Synchronization and distributed agreement in real-time systems
Ramanathan, Parameswaran
ProQuest Dissertations and Theses; 1989; ProQuest

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Order Number 9013992

Synchronization and distributed agreement in real time systems

Ramanathan, Parameswaran, Ph.D.

The University of Michigan, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SYNCHRONIZATION AND DISTRIBUTED AGREEMENT IN REAL-TIME SYSTEMS

by

Parameswaran Ramanathan

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1989

Doctoral Committee:

Professor Kang G. Shin, Chairman
Assistant Professor Chaitanya K. Baru
Associate Professor John R. Birge
Assistant Professor Richard B. Brown
Professor John P. Hayes

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

RULES REGARDING THE USE OF

MICROFILMED DISSERTATIONS

Microfilmed or bound copies of doctoral dissertations submitted to The University of Michigan and made available through University Microfilms International or The University of Michigan are open for inspection, but they are to be used only with due regard for the rights of the author. Extensive copying of the dissertation or publication of material in excess of standard copyright limits, whether or not the dissertation has been copyrighted, must have been approved by the author as well as by the Dean of the Graduate School. Proper credit must be given to the author if any material from the dissertation is used in subsequent written or published work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

© Parameswaran Ramanathan 1989
All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my parents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENTS

It is an impossible undertaking to thank all those who have contributed to the success of this dissertation. I would, however, like to take this opportunity to thank a few who have influenced me the most.

First and foremost, I would like to thank my advisor, Professor Kang G. Shin. Professor Shin's constant support helped me wade through the frustrations and the triumphs of a graduate program. Through his guidance I have gained invaluable expertise on approaching, analyzing and solving problems that are presented to me. I would also like to express my appreciation to other members of the doctoral committee, Professors Chaitanya Baru, John Birge, Richard Brown and John Hayes for their constructive criticisms on this dissertation. Thanks also goes to the National Aeronautics and Space Administration, Office of Naval Research, and Horace H. Rackham School of Graduate Studies for providing the funds necessary for this research.

Heartfelt thanks to James Dolter and Dilip Kandlur for all the daily discussions that helped me formulate the ideas in this dissertation. This work could not have been accomplished without their constant encouragement. I would also like to acknowledge Dilip Kandlur for his assistance in formulating some of the ideas in software synchronization that are presented in Chapter 2. Thanks also to Alan Olson and Daniel Kiskis for the helpful comments on the early versions of this manuscript.

Finally, I would like to thank my parents for their moral support during this effort. Their contribution to this dissertation cannot be measured.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

Table

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Figure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

INTRODUCTION

1.1 Motivation

The potential for applying digital computers to real-time control is enormous. It is just beginning to be recognized that computers have a vital role in controlling many critical processes. This is especially true in applications such as the control of aircraft, spacecraft, life-support systems, nuclear power plants, automobile engines, and process control plants, where the time available for responding to the environment is always becoming shorter.

The control computers in these applications are required to be fast and highly reliable. These requirements are stringent because a single failure in the control system can be catastrophic. Fault tolerance is therefore a major issue in the design of these control computers. The exact nature of fault tolerance that needs to be incorporated depends on the application. Some applications require very high reliability for a short period of time while others require moderate reliability over a long period. For example, in a commercial transport aircraft the duration of a flight is around 10 hours and the allowable probability of failure per mission is specified as 10^{-9}. In contrast, in a manned space flight, the duration of a mission is relatively long, around 10 days, while the allowable probability of failure per mission is higher by about two orders of magnitude. Besides, in a commercial transport aircraft there is no facility for repair during a mission as compared to some facility for repair in a manned space flight. Several such factors along with the potential loss in case of a failure determine the fault tolerance capabilities that needs to be incorporated in a system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Given the fault tolerance requirements, it is still very difficult to show that the designed system will actually meet those requirements. Since the cost incurred upon failure is extremely high, the control computers for these applications are often designed under the worst-case assumption that the faulty components can do whatever they like. This includes altering, blocking, re-routing or in the worst case, sending conflicting information to different parts of the system. A component that exhibits this type of behavior is commonly referred to as being *malicious*, and the fault model is referred to as the *Byzantine fault model.*

As one can easily imagine, Byzantine faults pose serious problems in all three major activities of a real-time control system, namely, acquisition of data from the input sensors, the processing of acquired data and writing of output data to the actuators. For example, it is often necessary for the components in a real-time system to reach a consensus on the value read from an input sensor. In the absence of Byzantine faults, this consensus can be easily achieved by using some form of redundancy. For instance, multiple components can independently read and exchange the value of a sensor with other components, and then use an appropriate fault-tolerant "averaging" function to arrive at a mutually agreeable value. This problem however gets complicated when malicious components report conflicting values to different parts of the system, thus causing them to arrive at contradictory results.

Several techniques have been put forth for ensuring consensus among the non-faulty components in the presence of Byzantine faults [7, 9, 11, 12, 13, 37, 66]. Most of these solutions require an excessive amount of time to execute and are, therefore, unsuitable for real-time applications. The objectives of this dissertation are to address ways of eliminating/reducing the time overhead imposed by these solutions and simplifying the design of computer systems that are resilient to Byzantine faults. The emphasis is on developing fault-tolerant methods for synchronizing system components and ensuring distributed agreement among cooperating processes.

This work is significant because almost all time and mission critical applications use some form of distributed processing to meet their reliability and performance requirements. In these systems, the coordination among concurrent processes is by exchanging information. The information may in general be input data, output data, clock values, system status, or any other data relevant to the system. *Agreement* by definition is a consensus among the participating components on any such information and hence, important in all distributed systems. Likewise,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

synchronization is important in all distributed systems because: (i) it is essential for ensuring distributed agreement, (ii) it can simplify fault-tolerant algorithms for several design problems including inter-process communication, checkpointing and rollback recovery, resource allocation, and transaction processing and (iii) it can be used to implement features like deadlines and timeout that are essential for correct operation of any distributed real-time system.

1.2 Prior Work

Techniques for designing computer systems that can tolerate Byzantine faults are well-understood. The following four conditions have been shown to be necessary for any algorithm that is resilient to m Byzantine faults. This includes algorithms for synchronization as well as distributed agreement.

1. There must be at least $3m + 1$ participants in the algorithm [44].
2. Each participant must be connected to at least $2m + 1$ other participants through disjoint communication paths [7].
3. The protocol must consist of a minimum of $m + 1$ rounds of communication among the participants [12].
4. The participants must be synchronized to within known skews of each other [8, 16].

Based on these theoretical developments several ultra-reliable computer systems have been built over the past decade. This includes systems such as the Fault-Tolerant MultiProcessor (FTMP) [25, 62], Software Implemented Fault Tolerance (SIFT) [20], Fault Tolerant Processor (FTP) [61], Multicomputer Architecture for Fault Tolerance (MAFT) [30], and Advanced Information Processing System (AIPS) [26]. A comparison between these systems is shown in Table 1.1.

As indicated in the table, SIFT is a six processor system with direct connection between all pairs of processors. It uses a software synchronization scheme to maintain the clocks to within known skews of each other. In comparison, FTMP is a ten processor system organized in the form of three triads and a spare. The communication between the triads is through a 5-way redundant

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

System	# of proc.	Topology	Sync. algorithm	Multiprocessing
SIFT	6	Fully connected	Software	Yes
FTMP	10	Broadcast bus	Hardware	Yes
FTP	≤ 4	See note 1	Hardware	No
MAFT	8	Broadcast bus	Software	Yes
AIPS	See note 2	Broadcast bus	See note 3	Yes

Note 1: Special data exchange network.

Note 2: Limited by contention for the broadcast bus.

Note 3: Still being designed.

Table 1.1: Comparison of a few ultra-reliable systems

broadcast bus with three active and two spares. Unlike SIFT, it uses a hardware synchronization scheme to maintain the clocks close to one another.

In direct contrast to FTMP and SIFT, FTP is a much simpler design that can execute only one instruction stream at any given time (the redundant processors are solely for fault tolerance). However, like FTMP it uses a hardware scheme to tightly synchronize its clocks. On the other hand, MAFT is a distributed system comprised of eight semi-autonomous nodes connected through a broadcast bus network. It uses a software scheme similar to that of SIFT to maintain a loose synchronization between its clocks. AIPS is a distributed system comprised of fault tolerant multiprocessors interconnected through a redundant virtual bus. The number of fault tolerant multiprocessors in AIPS is limited by the contention for the virtual bus.

At the outset, these systems may seem to be widely varied from each other. However, there are several common features among them that are typical to the existing algorithms for tolerating Byzantine faults. First, all of the above systems except AIPS are small and tightly coupled multiprocessors. Second, they can tolerate a maximum of one Byzantine fault at any given time. Third, the schemes used in these systems cannot often be extended to a multiple fault scenario,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

e.g, the hardware synchronization algorithm used in FTMP and FTP. Finally, even though all the above systems have extensive capabilities for reconfiguration, there exists no fault detection algorithm that is guaranteed to always identify the malicious components.

In contrast, the goal of this dissertation is to consider systems that are large and loosely-coupled. The systems are also assumed to be partially connected through a point-to-point interconnection network as opposed to a broadcast bus or a fully connected network. The reliability requirements of the systems are also assumed to be such that it may be necessary to tolerate more than one Byzantine fault. This work is significant because large partially connected distributed systems are well-suited for meeting the stringent performance and reliability requirements of a critical real-time application.

1.3 Research Objectives

The objectives of this dissertation are three-fold:

- develop algorithms for establishing a global time base in the system,
- illustrate the use of global time base to simplify other fault-tolerant algorithms, and
- reduce/eliminate the overheads for synchronization and distributed agreement through diagnosis of Byzantine faults.

The algorithm for establishing a global time base can be considered at two different levels: *system level* and *node level*. System level synchronization deals with establishing a time base among the nodes of a distributed system. This problem gets complicated when some of these nodes behave maliciously by providing conflicting clock values to different parts of the system. Additional problems arise in large distributed systems because the existing algorithms are not easily scalable. In contrast, node level synchronization deals with establishing a global time base among the components in a node by properly distributing a clock signal to all the components.

There are both software and hardware solutions for system level synchronization. The software solutions require nodes to exchange and adjust their clock values periodically. Since the clock values are exchanged via message passing, the time overhead imposed by the software solutions can be substantial, especially if a tight synchronization is desired. The hardware solutions, on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the other hand, use special hardware at each node to achieve a very tight synchronization with minimal overhead. However, the prohibitive cost of the additional hardware limits their usefulness to small distributed systems. Since the choice between the two solutions for a system depends on the application, both these approaches are considered in this dissertation. The emphasis is on reducing the cost of hardware solutions and the worst-case skews of software solutions to make both these alternatives viable even in large distributed systems.

In node level synchronization, the goal is to control the clock skew between the modules in a VLSI circuit to an acceptable small fraction of the clock period. Within a VLSI circuit the clock skews are mainly due to the difference in length of clock lines and due to the difference in the number of clock buffers inserted in the clock lines. The existing schemes for clock distribution are either applicable only in the case of a symmetric placement of the modules such as systolic arrays or they are too detailed and sensitive to the technology and the fabrication process. The emphasis here is on developing a general scheme that is applicable in non-symmetric VLSI circuits. The objective is to determine a layout of clock lines given the floorplan that minimizes the clock skew subject to minimum longest delay.

The use of a global time base to simplify fault-tolerant algorithms is illustrated by considering the checkpointing and rollback recovery schemes. These schemes are mainly used to recover from software errors when cooperating processes are executing concurrently in a distributed system. It is shown that the number of messages as well as the waiting time for checkpointing and rollback recovery can be reduced substantially by using the global time base. The only additional cost in this scheme as compared to other known schemes is the cost of establishing the time base.

The cost of synchronization, and hence the cost of establishing a global time base, depends on the maximum number of Byzantine faults to be tolerated. Given the reliability requirements of a system, the number of faults that needs to be tolerated can be reduced by preventing the accumulation of Byzantine faults by diagnosing them as soon as they occur. This work can be used to reduce the overheads imposed by synchronization algorithms and various other algorithms that are resilient to Byzantine faults. It is therefore presented in a more general context of distributed agreement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.4 Outline of the Dissertation

This dissertation is organized as follows. Chapter 2 deals with the problems related to synchronizing large distributed systems. It discusses solutions to two major problems in the existing hardware synchronization algorithms, namely, the assumption that there is a fully connected network of clocks and the assumption that the transmission delay from one clock to another is negligible. These two assumptions are difficult to satisfy in a large distributed system. The scheme discussed requires only 20–30 percent of the total number of interconnections required by a fully connected network for almost no loss in synchronization capabilities. This is achieved by grouping the clocks into clusters and treating the clusters as clock units as far as the system is concerned. The effects of the transmission delay are reduced by a technique traditionally used in the communication area called the returnable timing system.

Chapter 2 also discusses a hardware-assisted software solution for synchronizing large partially connected distributed systems. As compared to the hardware solution, this scheme requires minimal additional hardware at each node and does not require a separate network of clocks. It is suitable for synchronizing specific interconnection topologies like a hypercube or a C-wrapped hexagonal mesh, especially when a very tight synchronization is not essential. The skews achieved are, however, much tighter than that of the existing software schemes.

The problem of synchronizing different modules in a VLSI circuit is addressed in Chapter 3. The hierarchy created by clock buffers is used to distribute the clock signal to the modules. Unlike other related work in this area, both delay and skew are taken into account in determining the layout. The resulting scheme can be easily parallelized.

A checkpointing and rollback recovery algorithm that is based on the existence of a global time base is presented in Chapter 4. The existence of a global time base is coupled with the idea of pseudo-recovery block approach to develop an algorithm that has the advantages of maximum process autonomy, minimal wait for commitment, fewer messages and less memory requirement. A probabilistic model to evaluate these advantages is also developed.

A novel approach for reducing the overheads imposed by synchronization and distributed agreement algorithms is presented in Chapter 5. The reduction in overhead is achieved through diagnosis of Byzantine faults. The algorithm for diagnosis is such that there is an upper bound

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

on the number of times a node can exhibit its Byzantine behavior without being identified.

Chapter 6 is a discussion on future research directions based on the results presented in this dissertation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

SYSTEM LEVEL SYNCHRONIZATION

2.1 Introduction

A global time base is widely recognized as an important requirement in distributed real-time systems. It is essential for problems like distributed agreement and also for implementing features like deadlines and timeout. It can also be used to simplify solutions to several other problems like checkpointing, inter-process communication, resource allocation, and transaction processing.

The global time base can be established by synchronizing all the local clocks in the system. These synchronizing operations would not be much of a problem had all the clocks, including the faulty ones, behaved consistently with one another. However, when some of the faulty clocks can behave in an arbitrary manner, these synchronizing operations present some serious problems. For instance, a faulty clock can make it difficult for the other clocks to synchronize themselves by sending conflicting information during the course of synchronization. Lamport and Melliar-Smith referred to this kind of behavior as a *Byzantine fault* and were the first to develop a solution for synchronizing clocks in the presence of these faults [35, 36]. Since then, this problem has been studied extensively and several software and hardware solutions have been proposed.

The software solutions are flexible and economical but require additional messages to be exchanged solely for synchronization [22, 36, 43, 64]. Due to the dependence on message exchanges, the worst-case skews guaranteed by most of these solutions are greater than the maximum message transit delay between any two nodes in the system. The hardware solutions, on the other hand, use special hardware at each node to achieve a very tight synchronization with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

minimal time overhead [29, 33, 57, 67]. However, the cost of additional hardware precludes their use in large distributed systems unless a very tight synchronization is essential. The hardware solutions also require a separate network of clocks that is different from the interconnection network between the nodes of the distributed system [57].

The choice between hardware and software synchronization methods has to be made based on the application and the characteristics of the system. The goal of this dissertation is to make both methods viable alternatives for synchronizing large distributed systems. This goal is realized by addressing the specific drawbacks of these two methods, namely the cost of hardware solutions and the guaranteed worst-case skew of software solutions.

This chapter is organized as follows. A survey of related work in both hardware and software synchronization methods is presented in the following section. The terminology that is common to both these methods is introduced in the third section. The drawbacks in hardware and software synchronization methods are addressed in the fourth and the fifth sections. The sixth section discusses the relative merits of the schemes proposed.

2.2 Related Research

The principle of hardware synchronization algorithms is that of a phase-locked loop. Each individual clock is an output of a voltage controlled oscillator. The voltage applied to the oscillator comes from a phase detector whose output is proportional to the phase error between the phase of its clock (i.e., the output of the voltage controlled oscillator it is controlling) and a reference signal generated by using the other clocks in the system. Thus, by adjusting the frequency of each individual clock to the reference signal, the clocks can always be kept in lock-step with respect to one another.

As of today, there have been only two implementations of this approach [19, 62]. Both these implementations use four clocks and can tolerate only one Byzantine fault. An extension of the algorithms in [19, 62] to tolerate more than one of Byzantine fault is non-trivial and is described in [29, 33, 67]. The principle in [29, 33, 67] is the same as in [19, 62], but the selection of the reference signal is more complicated. Each node receives the clocks at other nodes as inputs and orders them as per the time of their arrival. The position of its own clock in this ordered sequence

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is used to correct its clock in the subsequent cycles. In [29], the number of clocks that are faster than its own clock is used to derive the control voltage while in [33] the phase difference between a chosen reference signal and its own clock is used as the control voltage. In [67], two reference signals are selected from the ordered sequence and a combination of the phase difference between its clock and the two reference signals is used as the control voltage.

There are several problems associated with synchronizing large distributed systems using the algorithms in [29, 33, 67]. The first problem is that all of the above algorithms require a fully connected network of clocks. Due to the large number of interconnections in a fully connected network, the reliability of synchronization will be determined by the failure rates of these interconnections rather than the failure rate of the clocks. Furthermore, there will be problems of fan-in and fan-out caused by the large number of interconnections. The second problem with the above algorithms is that they are based on the assumption that the transmission delays between the clocks are negligible as compared to desired worst-case skew. In a large system, the physical separation between a pair of clocks can be considerable enough to result in non-negligible transmission delays.

Unlike the hardware solutions, software solutions synchronize logical clocks instead of hardware clocks. Given that the hardware clocks drift at a bounded rate from real time, the software solutions periodically exchange and adjust the logical clocks to maintain them sufficiently close to one another [22, 36, 43, 64]. A software solution can be viewed as a process that runs on each node to maintain a time base for all the activities on the node. This clock process is responsible for synchronizing the local logical clock with the other logical clocks in the system.

In [22, 64], the clock process broadcasts a message to other clock processes to indicate that it is ready for resynchronization. This occurs either when its clock reaches a pre-determined time or when it sufficient number of messages from other processes to be sure that at least one other non-faulty process is ready for a resynchronization. During a resynchronization, a new logical clock is started in such a way that it is guaranteed to be close to the clocks started approximately at the same time at other nodes. In [36, 43], a clock process broadcasts its logical clock at a pre-determined time. It then waits for a fixed amount of time to collect similar messages from other processes. At the end of the waiting period, the clock process averages the arrival times of the messages from other clock processes using a fault-tolerant averaging function. The resulting

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

average is used to compute the adjustment to the local clock.

Because of the dependence of these software schemes on message passing, the guaranteed worst-case skew between the logical clocks is greater than the maximum message transit delay in the system [22, 36, 43, 64]. This is acceptable in a small fully connected system because the maximum message transit delays in such systems are typically quite small. However, in large distributed systems where the nodes are not fully connected, the clock messages may have to be relayed through several intermediate processes before they reach all other processes. This results in large message transit delays which in turn results in poor worst-case skews.

There are some software schemes in which this problem does not occur [2, 15]. In [15], the clocks are intended for simulating the concept of rounds for distributed agreement algorithms rather than to keep track of elapsed time intervals. Hence, their definition of a clock is different from the definition in the other schemes [22, 36, 43, 64]. The definition of a clock in [15] is not sufficient for real-time systems, since it cannot be used to implement important features like deadlines and timeout. The idea in [2] is to assume that the probability distribution of message transit delays is known and let each clock process make several attempts to read the other clocks. At the end of each attempt, it can calculate the maximum error that might occur if the clock value obtained in that attempt is used for determining the correction. By retrying a sufficient number of times, a clock process can read the other clocks to any given precision with probability as close to one as desired. This scheme is particularly suitable for systems that have a master-slave arrangement in which one clock has been designated or elected as a master and the other clocks act as slaves. This again is not suitable for a real-time system because the algorithms to detect a failure of the master and to elect a new one are fairly complex and time-consuming, especially in the presence of Byzantine faults. There is also a non-zero probability of not being able to synchronize the non-faulty clocks due to the probabilistic nature of the clock reading mechanism.

As a consequence, the usefulness of the existing hardware and software synchronization schemes is limited to small distributed systems. An extension of these schemes to large distributed systems is presented below. First, solutions are proposed to overcome the transmission delay and the interconnection problems in hardware synchronization. Then, a hardware-assisted software synchronization scheme is proposed that achieves skews that are about two to three orders of magnitude tighter than the skews in the existing software schemes. This can be used in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

systems that cannot afford the cost of additional hardware required in pure hardware solutions. Prior to describing these solutions some of the notations and the terminology are introduced.

2.3 Terminology

Definition 2.1: The time that is directly observable in some particular clock is called its *clock time*. This should be contrasted to the term *real time*, which is measured in an assumed Newtonian time frame that is not directly observable.

Definition 2.2: Let c be a mapping from clock time to real time, where $c(T) = t$ means that at clock time T the real time is t. Then, two clocks c_1 and c_2 are said to be *δ-synchronized* at a clock time T if and only if $|c_1(T) - c_2(T)| \leq \delta$. We adopt the convention of using lowercase letters to denote quantities that represent real time and uppercase letters to denote quantities that represent clock time.

Definition 2.3: A clock c is said to be a *good clock* during the real-time interval $[t_1, t_2]$ if it is a monotonic, differentiable function on $[T_1, T_2]$ where $c(T_i) = t_i$, $i = 1, 2$, and for all T in $[T_1, T_2]$:

$$\left| \frac{dc(T)}{dT} - 1 \right| < \frac{\rho}{2}$$

for some constant ρ. The constant ρ represents the drift rate of the good clocks.

Definition 2.4: A set of clocks are said to be *well-synchronized* if and only if any two non-faulty clocks in this set are δ-synchronized for some specified constant δ.

Definition 2.5: A well-synchronized set of clocks has a *re-synchronization interval.* In hardware (software) synchronization, the k^{th} re-synchronization interval is the time duration between the k^{th} and the $(k+1)^{th}$ tick (adjustment) of the fastest non-faulty clock. Note that the fastest non-faulty clock at the k^{th} tick (adjustment) can be different from the fastest non-faulty clock at the $(k+1)^{th}$ tick (adjustment). In hardware synchronization, the k^{th} re-synchronization interval is also called its k^{th} *global clock cycle.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

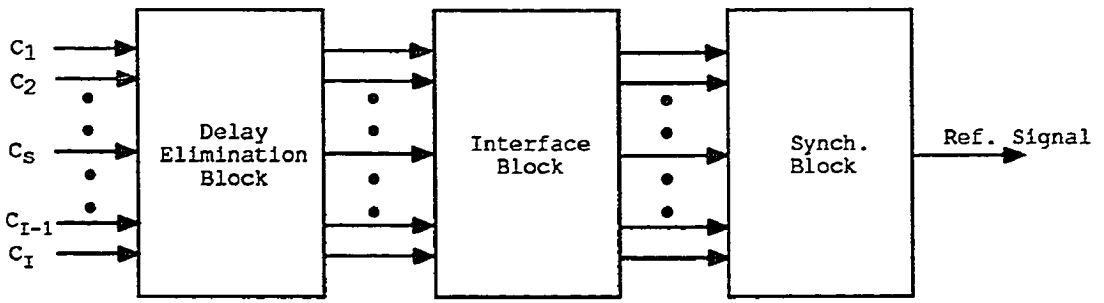

Figure 2.1: Block diagram of the hardware synchronization scheme

2.4 A Hardware Synchronization Scheme

Given a distributed system with N nodes, each of which has a clock of its own, the problem is to synchronize all the non-faulty clocks in the system to a specified fault tolerance m. The phase-locked algorithms in [29, 33, 67] solve this problem, but require the transmission delays to be negligible and a large number of interconnections. The effects of transmission delay in phase-locked loops have been studied in great detail in the communication area [41, 70] but not in the presence of Byzantine faults. It is shown below and in [59] that it is easy to incorporate the ideas from the communication area to take into account the presence of both Byzantine faults and non-negligible transmission delays.

2.4.1 Elimination of transmission delay effects

A block diagram of the hardware clock synchronization scheme is shown in Figure 2.1. It comprises three basic blocks : the *delay elimination* block, the *interface* block and the *synchronization* block. The delay elimination block receives all the other clocks as inputs and outputs analog voltages[1] proportional to the exact phase difference between the input clocks and its own clock. The interface block converts these analog voltages into a form suitable for the synchronization block. The synchronization block can be any one of the existing hardware synchronization circuits [29, 57, 67] and will not be discussed here. The details of other two blocks are discussed

[1] one for each of the other clocks.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in terms of the following notation.

s_i The clock signal from node i.

T_{gcc} The global clock cycle.

d_{ij} Transmission delay for s_j to reach node i

τ_{ij} Phase difference between the clocks at nodes i and j expressed in real time units.

δ Maximum phase difference between any two non-faulty clocks in the system.

e_{ij} Estimated phase difference at node i between the clocks at nodes i and j.

The delay elimination block

For clarity of presentation, consider the delay elimination block at node i. The delay elimination block consists of $I-1$ identical subblocks, i.e., one for each other clock it receives as input. For the interconnection strategy to be put forth later, typically $I \ll N$. Consider the subblock corresponding to clock input from node j (see Figure 2.2). It has two phase detectors and an averager. The inputs to the first phase detector, PD_1, are the clock signals s_i and s_j. Since the signal s_j encounters a delay in reaching node i, the phase difference detected by PD_1 does not represent the true phase difference τ_{ij}.

The inputs to the second phase detector, PD_2, are signals s_j and s_i returning from node j. Because of the transmission delays between the two clocks, the signal s_i returning from node j will be a delayed version of s_i. Since a delay in time is equivalent to a phase difference, the output of the second phase detector will also depend on the transmission delays. However, if the following four conditions hold, then it is shown in Theorem 2.1 that the average of the outputs of these two phase detectors is proportional to τ_{ij} irrespective of the transmission delays.

TD1. The two phase detectors in each subblock have identical gains.

TD2. $d_{ij} \approx d_{ji}$ for all i, j.

TD3. $\delta < \frac{T_{gcc}}{2}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

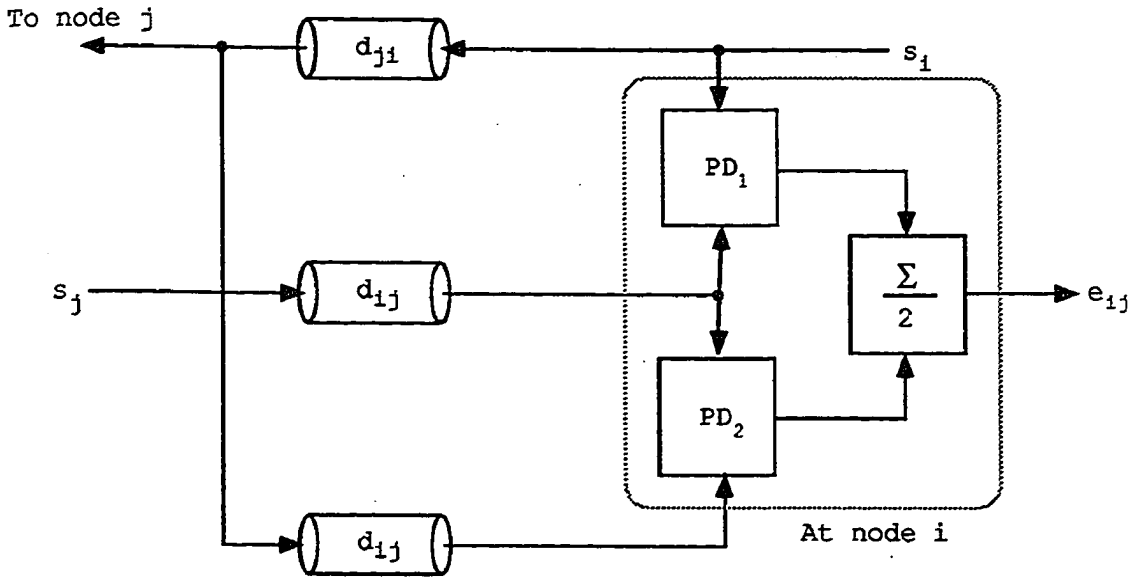

Figure 2.2: Delay Elimination Block

TD4. For all i, j, and some integer n, $nT_{gcc} < d_{ij} < nT_{gcc} + \frac{T_{gcc}}{2} - \delta$.

TD1 is a reasonable assumption because the effect of the difference in gains of phase detectors on the worst-case skew is usually much smaller than the effect of transmission delays. TD2 can be easily satisfied if the signals s_i, s_j and their returning signals are routed via almost identical paths. TD3 holds because hardware synchronization schemes achieve lock step synchronization. In fact, in most hardware synchronization schemes $\delta \ll \frac{T_{gcc}}{2}$. TD4 should be treated as a design constraint. Even though two clocks can be physically far apart from each other, they should not be allowed to be at arbitrary distances. The physical distance between any two clocks should be such that the transmission delays satisfy TD4.

TD4 is not mentioned in [41], because the output of the phase detector is assumed to always vary linearly with phase difference. This, however, is not realistic since phase differences between τ and $\tau + nT$, for any integer n, cannot be distinguished from each other by just considering the two signals, i.e., the outputs of phase detectors have a saw-tooth relationship with phase difference rather than a linear relationship [70].

Lemma 2.1: If integers n_1 and n_2 satisfy the following two conditions:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(1) $\frac{-T_{gcc}}{2} \leq (\tau_{ij} + d_{ji} + n_1 T_{gcc}) < \frac{T_{gcc}}{2}$

(2) $\frac{-T_{gcc}}{2} \leq (\tau_{ij} - d_{ji} + n_2 T_{gcc}) < \frac{T_{gcc}}{2}$

then $n_1 = -n_2$.

Proof: From TD4, there exists an integer n such that $nT_{gcc} < d_{ji} < nT_{gcc} + \frac{T_{gcc}}{2} - \delta$. We now show that $n_1 = -n$ and $n_2 = n$.

By definition, $-\delta \leq \tau_{ij} \leq \delta$. Hence,

$$\begin{aligned} nT_{gcc} - \delta &< \tau_{ij} + d_{ji} < nT_{gcc} + \frac{T_{gcc}}{2} \\ -nT_{gcc} - \frac{T_{gcc}}{2} &< \tau_{ij} - d_{ji} < -nT_{gcc} + \delta. \end{aligned} \tag{2.1}$$

Substituting for $\tau_{ij} + d_{ji}$ and $\tau_{ij} - d_{ji}$ from the above two equations, it is easy to verify that $n_1 = -n$ and $n_2 = n$ satisfy the conditions (1) and (2) (i.e., use TD3). Since integers satisfying (1) and (2) are unique, it follows that $n_1 = -n_2$. ■

Theorem 2.1: If the conditions TD1–TD4 hold, then the output of the averager, $e_{ij} \approx K\tau_{ij}$ for some constant K.

Proof: Let v_{ij}^1 and v_{ij}^2 denote the outputs of the phase detectors PD_1 and PD_2, respectively. Then for a saw-tooth phase detector [70],

$$v_{ij}^1 = K_1 (\tau_{ij} + d_{ij} + n_1 T_{gcc}) \tag{2.2}$$

$$\begin{aligned} v_{ij}^2 &= K_2 \{\tau_{ij} + d_{ij} - (d_{ij} + d_{ji}) + n_2 T_{gcc}\} \\ &= K_2 (\tau_{ij} - d_{ji} + n_2 T_{gcc}), \end{aligned} \tag{2.3}$$

where K_1 and K_2 are constants and n_1 and n_2 are integers so chosen that

$$\begin{aligned} -\frac{T_{gcc}}{2} &\leq \tau_{ij} + d_{ji} + n_1 T_{gcc} < \frac{T_{gcc}}{2} \\ -\frac{T_{gcc}}{2} &\leq \tau_{ij} - d_{ji} + n_2 T_{gcc} < \frac{T_{gcc}}{2}, \end{aligned}$$

respectively.

From Equations (2.2) and (2.3), the output of the averager is

$$e_{ij} = \frac{1}{2} [K_1(\tau_{ij} + d_{ij} + n_1 T_{gcc}) + K_2(\tau_{ij} - d_{ji} + n_2 T_{gcc})]. \tag{2.4}$$

From TD1, $K_1 = K_2 = K$. From Lemma 2.1, $n_1 = -n_2$. Substituting these relations in Equation (2.4), we get $e_{ij} = K\tau_{ij} + \frac{K}{2}(d_{ij} - d_{ji})$. The theorem then follows from TD2. ■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The interface block

The structure of the interface block depends on the synchronization block being used. For illustration, consider the interface block for the hardware synchronization circuit in [29]. In [29], the control voltage to the oscillator depends on whether more than m clocks are faster than the output of the oscillator. From the description of the delay elimination block, it is clear that the clock at node j is faster than the clock at node i if the voltage e_{ij} is negative. A comparator that outputs a TTL high voltage when the input is negative and a TTL low when the input is positive can be used along with the "greater than m detector" described in [29] to convert the output of the delay elimination block into the desired form.

The complexity and cost of the delay elimination and the interface blocks depend on the number of clock inputs. The following three subsections describe an interconnection strategy and a modified phase locked algorithm in which the number of inputs to each clock is considerably less than the total number of clocks in the system.

2.4.2 The interconnection strategy

The total number of interconnections is reduced by grouping the clocks into several clusters and then treating each cluster as a clock unit. Each clock is synchronized by using the phase-locked algorithm not only with all the clocks in its own cluster but also with one clock from each of the other clusters. As a result of this mutual coupling, the clusters remain synchronized with respect to one another and the system as a whole remains well-synchronized.

Let M be the total number of clusters in the system. Let p_i denote the number of clocks in the i^{th} cluster. Number the clusters from 1 to M, and also number the clocks in each cluster i from 1 to p_i. Let c_{ij} represent the j^{th} clock of the i^{th} cluster and let $q_{ik} \stackrel{\text{def}}{=} [(i-1) \bmod p_k] + 1$. Then each clock in the i^{th} cluster receives as inputs not only all the clocks from its own cluster but also the ${q_{ik}}^{th}$ clock from each cluster $k \neq i$. This interconnection for $N = 8$, $M = 4$ and $p_i = 2$ for all i is shown in Table 2.1, where a 1 in row i and column j indicates that the clock corresponding to column j is an input to clock corresponding to row i.

There is no particular sanctity associated with this number q_{ik}. As long as each clock receives a clock from every other cluster, the algorithm will work. However, the above formula ensures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TO	OUTPUTS FROM							
	c_{11}	c_{12}	c_{21}	c_{22}	c_{31}	c_{32}	c_{41}	c_{42}
c_{11}	1	1	1	0	1	0	1	0
c_{12}	1	1	1	0	1	0	1	0
c_{21}	0	1	1	1	0	1	0	1
c_{22}	0	1	1	1	0	1	0	1
c_{31}	1	0	1	0	1	1	1	0
c_{32}	1	0	1	0	1	1	1	0
c_{41}	0	1	0	1	0	1	1	1
c_{42}	0	1	0	1	0	1	1	1

Table 2.1: Proposed interconnection scheme for an 8 clock system

that the clock network is symmetric because the formula results in similar fan-out for all the clocks.

2.4.3 Modification of the phase-locked algorithm

There are two main differences between the above interconnection strategy and the one in [33]:

- A given clock may receive inputs from clocks to which its own output is not connected. This was not possible in [33], because every clock received inputs from every other clock in the system.
- Different clocks could receive different number of inputs. This again was not possible in [33], since every clock received exactly $N - 1$ inputs, where N is the total number of clocks in the system.

These two differences cause a minor change in the phase-locked algorithm, since a reference signal is generated for each clock based only on the inputs it receives. That is, if a clock receives I inputs (including itself), then it assumes there are I clocks in the system and functions accordingly. If the maximum number of faults to be tolerated is m, then it is shown in [33] that if (i) $I > 3m$ and (ii) the reference signal is generated as follows, then the non-faulty clocks will remain synchronized. Each clock first orders all the inputs it received in the order of arrival of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the clock ticks. This ordered set is called the *tick sequence* of the corresponding clock. Let x be the position of its own clock in its tick sequence. Then the reference signal chosen by this clock is the $f_x(I)^{th}$ clock (excluding itself) in its tick sequence where $f_x(I)$ is any function satisfying one of the following conditions [33]:

- $m+1 \leq f_x(N) \leq N-m-1$, for all $x = 1, \ldots, N$
- $f_x(N) \geq m-1+f_{N-m}(N)$, for all $x \leq m+1$,
- $f_{N-m}(N) \leq f_x(N) \leq f_{m+1}(N)$, for all $m+1 < x < N-m$,
- $f_i(N) \leq f_j(N)$ iff $i \leq j$.

This implies that if different clocks in the system receive different number of inputs then they will have a different function for generating the reference signal. This fortunately has no effect on the synchronizing capabilities of the network.

2.4.4 Minimization of the number of interconnections

Assume for the time-being that all the clusters have the same size, p. This implies $M \cdot p = N$. The goal is to minimize the total number of interconnections $J \equiv Mp(p-1) + Mp(M-1) = N(M+p-2)$ subject to the required level of fault tolerance, which can be stated as $M+p-2 \geq 3m$ from the Byzantine Generals paradigm and the interconnection strategy. Substituting for p in $M+p-2$ from $N = Mp$ and differentiating with respect to M, it is easy to show that $M+p-2$ increases with M. Therefore both the total number of interconnections and the fault tolerance of the network increase with M. So minimizing J is equivalent to minimizing M. By solving for M in the fault tolerance condition and combining it with the above result, we can get a unique value for M that minimizes the total number of interconnections while satisfying the fault tolerance requirement.

However, the assumption that all clusters are of the same size is not appropriate for all values of N. For instance, if N were a prime number, then it would not be possible to find two factors M and p other than N and 1. This means that if we restrict ourselves to clusters of a single size, only fully connected networks are possible. On the other hand, any N can be decomposed into clusters of two different sizes, say p_1 and p_2. This will result in fewer interconnections than a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

fully connected network. There is no need to consider clusters of more than two sizes due to the reasons outlined below.

As p_1 and p_2 decrease in magnitude, the network tends to change towards a fully connected network, i.e., the fault tolerance of the network increases and so does the total number of interconnections. These are the two opposing factors which determine the values of p_1 and p_2. Had there been no fault tolerance condition the minimum number of interconnections would have occurred at $p_1 = p_2 = \sqrt{N}$. From symmetry considerations, we know that even under fault tolerance conditions the minimum number of interconnections will occur when $p_1 = p_2$. But since for any general N it may not be possible to find a p_1 and p_2 such that $p_1 = p_2$, the minimum number of interconnections occur when $p_1 - p_2 = 1$. Since any N can be decomposed into this form, at worst, it is necessary to have clusters of two different sizes p_1 and $p_1 - 1$. An algorithm to partition the clocks into clusters of two different sizes such that the total number of interconnections is minimized is described below.

Let the N clocks be partitioned into M_1 clusters of p_1 clocks each and M_2 clusters of p_2 clocks each, where $p_1 \geq p_2$. This implies $M_1 p_1 + M_2 p_2 = N$. Number the clusters with p_1 clocks (henceforth referred to as CP1) from 1 to M_1 and the clusters with p_2 clocks (CP2) from $M_1 + 1$ to $M_1 + M_2$. Also number the clocks in a given cluster from 1 to p_1 or 1 to p_2 correspondingly.

Let $q_{i1} = [(i-1) \bmod p_1] + 1$ and $q_{i2} = [(i-1) \bmod p_2] + 1$. Then each clock in cluster i receives the ${q_{i1}}^{th}$ clock from each CP1 and ${q_{i2}}^{th}$ clock from each CP2 in addition to all the clocks from its own cluster. The problem is to determine M_1, p_1, M_2, and p_2 that minimize the total number of interconnections and meet the specified fault tolerance requirement. The solution to this problem is more difficult than the case of single cluster size because there are three independent variables M_1, p_1, p_2 as opposed to only one.

The total number of interconnections in this scheme can be derived as follows.

- The total number of inputs to each clock in CP1: $M_1 + M_2 + p_1 - 1$.
- The total number of inputs to each clock in CP2: $M_1 + M_2 + p_2 - 1$.

Consequently, the total number of interconnections

$$\begin{aligned} J &= M_1 p_1 (M_1 + M_2 + p_1 - 1) + M_2 p_2 (M_1 + M_2 + p_2 - 1) \\ &= N (M_1 + M_2 - 1) + M_1 p_1^2 + M_2 p_2^2. \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The decomposition problem can then be formally stated as:

Problem D: Determine non-negative integers M_1, p_1, M_2, p_2 which minimize

$$J = N(M_1 + M_2 - 1) + M_1 p_1^2 + M_2 p_2^2$$

subject to

$$M_1 p_1 + M_2 p_2 = N$$

$$M_1 + M_2 + p_2 - 2 \geq 3m$$

$$p_1 - p_2 \geq 0$$

$$p_1 \leq 2(M_1 + M_2 - 1).$$

Since there are only finitely many integers between 0 and N, there are only finitely many possible solutions for M_1, M_2, p_1, p_2. Thus, there definitely exists an integer solution to the above problem. For small N we can determine the solution by enumerating all the possible solutions and choosing the one that gives the minimum value for J. For larger N we can take recourse to non-linear integer programming methods [24].

2.4.5 Proof of correctness

In this section, it is shown that the above interconnection scheme coupled with the modified phase-locked algorithm ensures synchronization of a N clock system in the presence of m faults when $N \geq 3m + 1$.

Let CK denote the set of clocks in the system. Consider a clock from this set. There are two possibilities: either this clock is connected only to all the clocks in its own cluster or it is also connected to at least one cluster other than its own. CK can therefore be decomposed into two subsets A and B such that $A \cap B = \emptyset$, where A is the set of all clocks which are connected to at least one cluster other than its own, and B is the set of all clocks connected only to its own cluster. Let CL_i, $i \in L \equiv \{1,2,\ldots,M\}$, be one of the M clusters in the system. Due to the interconnection strategy adopted, for any cluster pair, CL_i and CL_j, there is one and only one clock in CL_i which serves as an input to all the clocks in CL_j. Denote this clock by the ordered pair (i,j). There is one more clock link between CL_i and CL_j, but this clock is from CL_j to CL_i, i.e., it is the input to CL_i from CL_j, and so will be denoted by $(j,i) \in CL_j$. Also, every

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

such ordered pair of clusters uniquely represents a clock in set A. On the other hand, a clock in set A can have more than one such ordered pair representation but definitely has one such representation. Based on this observation, partition the clocks into four groups with respect to any given cluster, CL_j, as follows:

$$\begin{aligned} IN_j &\equiv \{i : i \in L,\ (i,j) \text{ is non-faulty}\} \\ IF_j &\equiv \{i : i \in L,\ (i,j) \text{ is faulty}\} \\ ON_j &\equiv \{i : i \in L,\ (j,i) \text{ is non-faulty}\} \\ OF_j &\equiv \{i : i \in L,\ (j,i) \text{ is faulty}\}. \end{aligned}$$

If $i = k$, then c_{ij} and c_{kl} are two non-faulty clocks of the same cluster and they are assumed to be δ-synchronized. Theorem 2.2 shows that when $i \neq k$, the interconnection scheme is such that c_{ij} and c_{kl} are 3δ-synchronized irrespective of the location of the faults in the system. Informally, the idea of the proof is to show that irrespective of the location of the faults, there exists a non-faulty link from either CL_i to CL_k or vice versa with at most two hops.

Lemma 2.2: For any two clusters CL_i and CL_k satisfying $|OF_i| < |IN_k|$, there exists a non-faulty path from CL_i to CL_k with at most two hops. See Figure 2.3 for an illustration. In the figure, $|\bullet|$ denotes the cardinality of a set.

Proof: First, suppose that the clock (i,k) is non-faulty. Then, irrespective of the location of the faults there is a direct path from CL_i to CL_k containing exactly one hop. Therefore, the more interesting case occurs when the clock (i,k) is faulty. In this case the lemma is proved via a contradiction.

Assume that there exists no cluster $CL_q \in IN_k$ such that (i,q) is non-faulty. This means for any cluster $CL_s \in IN_k$ the clock (i,s) is faulty. In other words, there are at least $|IN_k|$ faulty clock outputs from CL_i, i.e., $|OF_i| \geq |IN_k|$, a contradiction. ■

Theorem 2.2: Let c_{ij} and c_{kl} be any two non-faulty clocks in the system. Also let δ be the maximum skew that can arise between any two non-faulty clocks in the same cluster as a result of applying the phase-locked algorithm. Then, $|c_{ij} - c_{kl}| \leq 3\delta$ for all i, j, k, and l in $L \equiv \{1,2,\ldots,M\}$ under the condition $p_{max} \leq 2M - 2$, where $p_{max} = \max\{p_1, p_2, \ldots, p_M\}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

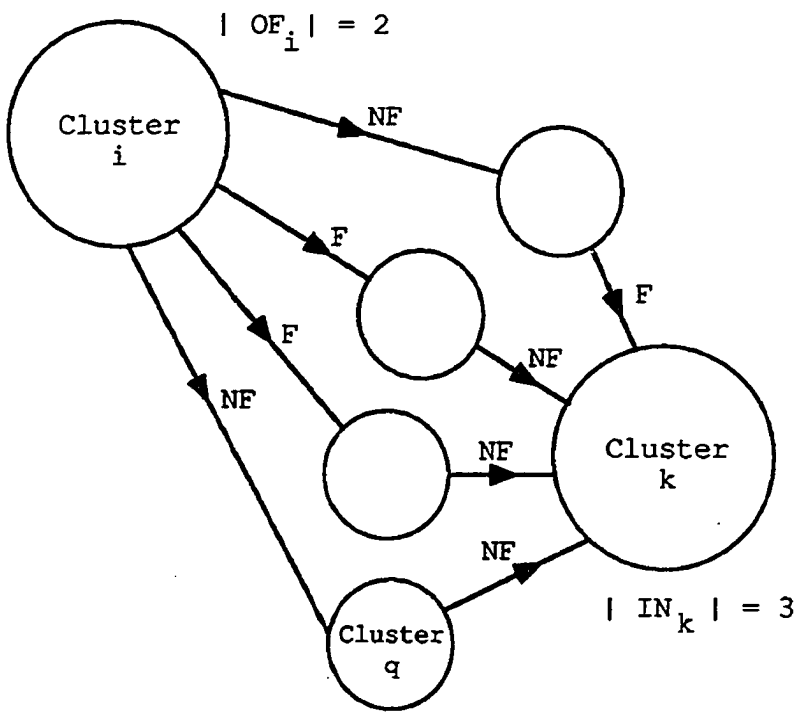

Figure 2.3: Illustration of case $|OF_i| < |IN_k|$ for Lemma 2.2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: Let $|IF_k| = x$ and let m be the maximum number of faults that needs to be tolerated. Since there are a total of M clusters in the system, it follows from the interconnection strategy that the total number of external inputs to any cluster is $M-1$. In other words,

$$|IN_k| = M - x - 1, \quad \text{or} \quad |IF_k| + |IN_k| = M - 1 \quad \text{for all } k \in L.$$

Consider the following two cases.

Case 1: $|OF_i| < \min\{M - x - 1,\ m + 1\}$.

Since $|OF_i| < M - x - 1 = |IN_k|$, by Lemma 2.2 there exists a cluster CL_q such that clocks (i,q) and (q,k) are both non-faulty. From the triangle inequality we get,

$$|c_{ij} - c_{kl}| \leq |c_{ij} - (i,q)| + |(i,q) - (q,k)| + |(q,k) - c_{kl}| \leq 3\delta.$$

Case 2: $M - x - 1 \leq |OF_i| \leq m$.

In this case it is possible that there is no $CL_q \in IN_k$ such that (i,q) is non-faulty, i.e., there is no non-faulty link from CL_i to CL_k (for example, see Figure 2.4). In such a case, it is shown that there is always a non-faulty path from CL_k to CL_i.

Let $r \equiv \left\lceil \frac{M}{p_{min}} \right\rceil$ where $\lceil x \rceil$ is the smallest integer not less than x and $p_{min} = \min\{p_1, p_2, \ldots, p_M\}$. Then, according to the interconnection strategy, every clock in the system could go to at most r different clusters. Using this along with the hypothesis $M - x - 1 \leq |OF_i|$, there are at least $\left\lceil \frac{M-x-1}{r} \right\rceil$ faulty clocks in CL_i. Thus, there are at most $m - \left\lceil \frac{M-x-1}{r} \right\rceil$ faulty clocks in the inputs to CL_i, i.e.,

$$|IF_i| \leq m - \left\lceil \frac{M-x-1}{r} \right\rceil, \quad \text{or} \quad |IN_i| \geq M - 1 - m + \left\lceil \frac{M-x-1}{r} \right\rceil. \tag{2.5}$$

Since $|IF_k| = x$, there are at most $m - x$ faulty clocks in CL_k and, thus,

$$|OF_k| \leq r(m - x). \tag{2.6}$$

If possible let there be no non-faulty path from CL_k to CL_i of length less than or equal to two hops. Then, from Lemma 2.2, $|OF_k| \geq |IN_i|$. Therefore, from Equations (2.5) and (2.6),

$$\begin{aligned} r(m-x) &\geq (M-1) - m + \left\lceil \frac{(M-x-1)}{r} \right\rceil \geq (M-1) - m + \frac{(M-x-1)}{r} \\ \text{or} \quad r^2 \cdot m + r \cdot m &\geq (r+1)(M-1) + (r^2-1)x. \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

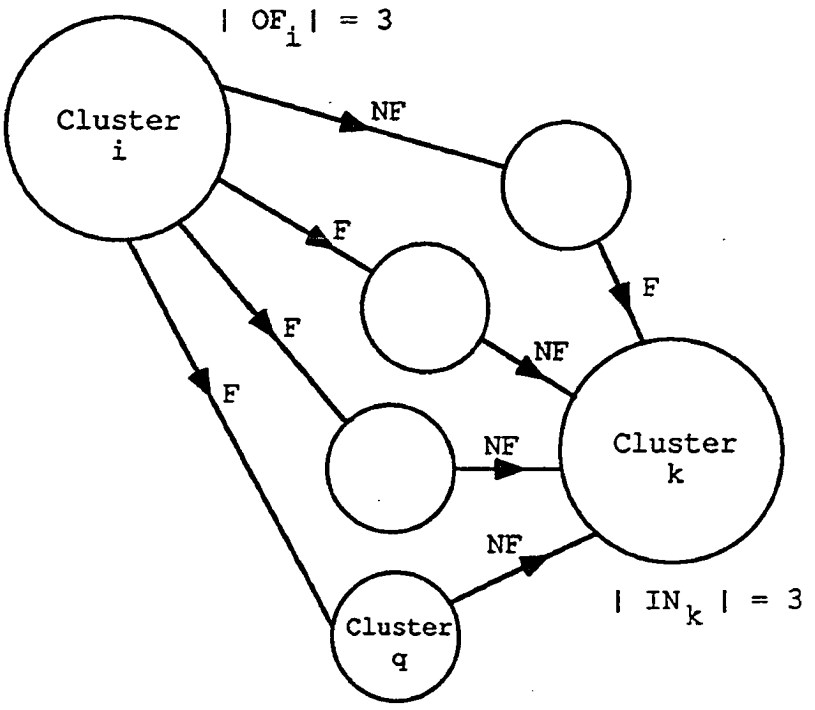

Figure 2.4: Illustration of the case $|OF_i| \geq |IN_k|$ for Theorem 2.2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since $x \leq m$, it follows that $m \geq M-1$. Therefore, the maximum number of inputs to any clock is $M + p_{max} - 1$ where $p_{max} = \max\{p_1, p_2, \ldots, p_M\}$. From the Byzantine Generals paradigm $\frac{M + p_{max} - 1}{3} > m$, leading to

$$M - 1 \leq m < \frac{M + p_{max} - 1}{3} \quad \text{or} \quad p_{max} > 2M - 2.$$

which contradicts the hypothesis. In other words, $|OF_k| < |IN_i|$. Then by Lemma 2.2 there exists at least one $q \in IN_i$ such that the clock (k,q) is non-faulty. Applying triangle inequality, for all i, j, k, and l

$$|c_{kl} - c_{ij}| \leq |c_{kl} - (k,q)| + |(k,q) - (q,i)| + |(q,i) - c_{ij}| \leq 3\delta. \quad \blacksquare$$

The condition $p_{max} \leq 2M-2$ in the above theorem implies that the connectivity is sufficiently large to ensure a good synchronization. The maximum fault tolerance is achieved in a fully connected network, i.e., when $p_{max} = 1$ and $M = N > 1$, in which case case the above condition is satisfied. In order to reduce the number of interconnections it is necessary to compromise on the maximum number of faults that can be tolerated. As shown earlier it is sufficient to have $p_{max} \leq \sqrt{N}$ to ensure the least number of interconnections under any fault tolerance specification when all the clusters are of the same size. The condition $p_{max} \leq 2M - 2$ is a generalization of the condition $p_{max} \leq \sqrt{N}$ because $p_{max} \leq \sqrt{N}$ implies $p_{max} \leq 2M - 2$.

2.4.6 Numerical example

The following results were obtained by solving Problem D using simple enumeration techniques. Table 2.2 illustrates the variation in the total number of interconnections with respect to the size of the system when the fault tolerance requirement is kept constant. The table also gives the percentage reduction in the number of interconnections as compared to a fully connected network. The plot of the variation in percentage reduction with respect to the size of the system and hence with respect to the number of clocks is given in Figure 2.5.

Figure 2.5 indicates that for a given fault tolerance condition the total number of interconnections increases proportionately to the size of the system. This is because when the size increases, the number of clocks in the system increases, and so more interconnections are needed to keep

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

N	m	M_1	p_1	M_2	p_2	J	% Reduction
20	3	2	3	7	2	206	45.79
30		6	5	0	0	300	65.52
40		2	6	4	7	468	70.00
50		1	8	6	7	658	73.14
62		2	7	6	8	916	75.78
64		1	8	7	8	960	76.19
100		10	10	0	0	1900	80.19
20	5	4	2	12	1	328	13.68
30		1	2	14	2	480	44.83
40		5	2	10	3	670	57.05
50		6	3	8	4	832	66.04
62		2	6	10	5	1004	73.45
64		4	6	8	5	1048	74.01
100		10	10	0	0	1900	80.19
20	7	-	-	-	-	-	-
30		14	1	8	2	676	22.30
40		4	1	18	2	916	41.28
50		8	3	13	2	1124	54.12
62		2	4	18	3	1372	63.72
64		4	4	16	3	1424	64.68
100		8	5	10	6	2260	77.17

Table 2.2: Variations in number of interconnections with size of the system

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

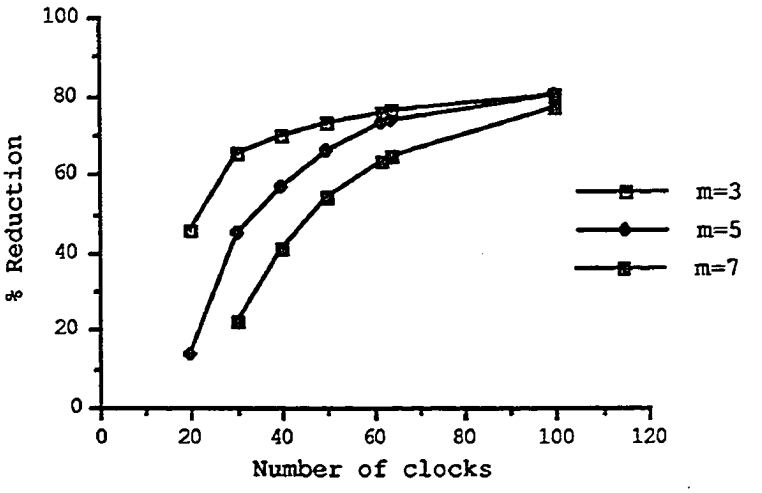

Figure 2.5: % Reduction over a fully connected network versus size of the system

them synchronized. However, while the total number of interconnections increases, the percentage reduction in the total number of interconnections also increases with the size of the system. This is significant because the fault tolerance condition will not usually increase proportionately with the size of the system. In such a case, reduction in the total number of interconnections will be one of the main concerns and our scheme works well in that situation.

Another aspect that is clear from Table 2.2 is that reductions up to 80% can be achieved for relatively less stringent fault tolerance condition. However, under those conditions the interconnection scheme in [62] requires fewer interconnections as compared to the scheme here (see Table 2.3). But as the fault tolerance specification becomes more stringent the percentage difference between the number of interconnections in both schemes drops rapidly to a very low value and then remains almost constant through the entire range. This aspect is clearly depicted by the plots in Figure 2.6. In other words, the scheme in [62] is only marginally better than the scheme put forth here under almost all fault tolerance specifications.

2.4.7 Comments

The above hardware synchronization scheme provides a tradeoff between fault tolerance and the total number of interconnections. This scheme can be used to determine a symmetric clock

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

N	m	M_1	p_1	M_2	p_2	J	J_{FTMP}	%Increase
20	2	5	4	0	0	160	120	33.30
	3	2	3	7	2	206	180	14.44
	4	6	1	7	2	274	240	14.17
	5	4	2	12	1	328	300	9.33
	6	20	1	0	0	380	360	5.55
62	3	2	7	6	8	916	558	64.16
	5	2	6	10	5	1004	930	7.90
	7	2	4	18	3	1372	1302	5.38
	8	10	2	14	3	1592	1488	6.99
	9	8	3	19	2	1760	1674	5.14
	12	12	1	25	2	2344	2232	5.01
	18	7	2	48	1	3424	3348	2.27
	20	20	1	0	0	3782	3720	1.67
64	3	8	8	0	0	960	576	66.67
	5	4	6	8	5	1048	960	9.17
	7	4	4	16	3	1424	1344	5.95
	9	10	3	17	2	1822	1728	5.44
	12	10	1	27	2	2422	2304	5.12
	18	9	2	46	1	3538	3456	2.37
	21	64	1	0	0	4032	4032	0.00
100	3	10	10	0	0	1900	900	111.11
	6	10	10	0	0	1900	1800	5.55
	10	16	3	13	4	3152	3000	5.07
	15	10	3	35	2	4630	4500	2.88
	20	22	1	39	2	6178	6000	2.97
	25	24	2	52	1	7648	7500	1.97
	30	9	2	82	1	9118	9000	1.31
	33	100	1	0	0	9900	9900	0.00

Table 2.3: Variations in number of interconnections with fault tolerance specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

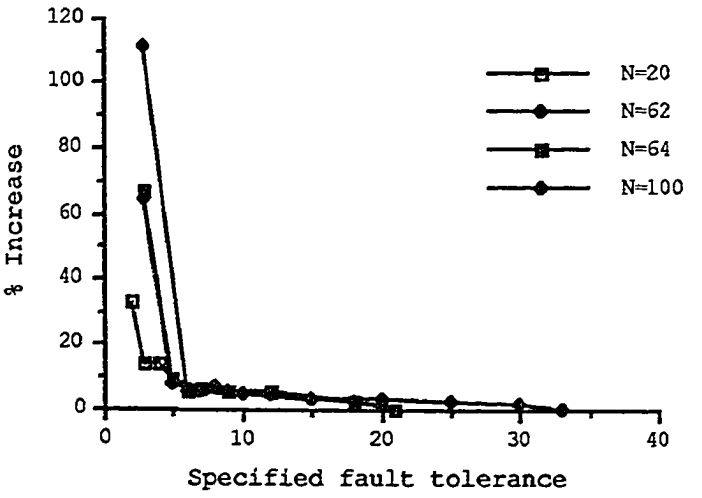

Figure 2.6: % Increase versus specified fault tolerance

network that requires a near minimal number of interconnections for any fault tolerance specification. In large systems, up to 80% reduction can be achieved in the total number of interconnections as compared to other hardware solutions with worst-skews in the order of 30–50 ns.

In spite of this considerable reduction, the cost of additional hardware can be substantial for many applications. The scheme requires a network of clocks that is different from the interconnection network between the nodes of the system. In addition, each clock is required to have two phase-detectors for every clock input and some circuitry to select a reference signal and adjust the local clock.

For systems that cannot afford the cost of additional hardware in the above approach, a software synchronization scheme is proposed below. This scheme strikes a balance between the hardware requirement and the clock skews attainable. Unlike other software solutions, the guaranteed worst-case skews in this scheme can be made insensitive to the message transit delay in the system. The scheme is also particularly suitable for large partially connected distributed systems with topologies that support simple broadcast algorithms. Examples of such topologies include the hypercube and the mesh interconnection structures.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.5 A Software Synchronization Scheme

The scheme proposed here is similar to the interactive convergence algorithm (CNV) in [36]. The differences arise mainly because CNV is intended for a fully connected system as opposed to a partially connected system like the hypercube [53] or the mesh topologies [1]. Although CNV could be used in a partially connected system, it is not well suited for such a system since the worst-case skews in CNV are greater than the maximum message transit delay in the system. In contrast, the worst-case skews in the scheme below are orders of magnitude smaller than the maximum time required for a reliable broadcast and they do not depend on this maximum time beyond a certain limit.

Algorithm CNV assumes that the clocks are initially synchronized and that they drift apart only by a bounded amount during a resynchronization interval. Each node executes a clock process (CP) that is responsible for maintaining a time base for all the activities on that node. In every resynchronization interval, each CP reads the value of the clock at all other nodes. If the value of a clock read differs from the clock at its node by an amount greater than a threshold, CP replaces that value by its own clock value. CP then computes the average of all such values and sets the local clock to that average. In [36], it is shown that this algorithm can achieve synchronization, and requires a minimum of $3m + 1$ nodes to tolerate m faults.

Three major problems arise when this algorithm is used in a distributed system that is not fully connected. First, the clock message received at a node may be corrupted by a faulty intermediate node. Second, the queueing delay for clock messages may cause a substantial difference between the real time at which a CP sends the clock value and the real time at which another CP receives that message. Therefore, subtracting the clock value in the received message from the current clock value will not reflect the actual skew that exists between the clocks of the two nodes. This problem is aggravated when system is partially connected because the clock message has to be relayed through intermediate nodes. Third, there may be a delay between the receipt of a clock message from the network and the time at which it is processed. A similar delay is possible between the time at which a CP sends the message and the time at which the message is placed on the network. The first problem is eliminated in the proposed scheme by using a broadcast algorithm that delivers multiple copies of the message to all CPs through node-disjoint paths.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The second problem is reduced by requiring the CP at each intermediate node to append to the message the delay (according to the local clock) incurred at that node. The third problem is handled by recording the time at which a clock message is sent or received.

In the proposed scheme, a CP broadcasts the local clock value to all other CPs at a specified time in the resynchronization interval according to its clock. The broadcast algorithm is such that all CPs receive multiple copies of the clock message through node-disjoint paths. The number of copies used in the broadcast algorithm depends on the maximum number of faults to be tolerated and the fault model for the system. When a CP receives a clock message sent by some other CP, it records the time (according to its local clock) at which the message was received. Then, in accordance with the broadcast algorithm, it relays the message to other CPs. Before relaying the message, it appends to the message the time elapsed (according to its own clock) since the receipt of the message. At the end of a resynchronization interval, it computes the skews between the local clock and the clock of the source node for each one of the copies it has received. It then selects the $(m+1)^{th}$ largest value as an estimate of the skew between the two clocks. The average of the estimated skews over all nodes is used as the correction to the local clock. As in CNV a minimum of $3m+1$ nodes are required to tolerate m Byzantine faults.

The above steps are explained in terms of four concurrent tasks: CLOCK_SEND, CLOCK_RELAY, CLOCK_RECEIVE and CLOCK_CORRECTION. An efficient implementation of these tasks requires some hardware support and a reliable broadcast mechanism.

2.5.1 Hardware support

A clock message contains the following information: the node at which the broadcast was initiated (namely, the initiator), the node from which the message was most recently relayed (namely, the immediate sender), and five words W_1, W_2, ..., W_5 shown below. The relative ordering of these words in the message depends on the format being used in the system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

W_1: Clock time of the initiator when this message was transmitted.

W_2: Clock time of the immediate sender when it received this message.

W_3: Clock time of the immediate sender when it relayed this message.

W_4: Accumulated transit delay not counting the delay at the immediate sender.

W_5: The time on the local clock when this message was received.

The hardware circuitry at each node helps in updating these words. When a clock message is received it updates W_5 with the local time. An interrupt is then generated to the processor that runs the clock tasks at that node. After the clock tasks have completed their processing, the message is scheduled for forwarding. While the message is being transmitted the hardware circuitry inserts the local clock time in W_3. The other operations performed on these words are explained in the formal description of the clock tasks later in this section.

A special circuit is required to maintain a continuous monotone logical clock at each node. A possible solution is as follows. A high frequency clock generated by a crystal oscillator is divided by a programmable counter and the divided clock is used to increment a counter that contains the logical clock. The factor used to divide the high frequency clock depends on the correction that needs to be applied to the logical clock. If at the end of a resynchronization interval the correction is positive (negative), then the factor used to divide the high frequency clock is increased (decreased) from its nominal value. For example, a 32 MHz clock can nominally be divided by 32 to get a logical clock whose resolution is 1 μs. This factor can be changed either to 36 or 28 depending on whether the correction is positive or negative, respectively.

2.5.2 Reliable broadcast

A reliable broadcast mechanism is especially important in a partially connected distributed system because a process may have to deliver its clock value to the other processes via several intermediate nodes. However, since the non-faulty CPs need not agree on the clock value at a faulty node, it is not necessary to use an interactive consistency algorithm [9, 37] to correctly deliver the clock values.

In the presence of m Byzantine faults, the following scheme can be used to reliably deliver

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the clock values. For each pair of nodes, determine a set of $2m + 1$ node-disjoint paths. (If there does not exist $2m + 1$ disjoint paths in the system, it cannot tolerate up to m Byzantine faults [7].) The CP at a non-faulty node then individually sends $2m + 1$ copies of the local clock value to the CP at each node through the pre-determined paths for that node. As shown in [71] this scheme does not require authentication. It requires: (i) the network to be $2m + 1$ connected, and (ii) a process should be able to identify the node from which a received message originated and the node from which it was most recently relayed. These two requirements are necessary for any algorithm that is resilient to m Byzantine faults when authentication is not available [7]. It is shown in Theorem 2.3 that a CP can reliably estimate the skew between the local clock and the clock at another node from the multiple copies of the clock message it receives from the CP at that node.

The above scheme, albeit very general, requires each CP to send large number of messages, thus resulting in substantial time overhead. However, for interconnection topologies like the hypercube [53] or a C-wrapped hexagonal mesh [1] there exist reliable broadcast algorithms that do not require many messages to be sent by a process [27, 47]. As in the scheme described above, they deliver $2m + 1$ copies of the clock message through node-disjoint paths. But they make use of the properties of the topology to reduce the number of messages that have to be sent by a process. Due to the importance of a reliable broadcast mechanism for distributed algorithms that are resilient to Byzantine faults, one can argue that most fault-tolerant distributed systems will use topologies that support simple broadcast algorithms similar to the ones in [27, 47]. Consequently, broadcast mechanism will be assumed to exist in the distributed systems being synchronized.

2.5.3 Detailed description

A CP is comprised of four concurrent tasks. The operations of these four tasks required can be described in terms of the following basic functions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

local_time( )	Returns the current time on the local clock.
btime(p)	Returns the time at which p is supposed to broadcast its clock message in the current resynchronization interval.
initiate_bcast(k,t)	Initiates broadcast of k copies of a clock message with time set at t.
w1(M), ..., w5(M)	The five clock words in the message M.
receive( )	Blocks until a clock message arrives and returns the received message.
initiator(M)	Returns the node at which the broadcast of message M was initiated.
select(*skews*, ℓ)	*skews* is an array of estimated skews with respect to a particular node computed from copies of clock messages received from the CP at that node. The function select(*skews*, ℓ) orders these skews and returns the ℓ^{th} value.

CLOCK_SEND

The CLOCK_SEND task is responsible for initiating a reliable broadcast in each resynchronization interval as shown in Figure 2.7. The time at which the broadcast takes place depends on the node and is staggered throughout the resynchronization interval. This alleviates the problem of transient loading which would otherwise occur due to the almost simultaneous broadcast of clock messages. The reliable broadcast delivers $2m+1$ copies of the message to each CP through node-disjoint paths if all CPs adhere to the broadcast algorithm. However, if some nodes/links are faulty, then fewer than $2m+1$ copies may be delivered. Given that there are at most m faulty nodes in the system, each CP will receive at least $m+1$ copies relayed through non-faulty nodes. It is shown in Theorem 2.3 that the CPs at all non-faulty nodes can reliably estimate the clock skew from the copies received.

Contention for network resources there may cause a time delay between the initiation of a broadcast (i.e., line 3 of Figure 2.7) and the time at which the copies are transmitted on the network. To account for this delay, the time on the local clock is inserted in W_1 and W_2 while the broadcast is being initiated. During transmission, the hardware circuitry at that node will

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
1.  loop
2.      if (local_time( ) = btime(p))
3.          initiate_bcast(2m + 1, local_time( ));
4.      endif;
5.  endloop;
```

Figure 2.7: CLOCK_SEND task at node p

```
1.  constant MAX_BROADCAST_TIME = U;
2.  loop
3.      M := receive( );
4.      source := initiator(M);
5.      if ( |local_time( ) – btime(source)| ≤ MAX_BROADCAST_TIME )
6.          w4(M) := w4(M) + w3(M) – w2(M);
7.          w2(M) := w5(M);
8.          relay the message according to the broadcast algorithm;
9.      endif;
10. endloop;
```

Figure 2.8: CLOCK_RELAY task

insert the current time in place of W_3. The delay between the initiation of the broadcast and the time at which the copy was transmitted, indicated by $W_3 - W_2$, will be accumulated in W_4 by the CLOCK_RELAY task of the node that is receiving this message.

CLOCK_RELAY

The CLOCK_RELAY task (Figure 2.8) is activated when a clock message broadcast by some other CP is received. It is responsible for relaying the message to other CPs as per the broadcast algorithm. It is also responsible for updating the words $W_1, \ldots, W_5$ in the clock message.

The CLOCK_RELAY task checks whether the initiator specified in the message is supposed to be broadcasting its clock value at this time instant. This is used to suppress spurious messages generated from faulty nodes. This check is possible if there is a bound on the time to complete a broadcast, say U. If the message is authentic, $(W_3 - W_2)$ is added to W_4 to accumulate the transit delay incurred by the message at the immediate sender. This node now becomes the immediate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
1.  constant MAX_BROADCAST_TIME = U;
2.  loop
3.      M := receive( );
4.      source := initiator(M);
5.      if ( |local_time( ) – btime(source)| ≤ MAX_BROADCAST_TIME )
6.          copy := Current_copy[source];
7.          Delta[source][copy] := w5(M) – (w4(M) + w3(M) – w2(M)) – w1(M);
8.          Current_copy[source] := Current_copy[source] + 1;
9.      endif;
10. endloop;
```

Figure 2.9: CLOCK_RECEIVE task

sender for the next step.

CLOCK_RECEIVE

When a message is received, the CLOCK_RECEIVE task (Figure 2.9) first verifies the authenticity of the message. It then computes the skew between the local clock and the clock at the node where the broadcast was initiated and adjusts the skew by subtracting the accumulated delay. This operation is performed for each copy that is received and the resulting skews are passed on to the CLOCK_CORRECTION task.

CLOCK_CORRECTION

The CLOCK_CORRECTION task (Figure 2.10) corrects the local clock based on the skews calculated by CLOCK_RECEIVE. It first selects the estimated skew for a particular node from the values computed for the copies received from that node. If the estimated skew is greater than a threshold, Δ, it is replaced by zero. The choice of Δ follows from the proof of correctness (see Lemma 2.7). The correction applied to the local clock is the average of the estimated skews over all nodes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
1.  constant THRESHOLD = Δ;
2.  loop
3.      if (local_time( ) = resync_time( ))
4.          total_skew := 0;
5.          for q = 1 to NUMBER_OF_PROCESSES
6.              Skew[q] := select(Delta[q], m + 1);
7.              if (Skew[q] > THRESHOLD)
8.                  Skew[q] = 0;
9.              endif;
10.             total_skew := total_skew + Skew[q];
11.         endfor;
12.         correction := total_skew / NUMBER_OF_PROCESSES;
13.     endif;
14.     initialize Current_copy, etc. for the next resynchronization interval;
15. endloop;
```

Figure 2.10: CLOCK_CORRECTION task

2.5.4 Proof of correctness

Before formally proving the correctness of the above software scheme it is necessary to introduce the following additional notations and terminology. These notations are similar to the ones in [36].

c_p	The clock at node p.
R	Resynchronization interval.
$T^{(0)}$	The time at which the system began its operation.
$T^{(i)}$	Time for i^{th} resynchronization, i.e, $T^{(0)} + iR$.
$R^{(i)}$	The interval $[T^{(i)}, T^{(i+1)}]$.
$c_p^{(i)}(T)$	Logical clock at node p in the interval $R^{(i)}$, where $c_p^{(i)}(T) = c_p(T + \xi_p^{(i)})$ for some constant $\xi_p^{(i)}$ representing a change in p's clock during $R^{(i)}$.

For notational convenience, let $\xi_p^{(0)} = 0$ and $c_p^{(0)} = c_p$. A node p will be referred to as being *non-faulty up to time* $T^{(i+1)}$ if it is non-faulty during the real time interval $[c_p^{(0)}(T^{(0)}),\ c_p^{(i)}(T^{(i+1)})]$. Using the above notation the two clock synchronization conditions can be stated as:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

S1. If nodes p and q are non-faulty up to time $T^{(i+1)}$, then $\forall\, T \in R^{(i)}$

$$\left|c_p^{(i)}(T) - c_q^{(i)}(T)\right| < \delta \qquad \text{for some constant } \delta.$$

S2. If node p is non-faulty up to time $T^{(i+1)}$, then

$$\left|\xi_p^{(i+1)} - \xi_p^{(i)}\right| < \Sigma \qquad \text{for some constant } \Sigma.$$

Intuitively, the first condition states that the difference in clock times on two non-faulty nodes is always bounded. The second condition states that there is a bound on the amount by which a clock can change its value during one resynchronization interval. We now show that the software scheme satisfies S1 and S2 if the following assumptions hold.

A1. For all nodes p and q, $\left|c_p(T^{(0)}) - c_q(T^{(0)})\right| < \delta_0$, where δ_0 is a constant.

A2. If node p is non-faulty during the time interval $[t_1, t_2]$, then c_p is a good clock during the interval. That is,

$$|c_p(t) - c_p(t_1)| < \frac{\rho}{2}(t - t_1) \qquad \forall\, t \in [t_1, t_2] \text{ for some constant } \rho.$$

A3. In a reliable broadcast initiated from a non-faulty node, all copies relayed through non-faulty nodes are delivered within a time bound U.

A4. Suppose S1 and S2 hold for i and node p is non-faulty up to time $T^{(i+1)}$. Also suppose that the CP at a non-faulty node q sends a message containing q's clock value to p at some time T_0 in $R^{(i)}$ according to q's clock. This message reaches p through a relay (possibly empty) of nodes. The CP at each relay node alters the words $W_1, \ldots, W_5$ in the message as described in CLOCK_RELAY. From the message received, the CP at p computes a value Δ_{qp} as described in CLOCK_RECEIVE. If all the relay nodes are non-faulty, then

$$\left|c_p^{(i)}(T_0 + \Delta_{qp} + Q) - Q - c_q^{(i)}(T_0)\right| < \epsilon$$

where $Q = W_5 - (W_4 + W_3 - W_2)$ is an estimate of the total transit delay incurred by the message and ϵ is a constant.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A5. The upper bound, U, on the time required to complete a reliable broadcast is such that $U \leq \frac{R}{N}$, where R is the resynchronization interval, and N is the number of nodes in the system.

Intuitively, A1 states that clocks are initially δ_0-synchronized to each other. A scheme to achieve this initial synchronization is described later. A2 states that if a node is non-faulty, then so is its clock. In other words, no distinction is made between a faulty clock and a faulty node. A3 states that there is an upper bound U on the time to complete a reliable broadcast. In A4, $c_p^{(i)}(T_0 + \Delta_{qp} + Q)$ is the real time at which the message is received at p and $c_q^{(i)}(T_0)$ is the real time at which the message was sent from q. Since Q is the total delay incurred by the message in transit, $c_p^{(i)}(T_0 + \Delta_{qp} + Q) - Q - c_q^{(i)}(T_0)$ is the error in estimating the skew at p based on this message. Therefore, A4 states that if all CPs adhere to the scheme for broadcasting and relaying the clock messages, then the error in estimating the skew is bounded. This assumption is reasonable because the errors occur mainly due to: (i) the transit delay incurred at an intermediate node is measured using a discrete logical clock as opposed to a continuous clock, and (ii) the logical clock at an intermediate node drifts from real time while measuring the delay incurred at a node. The existence of a bound then follows from the observation that the time required for a broadcast is bounded, the drift rates of non-faulty clocks are bounded, and the non-faulty clocks have a known finite resolution.

A5 states that a broadcast initiated by a CP will always complete before the next CP initiates its broadcast. As a result, at most one CP will be broadcasting its clock value at any given time. Since in our scheme only one CP broadcasts its clock message at any given instant and since the increase in delay is more than linear with increase in load, requiring $U \leq \frac{R}{N}$ when only one CP is broadcasting is less restrictive than $U \leq R$ when all CPs can broadcast.

Given these assumptions it can be proven that the non-faulty clocks will remain synchronized throughout the operation of the system. Due to the inherent complexity of the notation, an informal explanation of the lemmas is presented first. The lemmas are similar to the ones in [36]. The differences arise mainly because the worst-case skews are shown to be less than the maximum transit delay during a broadcast and because of the modeling assumption that a CP may have to use a few relay nodes to convey its clock value to other CPs. A few additional lemmas are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

required to show that the faulty relay nodes cannot prevent synchronization.

Lemma 2.3 states that for a good clock Π units of clock time elapse in approximately Π units of real time. Lemma 2.4 deals with the perceived skew between two non-faulty nodes, stating that there is a bound on the estimated skew between two non-faulty nodes. This bound is used in Lemma 2.8 to prove that the impact of faulty nodes can be controlled. In Lemma 2.6 it is shown that it does not matter (to a limited extent) at what time the CP at a non-faulty node initiates a broadcast of its clock value in a resynchronization interval. Lemma 2.7 states that the difference between the skew perceived at p between p and r and the skew perceived at q between q and r is bounded when p, q and r are non-faulty. Theorem 2.3 and Lemma 2.5 deal with the case when some of the relay nodes are faulty. Finally, Theorem 2.4 combines these lemmas to prove the correctness of the scheme.

Lemma 2.3: If Clock Synchronization Condition S1 holds for i and node p is non-faulty up to time $T^{(i+2)}$, then for any Π such that $|\Pi| < R$ and any T in $R^{(i)}$:

$$\left|c_p^{(i)}(T+\Pi) - [c_p^{(i)}(T)+\Pi)]\right| < \frac{\rho}{2}\Pi.$$

Proof: Follows easily from A2. ■

Lemma 2.4: Suppose S1 holds for i, and nodes p and q are non-faulty up to time $T^{(i+1)}$. If the CP at q sends its clock value to the CP at p through several non-faulty relay nodes, then skew Δ_{qp} estimated at p for this message using the proposed algorithm is such that

$$|\Delta_{qp}| \leq \frac{\delta + \epsilon + \frac{\rho}{2}\cdot U}{1-\frac{\rho}{2}}.$$

Proof: Let T_0 be the time according to q's clock at which it sent the message to p. Also let Q denote the total delay encountered by the message as indicated by words $W_1, \ldots, W_5$. Then by repeated application of triangle inequality we get,

$$\begin{aligned}
|\Delta_{qp}| &= \left|c_p^{(i)}(T_0) - c_p^{(i)}(T_0+Q+\Delta_{qp}) + Q + c_p^{(i)}(T_0+Q+\Delta_{qp}) - c_p^{(i)}(T_0) - Q - \Delta_{qp}\right| \\
&\leq \left|c_p^{(i)}(T_0) - c_p^{(i)}(T_0+Q+\Delta_{qp}) + Q\right| + \left|c_p^{(i)}(T_0+Q+\Delta_{qp}) - [c_p^{(i)}(T_0)+Q+\Delta_{qp}]\right| \\
&\leq \left|c_p^{(i)}(T_0) - c_q^{(i)}(T_0)\right| + \left|c_q^{(i)}(T_0) - c_p^{(i)}(T_0+Q+\Delta_{qp}) + Q\right| + \frac{\rho}{2}|Q| + \frac{\rho}{2}|\Delta_{qp}|.
\end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This implies

$$\begin{aligned}\left(1-\frac{\rho}{2}\right)\cdot|\Delta_{qp}| &\leq \left|c_p^{(i)}(T_0)-c_q^{(i)}(T_0)\right|+\left|c_q^{(i)}(T_0)-c_p^{(i)}(T_0+Q+\Delta_{qp})+Q\right|+\frac{\rho}{2}|Q| \\ &\leq \delta+\epsilon+\frac{\rho}{2}|Q| \qquad \text{(hypothesis and A4)} \\ &\leq \delta+\epsilon+\frac{\rho}{2}U \qquad \text{(by A3).}\end{aligned}$$

Hence proved. ■

Informally, Lemma 2.4 states that the Δ_{qp} calculated from a clock message is bounded if all the intermediate nodes through which the message was relayed are non-faulty. However, all the intermediate nodes may not necessarily be non-faulty and a CP may not be able to identify the messages that have been relayed through faulty intermediate nodes. Similarly, A4 holds only if all the intermediate nodes are non-faulty. These two problems are overcome by using a reliable broadcast that delivers multiple copies of a clock message to all CPs through node-disjoint paths. A receiving CP estimates the skew Δ_{qp} from the multiple copies it has received as in CLOCK_RECEIVE. In Theorem 2.3 and its corollary it is shown that the estimated Δ_{qp} satisfies the bound specified in Lemma 2.4. Furthermore, the bound specified in A4 can be changed to account for the fact that the selected copy may have passed through some faulty nodes.

Theorem 2.3: Suppose q uses CLOCK_RECEIVE to estimate the skew between p's clock and its own clock based on the messages it receives in a reliable broadcast initiated from p. Then, the estimated skew is either equal to the computed skew from a copy that was relayed only through non-faulty nodes, or there are two copies that have been relayed only through non-faulty nodes such that the estimated skew lies between the computed skews for these two copies.

Proof: Since the system is assumed to have a maximum of m faults, and since the reliable broadcast tries to deliver $2m+1$ copies, at least $m+1$ copies are relayed through non-faulty nodes. By A3 these $m+1$ copies will be delivered within U time units. Consequently, the CP at q can unambiguously determine the number of copies it will receive.

Suppose $2m+1-t$ copies are received at q for some $1 \leq t \leq m$. There are two possible cases: (i) $t < m$ and (ii) $t = m$. When $t < m$ there are more than $m+1$ copies. Since the skew estimated at q is the $(m+1)^{th}$ largest value, there are m computed skews smaller than and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

greater than the estimated skew. The statement of the theorem follows from this observation and the hypothesis that there are only m faults in the system.

When $t = m$ all the copies that have been relayed through faulty nodes are lost. So all the copies that have been received were relayed only through non-faulty nodes. Hence proved. ■

Corollary 2.1: Suppose p uses CLOCK_RECEIVE to estimate the skew, Δ_{qp}, between q's clock and its own clock based on the messages its receives in a reliable broadcast initiated by q. Then,

$$|\Delta_{qp}| \leq \frac{\delta + \epsilon + \frac{\rho}{2} \cdot U}{1 - \frac{\rho}{2}}.$$

Proof: Follows from Theorem 2.3 and Lemma 2.4. ■

Lemma 2.5: Suppose S1 holds for i, and nodes p and q are non-faulty up to time $T^{(i+1)}$. Also suppose the CP at q uses a reliable broadcast at time T_0 according to its clock to convey q's clock value and p uses CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew Δ_{qp}. If Q is the total transit delay as computed from the copy of the clock message that was used to estimate the skew, then

$$\left| c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0) \right| \leq \hat{\epsilon} \equiv \epsilon + \frac{\rho(\delta + \epsilon + U)}{(1 - \frac{\rho}{2})}.$$

Proof: If the copy used to estimate the skew was relayed only through non-faulty nodes, then the lemma follows from Lemma 2.5. So the more interesting case is when it was relayed through at least one faulty node. In this case it follows from Theorem 2.3 there are at least two copies relayed only through non-faulty nodes, say through paths P_1 and P_2, such that the estimated skew lies between the computed skews from the two copies. Let Q_1 and Q_2 be the accumulated transit delays, and let $\Delta_{qp}(P_1)$ and $\Delta_{qp}(P_2)$ be the computed skews from the clock messages that were relayed through paths P_1 and P_2, respectively.

Without loss of generality one can assume that $\Delta_{qp}(P_1) \leq \Delta_{qp} \leq \Delta_{qp}(P_2)$. Starting with the inequality $\Delta_{qp}(P_1) \leq \Delta_{qp}$ we get

$$c_p^{(i)}(T_0) + Q + \Delta_{qp}(P_1) \leq c_p^{(i)}(T_0) + Q + \Delta_{qp} - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + c_p^{(i)}(T_0 + Q + \Delta_{qp}).$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This is equivalent to

$$c_p^{(i)}(T_0) - c_q^{(i)}(T_0) - \frac{\rho(\delta+\epsilon+U)}{2(1-\frac{\rho}{2})} + \Delta_{qp}(P_1) \;\leq\; c_p^{(i)}(T_0+Q+\Delta_{qp}) - Q - c_q^{(i)}(T_0)$$

from Lemma 2.3, A4, and Corollary 2.1. On simplifying the left hand side by using Lemma 2.3 and A4, we get

$$-\epsilon - \frac{\rho(\delta+\epsilon+U)}{(1-\frac{\rho}{2})} \leq c_p^{(i)}(T_0+Q+\Delta_{qp}) - Q - c_q^{(i)}(T_0).$$

This proves one side of the required inequality. To prove the other side, we can start with $\Delta_{qp} \leq \Delta_{qp}(P_2)$ and show that

$$c_p^{(i)}(T_0+Q+\Delta_{qp}) - Q - c_q^{(i)}(T_0) \leq \epsilon + \frac{\rho(\delta+\epsilon+U)}{(1-\frac{\rho}{2})}.$$

Hence the lemma follows. ■

Lemma 2.6: Suppose S1 holds for i, and nodes p and q are non-faulty up to time $T^{(i+1)}$. Also suppose that the CP at q uses a reliable broadcast algorithm and that CP at p estimates the skew, Δ_{qp}, as in CLOCK_RECEIVE and CLOCK_CORRECTION. If Q is the total transit delay as computed from the copy used to estimate the skew, then for any Π such that $|\Pi| < R$ and any T in $R^{(i)}$:

$$\left|c_p^{(i)}(T+\Pi+Q+\Delta_{qp}) - Q - c_q^{(i)}(T+\Pi)\right| \leq \hat{\epsilon} + 2\rho R.$$

Proof: Let T_0 be as in A4. Then,

$$\begin{aligned}
&\left|c_p^{(i)}(T+\Pi+Q+\Delta_{qp}) - Q - c_q^{(i)}(T+\Pi)\right| \\
&\quad= \left|c_p^{(i)}(T_0+T+\Pi+Q+\Delta_{qp}-T_0) - Q - c_q^{(i)}(T_0+T+\Pi-T_0)\right| \\
&\quad\leq \left|c_p^{(i)}(T_0+Q+\Delta_{qp}) - Q - c_q^{(i)}(T_0)\right| + \rho\,|T-T_0+\Pi| \\
&\quad\leq \hat{\epsilon} + 2\rho R \qquad \text{(from Lemma 2.5).} \quad ■
\end{aligned}$$

Lemma 2.7: Suppose S1 holds for i, and nodes p, q and r are non-faulty up to time $T^{(i+2)}$. Also suppose the CP at r uses a reliable broadcast to convey its clock value, and suppose p and q use CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew between their clock and r's clock. Let $\bar{\Delta}_{rp}(a)$ and $\bar{\Delta}_{rq}(b)$ be the estimated skews at p and q, respectively, at the end of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

line 9 in CLOCK_CORRECTION with threshold $\Delta = \frac{(\delta + \epsilon + \frac{\rho}{2} \cdot U)}{(1 - \frac{\rho}{2})}$. Let $Q^{(a)}$ and $Q^{(b)}$ denote the total transit delay computed at p and q, respectively, from the copy used to estimate the skew. Then for any T in $R^{(i)}$:

$$\left| c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) - [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)] \right| \leq 2(\hat{\epsilon} + 2\rho R) + \frac{\rho}{(1 - \frac{\rho}{2})}(\delta + \epsilon + U).$$

Proof: It follows from Corollary 2.1 that when Δ is chosen as in the hypothesis, $|\Delta_{rq}(a)| \leq \Delta$ and $|\Delta_{rq}(b)| \leq \Delta$. So $\bar{\Delta}_{rp}(a) = \Delta_{rp}(a)$ and $\bar{\Delta}_{rq}(b) = \Delta_{rq}(b)$, where $\Delta_{rp}(a)$ and $\Delta_{rp}(b)$ are the estimated skews at the end of line 6 of CLOCK_CORRECTION. For notational simplicity, let $\Delta_{rp} \equiv \Delta_{rp}(a)$, and $\Delta_{rq} \equiv \Delta_{rq}(b)$. Then,

$$\begin{aligned}
\left| c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) \right. & - \left. [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)] \right| \\
&= \left| c_p^{(i)}(T) + \Delta_{rp} + Q^{(a)} - Q^{(a)} + Q^{(b)} - [c_q^{(i)}(T) + \Delta_{rq} + Q^{(b)}] \right| \\
&\leq \left| c_p^{(i)}\left(T + \Delta_{rp} + Q^{(a)}\right) - c_q^{(i)}\left(T + \Delta_{rq} + Q^{(b)}\right) + Q^{(b)} - Q^{(a)} \right| \\
&\qquad + \frac{\rho}{2}\left|\Delta_{rp} + Q^{(a)}\right| + \frac{\rho}{2}\left|\Delta_{rq} + Q^{(b)}\right| \\
&\leq \left| c_p^{(i)}\left(T + \Delta_{rp} + Q^{(a)}\right) - c_q^{(i)}\left(T + \Delta_{rq} + Q^{(b)}\right) + Q^{(b)} - Q^{(a)} \right| \\
&\qquad + \frac{\rho}{(1 - \frac{\rho}{2})}(\delta + \epsilon + U) \\
&\leq \left| c_p^{(i)}\left(T + \Delta_{rp} + Q^{(a)}\right) - c_r^{(i)}(T) - Q^{(a)} \right| \\
&\qquad + \left| c_q^{(i)}\left(T + \Delta_{rq} + Q^{(b)}\right) - c_r^{(i)}(T) - Q^{(b)} \right| \\
&\qquad + \frac{\rho}{(1 - \frac{\rho}{2})}(\delta + \epsilon + U) \\
&\leq 2(\hat{\epsilon} + 2\rho R) + \frac{\rho}{(1 - \frac{\rho}{2})}(\delta + \epsilon + U) \quad \text{from Lemma 2.6.} \quad \blacksquare
\end{aligned}$$

Lemma 2.8: Suppose S1 holds for i, and nodes p and q are non-faulty up to time $T^{(i+2)}$. Also suppose that p and q use CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew between their clock and the clock at node r. Let $\bar{\Delta}_{rp}(a)$ and $\bar{\Delta}_{rq}(b)$ be the estimated skews at p and q respectively at the end of line 9 in CLOCK_CORRECTION. Then for any T in $R^{(i)}$:

$$\left| c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) - [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)] \right| \leq \delta + 2\Delta.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: By the hypothesis that S1 holds for i, we have

$$\left|c_p^{(i)}(T) - c_q^{(i)}(T)\right| < \delta.$$

Since $\bar{\Delta}_{rp}$ and $\bar{\Delta}_{rq}$ are by definition no larger than Δ, the result follows. ■

Theorem 2.4: If A1–A5 hold and if

1. $3m < N$ and the interconnection network is $2m+1$ connected
2. $\delta > \max\left\{\frac{(N-m)}{N}\left(2(\tilde{\epsilon}+2\rho R)+\frac{\rho(\delta+\epsilon+U)}{(1-\frac{\rho}{2})}\right)+\frac{m}{N}(\delta+2\Delta)+\rho(R+\Delta), \delta_0+\rho R\right\}$.

then algorithm presented in the previous section satisfies Conditions S1 and S2 with $\Sigma = \Delta$.

Proof: Let Δ_p and Δ_q be the correction computed at p and q respectively at the end of line 12 in CLOCK_CORRECTION. Condition S2 holds because the correction to the local clock is the average of N terms, each less than Δ. Condition S1 can be proved by induction on i. For $i = 0$, A1 implies that two non-faulty clocks that are synchronized to within δ_0 at time $T^{(0)}$ will remain synchronized to within $\delta_0 + \rho R$ at time $T^{(1)} = T^{(0)} + R$. Condition S1 follows from A1 and the hypothesis.

Now suppose that S1 holds for i. We need to prove that it holds for $i+1$. Assume that p and q are both non-faulty up to time $T^{(i+2)}$. For clarity of presentation, we will use $\bar{\Delta}_{rp}$ and $\bar{\Delta}_{rq}$ without explicitly specifying the nodes through which the clock values were relayed from r to p and q. Also let T denote $T^{(i+1)}$. Then for any T' in $R^{(i+1)}$ we have

$$\begin{aligned}
\left|c_p^{(i+1)}(T') - c_q^{(i+1)}(T')\right| &\leq \left|c_p^{(i+1)}(T) - c_q^{(i+1)}(T)\right| + \rho R \qquad \text{by A2} \\
&= \left|c_p^{(i)}(T+\Delta_p) - c_q^{(i)}(T+\Delta_q)\right| + \rho R \qquad \text{from the algorithm} \\
&\leq \left|c_p^{(i)}(T) + \Delta_p - [c_q^{(i)}(T) + \Delta_q]\right| + \rho R + \frac{\rho}{2}(\Delta_p + \Delta_q) \qquad \text{from Lemma 2.3} \\
&= \left|\frac{1}{N}\sum_{r=1}^{N}(c_p^{(i)}(T) + \bar{\Delta}_{rp} - [c_q^{(i)}(T) + \bar{\Delta}_{rp}]\right| + \rho(R+\Delta) \\
&\leq \frac{1}{N}\sum_{r=1}^{N}\left|(c_p^{(i)}(T) + \bar{\Delta}_{rp} - [c_q^{(i)}(T) + \bar{\Delta}_{rp}]\right| + \rho(R+\Delta) \\
&\leq \frac{(N-m)}{N}\left(2(\tilde{\epsilon}+2\rho R)+\frac{\rho(\delta+\epsilon+U)}{(1-\frac{\rho}{2})}\right)+\frac{m}{N}(\delta+2\Delta)+\rho(R+\Delta). \qquad \blacksquare
\end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Initial Synchronization

There are several ways of ensuring that A1 is satisfied. The simplest scheme is to assume that all nodes are non-faulty at startup and select one of them as a master. After making sure all nodes are operational, the master can broadcast its clock value to all other nodes. The CPs can use a scheme similar to the one described above to correct the local clock based on the value they receive from the master.

If it is not possible to ensure that all nodes are non-faulty at startup, then a more complicated scheme is necessary for initial synchronization. First, use an existing initial synchronization algorithm such as the ones in [22, 43]. At the end of this phase the worst-case skew will be larger than the maximum transit delay. Repeatedly use the algorithm proposed here until the skew is below the required limit. Before each repetition, calculate the worst-case skew at that iteration and set the threshold Δ to that value. This scheme will converge exponentially as long as the total number of nodes $N > 3m + 1$, which will usually hold in large distributed systems. The proof that this will converge follows easily from Theorem 2.4.

Numerical examples

The expression for δ in Theorem 2.4 can be simplified by making certain approximations that hold in most distributed systems, namely, $\rho \cdot U \ll \delta$, $\Delta \ll R$, $\epsilon \ll U$, and $\rho \cdot U \ll \epsilon$. These approximations are reasonable, since, for most practical applications, ϵ is around 20–30 μs, ρ is around 10^{-6}, U is around 200–300 ms and R is in the order of seconds. From the exact expression, it also clear that $\delta \ll U$. Using these approximations, the second hypothesis in Theorem 2.4 can be restated as

$$\delta > \max\left\{\frac{2(N-m)(\epsilon+2\rho R)+2m\epsilon+\rho RN}{(N-3m)}, \delta_0+\rho R\right\}. \tag{2.7}$$

One can now characterize δ^*, the minimum worst-case skew that can be guaranteed in any given system using our scheme, i.e., δ^* is the minimum value of δ that satisfies Equation (2.7) given N, m, ρ, ϵ, and U. Since N, m, ρ, ϵ and U are characteristics of the system, the only parameter that can be controlled is R. It is easy to verify that given R the minimum δ which

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

satisfies the Equation (2.7) increases with R. From A5 we know $R \geq N \cdot U$ and therefore,

$$\delta^* = \max\left\{\frac{2(N-m)(\epsilon+2\rho NU)+2m\epsilon+\rho N^2U}{(N-3m)},\ \delta_0+\rho NU\right\}. \tag{2.8}$$

Clearly $\delta^* \ll U$ for most practical systems. As a result, the minimum worst-case skew that can be guaranteed is substantially less than the minimum worst-case skew guaranteed by other software synchronization algorithms [22, 36, 43, 64] where the worst-case skews are at least as large as U. This implies the skews that can be guaranteed in our scheme are very tight as compared to existing software synchronization algorithms.

In addition to this significant advantage, another major advantage of our scheme is that δ^* increases gradually with U. This should be contrasted to an almost linear increase in the minimum worst-case skew with respect to U in the existing software synchronization algorithms. Some of these aspects are illustrated with examples for C-wrapped hexagonal mesh and hypercube topologies.

A C-wrapped hexagonal mesh topology [1, 65] is a regular, homogeneous graph in which each node has six neighbors. The graph can be visualized as a simple hexagonal mesh with wrap links added to the nodes on the periphery. A simple hexagonal mesh looks like a set of concentric hexagon with a central node, where each hexagon has one more node on each edge than the one immediately inside of it. It can be defined succinctly as follows.

Definition 2.6: A *C-wrapped hexagonal mesh* of dimension e is comprised of $3e(e-1)+1$ nodes, labeled from 0 to $3e(e-1)$, such that each node s has six neighbors $[s+1]_{3e^2-3e+1}$, $[s+3e-1]_{3e^2-3e+1}$, $[s+3e-2]_{3e^2-3e+1}$, $[s-1]_{3e^2-3e+1}$, $[s-3e+1]_{3e^2-3e+1}$, and $[s-3e+2]_{3e^2-3e+1}$, where $[a]_b$ denotes a mod b.

Figure 2.11 shows a C-wrapped hexagonal mesh of dimension 3. This topology is currently being used in the HARTS [1] and the Mayfly systems [4]. An attractive feature of this topology is that it supports a simple reliable broadcast mechanism that is resilient to two Byzantine faults with minimal overhead [27].

A hypercube topology is also a regular, homogeneous graph. Each node in an n-dimensional hypercube, Q_n, has n neighbors. A Q_n also supports a simple reliable broadcast algorithm that delivers n copies of the message through node-disjoint paths [47]. It has been used as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

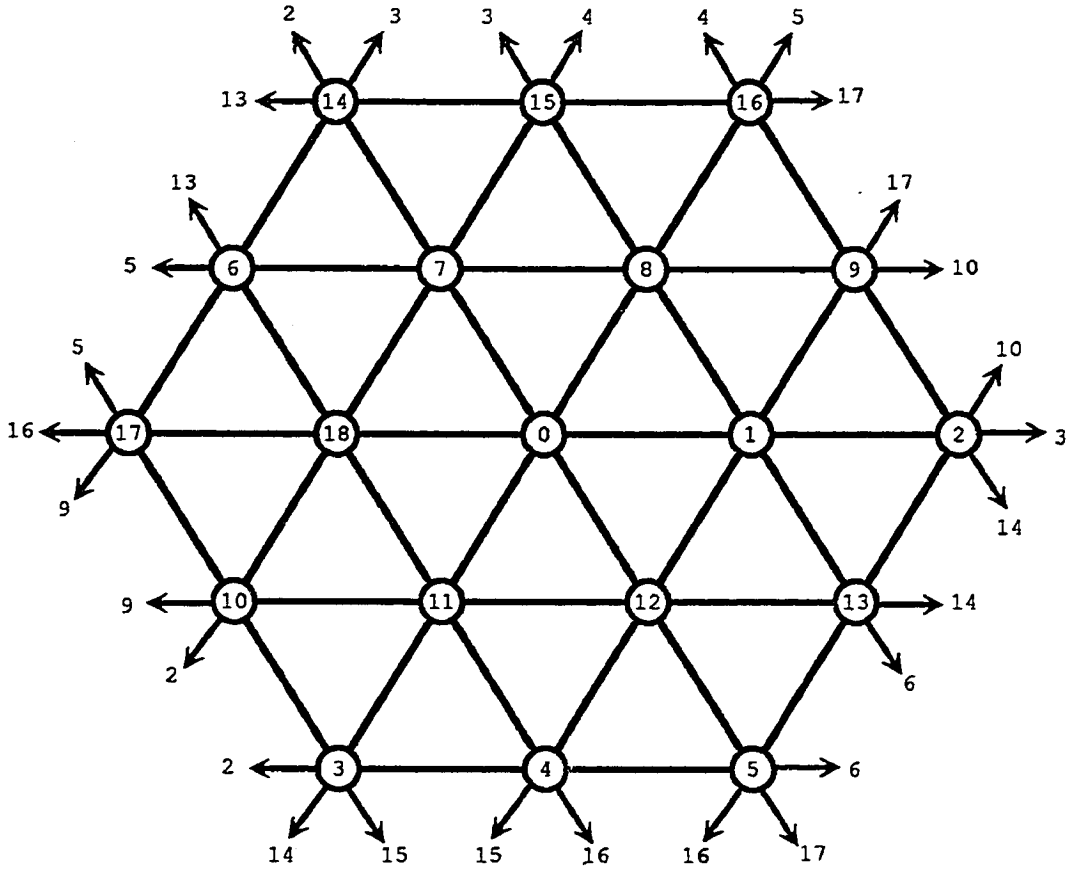

Figure 2.11: A C-wrapped hexagonal mesh of dimension 3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

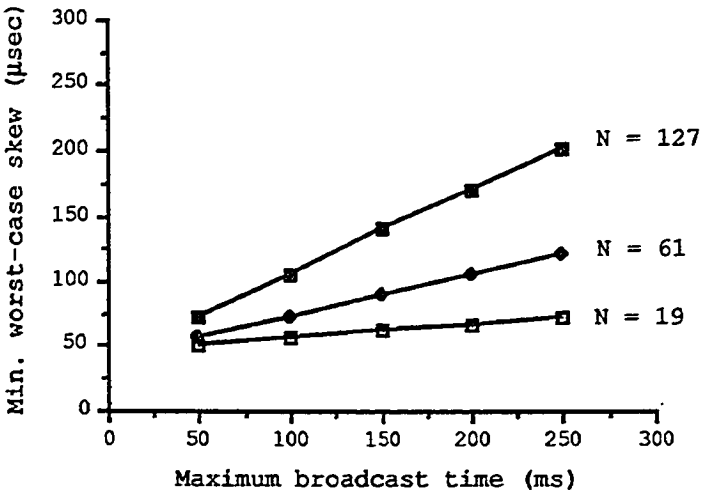

Figure 2.12: δ^* versus U for a C-wrapped hexagonal mesh when $m = 2$, $\rho = 10^{-6}$, $\epsilon = 20\mu s$

an interconnection topology in both research [54] and commercial systems by Intel, NCUBE, Floating Point Systems, Ametek, Thinking Machine, to name a few. It can be defined formally as follows.

Definition 2.7: A *hypercube* of dimension n is comprised of 2^n nodes, labeled from 0 to $2^n - 1$, such that two nodes are neighbors if and only if the binary representation of their labels differ by one and only one bit.

Figures 2.12 and 2.13 show the variation in δ^* with respect to U for the C-wrapped hexagonal mesh and hypercube topologies, respectively. It is evident from these figures that skews in the order 100 μs can be guaranteed even in large systems. Also evident is the fact that when U varies from 50 ms to 250 ms the minimum worst-case skew in our scheme varies from 50 μs to 700 μs as compared to 50 ms to 250 ms in [22, 36, 43, 64].

Figures 2.14 and 2.15 show the variation in the worst-case skews that can be guaranteed in the above two topologies with respect to maximum time required for a broadcast when the resynchronization interval R is significantly greater than the minimum resynchronization interval required, i.e., when $R > N \cdot U$. These two figures illustrate how the worst-case skews can be made insensitive to maximum time required for broadcast at the cost of tightness of synchronization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

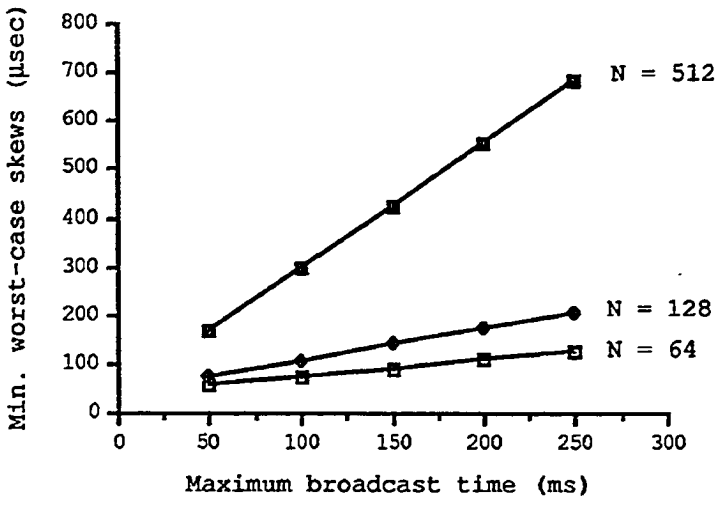

Figure 2.13: δ^* versus U for a hypercube when $m = 2$, $\rho = 10^{-6}$, $\epsilon = 20\mu s$

This is in sharp contrast to the existing software synchronization algorithms where the worst-case skews increase linearly with the maximum time required for a broadcast.

2.6 Discussion

Two schemes were proposed for synchronizing the local clocks on the nodes of a distributed system. The first scheme was a pure hardware solution that can be used to tightly synchronize large distributed systems. This scheme requires considerably fewer interconnections than the existing hardware solutions. It also accounts for the transmission delay between the clocks, thereby allowing the clocks to be physically far apart from each other.

The second scheme was a software solution which requires some hardware support at each node. This support is similar to the commonly available support to ensure timely delivery of messages. The guaranteed worst-case skews in this scheme are in the order of 100–200 μs as compared to 20–30 ns in the hardware scheme. The skews still about two to three orders of magnitude tighter than other software solutions. Additionally, the worst-case skews are insensitive to maximum time required for a broadcast. Furthermore, the broadcast of clock values from

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

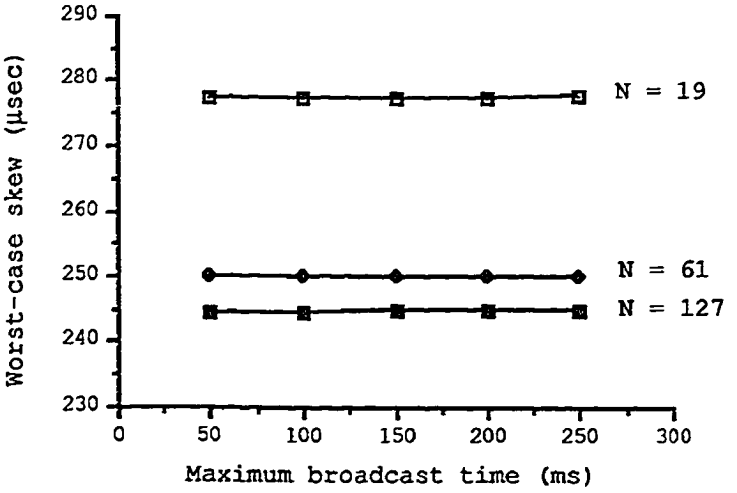

Figure 2.14: δ versus U for a C-wrapped hexagonal mesh when $R = 40$ s, $m = 2$, $\rho = 10^{-6}$, and $\epsilon = 20\mu s$

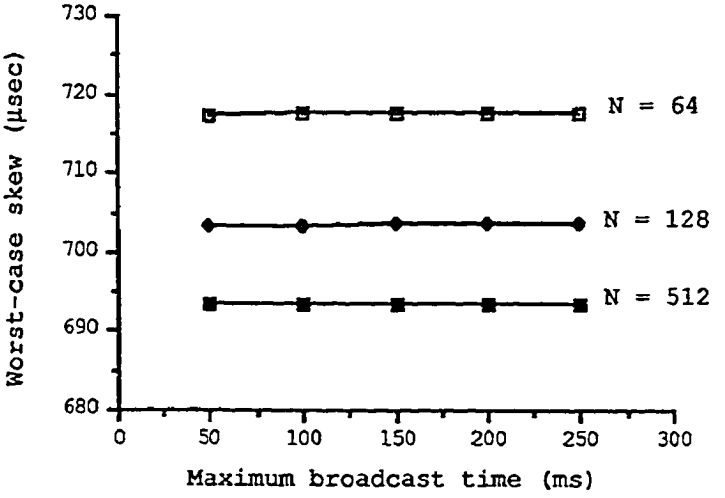

Figure 2.15: δ versus U for a hypercube when $R = 130$ s, $m = 2$, $\rho = 10^{-6}$, and $\epsilon = 20\mu s$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

different nodes are staggered throughout the resynchronization interval instead of being lumped at the end of the interval. This avoids sudden increase in network traffic caused by almost simultaneous broadcast of clock values from all nodes.

By using either of the above schemes it is possible to establish a global time base among all the nodes of a distributed system. The problem of establishing a time base among the components in a VLSI circuit is addressed in the following chapter.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

CLOCK DISTRIBUTION WITHIN VLSI CIRCUITS

3.1 Introduction

The focus of this chapter is on node level synchronization which deals with distributing a clock signal to the components within a node of a distributed system. This work complements the issues discussed in the previous chapter for establishing a global time base in the system. Since advances in semiconductor technology have now made it possible to integrate complete and powerful nodes on a single VLSI chip, the emphasis of this chapter will be on distributing a clock signal to the functional elements in a VLSI circuit.

In the context of a VLSI circuit, a *clock skew* can be defined as the undesired difference between the arrival times of the clock signal at any pair of functional elements where these arrival times are expected to be identical or separated by less than a specified interval. *System clock skew* is the maximum of these undesired differences when considering all pairs of functional elements. It is essential to control the system clock skew to an acceptable small fraction of the clock period because it can otherwise hamper the correct and dependable operation of the circuit.

There are four main factors responsible for this clock skew: (i) differences in the length of lines that deliver the signal to two different functional elements, (ii) differences in delays through any active elements inserted in these clock lines (e.g., clock buffers), (iii) differences in line parameters (e.g., resistivity, dielectric constant) that determine the line time constant, and (iv) differences in threshold voltages of different elements. These factors can be eliminated by using an appropriate clock distribution scheme.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Wann and Franklin [69] proposed a symmetric distribution scheme that ensures equal line lengths for all elements. However, this scheme is applicable only when all the elements are identical and placed in the form of a symmetric array. This scheme is not suitable for distributing clocks in a general VLSI circuit where the functional elements are typically of different sizes and where the placement of the elements is governed by their interconnection and usually not in an array form. To the best of our knowledge, there exists no distribution scheme that will guarantee equal line lengths under this non-symmetric placement.

Several studies have been undertaken to minimize the clock skew due to the other three factors [17, 60, 68]. The elimination of process-dependent skew in CMOS chips by careful adjustment of FET parameters is discussed in [60]. A clock distribution scheme for a standard cell/macrocell design is presented in [17]. The chip is first partitioned hierarchically into several functional elements. Then, accurate resistive and capacitive interconnect parasitics at the intra-element level are extracted and used to characterize the interconnect impedances between the functional elements [45]. The clocks to each functional element are distributed through a central finely-tuned clock buffer circuit that compensates for the variation in interconnect and fan-out loading of each of the functional elements. These two schemes have the disadvantage of being very sensitive to the fabrication technology. In [5], a method was introduced to reduce clock delays in VLSI structures by using a folding technique to derive a placement for the clock buffer circuits. However, the scheme results in an appreciable reduction in delays only in the case of long distribution lines.

In this chapter a non-symmetric distribution scheme is proposed that will minimize the clock skew due to line lengths for functional elements with different sizes and arbitrary placement. It also accounts for the difference in delays caused by the clock buffers. Unlike the other related work in this area, both delay and skew are considered in determining routes for the clock. This is important because layouts that have minimum skew might have long clock lines that result in large delays, and hence, degraded performance. On the other hand, layouts that have minimum delay (i.e., minimum line length to each element) might have large skews because of large difference in line lengths. This can affect the correctness of the circuit. The objective used here for determining the clock layout minimizes the skew subject to minimum longest delay.

The scheme assumes that a floorplan of the modules is given. This floorplan would have been determined by using some placement algorithm based on the signals interconnecting the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sub-modules that constitute that module. For this floorplan, the proposed scheme identifies an entry point from which the clock is distributed and the layout of the clocks from the entry point to each of the sub-modules. The entry point is selected in such a way that the delay to the farthest sub-module from that entry point is minimum. This will minimize the maximum delay in distributing the clock signals to the sub-modules. The optimal layout of the clock lines from the selected entry point is determined by an exhaustive search of all paths with intelligent pruning. The main advantage of the scheme is that the computation of the optimal layout can be easily parallelized. This makes it suitable for determining the clock layout with a parallel computer system.

This chapter is organized as follows. The clock layout problem is formally defined in the following section. An algorithm to select the entry point is presented in the third section. The fourth section describes an algorithm to determine the optimal clock layout from the selected entry point. An implementation of the proposed scheme is discussed in the fifth section. The chapter concludes with the sixth section.

3.2 Problem Formulation

As mentioned earlier, the two foremost factors responsible for clock skew in a VLSI circuit are the difference in the length of lines for delivering clock signals to the different functional elements and the difference in the number of clock buffers inserted in the clock lines [69]. These two factors can be eliminated by using an appropriate layout of the clock lines. There are two basic approaches for determining this layout: (i) a *flat routing* scheme, and (ii) a *hierarchical routing* scheme. In the flat routing scheme, the route of clock signals to all the processing elements are considered simultaneously, whereas in the hierarchical scheme, the processing elements are first grouped into several sub-modules, then sub-modules into modules, and so on. The clock is distributed recursively for the different levels of hierarchy. The main advantage of the hierarchical scheme is the reduction in complexity and the execution times for distributing the clock signals. However, the route thus obtained might be sub-optimal because local optimality does not necessarily imply global optimality.

The scheme proposed here and also in [48] is a combination of both approaches. As in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a flat routing scheme, the distribution of clock signals to all the sub-modules is considered simultaneously. However, the complexity of routing is reduced significantly by making use of the hierarchy created by the clock buffers. This hierarchy is used to partition the distribution of clock signals into several levels. The routing of signals at all levels is carried out in parallel. The delays introduced by the clock buffers are also taken into account when determining the layout of clock lines.

A *module* is a group of processing elements to which the clock signal is being distributed. A module may be composed of all the processing elements in the VLSI circuit (flat routing scheme), or only some of the processing elements in the circuit (hierarchical routing scheme). The processing elements contained in the module will be referred to as the *sub-modules*. A *floorplan* is a relative placement of the sub-modules in the routing area. A *channel* is a rectangular zone of empty area between the sub-modules through which the signals can be routed. A *track* is a sub-unit within a channel through which a single signal can be routed. A channel is said to be *horizontal* (*vertical*) if the tracks within the channel are horizontal (vertical). A channel is said to be *peripheral* if it lies in the periphery of the floorplan. There are three possible types of intersection between a horizontal and a vertical channel: L-type, T-type, and +-type. A *decision point* is the point of a T-type or a +-type intersection.

The floorplan can be represented by an undirected, weighted graph referred to as a *placement graph*. The vertices of the placement graph are the decision points in the floorplan, the input clock terminals of all the sub-modules, and the output clock terminals of all clock buffers. Clock buffers are sub-modules whose output drives the input clock signals of other sub-modules. Vertices v_l and v_k are connected by an edge if and only if there is a channel from v_l to v_k not containing any other vertex of the placement graph. Thus, there is a one-to-one correspondence between the edges of the placement graph and the channels in the floorplan. If a channel is viewed as a line instead of a region, then one can also associate a one-to-one correspondence between the points in the channel and the points on the corresponding edge. This is possible if it is sufficient to minimize the skew during global routing and the circuit can tolerate the additional skews that may be introduced when specific tracks are assigned to the clock lines. If the circuit cannot tolerate the additional skews, then the following heuristic can be adopted. Reserve a few adjacent tracks in each channel for the clock lines and route the clock lines before routing the other signals.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Once the clock lines have been routed, the unused tracks can be used for the other signals. If the floorplan has to be significantly altered due to incomplete routing of the other signals, then the algorithm has to be re-executed for the new floorplan.

The weight of an edge $\{v_l, v_k\}$ is equal to the physical distance in the channel between the two points corresponding to v_l and v_k except when either v_l or v_k is an input clock terminal of one of the sub-modules. It is therefore proportional to the delay that will be induced on a clock signal if that channel is used as a part of the clock layout. The weight of edges $\{v_l, v_k\}$ in which one of v_l or v_k is an input clock terminal will be used to account for the delays induced by clock buffers. The assignment of weights to these edges will be discussed later. One needs to consider not only the edges of the placement graph but also the line segment between two arbitrary points on the same edge. The line segment between two points u and v on the same edge will be denoted by $[u, v]$. The weight of $[u, v]$ is equal to the physical distance in the channel between the points corresponding to u and v. The weight of $[u, v]$ is sometimes referred to as the distance between u and v and is denoted by $d(u, v)$.

Let P be a floorplan of a module M that is composed of n sub-modules M_1, M_2, ..., and M_n. There is an entry point into the floorplan from which the clock is distributed to all the sub-modules. However, all sub-modules may not receive their clock signal directly from this entry point. Instead, some of the sub-modules will receive their clock signal indirectly from the output of some other sub-module (i.e., clock buffer) in M. For simplicity of presentation, introduce a fictitious clock buffer that drives the entry point to the floorplan and assume that the clock signals of all the sub-modules are driven by a clock buffer. For each sub-module M_i, $1 \leq i \leq n$, it is then possible to associate a unique level, *Level*(M_i), that is equal to the number of clock buffers in the route from the clock entry point to the sub-module. In other words, sub-modules directly driven from the entry point are said to be in *Level 1*, while those driven by sub-modules in Level 1 are said to be in *Level 2*, and so on. Let *max_level* $= \max_{1 \leq i \leq n}$ *Level*(M_i), and *Buffer*(M_i) be the clock buffer, say, M_j that drives the clock signal of M_i. Define *Buffer*(M_i) of a sub-module M_i at level 1 as the fictitious clock buffer that drives the clock entry point to the floorplan.

Definition 3.1: In a placement graph, a *simple path* from s to t is a finite sequence of distinct points $u_0 u_1 \cdots u_r$, $r \geq 1$, such that: (i) $u_0 = s$ and $u_r = t$, (ii) if $r > 1$, then u_1, u_2, ..., u_{r-1}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are vertices of the placement graph, and (iii) $\forall\, j \in \{0, \ldots, r-1\}$, u_j and u_{j+1} lie on a common edge.

We will sometimes use the term "a simple path from M_i to M_j" to refer to the simple path from the output clock terminal of M_i to the input clock terminal of M_j.

Definition 3.2: A *c-route* from an entry point e to a sub-module M is a concatenation of simple paths from e to B_1, B_1 to B_2, B_2 to B_3, ..., B_{l-1} to B_l, and B_l to M, where B_1, B_2, ..., B_l are such that $B_l = \textit{Buffer(M)}$ and $B_{j-1} = \textit{Buffer}(B_j)$ for all $l \geq j \geq 2$.

Definition 3.3: The sum of the distances between adjacent points in a c-route will be referred to as its *c-length.*

Definition 3.4: A *c-layout* from an entry point e is a tuple of c-routes from e to all sub-modules in the floorplan.

Let SM_l denote the set of all sub-modules in level l. Let $\eta_i^{P,e}$ denote the c-length of the shortest simple path from *Buffer(M_i)* to M_i in floorplan P when e is the clock entry point. Define $\mu_l^{P,e}$ to be $\max_{i \in SM_l} \eta_i^{P,e}$. We are now in a position to define the weights of the edges $\{v_l, v_k\}$ in the placement graph in which one of v_l or v_k is an input clock terminal of a sub-module. Let I_j be the input clock terminal of sub-module M_j. If *Level(M_j)* is *max_level* or if M_j is not a clock buffer, then the weight of the edge $\{v_k, I_j\}$ is equal to the physical distance between v_k and I_j in the channel, else the weight of the edge is equal to $\mu_{l+1}^{P,e} + b + c$, where $l = \textit{Level}(M_j)$, b is a constant corresponding to the delay introduced by M_j, and c is equal to the distance in the channel between the points v_k and I_j. The intuitive reason for assigning the weights to these edges in this manner is explained later in Example 3.1.

There are two main objectives in determining a c-layout. The first objective is to minimize the maximum difference in the c-length to the different sub-modules. The second objective is to minimize the c-length to all sub-modules. The first objective by itself is not sufficient for a good layout because a c-layout with minimum skew might have long c-routes that, in turn, result in large delays. Similarly, the second objective by itself is not sufficient for a good layout because a c-layout with minimum delay might have very large skews. Consequently, both these objectives are considered here when determining the c-layout.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

More formally, let E be the set of all points in the periphery of the floorplan. It is the set of points from which an entry point, e, is to be selected. Let $Y^{P,e}$ denote a c-layout from e in the floorplan P. Let $L_i(Y^{P,e})$ denote the c-length of the simple path from *Buffer(M_i)* to M_i in the c-route from e to M_i in $Y^{P,e}$. Then clock distribution problem can be stated as:

Determine e such that

$$\mu_1^{P,e} \leq \mu_1^{P,s} \qquad \forall s \in E$$

Determine $Y^{P,e}$ that minimizes

$$\sum_{i \in SM_l} \left| L_i(Y^{P,e}) - \mu_l^{P,e} \right| \qquad \forall 1 \leq l \leq max\_level.$$

Since $\mu_1^{P,e}$ represents the delay from the entry point to the farthest block (from that entry point), the first criterion minimizes the longest delay. The second criterion tries to increase the length of clock lines from the entry point to all the sub-modules to equal line length to the farthest sub-module. Consequently, the overall objective identifies a layout of clock lines that has minimum skew subject to minimum worst case delay.

A drawback of our scheme is that the delays introduced by clock buffers have to be estimated prior to executing the proposed algorithm. The difference between the actual and the estimated delays of these clock buffers can result in additional skews. These skews can be reduced by using fewer clock buffers in the circuit. This approach has been used in the MIPS architecture [28] where a single large clock buffer is used to drive the entire circuit. In our scheme, this approach will increase the execution time required to determine the optimal c-layout. However, the scheme will account for the skews due to difference in line lengths which is a major factor contributing to the skews.

Example 3.1: Consider the floorplan in Figure 3.1. The module M is comprised of ten sub-modules M_1, M_2, ..., M_{10}. Sub-modules M_2 and M_6 are clock buffers whose input is driven directly from the clock entry point of M. In addition, the input of sub-module M_8 is also driven directly from the entry point. M_2 drives the input clock signals of sub-modules M_1, M_3, M_4, and M_5 while M_6 drives the input clock signals of sub-modules M_7, M_9, and M_{10}. The sub-modules have been placed within the given area in such a way that there is ample space to route all the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

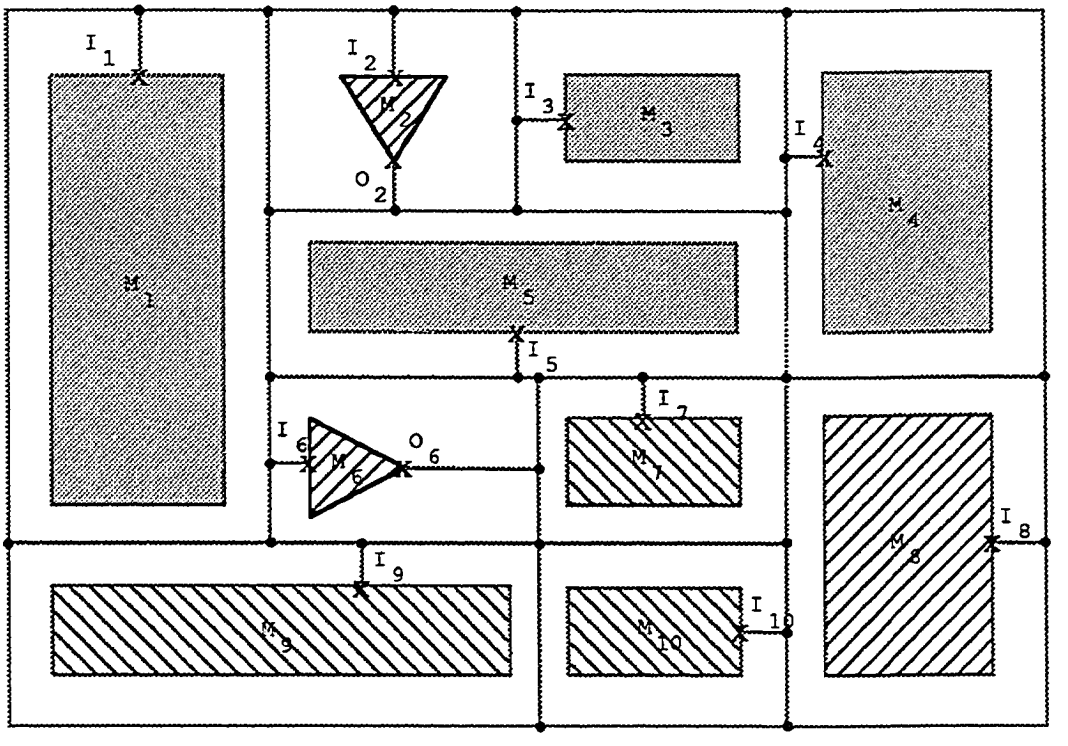

Figure 3.1: Floorplan of sub-modules in M.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

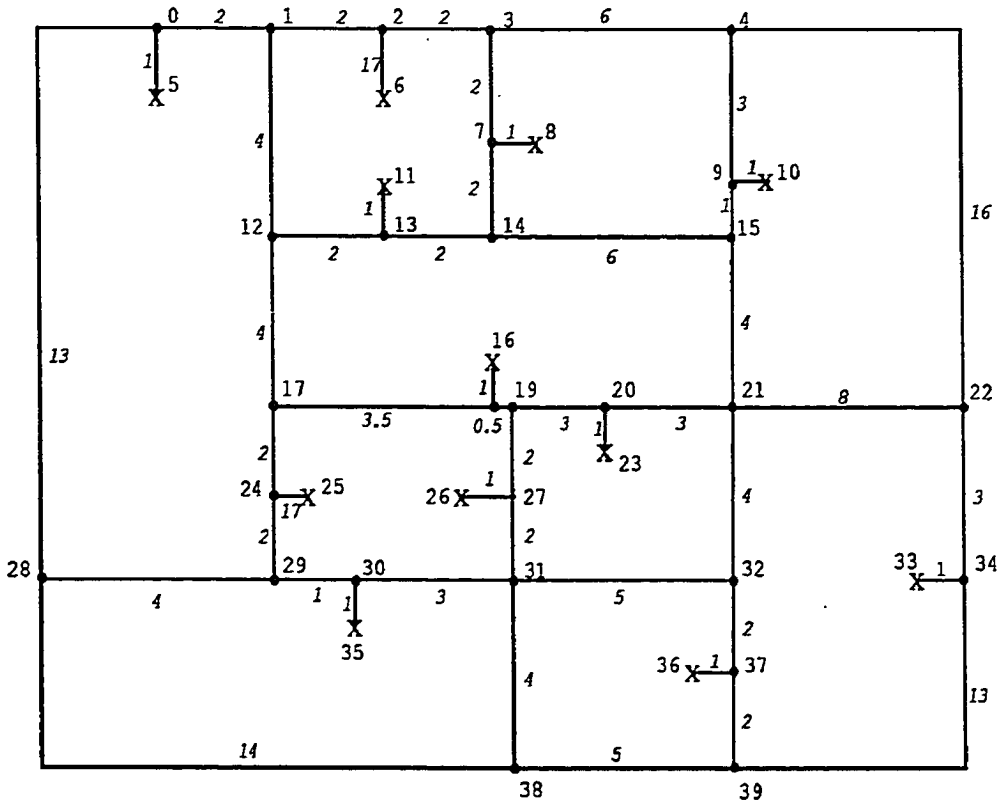

Figure 3.2: Placement graph.

signals between them. The problem is to determine a clock entry point for the module M, the route of the clocks from this entry point to the sub-modules M_2, M_6 and M_8, and the route of the clocks from the output of M_2 and M_6 to the input clock terminals of the other sub-modules.

The placement graph for Figure 3.1 is shown in Figure 3.2. In this figure the vertices corresponding to the decision points are indicated by • whereas those corresponding to the clock terminals of the sub-modules are indicated by ×. The weight of the edges in this graph are also indicated in italics.

This example illustrates the reason for assigning larger weights to the edges $\{v_j, v_k\}$ in which either v_j or v_k is an input terminal of a clock buffer. Consider the sub-module M_7. The clock input to M_7 is not directly driven by the clock entry point but by the output of M_6. Therefore, total delay from the clock entry point to the clock input of M_7 is equal to the sum of the delays from the entry point to the input of M_6, delay introduced by M_6 and the delay from the output of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

M_6 to the input of M_7. It follows from the objective for determining the c-layout that the routing delay from the output of M_6 to the input of M_7 will be almost equal to $\mu_2^{P,e}$. If b is the delay introduced by M_6, then $\mu_2^{P,e} + b$ is almost equal to the delay from the input of M_6 to the input of M_7. If c is the distance between the end-points of the edge on the channel, then by assigning $\mu_2^{P,e} + b + c$ as the weight of the edge $\{24, 25\}$ we can determine the c-route from the entry point to M_6 in parallel with the c-route from M_6 to M_7. In this example, b is assumed to be 5 and c is assumed to be 1. ■

3.3 Selection of the Entry Point

As discussed in the previous section that there are two aspects to the clock distribution problem. The first problem is to identify one of the points as the entry point, and the second problem is to determine a c-layout for the clock. An algorithm to select the entry point is discussed in this section. The next section will address the problem of determining the c-layout given the entry point. The notations used in the rest of this chapter are summarized below. Some of these notations were introduced earlier but they are included here for completeness.

M	Module within which the clock is being routed.
M_i	i^{th} sub-module of M.
P	The given floorplan of sub-modules in M.
G^P	Placement graph for floorplan P.
E	Set of points in the periphery of the floorplan.
SM_l	Set of sub-modules in level l.
$\eta_i^{P,x}$	The c-length of the shortest simple path from *Buffer(M_i)* to M_i when x is the entry point.
$\mu_l^{P,x}$	$\max_{i \in SM_l} \eta_i^{P,x}$.
$R_i^{P,x}$	A c-route to M_i from source point x in P.
$Y^{P,x}$	A c-layout from x in floorplan P.
$L_i(Y)$	The c-length of simple path from *Buffer(M_i)* to M_i in the c-layout $Y^{P,e}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$d(x,y)$ Distance between points x and y on the same edge in G^P.

$[x,y]$ Line segment between points x and y on the same edge.

The goal is to select a point in E that minimizes the longest delay, i.e., select $e \in E$ such that $\mu_1^{P,e} \leq \mu_1^{P,s}$ for all $s \in E$. Clearly, an algorithm to select such an entry point would depend on the variation of $\mu_1^{P,s}$ with respect to $s \in E$. Theorem 3.1 characterizes this variation.

Without loss of generality any point in E can be chosen as the origin and the other points can be represented as the distance from the origin along the periphery of the floorplan. In other words, E can be represented as a set of real numbers from 0 to $\|E\|$, where $\|E\|$ is the perimeter of the floorplan. Let $D(s_1)$ denote the distance of s_1 from the origin along the periphery.

Theorem 3.1: If $s_1 \in E$ and $s_2 \in E$ are two points on the same edge of a placement graph with $D(s_1) < D(s_2)$, then for all $x \in [s_1, s_2]$ and all $M_i \in SM_1$ the following equation is satisfied.

$$\eta_i^{P,x} = \begin{cases} \eta_i^{P,s_1} + D(x) - D(s_1) & \text{if } D(x) - D(s_1) \leq \dfrac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2} \\ \eta_i^{P,s_2} + D(s_2) - D(x) & \text{if } D(x) - D(s_1) > \dfrac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2} \end{cases}$$

Proof: There are three possible cases. First, there is a shortest c-route, say C, from s_1 to M_i that contains the edge $\{s_1, s_2\}$. In this case, all $x \in [s_1, s_2]$ lie on C. So the part of C from x to M_i is a shortest c-route from x to M_i. The result then follows because $d(x, s_2) = D(x) - D(s_2)$ for all $x \in [s_1, s_2]$.

Second, there is a shortest c-route from s_2 to M_i that contains the edge $\{s_1, s_2\}$. The proof is similar to the first case except the roles of s_1 and s_2 are reversed.

Third, there is no shortest path from s_1 and s_2 that contains the edge $\{s_1, s_2\}$. In this case, there is a point $z \in [s_1, s_2]$ such that for all $x \in [s_1, z]$ the shortest c-route from z to M_i is the concatenation of the segment $[z, s_1]$ and the shortest c-route from s_1 and for all $x \in (z, s_2]$ the shortest c-route is the concatenation of the segment $[z, s_2]$ and the shortest c-route from s_2. The result follows easily from this observation.

Note that the fourth case where the shortest c-routes from both s_1 and s_2 contain the edge $\{s_1, s_2\}$ is not possible. ■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Corollary 3.1: Let $s_1 \in E$ and $s_2 \in E$ be two points on the same edge of a placement graph such that $\eta_i^{P,s_1} > \eta_j^{P,s_2}$. Then the functions $\eta_i^{P,x}$ and $\eta_j^{P,x}$ of x intersect in the interval $[s_1, s_2]$ iff $\frac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2} < \frac{\eta_j^{P,s_2} - \eta_j^{P,s_1} + D(s_2) - D(s_1)}{2}$. Furthermore, if they intersect, then point of intersection is given by $\frac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2} + \frac{\eta_i^{P,s_1} - \eta_j^{P,s_1}}{2}$.

Proof: It follows from Theorem 3.1 that $\eta_i^{P,x}$ is a straight line of slope $+1$ from $D(s_1)$ to $D(s_1) + \frac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2}$ and a straight line of slope -1 from $D(s_1) + \frac{\eta_i^{P,s_2} - \eta_i^{P,s_1} + D(s_2) - D(s_1)}{2}$ to $D(s_2)$. Similarly, $\eta_j^{P,x}$ is a straight line of slope $+1$ from $D(s_1)$ to $D(s_1) + \frac{\eta_j^{P,s_2} - \eta_j^{P,s_1} + D(s_2) - D(s_1)}{2}$ and a straight line of slope -1 from $D(s_1) + \frac{\eta_j^{P,s_2} - \eta_j^{P,s_1} + D(s_2) - D(s_1)}{2}$ to $D(s_2)$. The result follows from this observation. ■

The algorithm to determine the entry point is based on Theorem 3.1 and Corollary 3.1. The basic idea of the algorithm is as follows. Let $s_1, s_2, \ldots, s_n$ be the decision points in E such that $D(s_i) < D(s_{i+1})$ for $1 \leq i \leq n-1$. Let $s_{n+1} \equiv s_1$. First determine the shortest c-routes to the sub-modules in level 1 from each of the points $s_1, s_2, \ldots, s_n$. Then, start at s_1 and move along the periphery of the floorplan from s_1 to s_2, s_2 to s_3, $\ldots s_{n-1}$ to s_n, and then back to s_1. To go from s_i to s_{i+1} it may be necessary to move through several other intermediate points. Each move is comprised of determining the best entry point based on the path already traveled and then selecting the next point to move. The pseudo-code of the algorithm is shown in Figure 3.3.

Theorem 3.2: Algorithm ENTRY is correct, i.e., $\mu_{opt} \leq \mu_1^{P,s}$ for all $s \in E$.

Proof: Consider an execution of procedure select. After executing the first two statements, s will be equal to u and i will be such that $M_i \in SM_1$ is the farthest sub-module from u. Suppose that the execution of the algorithm is correct till this point, i.e., $\mu_{opt} = \min_{s \in [s_1, u]} \mu_1^{P,s}$. We will show by induction on the executions through the while loop that the algorithm is correct at the end of procedure select.

Consider an execution through the while loop. There are two possible cases. First case, $D(s) - D(u) < \frac{\eta_i^{P,v} - \eta_i^{P,u} + D(v) - D(u)}{2}$. In this case, from Theorem 3.1, $\eta_i^{P,x}$ is increasing at u, and hence, so is $\mu_1^{P,x}$. Consequently, there is no need to update μ_{opt}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Global Variables
  s_{opt}: the best entry point;
  μ_{opt}: $\mu_1^{P,s_{opt}}$;

procedure select(u, v)
  Variables
  z_{ij}: point of intersection between $\eta_i^{P,x}$ and $\eta_j^{P,x}$;
begin
  $s = u$;
  $i = \arg\max\{\eta_j^{P,s} : M_j \in SM_l\}$; /* value of j that results in maximum $\eta_j^{P,s}$ */
  while ($s \neq v$) **do**
    if $\left(D(s) - D(u) < \dfrac{\eta_i^{P,v} - \eta_i^{P,u} + D(v) - D(u)}{2}\right)$ **then** /* $\eta_i^{P,x}$ is increasing at s */
      $s = D^{-1}\left(D(u) + \dfrac{\eta_i^{P,v} - \eta_i^{P,u} + D(v) - D(u)}{2}\right)$;
    else
      $s = \min\{z_{ij} : M_j \in SM_1, z_{ij} > s\}$;
      if ($\mu_{opt} > \mu_1^{P,s}$) **then**
        $\mu_{opt} = \mu_1^{P,s}$; $s_{opt} = s$;
      endif
      $i = \arg\min\{z_{ij} : M_j \in SM_1, z_{ij} > s\}$; /* value of j that results in minimum z_{ij} */
    endif
  endwhile
  return(μ_{opt}, s_{opt});
end;

/* Let $\{s_1, s_2, \ldots, s_n\}$ be the decision points in E such that $D(s_i) < D(s_{i+1})$ */
/* for $1 \leq i \leq n-1$. Let $s_{n+1} = s_1$. */

main( )
begin
  determine shortest c-route to all $M_j \in SM_1$ from $s_1, s_2, \ldots, s_n$;
  $\mu_{opt} = \mu_1^{P,s_1}$;
  for $i = 1$ to n **do**
    select(s_i, s_{i+1});
  endfor;
  print μ_{opt} and s_{opt};
end.

Figure 3.3: Algorithm ENTRY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

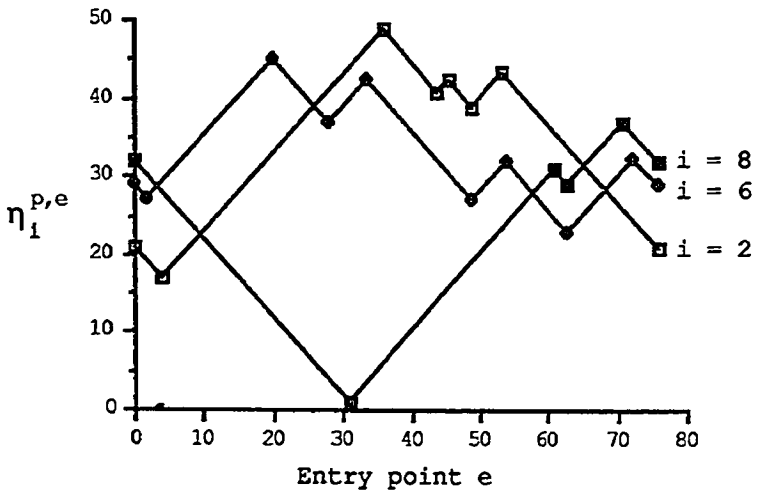

Figure 3.4: Variation in $\eta_i^{P,e}$ for the floorplan in Figure 3.1

Second case, $D(s) - D(u) \geq \dfrac{\eta_i^{P,v} - \eta_i^{P,u} + D(v) - D(u)}{2}$. In this case, from Theorem 3.1, $\eta_i^{P,x}$ is decreasing at s and hence so is $\mu_1^{P,x}$. It will decrease till $\eta_i^{P,x}$ intersects $\eta_j^{P,x}$ for some $j \neq i$. At the point of intersection the c-length of the shortest simple path to M_i and M_j are equal. After the point of intersection M_j becomes the farthest block. These are precisely the steps executed in the else clause.

Therefore, if the value of μ_{opt} was correct at the start of the while loop, it is correct at the end of each execution of the while loop. The theorem thus follows by induction. ■

Example 3.1 (cont'd.): Consider the floorplan in Figure 3.1. Recall that only sub-modules M_2, M_6 and M_8 are in level 1. Figure 3.4 shows the variation in $\eta_i^{P,e}$ for the three sub-modules in this floorplan. The plots are based on vertex 0 (see Figure 3.1) as the origin. Clearly, $\eta_i^{P,e}$ for $i = 2, 6$ and 8 are comprised of straight lines of slopes $+1$ and -1 as indicated by Theorem 3.1. Since in this example $\mu_1^{P,e} = \max\{\eta_2^{P,e}, \eta_6^{P,e}, \eta_8^{P,e}\}$ and $\mu_{opt} = \min_{e \in [0,76]} \mu_1^{P,e}$, it follows by either graphically (see Figure 3.5) or by using ENTRY that the optimal entry point is at distance 3.5 units along the periphery from the origin and $\mu_{opt} = 28.5$ units. ■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

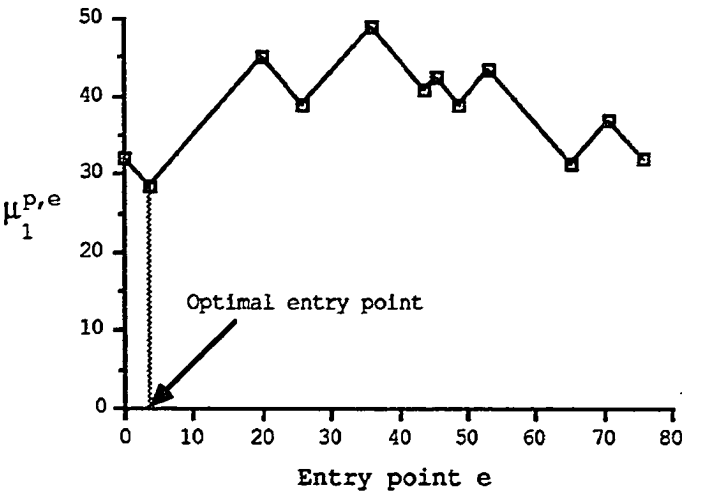

Figure 3.5: Variation in $\mu_1^{P,e}$ for the floorplan in Figure 3.1

3.4 Optimization Problem

The problem addressed in this section can be stated as follows: given a floorplan P and an entry point e determine a c-layout $Y^{P,e} \equiv (R_1^{P,e}, R_2^{P,e}, \ldots, R_m^{P,e})$ that minimizes $\sum_{i \in SM_l} \left| L_i(Y^{P,e}) - \mu_l^{P,e} \right|$ for all $1 \leq l \leq max\_level$. Since there is no inter-dependence between the terms of the summation,

$$\min_{Y^{P,e}} \sum_{i \in SM_l} \left| L_i(Y^{P,e}) - \mu_l^{P,e} \right| \iff \min_{R_i^{P,e}} \left| L(R_i^{P,e}) - \mu_l^{P,e} \right| \quad \forall\, i \text{ such that } M_i \in SM_l. \quad (3.1)$$

where $L(R_i^{P,e}) = L_i(Y^{P,e})$ is used just to emphasize that the right-hand side depends only on M_i. In the rest of this section we will concentrate only on the right-hand side of Equation (3.1).

Consider the placement graph G^P. Let e be the selected entry point. The point e can be assumed to be a vertex of G^P without loss of generality. (If e is not a vertex of G^P, then modify G^P to include e as a vertex by augmenting the vertex set of G^P with e and replacing the edge $\{v_1, v_2\}$ containing e by two edges $\{v_1, e\}$ and $\{e, v_2\}$ of weights $d(v_1, s)$ and $d(e, v_2)$, respectively.) Since determining a $R_i^{P,e}$ that minimizes $\left| L(R_i^{P,e}) - \mu_l^{P,e} \right|$ is equivalent to finding a simple path in a graph of length closest to a given length $\mu_l^{P,e}$, the problem is NP-hard [18]. An exhaustive search with intelligent pruning will therefore be used to find the optimal c-layout. The reduction in search space due to pruning is illustrated through examples in the following section.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ROUTE, the algorithm for searching all the paths, is recursive in nature. The basic idea of the algorithm is as follows. Consider a clock buffer B_l that is at level l. It drives the clock signals of some of the sub-modules in level $l+1$. Let *Drives*(B_l) denote the set of sub-modules driven by B_l. For clarity of presentation, it is convenient to describe the algorithm as if there were only one sub-module in *Drives*(B_l). Extending the description to the case when *Drives*(B_l) contains more than one sub-module is relatively simple. So let M_i be the only sub-module in *Drives*(B_l).

The algorithm starts out with a partial path that contains only the output clock terminal of B_l. At each step of the algorithm a vertex is added to the partial path until either the clock terminal of the M_i is reached, or when there is no completion of the partial path that will result in a c-route with better objective value than the best known objective at that step. The algorithm terminates when it is clear that every path from the output of B_l to the clock terminal of M_i has been either investigated or pruned. Although the algorithm is exponential in nature, our simulations show that the execution time of this algorithm is small even for large designs.

A pseudo-code of the algorithm is shown in Figure 3.6. It is described in terms of the following three functions.

cost(p)	Returns the length of the path p.
mincost(v, M_i)	Returns the length of the shortest c-route from vertex v to sub-module M_i.
objective(p)	Returns the value of the objective for the path p, i.e., $\left\lvert cost(\,p\,) - \mu_l^{P,s} \right\rvert$, where s is the output of M_i.

The correctness of ROUTE is proved formally in Theorem 3.3. The pruning of the search space occurs in the formation of the set of vertices, $N(\omega u)$. At an intermediate step in the algorithm a partial c-route ωu is to be extended to a valid c-route. At this step an attempt is made to extend the partial c-route by appending one more vertex to it. The first two conditions[1] in the formation of $N(\omega u)$ ensures the vertex being appended is adjacent to u and does not already lie on the path. The third condition eliminates vertices from which there is no better completion of the partial c-route.

Theorem 3.3: Algorithm ROUTE is correct, i.e., at the completion of the algorithm

[1] Those conditions for $N(\omega u)$ in the pseudo-code of ROUTE.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
procedure search( p );
path p;
begin
    let p := ωu;
    if u is the clock terminal of M_i then
        if ( objective(p) < min_obj) then
            min_obj:= objective(p);
            min_p := p;
        endif;
        return ( min_obj, min_p );
    endif;
    let N(ωu) ≡ {v : {u,v} ∈ E^P, v ∉ ω, cost(p) + mincost(v, M_i) − μ_i^{P,s} ≤ min_obj};
    for each v ∈ N(ωu) do
        search ( pv );
    endfor;
    return ( min_obj, min_p );
end;

main ( )
begin
    search(s); /* s is the clock output of M_i */
    print min_obj and min_p;
end.
```

Figure 3.6: Algorithm ROUTE

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$min\_obj = \min_{\text{c-routes } p} \left| cost(p) - \mu_l^{P,s} \right|.$$

Proof: Suppose not. Then there exists a path $p^* = su_1u_2\cdots u_nI_i$ such that $\left|cost(p^*) - \mu_l^{P,s}\right| < min\_obj$ and I_i is the input clock terminal of M_i. Then,

$$\begin{aligned} min\_obj &> \left| cost(p^*) - \mu_l^{P,s} \right| \\ &= \left| cost(su_1) + cost(u_1u_2\cdots I_i) - \mu_l^{P,s} \right| \\ &\geq \left| cost(su_1) + mincost(u_1, M_i) - \mu_l^{P,s} \right| \\ &\geq cost(su_1) + mincost(u_1, M_i) - \mu_l^{P,s} \end{aligned} \tag{3.2}$$

Equation (3.2) implies $u_1 \in N(s)$. By using a similar argument one can easily show by induction that $u_j \in N(su_1u_2\cdots u_{j-1})$ for all $j = 1, \ldots, n+1$, $u_{j+1} \equiv I_i$. This implies p^* was examined by ROUTE and therefore $min\_obj \leq \left|cost(p^*) - \mu_l^{P,s}\right|$. Contradiction. ■

3.5 Implementation

ENTRY and ROUTE were implemented in C on an Apollo DN4000. The placement graph was the input to the program and the output was the optimal entry point and the route of clocks to all the sub-modules. This implementation was used to distribute clocks in several examples. The results for two examples are shown here.

The first example is that of the floorplan in Figure 3.1. As mentioned earlier, in this floorplan, sub-modules M_2, M_6 and M_8 are driven directly from the entry point, M_1, M_3 and M_5 are driven by M_2, and M_7, M_8, and M_{10} are driven by M_6. The first task is to identify the entry point. Once the entry point has been determined, the problem of determining the c-layout can be partitioned into three independent sub-problems: (i) route of the clock signal from the entry point to M_2, M_6 and M_8, (ii) route of the clock signal from M_2 to M_1, M_3 and M_5, and (iii) route of the clock signal from M_6 to M_7, M_8 and M_{10}. These three sub-problems can be solved in parallel with each other.

It follows from Figure 3.5, that the optimal entry point is at a distance 3.5 units from vertex 0. The route of the clocks from this entry point is shown in Figure 3.7. In this layout it is easy to see how the increased weight on the edges $\{2,6\}$ and $\{24,25\}$ accounts for the delays induced

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

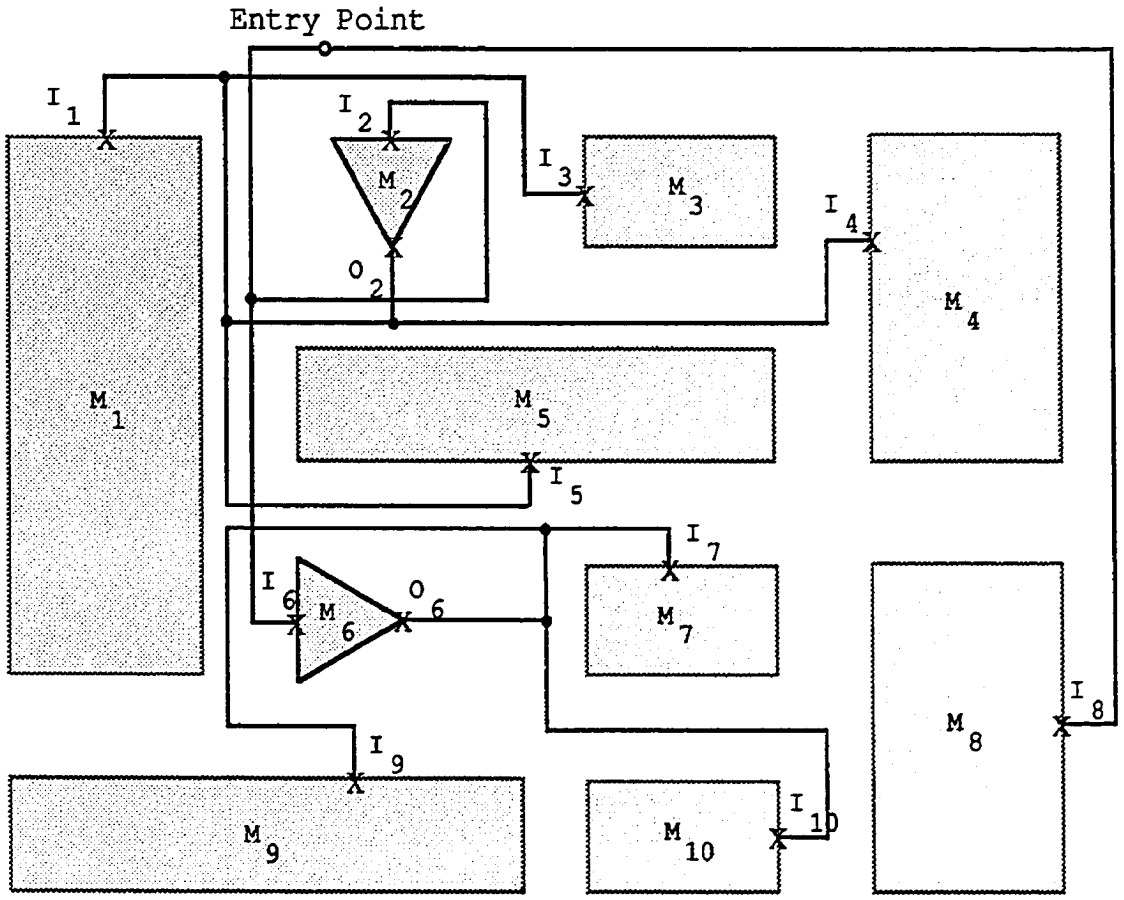

Figure 3.7: Optimal clock layout for floorplan in Figure 3.1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

by the clock buffers M_2 and M_6, respectively. For instance, the delay to M_8 from the entry point is 28.5 units. The delay to M_1 from the entry point is the delay to M_2 (16.5 units) plus the delay introduced by M_2 (5 units) plus the delay from M_2 to M_1 (10 units), i.e., a total delay of 31.5 units. As this was a small example, the CPU time required to determine all these routes was less than a second.

The second example was a custom VLSI chip called the *routing controller* [14]. It is a microprogrammable unit designed as an intelligent front-end interface to implement the low-level message routing algorithms in an experimental distributed real-time system called Hexagonal Architecture for Real-Time Systems (HARTS). It is comprised of six almost identical "ports"[2] interconnected through a time-slice bus. One of these ports has been implemented using the CONCORDE™ silicon compiler in a $2\mu m$ CMOS process and contains around 20,000 transistors.

A floorplan of the sub-modules (along with the routing of the other signals) in the single port implementation is shown in Figure 3.8. This floorplan is comprised of sub-modules such as two 64×16 bit static RAM, 8×8 bit FIFO, datapaths containing registers, ALU, gates and flip-flops, a single-stage pipelined microsequencer, controlling PLAs, a few logic gates and several clock buffers. Each of these sub-modules was generated by the silicon compiler. The routing of clock signals within these sub-modules were left to the silicon compiler.

The routing of clock signals to the sub-modules was also initially carried out using the placement and routing tools in CONCORDE™. Since these placement and routing tools do not take into account the clock skew problem, the skew (in terms of difference in line lengths) was around 4000 microns between the sub-modules driven by one of the clock buffers. By using ROUTE, the skew can be reduced to 20 microns for the same set of sub-modules and the algorithm requires only 4.5 seconds of CPU time on an Apollo DN4000 with 16 MB of memory. This should be contrasted to the several hours of CPU time that was required to route the other signals in the floorplan.

These two examples clearly indicate that algorithms ENTRY and ROUTE can be used to distribute clocks even in large VLSI circuits.

---

[2] The six ports differ from each other only in their address recognition modules.

™CONCORDE is a trademark of Seattle Silicon Technology Inc.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

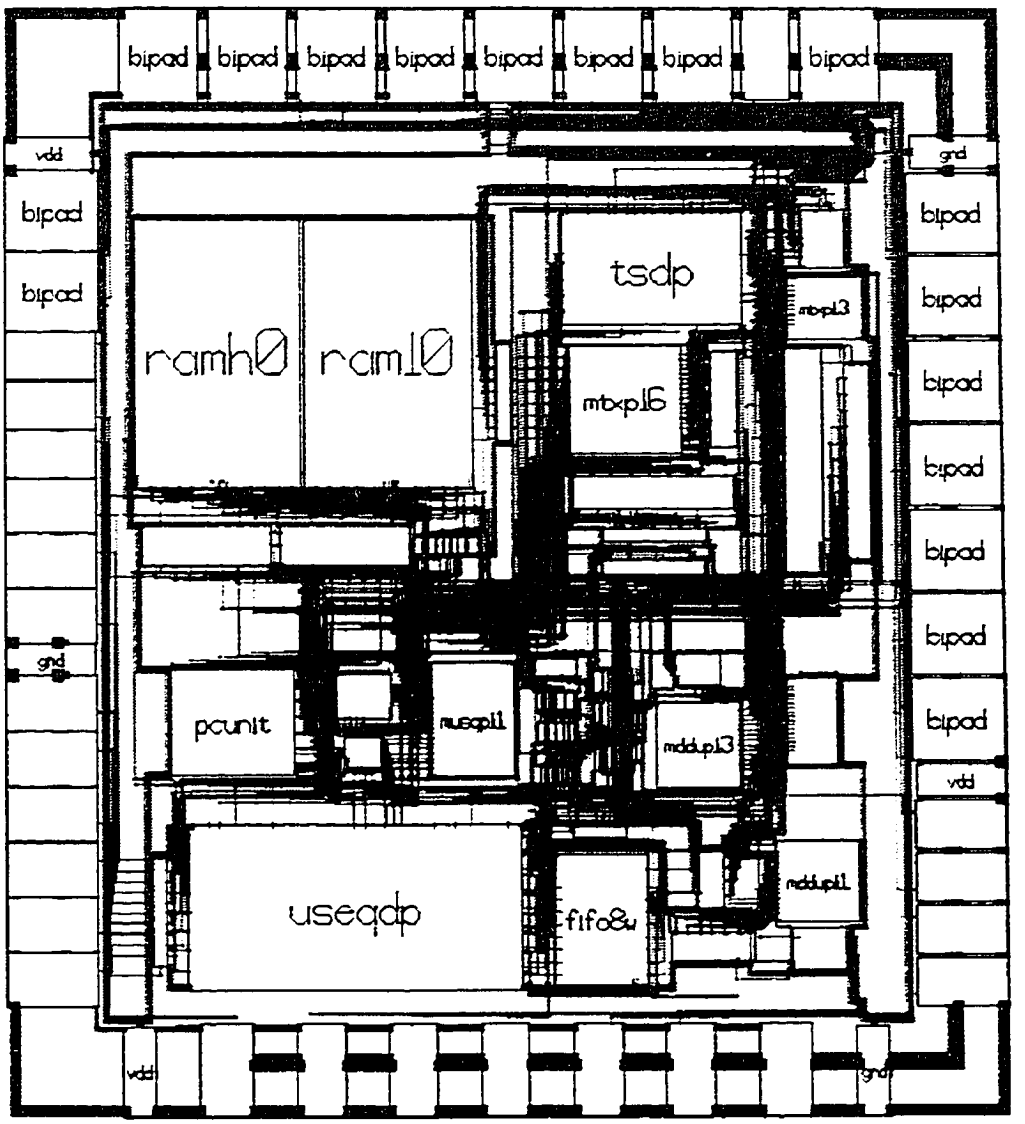

Figure 3.8: Floorplan of the custom VLSI chip

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.6 Discussion

A scheme for laying out the clock lines within a general VLSI circuit was presented. The unique feature of the scheme is that both delay and skew were considered in determining the clock layout. The delays and the skews were computed based on the length of clock lines as well as the number of clock buffers, thus accounting for all the significant factors that affect the timing within a chip. The proposed scheme is also easily parallelizable.

Although the scheme was described for a VLSI circuit it can be extended to large systems such as printed circuit boards. The chips on a printed circuit board are similar to the modules within a VLSI circuit. The scheme can therefore be used to establish a time base among the components within a node of a distributed system even if the node cannot be implemented on a single chip.

Consequently, one can establish a global time base in a distributed system at both system level as well as at the node level. The following chapter discusses the use of a global time base in simplifying solutions to an example design problem in real-time systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

CHECKPOINTING AND ROLLBACK RECOVERY USING COMMON TIME BASE

4.1 Introduction

Recall that the three objectives of this dissertation are to develop schemes for establishing a global time base, to illustrate the use of global time base in simplifying solutions of some design problems in real-time systems, and to reduce the overhead of establishing the time base through diagnosis of Byzantine faults. Chapters 2 and 3 dealt with the first objective. This chapter illustrates the use of global time base in checkpointing and rollback recovery.

A checkpointing and rollback recovery scheme is important because concurrent processes in a critical real-time system are often required to complete their execution prior to an imposed deadline. These deadlines cannot be easily met if the processes always have to restart their execution in case of a failure. A checkpointing scheme avoids this restart by requiring processes to save their states several times during their execution so that they can roll back to the most recently saved state and resume their execution in case of an error. In order to be able to successfully resume execution from a saved state it is essential that all cooperating processes coordinate while saving their states. Although several techniques have been put forth for achieving this coordination [31, 32, 50, 51], it is still the primary cause of the overhead imposed by the checkpointing algorithm.

A new approach for achieving this coordination is proposed here. Unlike conventional algorithms, this approach is based on the existence of a global time base. The global time base is used

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to predict the relative behavior of the cooperating processes. Since the execution of a process is controlled by the clock of the node in which it is executing and since the clocks of the nodes are kept in synchrony by using a synchronization algorithm, the relative behavior of the processes can be predicted more easily and accurately than when the processes operate completely asynchronous to each other. This additional ability to predict the relative behavior of the processes is used as a key element to reduce the time and space overhead of the algorithm. In particular, the scheme has the advantages of maximum process autonomy, minimal wait for commitment for saving of states, fewer messages to be exchanged and reduced memory requirement.

This chapter is organized as follows. A survey of related work is presented in the following section. The proposed checkpointing scheme is described in the third section. The scheme is discussed in terms of the necessary architectural support, a formal description, analysis of the overhead and some numerical examples. The chapter concludes in the fourth section.

4.2 Related Research

As mentioned earlier, concurrent processes save their states several times during their execution so that they can roll back to the most recently saved state and resume their execution in case of an error. Unfortunately, the rollback of a process can result in a cascade of rollbacks that can push the processes back to their beginnings, i.e, the *domino effect*, unless the cooperating processes save their states in coordinated manner. In order to avoid the domino effect, Randell proposed a *conversation scheme* in [50, 51]. Since then, Kim has proposed a similar but more flexible scheme in [31]. Koo and Toueg have also proposed a similar scheme [32] that requires much less overhead as compared to the Randell's scheme.

In the conversation scheme, cooperating processes enter a *conversation* in order to interact with each other. Within a conversation, processes coordinate to save their states before they interact. Processes exit a conversation only after every process in the conversation has passed its *acceptance test*. If a process does not pass its acceptance test, then all the processes in the conversation roll back to the saved state and redo the computation using an alternative process. This rollback or checkpointing scheme usually results in an additional wait for the faster processes, thus degrading the utilization of nodes on which the processes are executing. Furthermore, in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

order to indicate the passing of an acceptance test, each process has to broadcast a message to all the other processes in the conversation, thereby introducing additional overheads.

To overcome the under-utilization problem, a *pseudo-recovery block* (PRB) approach was suggested in [55]. In this approach, processes do not wait for each other to coordinate the saving of states. Instead, after passing an acceptance test, a process broadcasts a message to all other processes to establish a pseudo-recovery point. A *pseudo-recovery point* (PRP) is defined as a recovery point established *without* a preceding acceptance test. So, when a process receives a message to establish a PRP, it completes its current step and then saves its state without an acceptance test. Consequently, a *pseudo-recovery line* is created whenever a process establishes a recovery point. On detecting an error, processes roll back to a pseudo-recovery line that has been validated later by an acceptance test in each of the processes. Though there is no waiting for commitment in this scheme, substantial overheads in time and space are introduced because of the large number of PRP's each process has to establish.

The approach proposed here avoids domino effect with the help of a *global time base*. The idea of using a global time base for simplifying fault tolerant algorithms was first suggested by Lamport [34]. He suggested a simple clock driven algorithm, called the *state machine approach*, that was based on absolute time instead of timeout [34]. His approach is based on the following observation. Given that the nodes of the system are tightly synchronized and given that there is an upper bound on the message transit delay, there exists a bound Δ such that when a non-faulty process broadcasts a message at time T according to its clock, it is delivered to all other non-faulty processes before the time $T + \Delta$ according to their clocks. As a consequence, when the clock strikes $T + \Delta$ each process can safely execute its part of all the actions initiated at time T and broadcast the actions it would like to initiate at time $T + \Delta$.

Lamport developed simple solutions for design problems such as resource allocation, distributed semaphore management, replicated database management, and transaction processing based on this idea. These solutions, however, can be guaranteed to be correct only if there is an upper bound, U, on the message transit delay. Unfortunately, such a bound may not exist in all distributed systems due to contention for various network resources. Hasegawa et. al use negative acknowledgments to extend Lamport's approach to consider the situation in which the probability of message transit delay exceeding a chosen bound U is non-zero [23]. More recently, Lee and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Davidson use absolute time to implement communication primitives in the presence of deadlines in a fault free situation [38, 39]. Absolute time was also used in [21] to implement a fault tolerant distributed directory and communication service. It uses atomic broadcasts [3] and group membership algorithms along with absolute time to update a replicated database synchronously. Since it is not difficult to equip the system with a global time base, the use of absolute time is an attractive alternative for checkpointing and other design problems.

4.3 Checkpointing and Rollback Recovery Scheme

The basic idea of the scheme proposed here and also in [46] is as follows. The execution of each process is first synchronized to the clock of the node on which it is executing. An immediate consequence of this synchronous execution is that each step in a process takes a known number of clock cycles. This fact is coupled with the existence of the global time base to predict the relative behavior of the cooperating processes. The expected time for processes to reach their acceptance tests is first estimated and these estimated times are used to determine the *times* at which all the cooperating processes are asked to establish the pseudo-recovery points.

In order to ensure that a PRP is error free, each process is expected to pass an acceptance test in between two successive PRP's. If a process does not pass an acceptance test before the time for the next PRP, then all the other processes must wait for the process to pass its acceptance test. Since the clocks of all nodes are tightly synchronized and since the processes are synchronous to the clocks of the nodes on which they are executing, the times for establishing the PRP's can be so chosen that a process will *almost always* pass an acceptance test before the next PRP. If all processes complete their acceptance test before the next PRP, then there is no wait for commitment for establishing a PRP. This fact can also be used to reduce the number of messages processes have to exchange to establish the pseudo-recovery line. Later in this section analytic expressions are derived for computing the probability of a process sending a message and the expected waiting time. These expressions indicate that for a typical numerical example, the expected wait time is only 10-15% of the expected wait time in the Randell's scheme [50]. Similarly, the probability of a process sending a message for checkpointing purposes is only around 4–5%.

Since an acceptance test validates a PRP, each process has to establish only one PRP per

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

pseudo-recovery line and preserve only two PRP's to be able to recover from an error without a cascade of rollbacks. This should be contrasted to the $N-1$ PRP's per recovery line in the PRB approach, where N is the number of cooperating processes running concurrently on the system. As a result, the scheme incurs considerably less space overhead as compared to other algorithms.

A formal description of the scheme is presented below. We begin with the architectural support that is necessary for the scheme and then present a pseudo-code representation of the various steps involved. The overheads imposed by the scheme are then analyzed, followed by some numerical examples.

4.3.1 Architectural support

To coordinate the establishment of the pseudo-recovery points each process is provided with a *pseudo-clock*. The pseudo-clock of each process can be thought of as a counter that normally increments at every pulse of the corresponding real clock. However, as will be described later, there are situations where the pseudo-clocks do not increment. In fact, there are situations in which the pseudo-clocks are forced to roll back instead of keeping up with real time. The pseudo-clocks of all the processes are always kept tightly synchronized to each other, by keeping the hardware clocks of all the nodes in lock-step synchronization and by ensuring that when a pseudo-clock rolls back or stops incrementing all the other pseudo-clocks also do the same.

Each process is also provided with four interrupts: *Pseudo-Recovery Interrupt* (PRI), *Acceptance Test Interrupt* (ATI), *Error Interrupt* (EI) and *Timer Interrupt* (TI). A process receives a PRI when its pseudo-clock reaches a value R_j for some $j \in \{1, 2, ..., n\}$, where R_j's are clock values provided to the processes prior to their execution. It can thus be a hardware interrupt implemented with the help of a timer. On the other hand, the ATI is a software interrupt triggered when a process enters an acceptance test, whereas EI is either a hardware or a software interrupt triggered whenever an error is detected in the system during the execution of an acceptance test. It is generated in the process that detects the error and passed on to the other processes by sending messages. A TI is generated when an alarm set by a process expires before the alarm is canceled.

To preserve process autonomy, PRI and ATI are generated locally in each process. As a result, they are not generated at the same time in all the processes. However, since the pseudo-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

clocks of all the processes are always kept in tight synchrony by the checkpointing algorithm, the PRI's will be generated within a short time interval as determined by the small skews between the synchronized clocks. That is, the maximum time interval between the PRI's at two distinct processes is equal to the maximum skew that exists between the pseudo-clocks of all the processes which in turn is determined by the maximum skews that exist between the real clocks and the maximum number of cycles it takes to execute a single step in a process.

Unlike the PRI's, the ATI's in different processes do not necessarily occur within a short time interval. They are governed by the presence of loops, recursions, waiting times for shared resources, and other overheads in each of the processes. The checkpointing algorithm coordinates the establishment of the pseudo-recovery lines while requiring only a loose synchronization in the execution of the acceptance tests.

4.3.2 Formal description

Let $P_1, P_2, \ldots, P_N$ be the N cooperating processes in the system. They satisfy the following assumptions.

A1. Each process executes independently of others except when accessing shared resources.

A2. The processes are executing on reliable nodes.

A3. The processes are synchronous to the clock of the node on which they are executing.

A4. The programmer makes minimal effort to coordinate the interaction between the processes.

A5. There is a known upper bound on the message transit delays between any two processes.

A1 is a requirement rather than an assumption. A2 is made for convenience of presentation. Like almost all checkpointing schemes, the proposed scheme tolerates software design faults and not hardware operational faults. Hardware operational faults such as failure of nodes can be tolerated by using either redundant or spare nodes. Since the emphasis of the discussion

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

here is not on tolerating such hardware failures, we assume that the processes are executing on reliable nodes. Note that the use of redundant or spare nodes does not obviate the need for a rollback recovery scheme since software design faults are likely to affect all the redundant copies simultaneously. Similarly, a rollback recovery scheme cannot obviate the need for redundant or spare nodes since rollback recovery schemes require redundant or spare nodes to execute the alternate block in case of a hardware failure.

A3 and A4 are based on the availability of the global time base and distinguish the approach here from the other checkpointing algorithms. This is in contrast to the approach in which processes are assumed completely asynchronous in the sense that the time between successive "steps" in a given process are not necessarily bounded. A5 is required to distinguish between a failed process not sending a message and unusually large message transit delays. It is well-known that any form of synchronization is difficult to achieve, if not impossible, in the presence of both faulty processes and unbounded message transit delays [8, 16].

Let n_i be the number of acceptance tests in process P_i and let $n = \min_i n_i$. P_i chooses n out of its n_i acceptance tests for the checkpointing algorithm. Even though failure to pass any one of the n_i acceptance tests will trigger a rollback, only the chosen n acceptance tests will be considered by the checkpointing algorithm to establish the PRP's. The choice of these n acceptance tests does not have to be necessarily based on any criteria. However, the overheads involved in the checkpointing algorithm will be less if the processes choose their acceptance tests in such a manner that all processes execute a chosen acceptance test at the same time. So to minimize the overheads, P_i should choose an acceptance test that is closest (in estimated time) to an acceptance test of a process P_j that has only n acceptance tests. For clarity of presentation, assume that all processes have the same number of acceptance tests, namely, the chosen n acceptance tests. In particular, assume that an ATI occurs only when a process enters one of the chosen acceptance tests.

The expected time for a process to reach each of its acceptance tests is estimated prior to its execution. This is made possible by the existence of a global time base and the fact that the processes are synchronous to the real-time clock. However, it is only "expected" because: (i) of the presence of loops and recursions, and (ii) the waiting times for shared resources and the overhead due to interrupts are not known *a priori*. Using these expected times, the time at which

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

all processes establish their PRP's is calculated. The time for the j^{th} PRP is so chosen that all processes are expected to have completed their j^{th} acceptance test but not their $j+1^{th}$. Each process establishes its PRP when its pseudo-clock reaches the determined time (indicated by the arrival of a PRI).

However, before establishing a PRP, a process checks whether it has passed an acceptance test since the last PRP. Since the times for establishing the PRP are appropriately chosen in accordance with the predicted behavior of the process, the process would have almost always passed an acceptance test since the last PRP. In the rare instances when it has not passed an acceptance test, the process broadcasts a message to all other processes in the system indicating it. On receiving such a message every process stops incrementing its pseudo-clock and waits for those processes that have not yet passed their acceptance test. Once they pass their acceptance test, messages are once again sent to all processes indicating it. After all processes have passed their acceptance test, processes start incrementing their pseudo-clocks and resume their normal operation. Similarly, before executing an acceptance test, each process checks whether or not it has established a PRP since the last acceptance test. If it has not established one, it waits till it receives the next PRI, establishes a PRP and then starts executing the acceptance test. In order to describe this algorithm more formally, it is necessary to introduce the following primitives.

receive(*text*, *process_id*)	Receives the message *text* from the process whose identity is *process_id*.
broadcast(*text*, *process_id*)	Process *process_id* broadcasts the message *text* to all processes.
wait(*condition*)	Process halts until the condition becomes true.
alarm(*interval*, *handler*)	Cancel any previously set alarm and generate a timer interrupt after *interval* time units have elapsed. If *interval* has value 0, then do not generate any interrupt but cancel all previously set alarms. If the alarm expires execute the *handler*.

Implementing these four primitives in the presence of faults is non-trivial. There are several papers that have addressed this issue [34, 38]. Here we will assume the existence of such an implementation and concentrate on developing and analyzing a checkpointing algorithm using these primitives.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

PRI handler

This interrupt handler is executed when a process receives a PRI, i.e., when its pseudo-clock reaches a time to set up a PRP. A flowchart of the steps executed in this handler and a more formal description of those steps are in Figures 4.1 and 4.2, respectively. AT_flag indicates whether it has passed an acceptance test since the last PRP. If it has not passed an acceptance test, it sets the *slow* flag and broadcasts a message "not completed AT" to all the other processes. Otherwise, it checks the incoming messages to ensure that all other processes have also passed an acceptance test since the last PRP.

There are several ways of checking the incoming messages to ensure all other processes have passed their acceptance test. One easy way is to implement the *receive* primitive as a non-blocking call that initializes a simple dedicated hardware to interrupt the process on receiving a message from the process identified in the call. The processes can then assume that all processes would have passed their acceptance test and proceed with normal execution. If there is at least one process that has not passed an acceptance test since the last PRP, then a message "not completed AT" will be received by all other processes. On receipt of such a message (indicated by an interrupt), the processes roll back to the state at the time of last PRI. This scheme would be correct and efficient as long as the upper bound on message transit delay is less than the minimum time interval between two successive PRI's.

In rolling back to the time of last PRI, the processes also roll back their pseudo-clocks to the time of the last PRI and disable its increment until they get a message "completed AT" from all the slow processes. To prevent a failed process from permanently blocking a non-faulty process, each process sets an alarm for a duration of Max_Willing_To_Wait on receiving the "not completed AT" message. If a "completed AT" message is not received within this duration, then slow process is considered to have failed and a recovery action such as a rollback is undertaken.

It is not easy to determine an optimal value for Max_Willing_To_Wait. Choosing a small value would cause more non-faulty processes to be considered faulty while choosing a large value will cause non-faulty processes to wait for a long time in case a process is faulty. However, it is not difficult to choose reasonably good values depending on the exact nature of the cooperating processes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

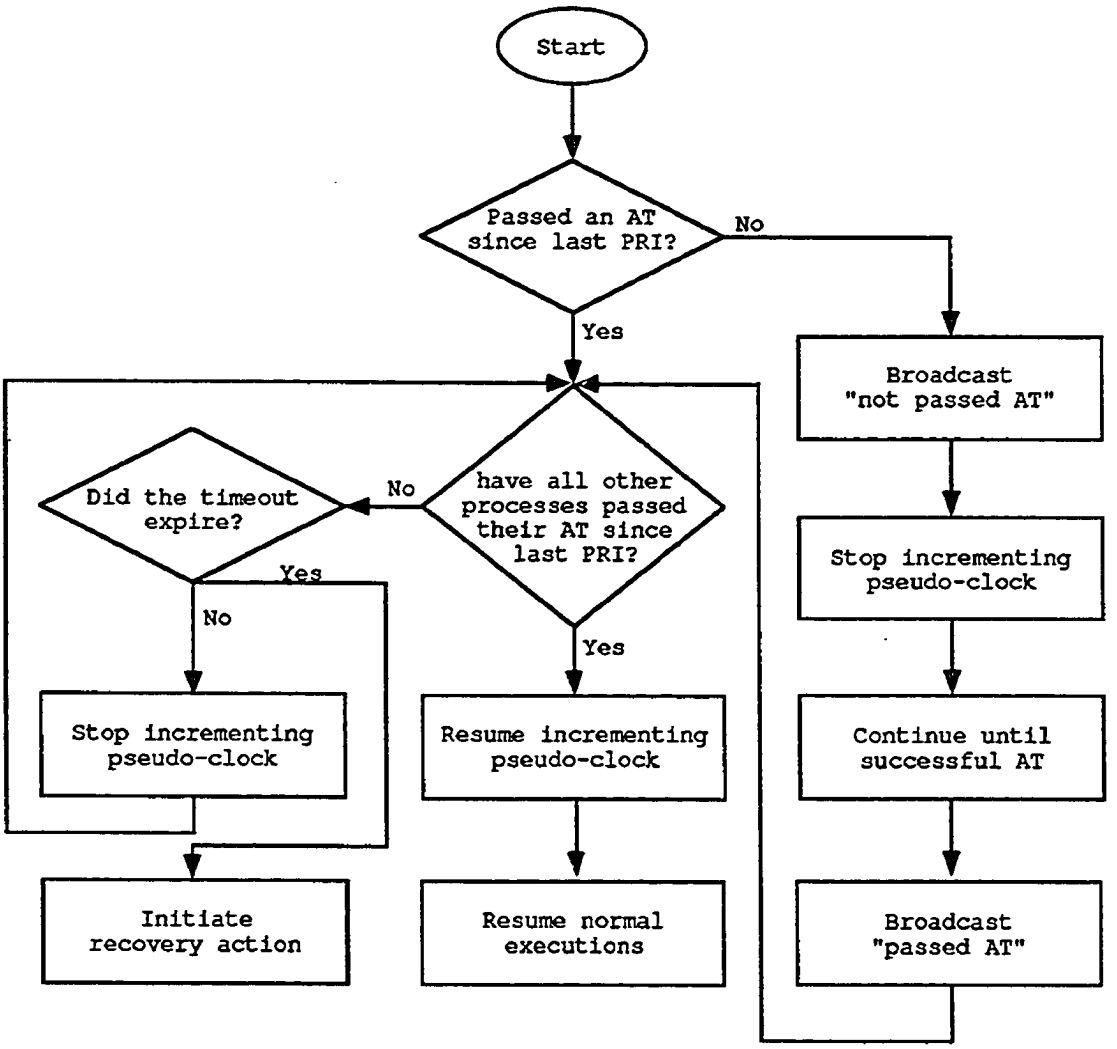

Figure 4.1: Flowchart of the interrupt handler for PRI

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
procedure pseudorecovery_interrupt;
    begin
        pseudo_clock_backup := pseudo_clock;
        if (not AT_flag) then
            slow := true;
            broadcast ("not completed AT", my_id);
            disable_clock := true;
            pseudo_clock := pseudo_clock_backup;
        else
            for k=1 to N, k ≠ i do
                if receive ("not completed AT", k) then
                    receive_flag := true;
                    count := count + 1;
                    alarm (Max_Willing_To_Wait, error_interrupt);
                endif;
            endfor;
            if (not receive_flag) then
                checkpoint_valid := checkpoint_new;
                checkpoint_new := current_state;
            else
                disable_clock := true;
                pseudo_clock := pseudo_clock_backup;
                while receive_flag do
                    if receive ("completed AT", k) then
                        count := count - 1;
                        if (count = 0) then
                            receive_flag := false;
                            alarm (0, error_interrupt);
                        endif
                    endif;
                endwhile;
                disable_clock := false;
                pseudo_clock := pseudo_clock_backup;
            endif;
        endif;
    AT_flag := false;
    end; /* pseudorecovery_interrupt */
```

Figure 4.2: Interrupt handler for PRI

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ATI handler

In order to prevent a process from running ahead of all the other processes, a process is allowed to establish only one of the chosen acceptance tests between any two successive pseudo-recovery lines. So if a process gets two ATI's before getting a PRI, then it must wait for a PRI (see Figures 4.3 and 4.4). This is ensured by checking the AT_flag variable whenever an ATI occurs.

If it is the first ATI after a PRI, then the process checks whether it is running slower than the others, i.e., are there some processes waiting for it to complete this acceptance test (indicated by *slow*). If so, it broadcasts a "completed AT" message to all other processes. If all processes have finished their acceptance tests, it proceeds with the normal execution. Otherwise, it waits for the others to finish.

EI handler

This handler is executed whenever an error occurs in the system. The errors are detected during the execution of an acceptance test[1]. The error handler rolls back the processes to a valid state and then allows them to proceed from that point using an alternative path [50, 51] (see Figure 4.5). In the above checkpointing scheme a valid state is the state of the system at the time of the second to last PRI because the algorithm guarantees an acceptance test in all processes between the second to last and the last PRI. Consequently, there is no need to interact with other processes to determine the state to which a process has to rollback. This simplifies the rollback procedure to a great extent.

Maintenance procedure for pseudo-clock

Figure 4.6 describes the procedure for incrementing the pseudo-clocks. When the disable_clock is true (which happens when one of the processes is waiting for some other process(es) to finish their acceptance test), the pseudo clock does not increment. Otherwise the pseudo clock increments once every clock tick.

[1] The chosen as well as the additional acceptance tests can trigger an EI.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

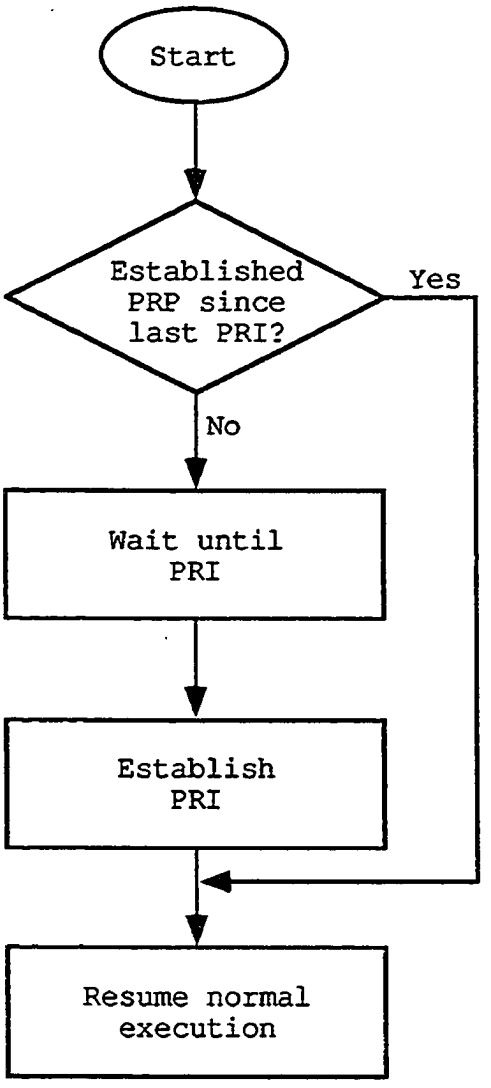

Figure 4.3: Flowchart of the interrupt handler for ATI

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
procedure at_interrupt;
    begin
        if AT_flag then
            wait (pseudorecovery interrupt)
            execute (acceptance test)
            else
                execute (acceptance test)
                if slow then
                    broadcast ("completed AT", my_id);
                    AT_flag := true;
                    slow := false;
                    pseudorecovery_interrupt;
                endif;
            endif;
    end; /* at_interrupt */
```

Figure 4.4: Interrupt handler for ATI

```
procedure error_interrupt;
begin
    current_state := checkpoint_valid;
end /* error_interrupt */
```

Figure 4.5: Interrupt handler of EI

```
procedure clock;
begin
    if (not disable_clock) then
        increment (C_i);
    endif;
end /* clock */
```

Figure 4.6: Maintenance procedure for pseudo-clock

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.7 illustrates the algorithm for three processes P_1, P_2 and P_3. The thick (thin) lines indicate the estimated (actual) time for establishing the j^{th} and $j+1^{th}$ acceptance tests and the dotted line indicates the time at which the j^{th} PRI was issued. In Figure 4.7(a), all processes complete their j^{th} acceptance test before receiving the j^{th} PRI and no process reaches the $j+1^{th}$ acceptance test before receiving the j^{th} PRI. So, no messages are sent and no process waits for another process to establish a checkpoint. In Figure 4.7(b), P_1 does not complete its j^{th} acceptance test before receiving the j^{th} PRI. So it sends a message to the other two processes when it receives the j^{th} PRI. In Figure 4.7(c), P_3 reaches its $j+1^{th}$ acceptance test before it receives the j^{th} PRI. It, therefore, waits till it receives the j^{th} PRI before executing the acceptance test. There are no messages exchanged in this situation.

The highlight of our approach lies in that the need for message exchange and waits in the above algorithm is minimized by the prediction of process execution behavior with a global time base. A detailed analysis of its performance is the subject of the next section.

4.3.3 Analysis

In this section, a probabilistic model is developed to characterize the checkpointing scheme described above. This model will be used to analyze the expected overhead of the proposed scheme. To facilitate the analysis, we shall use the following notation.

n	Number of checkpoints.
AT_j	The j^{th} acceptance test.
PRP_j	The j^{th} pseudo-recovery point.
W_{ij}	Random variable representing the uncertainty in the expected time for P_i to reach AT_j.
T_{ij} (A_{ij})	Time required by P_i to reach AT_j without (with) the checkpointing overhead.
S_j	$\max_i T_{ij}$
O_{ij}	Waiting time by P_i at PRP_j.
A^a_{ij} (A^d_{ij})	The real-time at which P_i receives (establishes) the j^{th} PRI.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

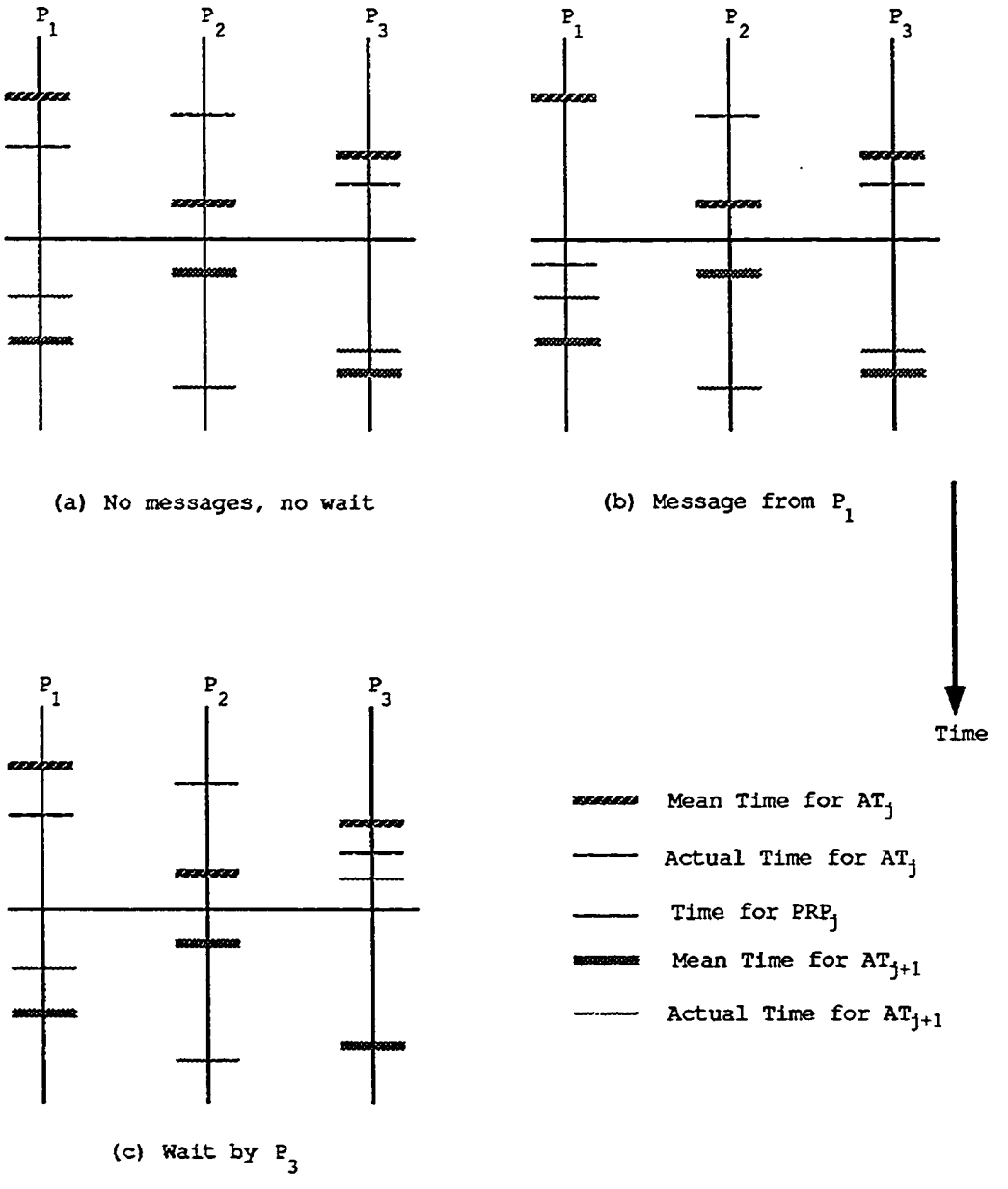

Figure 4.7: Illustration of checkpointing algorithm

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

f_X Density of a random variable X.

M_j $\max_i W_{ij}$

m_j $\min_i W_{ij}$

R_j The pseudo-clock time at which all processes receive their j^{th} PRI.

Probabilistic model

To model the performance of the checkpointing scheme, the following assumptions are necessary.

MA1. W_{ij} and W_{kl}, $k \neq i$, are independent of each other for all j, l.

MA2. Given W_{ij}, the time required by P_i to reach AT_j without waiting for other processes for checkpointing, denoted by, T_{ij}, has a density $f_{T_{ij}|W_{ij}}$.

MA3.

$$R_j = \begin{cases} M_j & \text{if } M_j > m_{j+1} \\ M_j + k_j * (m_{j+1} - M_j) & \text{if } m_{j+1} \geq M_j \end{cases}$$

where k_j is a design parameter.

MA4. $R_0 = A_{i0}^d = 0$ for all i.

A programmer inserts the acceptance tests within the processes. For each acceptance test, there is usually a desired point (time) in the process where the programmer would like to insert it. The desired point will be based on the number of acceptance tests to be inserted and the total estimated execution time for the processes [56]. The acceptance tests cannot, however, be inserted at any arbitrary point in the process since one may not be exactly aware of the state a process should be in at every point in its execution (for verification by an acceptance test). So the acceptance tests will be inserted at a feasible point closest to the desired point, i.e., the expected time to reach an acceptance test will differ from the desired time. One can model this uncertainty in the expected time to reach an acceptance test as a random variable distributed around the desired time for inserting that acceptance test.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

MA1 states that the random variable representing the uncertainty in the expected time for P_i to reach AT_j is independent of that for P_k to reach the AT_l for all $k \neq i$. This arises from the assumption that each process executes independently of others except when accessing shared resources. MA2 states that due to unpredictable waits for shared resources and other overheads the actual time for P_i to reach its j^{th} acceptance test will differ from the expected time. The distribution of the actual time of an acceptance test around the expected time is defined by MA2. MA3 defines the class of functions considered in the analysis for determining the times to establish the PRP's. Since the checkpointing algorithm requires every process to have completed the j^{th} acceptance test and not the $j+1^{th}$, it is reasonable to assume that the time for the j^{th} PRI should be somewhere in between M_j and m_{j+1}. This particular class of functions is chosen to make the analysis tractable. MA4 specifies the initializing conditions.

Out of the two factors that contribute to the checkpointing overhead, the overhead of saving states depends directly on the number of times a process has to save its state. It can be reduced considerably at the cost of additional hardware as shown in [40]. The other factor that contributes to the overhead, namely the overhead of waiting at PRP, occurs either if one of the n processes does not complete its j^{th} acceptance test before the j^{th} PRI or if the process receives the $j+1^{th}$ ATI before the j^{th} PRI. In the first case a process waits because some process was slower than expected while in the second case the process waits because it was much faster than expected. We will henceforth refer to the first case as a *slow_wait* and the second case as the *fast_wait*.

Since PRI's are based on time rather than points in execution whereas ATI's correspond to points in execution, the probability of a *slow_wait* depends on the time interval between the start of the process and the j^{th} PRI. When the overhead due to the checkpointing scheme is zero, P_i executes for R_j time units before the j^{th} PRI. But in practice, due to the waiting times at the previous checkpoints, P_i gets $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik}$ time units for execution before the j^{th} PRI. Since the processes do not increment their pseudo-clocks while waiting at the PRP's, it is proved below that $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$. To prove this result it is convenient to define a mapping C_i from real-time to the pseudo-time on process P_i such that $C_i(t) = T$ means that the pseudo-time on P_i at real-time t is T.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Theorem 4.1: Let A^a_{ij} and R_j be the respective real-time and the pseudo-time at which process P_i receives the j^{th} PRI. Also let O_{ik} be the P_i's waiting time at the k^{th} PRP. Then, $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$.

Proof: Since a process might have to wait for some other processes (including itself) to complete the k^{th} acceptance test before establishing the k^{th} PRP, A^a_{ik} is not necessarily equal to A^d_{ik}, where A^d_{ik} is the real-time at which P_i establishes the k^{th} PRP. However, since the processes do not increment their pseudo-clocks while some process is waiting, $C_i(A^a_{ik}) = C_i(A^d_{ik})$ for all i.

In the time interval between the establishment of the k^{th} PRP and the receiving of the $k+1^{th}$ PRI, the pseudo-clocks of all processes keep up with real-time. As a result,

$$C_i(A^a_{ik+1}) - C_i(A^d_{ik}) = A^a_{ik+1} - A^d_{ik} \qquad \text{for all } i, k.$$

Also, since a process that has not completed its k^{th} acceptance test when it received the k^{th} PRI continues to run while the others are waiting for it to complete the acceptance test, $O_{ik} \leq A^d_{ik} - A^a_{ik}$. From these observations, the theorem can be proved as follows.

$$\begin{aligned} A^a_{ij} &= \sum_{k=1}^{j}(A^a_{ik} - A^d_{ik-1}) + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &= \sum_{k=1}^{j}(R_k - R_{k-1}) + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &= R_j + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &\geq R_j + \sum_{k=1}^{j-1} O_{ik}. \end{aligned}$$

In other words, $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$. ■

The above theorem implies that the time interval between two successive PRI's does not depend on the waiting times at the previous PRI's. Consequently, the total time that a process gets to execute before receiving the j^{th} PRI does not depend on the waiting times at the previous PRI's. So the probability of a message being sent does not depend on the waiting times at each PRP and therefore it can be evaluated by considering the situation in which there is no checkpointing overhead, i.e., the probability of P_i sending a message can be evaluated by using T_{ij} instead of A_{ij}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Estimation of the overhead

The overhead due to *slow_wait* occurs when a process does not complete its j^{th} acceptance test before the time R_j while the overhead due to *fast_wait* occurs when a process reaches its $j+1^{th}$ acceptance test before R_j. The overhead due to saving of states occurs each time a process has to establish a pseudo-recovery point.

In terms of the notations introduced earlier, the wait time at the j^{th} PRP of process P_i can be expressed as

$$O_{ij} = \begin{cases} S_j - T_{ij} & \text{if } T_{ij} > R_j \\ S_j - R_j & \text{if } S_j > R_j,\ T_{ij} \leq R_j \text{ and } T_{ij+1} > R_j \\ S_j - T_{ij+1} & \text{if } S_j > R_j \text{ and } T_{ij+1} \leq R_j \\ R_j - T_{ij+1} & \text{if } S_j \leq R_j \text{ and } T_{ij+1} \leq R_j \\ 0 & \text{otherwise.} \end{cases} \quad (4.1)$$

The first case, namely $T_{ij} > R_j$, corresponds to the situation where P_i does not complete its AT_j before receiving the j^{th} PRI. P_i therefore waits from the time it completes its AT_j till the slowest process completes. The second case corresponds to situation where P_i completes its j^{th} acceptance test before the j^{th} PRI but some other process does not. So P_i waits for the slowest to complete from the time it receives the j^{th} PRI. The third and the fourth cases occur when P_i is so fast that it completes both AT_j and AT_{j+1} before receiving the j^{th} PRI. In the third case, a process other than P_i does not complete its AT_j before the j^{th} PRI. In this case, P_i waits from the time it receives the $j+1^{th}$ ATI till the slowest process completes its j^{th} acceptance test. In the fourth case, all processes complete their AT_j by R_j and therefore P_i resumes its normal execution at the time R_j. Finally, if none of the above situation occurs, then P_i does not wait at the j^{th} pseudo-recovery point. The goal is to select a R_j such that the final case occurs more often than the above four cases.

We can then prove the following result about the expected value of O_{ij}, i.e., the expected wait time of P_i at the j^{th} PRP.

Theorem 4.2:

$E[O_{ij}] \leq E[T_{ij} - R_j | T_{ij} > R_j] + E[R_j - T_{ij+1} | R_j > T_{ij+1}]$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: From Equation (4.1),

$$\begin{aligned} E[O_{ij}] &= E[S_j - T_{ij}|T_{ij} > R_j]P\{T_{ij} > R_j\} \\ &+ E[S_j - R_j|S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j]P\{S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j\} \\ &+ E[S_j - T_{ij+1}|S_j > R_j, T_{ij+1} \le R_j]P\{S_j > R_j, T_{ij+1} \le R_j\} \\ &+ E[R_j - T_{ij+1}|S_j \le R_j, T_{ij+1} \le R_j]P\{S_j \le R_j, T_{ij+1} \le R_j\}. \end{aligned}$$

Since

$$\begin{aligned} E[S_j - T_{ij+1}|S_j > R_j, T_{ij+1} \le R_j] &= E[S_j - R_j|S_j > R_j, T_{ij+1} \le R_j] \\ &+ E[R_j - T_{ij+1}|S_j > R_j, T_{ij+1} \le R_j] \quad \text{and} \\ E[S_j - T_{ij}|T_{ij} > R_j] &\le E[S_j - R_j|T_{ij} > R_j] \end{aligned}$$

we get,

$$\begin{aligned} E[O_{ij}] &\le E[S_j - R_j|T_{ij} > R_j]P\{T_{ij} > R_j\} \\ &+ E[S_j - R_j|S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j]P\{S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j\} \\ &+ E[S_j - R_j|S_j > R_j, T_{ij+1} \le R_j]P\{S_j > R_j, T_{ij+1} \le R_j\} \\ &+ E[R_j - T_{ij+1}|S_j > R_j, T_{ij+1} \le R_j]P\{S_j > R_j, T_{ij+1} \le R_j\} \\ &+ E[R_j - T_{ij+1}|S_j \le R_j, T_{ij+1} \le R_j]P\{S_j \le R_j, T_{ij+1} \le R_j\}. \end{aligned}$$

But $T_{ij} > R_j$ implies $S_j > R_j$ and $T_{ij+1} > R_j$. Therefore,

$$\begin{aligned} E[O_{ij}] &\le E[S_j - R_j|T_{ij} > R_j]P\{S_j > R_j, T_{ij} > R_j, T_{ij+1} > R_j\} \\ &+ E[S_j - R_j|S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j]P\{S_j > R_j, T_{ij} \le R_j, T_{ij+1} > R_j\} \\ &+ E[S_j - R_j|S_j > R_j, T_{ij+1} \le R_j]P\{S_j > R_j, T_{ij} \le R_j, T_{ij+1} \le R_j\} \\ &+ E[R_j - T_{ij+1}|S_j > R_j, T_{ij+1} \le R_j]P\{S_j > R_j, T_{ij+1} \le R_j\} \\ &+ E[R_j - T_{ij+1}|S_j \le R_j, T_{ij+1} \le R_j]P\{S_j \le R_j, T_{ij} \le R_j, T_{ij+1} \le R_j\}. \quad (4.2) \end{aligned}$$

Furthermore,

$$\begin{aligned} &E[S_j - R_j|S_j > R_j] = \\ &\quad E[S_j - R_j|T_{ij} > R_j]P\{S_j > R_j, T_{ij} > R_j, T_{ij+1} > R_j\} \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$+E[S_j - R_j|S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j]P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j\}$$
$$+E[S_j - R_j|S_j > R_j, T_{ij+1} \leq R_j]P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\} \quad \text{and}$$

$E[R_j - T_{ij+1}|T_{ij+1} \leq R_j] =$

$$E[R_j - T_{ij+1}|S_j > R_j, T_{ij+1} \leq R_j]P\{S_j > R_j, T_{ij+1} \leq R_j\}$$
$$+ E[R_j - T_{ij+1}|S_j \leq R_j, T_{ij+1} \leq R_j]P\{S_j \leq R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\}.$$

Hence from Equation 4.2,

$$E[O_{ij}] \leq E[S_j - R_j|S_j > R_j] + E[R_j - T_{ij+1}|R_j > T_{ij+1}].$$

Simplifying further,

$$E[S_j - R_j|S_j > R_j] = \sum_{i=1}^{N} E[T_{ij} - R_j|T_{ij} > R_j, T_{ij} \equiv S_j]P\{T_{ij} \equiv S_j\}.$$

Since all processes are equally probable of being the slowest $P\{T_{ij} \equiv S_j\} = \frac{1}{N}$. In other words,

$$E[O_{ij}] \leq E[T_{ij} - R_j|T_{ij} > R_j] + E[R_j - T_{ij+1}|R_j > T_{ij+1}]. \quad \blacksquare$$

Although exact analytic expressions for the expected overhead can be derived, evaluating those expressions to within an acceptable accuracy is very complicated. So, analytic expressions for the upper bound in Theorem 4.2 are derived instead of the exact expressions. $E[T_{ij} - R_j|T_{ij} > R_j]$ can be evaluated from the joint distribution of T_{ij} and R_j as follows.

$$P\{T_{ij} \leq t;\ R_j \leq z\} = \int_{t_1=0}^{z} P\{T_{ij} \leq t; m_{j+1} \leq \frac{z-(1-k_j)t_1}{k_j}|M_j = t_1\}f_{M_j}(t_1)dt_1. \tag{4.3}$$

Eq. (4.3) can be evaluated from the joint distribution of T_{ij}, M_j and m_{j+1}. Since W_{ij} and W_{kl} are assumed to be independent for all $k \neq i$, the joint distribution of T_{ij}, M_j and m_{j+1} can be expressed as:

$P\{T_{ij} \leq t; M_j \leq z_1; m_{j+1} > z_2\} =$

$$P\{T_{ij} \leq t; W_{ij} \leq z_1; W_{ij+1} > z_2\}P\{\forall\, l \neq i\ \ W_{lj} \leq z_1; W_{lj+1} > z_2\}$$

where,

$P\{\forall\, l \neq i\ \ W_{lj} \leq z_1; W_{lj+1} > z_2\} =$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\prod_{\substack{l=1 \\ l \neq i}}^{N} \int_{t_1=0}^{z_1} P\{W_{lj+1} > z_2 | W_{lj} = t_1\} f_{W_{lj}}(t_1) dt_1 \quad \text{and}$$

$$P\{T_{ij} \leq t; W_{ij} \leq z_1; W_{ij+1} > z_2\} =$$

$$\int_{t_1=0}^{z_1} P\{T_{ij} \leq t | W_{ij} = t_1\} P\{W_{ij+1} > z_2 | W_{ij} = t_1\} f_{W_{ij}}(t_1) dt_1.$$

Similarly, it is possible to evaluate the $E[R_j - T_{ij+1} | R_j > T_{ij+1}]$ from the joint distribution of T_{ij+1} and R_j.

$$P\{T_{ij+1} \leq t; \ R_j \leq z\} = \int_{t_1=0}^{z} P\{T_{ij+1} \leq t; m_{j+1} \leq \frac{z - (1-k_j)t_1}{k_j} | M_j = t_1\} f_{M_j}(t_1) dt_1. \tag{4.4}$$

Eq. (4.4) can be evaluated from the joint distribution of T_{ij+1}, M_j and m_{j+1}. Since W_{ij} and W_{kl} are assumed to be independent for all $k \neq i$, the joint distribution of T_{ij}, M_j and m_{j+1} can be expressed as:

$$P\{T_{ij+1} \leq t; M_j \leq z_1; m_{j+1} > z_2\} =$$

$$P\{T_{ij+1} \leq t; W_{ij} \leq z_1; W_{ij+1} > z_2\} P\{\forall \, l \neq i \ \ W_{lj} \leq z_1; W_{lj+1} > z_2\} \text{ where}$$

$$P\{\forall l \neq i \ \ W_{lj} \leq z_1; W_{lj+1} > z_2\} =$$

$$\prod_{\substack{l=1 \\ l \neq i}}^{N} \int_{t_1=0}^{z_1} P\{W_{lj+1} > z_2 | W_{lj} = t_1\} f_{W_{lj}}(t_1) dt_1 \quad \text{and}$$

$$P\{T_{ij+1} \leq t; W_{ij} \leq z_1; W_{ij+1} > z_2\} =$$

$$\int_{t_1=0}^{z_2} P\{T_{ij+1} \leq t | W_{ij+1} = t_1\} P\{W_{ij} \leq z_1 | W_{ij+1} = t_1\} f_{W_{ij+1}}(t_1) dt_1.$$

The overhead due to saving of states depends on: (i) the number of times a process has to save its state and (ii) the architecture of the system. In particular, it does not depend on R_j or any other parameter specific to the checkpointing algorithm. Since the algorithm requires a process to save its states only once every pseudo-recovery line as compared to $N-1$ in the PRB approach [55], where N is the number of cooperating processes in the system, this overhead is substantially less in the proposed algorithm than in the PRB approach.

The above equations can be used to determine a good value for the design parameter k_j. This value of k_j would have been optimal if the exact expressions are used for $E[O_{ij}]$ (instead of the upper bound) and if the objective is to minimize the expected wait time. On the other

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

hand, if the objective is to minimize a different objective such as probability of a long wait or the probability of a process sending a message, then an appropriate analytical expression has to be minimized. Here, we will presume that the objective is to select a k_j that minimizes the expected wait time at the j^{th} PRP. Since evaluating the exact expressions for the expected wait time is very complicated even for some simple distributions, we will minimize the upper bound specified in Theorem 4.2 to obtain a sub-optimal solution. In other words,

Minimize

$E[S_j - R_j | S_j > R_j] + E[R_j - T_{ij+1} | T_{ij+1} \leq R_j]$ **with respect to** R_j

Subject to:

$$R_j = \begin{cases} M_j & \text{if } M_j > m_{j+1} \\ M_j + k_j * (m_{j+1} - M_j) & \text{if } m_{j+1} \geq M_j \end{cases}$$

where k_j is a design parameter. Since k_j is the only design parameter in R_j, choosing value for R_j is equivalent to choosing a value for k_j. The value of k_j that minimizes the above objective can be determined numerically using iterative optimization techniques such as Fibonacci descent method [42].

4.3.4 Numerical examples

The overheads described in the previous section were evaluated using numerical integration techniques for some known distribution and the following results were obtained. To account for differences in the processes under consideration, the expected time for processes to reach the j^{th} acceptance test was assumed to be uniformly distributed over the interval $[j * inter\_at - a_j, j * inter\_at + a_j]$, where a_j is known parameter. Given the expected time for a process to reach its j^{th} acceptance test, the actual time (without the checkpointing overhead) was assumed to be uniformly distributed around the expected time with parameter b_j, i.e., $f_{T_{ij}|W_{ij}}(t|W_{ij} = t_1)$ is uniformly distributed over the interval $[t_1 - b_j, t_1 + b_j]$. This accounted for the variation in number of times certain loops were executed, waiting times for shared resources, interrupt service overhead, etc.

The expected waiting time at the sixth PRP for the best value of k_j (obtained by solving the minimization problem in the previous subsection) is shown in Table 4.1. The values in this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a_6	b_6	$E[O_{ij}]$ (Prop.)	$E[O_{ij}]$ (Rand.)	$\frac{\text{Prop.}}{\text{Rand.}}$ %	Prob. of mess. %
6.0	12.0	1.32	15.80	13.87	4.14
8.0	12.0	1.95	17.25	11.30	4.86
9.0	12.0	2.79	18.03	15.47	5.95
10.0	12.0	4.42	18.82	23.48	5.76
12.0	12.0	6.56	20.49	32.01	5.04
8.0	10.0	1.22	15.41	7.90	2.61
8.0	12.0	1.95	17.25	11.29	4.86
8.0	14.0	2.62	19.14	13.71	7.70
8.0	16.0	2.92	21.07	13.87	7.65
8.0	18 .0	4.06	23.03	17.64	13.53

Table 4.1: Expected overheads in the proposed checkpointing scheme

table correspond to a *inter_at* of 25. The minimization problem for determining the best k_j was solved using the Fibonacci descent method. This method would lead to the optimal solution if the objective being minimized is unimodal. Otherwise, the results would be upper bounds to the actual value.

The table also shows the variation in the expected waiting times with changes in the parameters a_6 and b_6. It is clear from the table that the expected waiting time increases with a_6 for a constant b_6. This is because an increase in a_6 corresponds to a greater variance between the processes, and hence it is more difficult to coordinate the completion of acceptance tests by the processes. Similarly an increase in b_6 results in an increase in the expected waiting time. This is because an increase in b_6 implies that the estimate of the execution time to reach the j^{th} differs more from the actual execution time. For the purpose of comparison, the table also contains the expected wait time in Randell's checkpointing scheme [50]. Even for large values of a_6 and b_6 the expected wait times in the proposed checkpointing scheme are much less than in Randell's scheme. For example, $a_6 = 12$, $b_6 = 12$ and $inter\_at = 25$ corresponds to the case where the actual execution time to reach the sixth acceptance could vary between 126 time units to 174 time units. Even

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for this severe variation in the actual execution time, the expected wait time is only 32% of the expected wait time in Randell's scheme.

In addition to the reduced wait times, it also has fewer message exchanges for checkpointing purposes. The sixth column in Table 4.1 shows the probability of process sending a message at the sixth PRP for the best value of k_j. These values should be contrasted to a 100% probability of message exchange in Randell's scheme.

4.4 Discussion

The checkpointing algorithm described above has several desirable features including reduced time and space overhead. Processes have to establish only one PRP per pseudo-recovery line and preserve only two PRP's. From the analysis presented in this chapter, the expected waiting times and the probability of a process exchanging messages are shown to be much less as compared to Randell's checkpointing scheme [50].

The additional overheads in this scheme as compared to others are (i) the need for a global time base and (ii) the need to know the expected times for reaching the acceptance tests *a priori*. If a hardware synchronization algorithm is used to establish the global time base, then the time overhead on the system is almost minimal. The cost of the additional hardware is easily compensated by the reduced overhead in the checkpointing algorithm.

The expected times for reaching acceptance tests have to be estimated only once for every process. This can be easily done by executing the process repeatedly prior to their actual execution (mission). Since processes are usually repeatedly executed prior to the mission to ensure that there are no bugs in the program, these estimates can be obtained at no extra cost. Hence, the proposed checkpointing algorithm has high potential use for real-time applications.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

DIAGNOSIS OF BYZANTINE FAULTS

5.1 Introduction

Distributed systems often require a means by which nodes or cooperating processes can arrive at a mutual agreement on some information. This information may be clock values, values read from an input sensor, an output to an external actuator, system status, or any other data relevant to the operation of the system. As in synchronization, guaranteeing this agreement in the presence of Byzantine faults is not easy.

The problem of distributed agreement can be defined formally as follows. Given an N node distributed system with a maximum of m faulty nodes, the problem is to ensure consensus among all the non-faulty nodes on the private value of a particular node, called the *initiator*, of the agreement. By consensus we mean the following two *interactive consistency* conditions are satisfied.

IC1. All non-faulty nodes agree on the same value.

IC2. If the initiator is non-faulty, then all the non-faulty nodes agree on the initiator's private value.

Implicit in IC1 and IC2 is the idea that the agreement is synchronous in the sense that all the non-faulty nodes reach this agreement at the same time. In other words, there must be some real time at which all the non-faulty nodes have decided on the private value of initiator, and this time must be known in advance. Also implicit in IC1 and IC2 is the idea that the algorithm used

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to ensure this agreement should not in any way depend on the anticipated behavior of the faulty nodes. The above two conditions should be satisfied even if the non-faulty nodes cannot identify the faulty nodes.

Over the past decade, numerous algorithms have been proposed for ensuring interactive consistency in a distributed system. Pease, Shostak, and Lamport were the first to propose solutions to the above problem under the Byzantine fault model [37, 44]. Since then this problem has been studied extensively and several good solutions have been proposed [7, 9, 10, 12, 13, 66]. In fact, many of these algorithms have achieved the theoretical lower bound on the number of messages (phases) necessary to tolerate a specified number of faults. In spite of that, the overheads imposed by these algorithms are still too large to be useful in real-time applications. In addition, these algorithms do not address the problem of identifying the faulty nodes. They implicitly assume that the faulty nodes remain in the system until the end of the mission and as a result a larger number of faults need to be tolerated to meet a specified reliability requirement as compared to the case when the faulty nodes are diagnosed and then removed from the system during a mission.

To overcome the above limitations this chapter describes an on-line diagnosis scheme. The proposed scheme is comprised of a modified distributed agreement algorithm and a fault location algorithm that can be used in conjunction with each other to identify the nodes with Byzantine faults. The basic idea of this diagnosis scheme is to let the nodes execute a modified distributed agreement algorithm (instead of one of the existing algorithms) to reach the necessary consensus throughout the mission. This is exactly like any other system that tolerates Byzantine faults except that the agreement algorithm is slightly modified to help in diagnosis. At the end of each execution, the nodes execute a fault location algorithm in which each node examines the messages it received in the current execution of the agreement algorithm, identifies a set of suspicious nodes and conveys their identity to the other nodes in the system. The information from all nodes along with similar information from previous executions of the fault location algorithm is used to identify a set of malicious nodes[1].

It may happen that only some of the nodes that exhibited Byzantine behavior in the current execution of the agreement algorithm are identified. It may also happen that some malicious

[1] Nodes with Byzantine faults will be referred to as *malicious nodes*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

nodes that exhibited their Byzantine behavior in a previous execution of the agreement algorithm are now identified due to the additional information. The fault location algorithm aggregates all such information and feeds it back to the subsequent executions of the distributed agreement algorithm. The agreement algorithm adapts itself based on the feedback information to expedite the identification of the faulty nodes. It is the incorporation of this diagnosis information into the distributed agreement algorithm that makes this approach unique.

This chapter is organized as follows. A brief survey of related work is presented in the following section. This includes a survey on the existing agreement algorithms because they play key role in the diagnosis scheme. The issues that arise in the diagnosis of Byzantine faults are presented in the third section. A formal description of a generic distributed agreement algorithm, the system model, modifications to the agreement algorithm and the fault location algorithm are described in the fourth section. This is followed by a brief discussion of the merits and limitations of the proposed scheme.

5.2 Related Research

As mentioned earlier, Pease et. al were the first to propose a solution that guaranteed IC1 and IC2 in the presence of Byzantine faults [37, 44]. They proved that it is not possible to ensure IC1 and IC2 by using a simple majority voting algorithm. They also showed that in the absence of any restriction on the behavior of the faulty nodes, it is necessary and sufficient to have a total of $3m + 1$ nodes in order to tolerate a maximum of m faults. However, their algorithms require $O(N^m)$ messages and $m + 1$ phases[2].

An algorithm that requires polynomial number of messages was proposed in [9]. It requires only $O(Nm + m^3)$ messages and $2m + 3$ phases. This algorithm was later shown to be the best possible to within a constant factor when $N > m^2$ [12]. A different class of algorithm referred to as the authenticated algorithm was considered in [12, 13]. In the authenticated algorithms, it is assumed that a non-faulty node can authenticate all the messages it generates or relays by using an unforgeable digital signature [6, 52]. In a typical authenticated algorithm, a non-

[2] Informally, a phase consists of a synchronized message exchange where nodes first broadcast messages (according to their state), then wait to receive messages sent by other nodes in the same phase, and change their state accordingly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

faulty node appends its signature to every messages it sends. This signature contains a sample portion of the message encoded in such a way that any receiver can verify the authenticity of the message. Since nodes cannot forge the signature of another non-faulty node, they cannot make an undetectable change in the content of a message or introduce an undetectable spurious message into the information exchange.

This restriction on the behavior of faulty nodes results in algorithms that are simpler, more efficient, and tolerate more faults in the system than algorithms without authentication. The authenticated algorithms require only $m+1$ nodes as opposed to $3m+1$ nodes to tolerate m faults. They also require only $O(Nm)$ messages and $m+1$ phases as opposed to $O(Nm+m^3)$ messages and $2m + 3$ phases in the unauthenticated case [13]. However, the cryptographic techniques used for generating digital signatures require some computational and communication overhead. Moreover, they are not unconditionally secure from attacks by malicious nodes. Malicious nodes can break such schemes by computing or guessing the signature of another node, although the probability of such an occurrence is very small.

Srikanth et. al proposed a simple broadcast primitive that simulated authentication without the use of digital signatures [63]. Using that broadcast primitive they derived an authenticated algorithm that requires $3m+1$ nodes, $2m+1$ phases and $O(Nm^2)$ messages. They also described an early stopping algorithm in [66] that improved upon Dolev's algorithm [11] in both number of phases and messages. These two algorithms [11, 66] do not satisfy the implicit assumption of synchronous agreement.

As indicated previously, the above agreement algorithms do not attempt to identify the malicious nodes. They concentrate on efficiently masking the faulty nodes during the course of any information exchange. The problem of identifying the faulty nodes has just started receiving some attention. Yang and Masson have considered system level diagnosis in the presence of Byzantine faults [71]. Their diagnosis scheme is based on the assumption that when a non-faulty node tests another non-faulty node it can correctly conclude that the node is fault free. The faulty nodes affect the diagnosis scheme by corrupting the exchange of the "syndromes" between the non-faulty nodes. In [49], Ramarao and Adams also based their algorithm on the assumption that there are tests non-faulty nodes can use to correctly conclude that other non-faulty nodes are fault free. Therefore, the schemes in [49] and [71] are useful only if one can develop diagnostics

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that will efficiently detect the existence of a faulty node. Since several different faults can cause a Byzantine behavior, it becomes practically impossible to run diagnostics that can detect all possible faults.

A more natural way of identifying the malicious nodes is to observe the messages exchanged between the nodes and detect faults by identifying messages that are not consistent with the message exchange protocol. Since all malicious nodes will eventually exhibit Byzantine behavior[3] by not adhering to the specified message exchange protocol, the malicious nodes can be identified by noting the discrepancies in the messages exchanged between the nodes. This process is complicated by the fact that a node can exhibit Byzantine behavior even while reporting about some other nodes, thus making it impossible to rely fully on any report. However, as shown later in this chapter and also in [58], if the agreement algorithm is modified appropriately, there is an upper bound on the number of times a faulty node can exhibit such Byzantine behavior without being identified.

5.3 Issues in diagnosis of Byzantine faults

The foremost issue that distinguishes diagnosis under the Byzantine fault model from that under any other fault model is the ability of malicious nodes to cause a non-faulty node to accuse another non-faulty node. In diagnosis schemes that do not consider Byzantine faults, when p accuses q we can conclude that either p or q is faulty. This is not necessarily the case when some nodes can exhibit Byzantine behavior.

This issue is not a consequence of using message exchanges in an agreement algorithm to diagnose malicious nodes. The malicious nodes can cause non-faulty nodes to accuse other non-faulty nodes even if the diagnosis is based on message exchanges in any distributed computation. Furthermore, the use of an authentication mechanism does not eliminate this problem.

The second issue in diagnosis under the Byzantine fault model is that malicious nodes can behave in any way they like. As a result, they cannot be diagnosed as soon as they exhibit their faulty behavior. Since the diagnosis scheme here is strongly based on these two issues, they are elaborated below.

---

[3] If not, it does not matter to the system, and thus, they will be treated as non-faulty.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

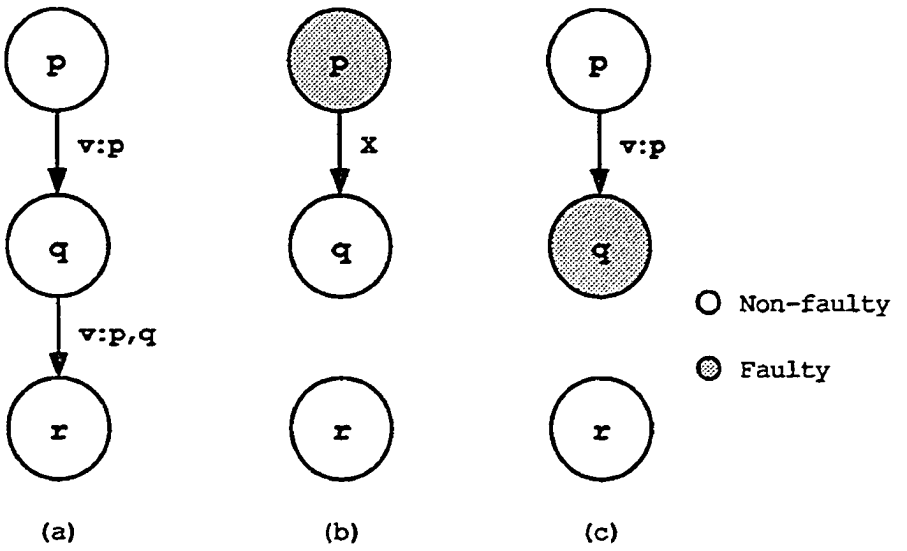

Figure 5.1: An example of a non-faulty node accusing another non-faulty node

Example 5.1: Consider the three cases shown in Figure 5.1. In the first case none of the nodes are faulty (Figure 5.1(a)). It therefore shows the correct execution of a distributed computation. In this case, p initiates the computation by sending an authenticated message of value v to q. On receiving this message q appends its own signature and forwards the message to r. In the second case p is the only faulty node (Figure 5.1(b)). In this case p initiates the computation by sending an inconsistent message X to q and q realizes that this is an inconsistent message and decides to ignore the message by not forwarding the message to r. In the third case q is the only faulty node (Figure 5.1(c)). Just as in the first case p initiates the computation by sending an authenticated message of value v to q. However, the faulty node q does not forward the message to r.

Since r perceives the same behavior in the latter two cases, it cannot distinguish between the two cases. But r can sense that something is wrong because the perceived behavior is different from that in the first case. Since the algorithm that r uses to generate the accusations is deterministic, r will generate the same accusations in the second and third cases. If r accuses q, then a non-faulty node would accuse another non-faulty node if the situation were as in Figure 5.1(b). On the other hand, if r accuses p, then a non-faulty node would accuse another non-faulty node if the situation were as in Figure 5.1(c). Hence, irrespective of which node (p or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

q) r accuses, there is a case in which r will accuse a non-faulty node.

Furthermore, in this example authentication cannot prevent a malicious node from causing a non-faulty node to accuse another non-faulty node. This result is true in general and is formalized in Theorems 5.1 and 5.2. ■

Theorem 5.1: A malicious node can cause a non-faulty node to accuse another non-faulty node if accusations are based on message exchanges in a distributed computation.

Proof: A general distributed computation can be modeled in terms of a function $\mathcal{G} : P \times \Lambda \times 2^M \times 2^M \times \mathcal{R}^+ \times P \rightarrow M$, where P is the set of nodes, Λ is the set of all possible states a node could be in, M is the set of all possible messages that can be exchanged during the execution of an agreement algorithm, and $\mathcal{R}^+$ is the set of non-negative reals. $\mathcal{G}(p, \sigma_t, S_t, R_t, t, q)$ has the semantics of the message sent by a node p at time t if it is in state σ_t and it has sent (received) all messages in S_t (R_t) in the time interval $[0, t)$.

Now consider the following two cases. In Case A, p is the only node that exhibits faulty behavior. In Case B, q is the only node that exhibits faulty behavior. We will then show that a non-faulty node r cannot distinguish between Cases A and B. Since the algorithm that r uses to generate accusations is deterministic, r will generate the same accusations in both cases. So irrespective of which node (p or q) r accuses, there is a case in which r will accuse a non-faulty node.

Case A: Suppose no node exhibits faulty behavior till time t_1. Also suppose that $\sigma^p_{t_1}$ and $\sigma^q_{t_1}$ are respective states of p and q at time t_1, and $S^p_{t_1}$ ($R^p_{t_1}$) and $S^q_{t_1}$ ($R^q_{t_1}$) are the sets of messages sent (received) by p and q, respectively, prior to t_1.

At time t_1 suppose that p exhibits its faulty behavior by sending a message $X \in M$ to q where $X \neq \mathcal{G}(p, \sigma^p_{t_1}, S^p_{t_1}, R^p_{t_1}, t_1, q)$. At some time $t_2 > t_1$, X will be received by q. This would cause q to send a message $\mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, R^q_{t_2}, t_2, r)$ to r. Note that q could decide not to send any message to r by setting $\mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, R^q_{t_2}, t_2, r)$ to a null message. Suppose that the message that q sends is different from the message that it would have sent had p not exhibited its faulty behavior at t_1, i.e.,

$$\mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, R^q_{t_2}, t_2, r) \neq \mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, C^q_{t_2}, t_2, r)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $C^q_{t_2} = (R^q_{t_2} - \{X\}) \cup \{\mathcal{G}(p, \sigma^p_{t_1}, S^p_{t_1}, R^p_{t_1}, t_1, q)\}$.

Case B: Suppose no node exhibits faulty behavior till time t_1 and let $\sigma^p_{t_1}$, $\sigma^q_{t_1}$, $S^p_{t_1}$, $R^p_{t_1}$, $S^q_{t_1}$, and $R^q_{t_1}$ be as in Case A. At time t_1, p correctly sends the message $\mathcal{G}(p, \sigma^p_{t_1}, S^p_{t_1}, R^p_{t_1}, t_1, q)$ to q. At some time $t_2 > t_1$, this message will be received by q. But now suppose q is faulty and it exhibits its faulty behavior at time t_2. Instead of sending r the message $\mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, R^q_{t_2}, t_2, r)$, suppose that q sends the message $\mathcal{G}(q, \sigma^q_{t_2}, S^q_{t_2}, D^q_{t_2}, t_2, r)$ to r, where $D^q_{t_2} = (R^q_{t_2} - \{\mathcal{G}(p, \sigma^p_{t_1}, S^p_{t_1}, R^p_{t_1}, t_1, q)\}) \cup \{X\}$.

Since no other node is assumed to exhibit its faulty behavior, the message that r receives in both cases are the same. So r cannot distinguish between Cases A and B. Since r has to accuse either p or q, it will accuse a non-faulty node in one of these two cases. This shows that even if the diagnosis were made using message exchanges in a general distributed computation, a malicious node can cause a non-faulty node to accuse another non-faulty node. ■

Theorem 5.2: Theorem 5.1 holds even if authentication is used for message exchanges.

Proof: Consider the example in the proof of Theorem 5.1. The question is whether both cases could occur if messages were authenticated. The answer is yes. This is because p is faulty in Case A. So if the message X does not contain the signature of p, then the message that q sends to r at time t_2 also cannot contain the signature of p. This implies that a faulty node q can generate the same message in Case B even if authentication is being used. Once again, r cannot distinguish between Cases A and B. Hence, it will accuse a non-faulty node in one of the two cases. ■

The accusations between two non-faulty nodes can, however, be eliminated by carefully processing the accusations made by all nodes in the fault location algorithm. During this elimination process some accusations made by non-faulty nodes on faulty nodes may also get eliminated, thus reducing the evidence against some faulty nodes. In spite of this the diagnosis scheme achieves the theoretical lower bound on the number of Byzantine behaviors required to identify all the malicious nodes.

Theorem 5.3: It is impossible to diagnose all malicious nodes as soon as they exhibit their faulty behavior.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: This can be reasoned as follows. Let p and q be any two nodes in the system such that p is faulty. Suppose that p sends a message to q that is not consistent with the message exchange protocol of the algorithm. When q accuses p of being faulty, the other nodes in the system cannot conclude that p is faulty because q could be a faulty node that generates a false accusation. There is no way to distinguish which of the two nodes p or q is actually faulty even if p repeats this faulty behavior infinitely often. ■

The modification to the distributed agreement algorithm takes care of this problem by preventing p and q from exchanging messages with each other once such a faulty behavior is detected.

5.4 Diagnosis Scheme

Given a distributed computing system, the goal is to develop a solution that not only masks malicious nodes but also detects and identifies them. Assume that the detection is based on the messages exchanged during a distributed agreement algorithm. This does not restrict the ability to detect Byzantine behavior because in a system that tolerates malicious nodes all distributed computations have to use an agreement algorithm for exchanging information.

However, distributed agreement algorithms as proposed in literature are not suitable for diagnosis of Byzantine faults. In fact, it can be easily shown that if the detection of Byzantine behavior is based solely on the existing distributed agreement algorithms, then it is possible for the malicious nodes to repeatedly exhibit their Byzantine behavior and yet remain unidentified. To overcome this problem, a modification is proposed to the existing distributed agreement algorithms. This modification is not specific to any particular agreement algorithm. As a result of using this modified agreement algorithm, it will be shown that there is an upper bound on the times of a node can exhibit its Byzantine behavior without being identified.

Let P be the set of nodes and let M be the set of all possible messages that can be exchanged during the execution of the agreement algorithm. M contains messages that are correctly formatted as well as messages that are incorrectly formatted. Each message has a value associated with it. The values of a message are taken from a set V. The value of a correctly formatted message represents the information that the nodes are trying to agree on. The value of an incorrectly formatted message is assumed to be an arbitrary value denoted by '?'. In addition to a value,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

each message may also be associated with a text. The text of a message contains information relevant to the agreement algorithm. Define a function $Value : M \rightarrow V$ such that $Value(mess)$ is the value of the message $mess \in M$.

A distributed agreement algorithm proceeds in *rounds*. Each round is comprised of three phases: *send phase*, *receive phase* and *compute phase*. Each round starts with the *send phase*. In this phase, nodes send all their messages for this round. The messages sent during the send phase depend on the messages sent or received in the previous rounds and on the compute phases of the previous rounds. In the receive phase, nodes receive all the messages sent to them in the current round. At the end of the receive phase, nodes enter a compute phase where they compute the messages to be sent in the next round. The algorithm terminates at the end of round *max_round*. There are some distributed agreement algorithms where such an upper bound on the number of rounds does not exist [11, 66]. The algorithms that have an upper bound are said to achieve *immediate* Byzantine agreement as opposed to *eventual* Byzantine agreement. In this dissertation only algorithms that achieve immediate agreement are considered.

The message exchange protocol for the distributed agreement algorithm can be specified in terms of a function $\mathcal{F} : P \times 2^M \times 2^M \times \mathcal{N} \times P \rightarrow M$, where $\mathcal{N}$ is the set of non-negative integers and $\mathcal{F}(p, S_k, R_k, k, q)$ is the message sent by a non-faulty node p to node q in round k when S_k (R_k) is the set of messages sent (received) by p prior to round k. $\mathcal{F}(p, S_k, R_k, k, q) = \epsilon$ has the semantics that no message is to be sent from p to q in round k when S_k and R_k are the messages sent and received prior to round k.

Definition 5.1: A message $mess$ sent by $p \in P$ to $q \in P$ in round l of the agreement algorithm is said to be *consistent* with respect to the message exchange protocol, $\mathcal{F}$, if $mess = \mathcal{F}(p, S_l, R_l, l, q)$, where S_l (R_l) are messages sent (received) by p prior to round l. A message that is not consistent is said to be *inconsistent*.

Definition 5.2: Node $p \in P$ is said to *accuse* a node $q \in P$, if p claims (in the fault location algorithm) that it received an inconsistent message from q during the agreement algorithm.

Definition 5.3: Nodes $p \in P$ and $q \in P$ are said to be a pair of *co-faulty* nodes if p non-faulty implies that q is faulty, and vice versa.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Definition 5.4: Node $p \in P$ is said to *suspect* $q \in P$ if p and q are a pair of co-faulty nodes.

Definition 5.5: An execution of the agreement algorithm is said to be *perfect* if all messages sent to the non-faulty nodes are consistent with respect to the message exchange protocol of the algorithm. If an execution of the agreement algorithm is not perfect, then it is said to be an *imperfect* execution.

There are usually several prerequisites for a distributed agreement algorithm. Since they are essential for masking the malicious nodes, assume that the system satisfies these prerequisites. In addition, assume that the system satisfies the following assumptions.

1. Byzantine behavior is caused by an underlying fault in the node. Hence, Byzantine behavior will recur whenever the fault is exercised.

2. A non-faulty node always sends the message correctly to the immediate receiver.

3. A node can always identify the node that last relayed this message irrespective of whether the message has the proper format or not.

4. Given (i) a node $p \in P$, (ii) a round k, $0 \leq k \leq max\_round$, (iii) S_k, the set of messages sent by p prior to round k, and (iv) R_k, the set of messages received by p prior to round k, any node $r \in P$ can determine $Value(\ \mathcal{F}(p, S_k, R_k, k, q)\ )$, the value of the message $\mathcal{F}(p, S_k, R_k, k, q)$, $\forall\ q \in P$.

The first assumption states that a faulty node will repeatedly exhibit Byzantine behavior. However, it does not restrict the kind of behavior a faulty node will exhibit each time it recurs. This is a reasonable assumption because a node that rarely exhibits Byzantine behavior would seldom harm the system. A node that exhibits Byzantine behavior only a few times will be masked but not always identified. The second and the third assumptions prevent a malicious node from snooping or altering the messages on direct links between two other nodes. They are easy to satisfy if nodes use dedicated hardware links instead of a broadcast medium to communicate with each other.

The fourth assumption states that the distributed agreement algorithm is deterministic in the sense that given all the information the value of the message to be sent can be determined by any

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

other node. This does not imply that a node can determine the entire message that some other node will generate. It can only determine the value of the messages. In particular, in the presence of authentication the entire message cannot be determined since the message would contain an unforgeable signature of p. This assumption is not a very restrictive one. To the best of our knowledge there is no existing agreement algorithm that does not satisfy this assumption.

5.4.1 Modifications to the agreement algorithm

The modifications to the agreement algorithm are clear by comparing Figures 5.2(a) and 5.2(b). Figure 5.2(a) contains the pseudo-code for an unmodified distributed agreement algorithm as executed by a node $p \in P$. The modified algorithm is shown in Figure 5.2(b).

In this modified algorithm, $CoFaulty(p)$ is the set of all nodes $q \in P$ such that p and q are a pair of co-faulty nodes. It is the feedback information from the fault location algorithm described in the following subsection. It is the use of this feedback information in the agreement algorithm that imposes an upper bound on the number of times a node can exhibit its Byzantine behavior without being identified.

Theorem 5.4 proves the correctness of the modified algorithm. Theorem 5.7 shows that this simple modification results in an upper bound on the number of times that a node can exhibit its Byzantine behavior without being identified.

Theorem 5.4: If the agreement algorithm in Figure 5.2(a) ensures IC1 and IC2, then the modified algorithm in Figure 5.2(b) also ensures IC1 and IC2.

Proof: The theorem follows from the argument given below. Consider a non-faulty node $p \in P$. In the modified algorithm, $p \in P$ does not send any message to nodes in $CoFaulty(p)$. On the other hand, the unmodified algorithm might require p to send a message to some nodes in $CoFaulty(p)$ in certain rounds.

The correctness of the unmodified algorithm cannot depend on the messages being sent to some faulty nodes. Since all nodes in $CoFaulty(p)$ are faulty if p is non-faulty[4], it does not matter whether p sends messages to them or not. ■

[4] This result will be proved later in Theorem 5.5.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Distributed Agreement Algorithm

Round 0

Send: If *initiator* then send $\mathcal{F}(p, \emptyset, \emptyset, 0, q)$ to each node $q \in P$.
Receive: Receive all the messages sent during the send phase.
Compute: Update S_1 and R_1. Compute the messages to be sent in the next round.

Round k, $1 \leq k < max\_round$

Send: Send $\mathcal{F}(p, S_k, R_k, k, q)$ to each node $q \in P$.
Receive: Receive all the messages sent during the send phase.
Compute: Update S_k and R_k. Compute the messages to be sent in the next round.

Round max_round

Send:
Receive:
Compute: Decide on the consensus value.

(a)

Modified Distributed Agreement Algorithm

Round 0

Send: If *initiator* then send $\mathcal{F}(p, \emptyset, \emptyset, 0, q)$ to each node $q \in P$, $q \notin CoFaulty(p)$.
Receive: Receive all the messages sent during the send phase.
Compute: Update S_1 and R_1. Compute the messages to be sent in the next round.

Round k, $1 \leq k < max\_round$

Send: Send $\mathcal{F}(p, S_k, R_k, k, q)$ to each node $q \in P$, $q \notin CoFaulty(p)$.
Receive: Receive all the messages sent during the send phase.
Compute: Update S_k and R_k. Compute the messages to be sent in the next round.

Round max_round

Send:
Receive:
Compute: Decide on the consensus value.

(b)

Figure 5.2: Unmodified and modified distributed agreement algorithms.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.4.2 Fault location algorithm

The fault location algorithm is executed in three phases. In the first phase, nodes scan the messages they have received during the course of the agreement algorithm and generate accusations from their point of view. In the second phase, each node that has a non-empty set of accusations initiates an execution of an agreement algorithm to convey its set of accusations to all the other nodes. In the third phase, nodes analyze the accusations they have received in the second phase and generate pairs of co-faulty nodes. After combining the generated pairs of co-faulty nodes with similar information from the previous executions of the algorithm, each node identifies a set of faulty nodes. The nodes can then re-execute the fault location algorithm for the agreement algorithms executed in the second phase. This process will terminate due to the upper bound on the number of times a node can exhibit its Byzantine behavior without identification.

In the fault location algorithm, $CoFaulty(p)$ is the set of nodes q such that nodes p and q are a pair of co-faulty nodes. $Faulty$ is the set of nodes that have been identified as faulty by the algorithm, and $\overline{P} = P - Faulty$. When the system starts, the sets $Faulty$ and $CoFaulty(p)$ are empty for all $p \in P$. Nodes are added/removed each time the algorithm is executed. The algorithm as executed by a node $p \in P$ is shown below.

Algorithm LOCATE

Phase 1

[1.1] Form a set $C(p) \subseteq V \times \overline{P} \times \overline{P} \times \{0, \ldots, max\_round\}$ such that $(v, p, q, l) \in C(p)$ if and only if v is the value of a message that p would receive from q in round l of the modified agreement algorithm if all the nodes in $\overline{P}$ are non-faulty.

[1.2] Form a set $R(p) \subseteq V \times \overline{P} \times (\overline{P} - CoFaulty(p)) \times \{0, \ldots, max\_round\}$ such that $(v, p, q, l) \in R(p)$ if and only if a message of value v was received from q in round l of the modified agreement algorithm.

[1.3] $Accusations(p) = [C(p) - R(p)] \cup [R(p) - C(p)]$.

Phase 2

[2.1] If $Accusations(p) \neq \emptyset$, then initiate the modified agreement algorithm to convey $Accusations(p)$ to all the other nodes in $\overline{P}$.

[2.2] If necessary, participate in the agreement algorithm initiated by other nodes in $\overline{P}$ and collect $Accusations(q)$ for all $q \in \overline{P}$.

[2.3] $Accusations = \bigcup_{q \in \overline{P}} Accusations(q)$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Phase 3

[3.1] Eliminate the accusations on non-faulty nodes by non-faulty nodes caused by malicious nodes as follows:

```
begin
    for each q ∈ P̄ do
        for l = 1 to max_rounds do
            for each r,s ∈ P̄ do
                if (v1,r,s,l) ∈ Accusations and (v2,q,r,k) ∈ Accusations, k < l, then
                    Accusations := Accusations − { (v1,r,s,l) };
            endfor;
        endfor;
    endfor;
    Suspicions := Accusations;
end;
```

[3.2] For each $q \in \overline{P}$ form a set $Suspect(q)$ as:

$$Suspect(q) = \{r : (v,q,r,l) \in Suspicions,\ r \notin CoFaulty(q)\}.$$

[3.3] Form a set of definitely faulty nodes, DF:

$$DF = \{q : (v,q,r,l) \in Suspicions,\ r \in CoFaulty(q)\}.$$

[3.4] For each $q \in \overline{P}$ update $CoFaulty(q)$ as follows:

$$CoFaulty(q) := CoFaulty(q) \bigcup Suspect(q) \bigcup \{r : q \in Suspect(r)\}.$$

[3.5] Update the set of definitely faulty nodes, DF:

$$DF = DF \bigcup \{q : \ |CoFaulty(q)| > m\}.$$

[3.6] Form a set of allowable fault sets that can give rise to the above $CoFaulty(\cdot)$.

$$PF = \{F : F \subseteq P,\ DF \subseteq F,\ |F \cup Faulty| \leq m,\ CoFaulty(q) \subseteq F,\ \forall\, q \notin F\}.$$

[3.7] Calculate $Faulty = \left(\bigcap_{F \in PF} F \right) \cup Faulty.$

[3.8] For all $q \in P$ set $CoFaulty(q) := CoFaulty(q) - Faulty.$ ■

In Phase 1, $C(p)$ is the set of all messages that p should have received during the course of the agreement algorithm and $R(p)$ is the set of messages it actually received. $Accusations(p)$ is the set of inconsistent messages that p has observed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In Phase 2, p initiates an agreement algorithm only if $Accusations(p)$ is non-empty. It follows from the discussions earlier, that even if p is non-faulty, $Accusations(p)$ may contain improper accusations on other non-faulty nodes. These improper accusations are eliminated in Phase 3. The operations in Step 3.1 can be informally explained as follows. If r accuses q in round k and s accuses r in round $l > k$, then eliminate the accusation by s and retain the accusation by r. This is because there is a possibility that a non-faulty node r sent an inconsistent message only because it received an inconsistent message from a faulty node q. If this were the case, then the accusation on r would have to be removed especially if s is non-faulty. This case cannot be distinguished from the five other possible cases: (i) r is faulty, and q and s are non-faulty, (ii) q and r are faulty, and s is non-faulty, (iii) q, r and s are faulty, (iv) q and s are faulty, and r is non-faulty, and (v) r and s are faulty, and q is non-faulty. Thus, the accusation on r would have to be deleted irrespective of the actual fault situation. If the fault situation is one of the above five cases, there is some loss of information towards the safe side. However, this loss in information does not affect the capability of diagnosing malicious nodes to a great extent.

Step 3.2 generates the set of nodes each $q \in \overline{P}$ suspects. In generating this set, the accusations of the form (v, q, r, l), $r \in CoFaulty(q)$, are ignored because a non-faulty node will not generate accusations of that form (see Step 1.2). From this argument it follows that any node that generates accusations of this form has to be faulty and hence Step 3.3. In Step 3.4, the set of co-faulty nodes is updated with the set of co-faulty nodes from the current execution. In Theorem 5.5 it is shown that these updates retain the property of co-faulty sets, i.e., even after the updates p is non-faulty implies all nodes in $CoFaulty(p)$ are faulty. This property of the sets $CoFaulty(\cdot)$ can be coupled with the fact that there is a maximum of m faulty nodes in the system, to conclude that any node q with $|CoFaulty(q)| > m$ is definitely faulty. This fact is used in Step 3.5.

In Step 3.6 all the possible fault sets that could have led to the current sets $CoFaulty(\cdot)$ are generated. From these sets some more faulty nodes are identified in Step 3.7. Some of these nodes may not have been suspected by $m + 1$ non-faulty nodes, but can still be concluded to be faulty due to their behavior. Finally, in Step 3.8, the sets $CoFaulty(\cdot)$ are updated to eliminate the nodes that have been identified as faulty.

Identifying each one of the sets in PF in Step 3.6 can be mapped onto a vertex cover problem. Since a vertex cover problem is known to be NP-hard, some heuristics may have to be used in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

place of Steps 3.6 and 3.7. Several good heuristics that reduced the complexity from $O(2^N)$ to $O(\max\{N^2, 2^{m^2}\})$ are described in [49]. If the maximum number of malicious nodes to be tolerated is small, then it may not be necessary to use any heuristics.

It should be noted that using heuristics instead of Steps 3.6 and 3.7 does not affect the correctness of the algorithm. Moreover, it also does not affect the upper bound on the number of faulty behaviors necessary for a guaranteed identification of all malicious nodes (see Theorem 5.7). Good heuristics only expedite the identification of the malicious nodes on the average.

A formal proof of correctness of LOCATE is presented later. An example to illustrate the various steps in the algorithm is given below.

Example 5.2: Consider a ten node example system in which at most three of them can be maliciously faulty. Denote these nodes by $p_1, p_2, \ldots, p_{10}$ and let p_1, p_2 and p_4 be the malicious nodes. Suppose further that the following faulty behavior is exhibited by the malicious nodes.

- In the first execution of the agreement algorithm, p_1 sends an inconsistent message to p_3. Also suppose that p_2 colludes with p_3 and accuses p_1 of sending an inconsistent message.
- In the second execution of the agreement algorithm, p_4 sends an inconsistent message to p_5 and p_6.

Consider the situation at the end of the first execution of the agreement algorithm. At the end of Step 3.4,

$$\begin{aligned} CoFaulty(p_1) &= \{p_2, p_3\} \\ CoFaulty(p_2) &= \{p_1\} \\ CoFaulty(p_3) &= \{p_1\} \\ CoFaulty(p_i) &= \emptyset \qquad \text{for } 4 \leq i \leq 10. \end{aligned}$$

As a result, $PF = \{ \{p_1\}, \{p_2, p_3\} \}$ and $Faulty = \emptyset$ at the end of Step 3.7.

Now consider the situation at the end of the second execution of the agreement algorithm. The sets $CoFaulty(p_1)$, $CoFaulty(p_2)$ and $CoFaulty(p_3)$ are as above while

$$\begin{aligned} CoFaulty(p_4) &= \{p_5, p_6\} \\ CoFaulty(p_5) &= \{p_4\} \\ CoFaulty(p_6) &= \{p_4\} \\ CoFaulty(p_i) &= \emptyset \qquad \text{for } 7 \leq i \leq 10. \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As a result, $PF = \{ \{p_1, p_4\}, \{p_2, p_3, p_4\}, \{p_1, p_5, p_6\} \}$ and again no node can be definitely identified as faulty.

Finally, suppose that the malicious nodes exhibit their Byzantine behavior once again in the third and the fourth executions of the agreement algorithm as follows:

- In the third execution, p_1 sends an inconsistent message to p_3.
- In the fourth execution, p_2 sends an inconsistent message to p_5 and p_7.

At the end of Step 3.4 we now get

$$\begin{aligned}
CoFaulty(p_1) &= \{p_2, p_3\} \\
CoFaulty(p_2) &= \{p_1, p_5, p_7\} \\
CoFaulty(p_3) &= \{p_1\} \\
CoFaulty(p_4) &= \{p_5, p_6\} \\
CoFaulty(p_5) &= \{p_2, p_4\} \\
CoFaulty(p_6) &= \{p_4\} \\
CoFaulty(p_7) &= \{p_2\} \\
CoFaulty(p_i) &= \emptyset \qquad \text{for } 8 \leq i \leq 10.
\end{aligned}$$

Although $|CoFaulty(q)| \leq 3 \ \forall \ q \in P$, all the three faulty nodes can be diagnosed this time because $PF = \{ \{p_1, p_2, p_4\} \}$. ■

Note that LOCATE was able to identify malicious nodes that have exhibited their faulty behavior only once. This is because the faulty behavior of a malicious node often increases the evidence against other malicious nodes. It also appears from the above example that if malicious nodes exhibit their faulty behavior often enough, then it is possible to diagnose all of them using the above algorithm. This notion is formalized in Theorem 5.7.

5.4.3 Proof of correctness

Lemma 5.1: For all $p \in \overline{P}$, if p is non-faulty, then every $q \in Suspect(p)$ is faulty.

Proof: Suppose not. Then, there exist non-faulty nodes p and q such that $q \in Suspect(p)$. This implies that at the end of Step 3.1 there exists a $(v_1, p, q, l) \in Suspicions$ for some $v_1 \in V$ and a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

round number $l \in \{0, 1, \ldots, max\_round\}$. Therefore, $(v_2, r, p, k) \notin Accusations$ for all $k < l$, $v_2 \in V$, and $r \in \overline{P}$. In other words, no faulty node exhibited its faulty behavior to p prior to round l, and p sent an incorrect message to q in round l. This is not possible. ■

Theorem 5.5: For all $p \in \overline{P}$, if p is non-faulty, then every $q \in CoFaulty(p)$ is faulty.

Proof: Follows from Lemma 5.1 and induction on successive executions of the distributed agreement algorithm. ■

Theorem 5.6: Algorithm LOCATE is correct, i.e., if $p \in Faulty$, then p is indeed faulty.

Proof: The result follows directly from the following facts: from Theorem 5.5 if p is non-faulty, then every $q \in CoFaulty(p)$ is faulty. So each element in PF does represent an allowable fault set that could have led to the current sets $CoFaulty(\cdot)$. Since PF contains all such allowable fault sets, a node that belongs to all these sets should indeed be faulty. Hence proved. ■

Lemma 5.2: If the $n+1^{th}$ execution of the agreement algorithm is imperfect and if $|Faulty(n+1)| = |Faulty(n)|$, then there exists a faulty node $q \in P$ such that $|CoFaulty(q, n)| < |CoFaulty(q, n+1)|$, where $Faulty(n)$ and $CoFaulty(q, n)$ denote the sets $Faulty$ and $CoFaulty(p)$ at the end of the n^{th} execution of the modified agreement algorithm.

Proof: Since the execution of the agreement algorithm is imperfect, there exists a non-faulty node p that receives a message that is not consistent with the message exchange protocol. This implies there exists a $v \in V$, $l \in \{0, \ldots, max\_round\}$, and $q \in \overline{P}$ such that $(v, p, q, l) \in Accusations$ at the end of Phase 2.

Consider Phase 3 of LOCATE. If $Accusations \neq \emptyset$, then $Suspicions \neq \emptyset$. Therefore, let $(v', p', q', l') \in Suspicions$. Since $|Faulty(n+1)| = |Faulty(n)|$, it follows from Step 3.6 that $DF = \emptyset$. This implies $q' \notin CoFaulty(p')$. Therefore, at the end of Step 3.2 $q' \in Suspect(p')$ and $p' \in Suspect(q')$. This in turn implies $|CoFaulty(p', n+1)| > |CoFaulty(p', n)|$ and $|CoFaulty(q', n+1)| > |CoFaulty(q', n)|$. The lemma then follows since both p' and q' cannot be non-faulty by Theorem 5.5. ■

Lemma 5.2 implies that a malicious node cannot exhibit its Byzantine behavior without increasing the evidence against at least one malicious node.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Theorem 5.7: A maximum of $m(m+1)$ imperfect executions of the agreement algorithm are needed to identify all the malicious nodes in the system.

Proof: Since each imperfect execution increases the cardinality of $CoFaulty(q)$ for some faulty node $q \in \overline{P}$, the cardinality of $CoFaulty(p)$ for all faulty node $p \in \overline{P}$ will become greater than m within $m(m+1)$ imperfect executions. The theorem then follows from Step 3.5. ■

We now show that any fault location algorithm requires at least $m(m+1)$ imperfect executions of the agreement algorithm for a guaranteed identification of all the m malicious nodes. In other words, the above algorithm is optimal.

Theorem 5.8: There is no fault location algorithm that can guarantee identification of all malicious nodes in less than $m(m+1)$ imperfect executions of the agreement algorithm.

Proof: Let $f_1, f_2, \ldots, f_m$ be the m malicious nodes in the system. Suppose initially only f_1 exhibits Byzantine behavior. After f_1 has been identified, only f_2 exhibits Byzantine behavior, and so on, till f_m. Also suppose each time a node exhibits its Byzantine behavior it does so by sending an inconsistent message to only one non-faulty node.

This implies each malicious node would have exhibited its Byzantine behavior $m+1$ times before it can be suspected by $m+1$ nodes irrespective of the fault location algorithm. Therefore, at least $m(m+1)$ Byzantine behaviors are necessary for a guaranteed identification of all the malicious nodes. ■

5.5 Discussion

The problem of identifying nodes with Byzantine faults was addressed. A modification to the distributed agreement and an associated fault location algorithm to identify all the malicious nodes in the system were proposed.

The modification proposed is not specific to any particular agreement algorithm. The key feature of the modification is that when it is used in conjunction with the fault location algorithm there is an upper bound on the number of times a node can exhibit its Byzantine behavior without being identified. More specifically, in a system that tolerates m Byzantine faults all the m faulty nodes can be identified within $m(m+1)$ imperfect executions of the agreement algorithm.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Furthermore, we have shown that this upper bound is optimal in the sense that $m(m+1)$ is also the theoretical lower bound on the number of Byzantine behaviors that are necessary for a guaranteed identification of all nodes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 6

DISCUSSION AND FUTURE WORK

The focus of this dissertation has been on problems related to designing distributed systems that are resilient to Byzantine faults. The main thrust of this work was to ensure synchronization and distributed agreement in the presence of Byzantine faults. Although these two problems have been addressed before, most of the existing solutions are economical only in small distributed systems. The key feature of this work is to develop solutions that are practical in small as well as large distributed systems.

We first presented solutions to eliminate some of the limitations in the existing algorithms for system level synchronization. The hardware schemes were extended to large distributed systems by addressing the associated interconnection and transmission delay problems. The solution to the transmission delay problem almost doubled the complexity of special circuitry at each node for every clock input. However, the number of inputs were reduced by as much 80 percent in some cases by the solution to the interconnection problem. Therefore, the combination of the two solutions resulted in a scheme that is considerably less expensive to implement.

In spite of this, the hardware schemes are quite expensive as compared to the software schemes and hence not suitable for all systems. For synchronizing systems in which hardware schemes are not cost-effective, a software scheme is proposed in this dissertation. The skews achieved by the scheme are not only very tight but are also insensitive to the maximum message transit delay in the system. These two advantages make the scheme ideal for synchronizing large distributed systems with homogeneous point-to-point interconnection topologies.

A combination of the hardware and software schemes discussed here can also be used to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

hierarchically synchronize a system to different degrees of tightness. As a case in point, real-time applications can often be partitioned into several groups of closely related tasks. The tasks belonging to the same group may require a very tight synchronization while the tasks belonging to different groups may be able to tolerate a looser synchronization. These applications can be easily mapped onto a distributed system comprised of nodes which are themselves multiprocessor systems. For example, in AIPS [26] fault tolerant uniprocessors are interconnected through a virtual bus to form clusters and the clusters are interconnected through gateways to form the entire system. A good scheme for synchronizing such a system would be to use the hardware scheme for intra-node and intra-cluster synchronization, and the software scheme for inter-cluster synchronization.

Node level synchronization was the next subject of the dissertation. A scheme was presented for delivering a clock signal to the components that constitute a "processor" of a node in a distributed system. This processor might either be a single chip or a set of few chips organized in a printed circuit board. The salient feature of the scheme is that the objective for determining the layout of the clock lines considers not only the maximum line lengths for delivering the clock signals but also the difference in the length of such lines. The delays introduced by the clock buffers were also taken into account when determining the layout by converting them into equivalent line lengths. This work is fairly unique because, to date, we are not aware of any scheme that considers the line lengths as well as the clock buffers in such a comprehensive manner.

A checkpointing scheme is then presented for efficiently recovering from software design errors during a mission. This is important because software design errors cannot be masked by using any form of redundancy. The scheme made use of the global time base to reduce the time as well as the space overhead it imposes. The reduction in the overheads were so significant that it further demonstrates the advantages of synchronization in a distributed system. It also suggests that there may be several other algorithms whose overhead can be reduced by making use of the global time base. Identifying these problems will further simplify the design of real-time systems and also amortize the cost of establishing a global time base.

Finally, a scheme was presented for identifying nodes with Byzantine faults. This scheme imposes an upper bound on the number of times a node can exhibit its Byzantine behavior

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

without being identified. This work is necessary because several algorithms for synchronization and distributed agreement already achieve the theoretical lower bound on the overheads they impose for tolerating a specified number of Byzantine faults. Therefore, the only way to reduce the overheads is to reduce the number of Byzantine faults to be tolerated. The identification of nodes with Byzantine faults is the first step towards realizing this goal. The next step is to determine the likelihood of a Byzantine fault occurring during a mission. This will eliminate the pessimistic assumption that all faults are Byzantine. It will also help avoid the unnecessary use of costly algorithms that are necessary to tolerate Byzantine faults.

Characterization of Byzantine faults is a difficult problem because the Byzantine fault model encompasses all possible faulty behaviors. For example, it is not clear how one would characterize the behavior of a faulty node that acts as an "evil intelligence". It might, however, be possible to characterize Byzantine faults that are a result of random hardware malfunctions. These random Byzantine faults (as opposed to intelligent Byzantine faults) can be characterized by first specifying the desired behavior and then studying the deviation from desired behavior in the presence of faults. Since desired behavior may be described at different levels of abstraction, a variety of deviations can be associated with a given hardware malfunction.

In order to justify the use of costly algorithms to tolerate Byzantine faults it is necessary to demonstrate that simple schemes such as encryption are not sufficient to mask all the deviations that occur at a given level of abstraction. Since faults occur infrequently it is not feasible to collect statistically significant data on deviations from desired behavior by simply observing the system. An approach to overcome this limitation is to monitor the behavior of the system after injecting a fault into it. This approach has the advantage that it can be carried out in simulation instead of the actual system. However, this approach would require further research in test generation and structured specification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

[1] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing and broadcasting in hexagonal mesh multiprocessors," To appear in *IEEE Trans. Comput.*

[2] F. Cristian, "Probabilistic clock synchronization," Technical Report RJ 6432 (62550), IBM Almaden Research Center, September 1988.

[3] F. Cristian, H. Aghili, and R. Strong, "Atomic broadcast: From simple message diffusion to Byzantine agreement," In *Proc. 15th Intl. Fault Tolerant Computing Symposium*, pp. 200–206, June 1985.

[4] A. Davis, R. Hodgson, B. Schediwy, and K. Stevens, "Mayfly system hardware," Technical Report HPL-SAL-89-23, Hewlett-Packard Company, April 1989.

[5] S. Dhar, M. A. Franklin, and D. F. Wann, "Reduction of clock delays in VLSI structures," In *Proc. Intl. Conf. Computer Design*, pp. 778–781, 1984.

[6] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. IT-22, pp. 644–654, November 1976.

[7] D. Dolev, "The Byzantine generals strike again," *Journal of Algorithms*, vol. 3, pp. 14–30, 1982.

[8] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of ACM*, vol. 34, no. 1, pp. 77–97, January 1987.

[9] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, "An efficient algorithm for Byzantine agreement without authentication," *Information & Control*, vol. 52, no. 3, pp. 257–274, March 1982.

[10] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, "Reaching approximate agreement in the presence of faults," *Journal of ACM*, vol. 33, no. 3, pp. 499–516, July 1986.

[11] D. Dolev, R. Reischuck, and H. R. Strong, "Eventual is earlier than immediate," In *Proc. 23rd Symp. Foundations of Computer Science*, pp. 196–203, November 1982.

[12] D. Dolev and R. Reischuk, "Bounds on information exchange for Byzantine agreement," *Journal of ACM*, vol. 32, no. 1, pp. 191–204, January 1985.

[13] D. Dolev and H. R. Strong, "Authenticated algorithms for Byzantine algorithms," *SIAM Journal of Computing*, vol. 12, no. 4, pp. 656–666, November 1983.

[14] J. W. Dolter, P. Ramanathan, and K. G. Shin, "A microprogrammable VLSI routing controller for HARTS," Technical Report CSE-TR-12-89, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, 1989.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[15] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of ACM*, vol. 35, no. 2, pp. 288–323, April 1988.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of ACM*, vol. 32, no. 2, pp. 374–382, April 1985.

[17] E. G. Friedman and S. Powell, "Design and analysis of a hierarchial clock distribution system for synchonous standard cell/macrocell VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 2, pp. 240–246, April 1986.

[18] M. R. Garey and D. S. Johnson, *Computer and intractability: A guide to the theory of NP–completeness*, Freeman, San Francisco, 1979.

[19] R. J. Gauthier, "The AIRLAB fault-tolerant processor: physical implementation," Contractor Report NAS1-18061, NASA, Langley Research Center, December 1986.

[20] J. Goldberg et al., "Development and analysis of SIFT," NASA contractor report 172146, NASA Langley Research Center, February 1984.

[21] A. Griefer and R. Strong, "DCF: Distributed communication with fault tolerance," In *Proc. 7th Symp. on Principles of Distributed Computing*, pp. 18–27, August 1988.

[22] J. Y. Halpern, B. Simons, R. Strong, and D. Dolev, "Fault-tolerant clock synchronization," In *Proc. 3rd Symp. on Principles of Distributed Computing*, pp. 89–102, 1984.

[23] S. Hasegawa, J. W. S. Liu, and M. Railey, "Reliable clock-driven process synchronization algorithms," In *Proc. 15th Intl. Fault Tolerant Computing Symposium*, pp. 207–212, June 1985.

[24] D. M. Himmelblau, *Applied non-linear programming*, pp. 274–292, New York: McGraw Hill, 1972.

[25] A. L. Hopkins, T. B. Smith, and J. H. Lala, "FTMP - a highly reliable fault-tolerant multiprocessor for aircraft," *Proc. of the IEEE*, vol. 66, no. 10, , October 1978.

[26] M. W. Johnson et al., "Advanced information processing systems (AIPS) system requirements (revision 1)," Technical Report CSDL-C-5709, Charles Stark Draper Lab. Inc., October 1984.

[27] D. D. Kandlur, "Networking issues in distributed real-time systems," Technical report, Real-Time Computing Laboratory, Dept. of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, May 1989.

[28] G. Kane, *MIPS R2000 RISC Architecture*, Prentice-Hall, Englewoods Cliffs, New Jersey, 1987.

[29] J. L. W. Kessels, "Two designs of a fault-tolerant clocking system," *IEEE Trans. Comput.*, vol. C-33, no. 10, pp. 912–919, October 1984.

[30] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Trans. Comput.*, vol. 37, no. 4, pp. 398–405, April 1988.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[31] K. H. Kim, "Approaches to mechanization of conversation scheme based on monitors," *IEEE Trans. Software Engg.*, vol. SE-8, no. 3, pp. 189–197, May 1982.

[32] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, vol. SE-13, no. 1, pp. 23–31, January 1987.

[33] C. M. Krishna, K. G. Shin, and R. W. Butler, "Ensuring fault tolerance of phase-locked clocks," *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 752–756, August 1985.

[34] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems," *ACM Trans. on Programming Languages and Systems*, vol. 6, no. 2, pp. 254–280, April 1984.

[35] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," CSL Technical Report 141, SRI International, 1982.

[36] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of ACM*, vol. 32, no. 1, pp. 52–78, January 1985.

[37] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Prog. Languages and Systems*, vol. 4, no. 3, pp. 382–401, July 1982.

[38] I. Lee and S. B. Davidson, "Adding time to synchronous process communications," *IEEE Trans. Computers*, vol. C-36, no. 8, pp. 941–948, August 1987.

[39] I. Lee and S. B. Davidson, "A performance analysis of timed synchronous communication primitives," Technical Report MS-CIS-87-101, Dept. of CIS, School of Engg. and Applied Science, Univ. of Pennsylvania, November 1987.

[40] Y.-H. Lee and K. G. Shin, "Design and evaluation of a fault-tolerant multiprocessor using hardware recovery blocks," *IEEE Trans. Comput.*, vol. C-33, no. 2, pp. 113–124, February 1984.

[41] W. C. Lindsey, A. V. Kantak, and A. Dobrogowski, "Network synchronization by means of a returnable timing system," *IEEE Trans. Communications*, vol. COM-26, no. 6, pp. 892–896, June 1978.

[42] D. G. Luenberger, *Linear and Non-linear Programming*, Addison Wesley, second edition, 1984.

[43] J. Lundelius-Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[44] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of ACM*, vol. 27, no. 2, pp. 228–234, April 1980.

[45] S. Powell, W. R. Smith, and G. Persky, "A parasitics extraction program for closely-spaced VLSI interconnects," In *Proc. Intl. Conf. Computer-Aided Design*, pp. 193–195, November 1985.

[46] P. Ramanathan and K. G. Shin, "Checkpointing and rollback recovery in a distributed system using common time base," In *Proc. 7th Symposium on Reliable Distributed Systems*, pp. 13–21, October 1988.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[47] P. Ramanathan and K. G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Comput.*, vol. 37, no. 12, pp. 1654–1657, December 1988.

[48] P. Ramanathan and K. G. Shin, "Clock distribution scheme for a general VLSI circuit," In *To appear in Intl. Conf. on Computer-Aided Design*, November 1989.

[49] K. V. S. Ramarao and J. C. Adams, "On the diagnosis of Byzantine faults," In *Proc. 7th Symposium on Reliable Distributed Systems*, pp. 144–153, October 1988.

[50] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Engg.*, vol. SE-1, no. 2, pp. 220–232, June 1975.

[51] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability issues in computing system design," *ACM Comput. Surveys*, vol. 10, no. 2, pp. 123–165, June 1978.

[52] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, February 1978.

[53] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. on Comput.*, vol. 37, no. 7, pp. 867–872, July 1988.

[54] C. L. Seitz, "The Cosmic Cube," *IEEE Trans. Comput.*, vol. C-34, no. 1, pp. 22–33, January 1985.

[55] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. Software Engg.*, vol. SE-10, no. 6, pp. 692–700, November 1984.

[56] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. Comput.*, vol. C-36, no. 11, pp. 1328–1341, November 1987.

[57] K. G. Shin and P. Ramanathan, "Clock synchronization of a large multiprocessor system in the presence of malicious faults," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 2–12, January 1987.

[58] K. G. Shin and P. Ramanathan, "Diagnosis of processors with Byzantine faults in a distributed computing system," In *Proc. 17th Intl. Fault Tolerant Computing Symposium*, pp. 55–60, July 1987.

[59] K. G. Shin and P. Ramanathan, "Transmission delays in hardware clock synchronization," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1465–1467, November 1988.

[60] M. Shoji, "Elimination of process-dependent clock skew in CMOS VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 5, pp. 869–880, October 1986.

[61] T. B. Smith, "Fault tolerant processor concepts and operation," In *Proc. 14th Intl. Fault Tolerant Computing Symposium*, June 1984.

[62] T. B. Smith and J. H. Lala, "Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer volume I: FTMP principles of operation," Contractor Report 166071, NASA, May 1983.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[63] T. K. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," Technical Report 84-623, Dept. Comput. Sci, Cornell Univ., July 1984.

[64] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Journal of ACM*, vol. 34, no. 3, pp. 626–645, July 1987.

[65] K. S. Stevens, "The communication framework for a distributed ensemble architecture," AI Technical Report 47, Schlumberger Research Laboratory, February 1986.

[66] S. Toueg, K. J. Perry, and T. K. Srikanth, "Fast distributed agreement," *SIAM Journal of Computing*, vol. 16, no. 3, pp. 445–457, June 1987.

[67] N. Vasanthavada and P. N. Marinos, "Synchronization of fault-tolerant clocks in the presence of malicious failures," *IEEE Trans. Comput.*, vol. 37, no. 4, pp. 440–448, April 1988.

[68] K. D. Wagner, "A survey of clock distribution techniques in high-speed computer systems," Technical Report CRC 86-20, Center for Reliable Computing, Stanford University, December 1986.

[69] D. F. Wann and M. A. Franklin, "Asynchronous and clocked control structures for VLSI based interconnection networks," *IEEE Trans. Comput.*, vol. C-32, no. 3, pp. 284–293, March 1983.

[70] M. W. Williard, "Analysis of system of mutually synchronized oscillators," *IEEE Trans. Communication Technology*, vol. COM-18, no. 5, pp. 892–896, October 1970.

[71] C. L. Yang and G. M. Masson, "A distributed algorithm for fault diagnosis in systems with soft failures," *IEEE Trans. on Comput.*, vol. 37, no. 11, pp. 1476–1480, November 1988.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.