)\,(1-\rho)\,(1-P(Actw))\,P(2nd)\,P(invalA)\,p_1$$

The difficult question is: what is the timestamp distribution of a second generation anti-message? Does this distribution change from generation to generation? Note that the three different types of density functions will not change from generation to generation. That is, the timestamp of an *Nth* generation anti-message will be either $Y = T_{cs} + \xi$ (conditioned on being within the aggressive window), $Z = X + \xi$ (where $T_{cs} < X < A$ and again conditioned on being within the aggressive window), or $D = \phi + \xi$ where ϕ is the minimum timestamp of all activities within the aggressive window (and $\phi + \xi < A$). This will not change because an activity can only enter into service aggressively at logical time T_{cs}, logical time X or logical time ϕ.

What can change is the probability that the timestamp of an anti-message will be a particular one of the three density functions. As shown above, the probability that a first generation anti-message has a *pdf* that is a conditional exponential, conditioned on being between T_{cs} and A is $\frac{P(CSanti) + P(Aanti)}{P(Anti)}$. The question is, what is the probability that the *pdf* of a second generation has this same timestamp distribution? If this probability is $\frac{P(CSanti) + P(Aanti)}{P(Anti)}$ (and the probabilities associated with the other density functions also remain the same), then we can develop a recurrence relation to compute the probability of an *Nth* generation anti-message. If this proportional probability changes from generation to generation, then a separate computation must be performed for each generation anti-message.

Because we cannot answer this question, we propose the following approximation.

Recall that an activity can enter into service aggressively at logical time T_{cs}, logical time $X \mid T_{cs} < X < A$, or logical time ϕ, where ϕ is the minimum among the $N \geq 1$ activities within the aggressive window. As we have discussed, the timestamp of an anti-message produced as a result of the invalidation of this activity will be the sum of the logical time the activity entered into service and ξ. The smaller the expected value of the timestamp of the anti-message, the more likely it will be that the anti-

message will invalidate another activity (and therefore the more likely it is to propagate the error through the system).

For this reason we assume the timestamp of the anti-message is $D = \phi + \xi$. Note this represents the worst case assumption regarding the logical time of the activity placed into service aggressively.

Define *TSInvalA* as the probability that D is less than T_{cs}. Again we do not elaborate on the computation of this probability. Define *TSInvalB* as the probability that D is less than $X \mid T_{cs} < X < A$. Define *TSInvalC* as the probability that D is less than ϕ. Given these probabilities, and the approximation for the timestamp distribution of an anti-message, we can define a recurrence relation to predict the probability of an *Nth* generation anti-message.

$$
\begin{aligned}
P(Anti_N) = {} & P(Anti_{N-1})\, \rho\, P(C)\, P(Act\,1)\, P(2nd)\, P(TSInvalA)\, p_1 \\
& + P(Anti_{N-1})\, \rho\, P(C)\, P(Act\,2)\, P(C\_2nd)\, P(TSInvalB)\, p_1 \\
& + P(Anti_{N-1})\, (1-\rho)\, P(Actw)\, P(S\_2nd)\, P(TSInvalC)\, p_1 \\
& + P(Anti_{N-1})\, (1-\rho)\, (1 - P(Actw))\, P(2nd)\, P(TSInvalA)\, p_1
\end{aligned}
\tag{A.28}
$$

Now we have shown how to calculate the probability of an *Nth* generation anti-message. In Chapter 5 we gave our predicted and observed probabilities of a first generation anti-message. Also, we gave some predicted and observed probabilities of a second generation anti-message. This concludes our discussion of the probability of producing an anti-message in an open system.