The Automated Partitioning of Simulations for Parallel Execution

David M. Nicol, Paul F. Reynolds, Jr.

Computer Science Report No. TR-85-15
August 1985

This research was supported by grants from the IBM Corporation, and Virginia's Center for Innovative Technology.

Abstract

One of the most active areas in computer science research today is the study of parallel computation. A critical problem in this field is that of *partitioning*: the decomposition of a problem, program, or algorithm into pieces suitable for parallel execution. We consider the problem of partitioning *simulations* automatically for parallel execution. Before we can study the partitioning of simulations, we must be able to analyze simulations. Towards this end, we construct a formal model of simulations, study a simulation's run-time behavior, and explore the effect of synchronization on run-time performance. We develop static and statistical methods of analyzing a distributed simulation's performance. We then propose a simulation partitioning algorithm and examine the performance of partitions it creates. Finally, we develop an optimal dynamic repartitioning decision process.

TABLE OF CONTENTS

LIST OF DEFINITIONS

LIST OF LEMMAS

LIST OF THEOREMS

LIST OF COROLLARIES

LIST OF FIGURES

Chapter 1

Introduction

One of the most active areas in computer science research today is the study of parallel computation. A critical problem in this field is that of *partitioning*: how should we decompose a problem, program, or algorithm into pieces suitable for parallel execution? Our research is motivated by the observation that *simulations* form an interesting and important class of computer programs which might be amenable to parallel computation; many systems that we simulate have inherently concurrent activity which might be simulated in parallel. In this dissertation we develop an analytic framework supporting the study of simulation partitioning. As simulation partitioning raises issues not considered before in a partitioning context, this research is prerequisite for a study of simulation partitioning algorithms. We then use this framework to develop a *static* simulation partitioning algorithm, and an optimal *dynamic* repartitioning decision algorithm.

In this chapter we define the fundamental problems in partitioning a simulation; we outline the existing related research, illustrating the context in which our research should be taken. Finally, we present a detailed overview of this thesis.

1.1. Partitioning Simulations

We can view a simulation program as a collection of software modules. Each software module is responsible for some control function, or for emulating the behavior of some physical process. A *partition* of a simulation is an assignment of each software module to some processor in a multi-processor (we presume a multiple instruction multiple data, or MIMD, computer system). Every execution of a software module is performed in its assigned processor. The objective of a simulation partitioning algorithm is to find an assignment of modules such that the overall execution time is significantly reduced.

Partitioning simulations is substantially different from other partitioning problems considered in the literature. Problems include (i) the computational requirements of a simulation are not static; (ii) a simulation cannot be treated as a fixed task structure under some precedence relation; (iii) before we can treat partitioning, we have to understand the relationship between synchronization and the simulation execution time; (iv) if the simulation's execution requirements are known, we have to be able to calculate the simulation's execution time; (v) if the simulation's execution requirements are not known, we have to be able to estimate the simulation's average execution time; (vi) we have to be able to compare two partitions of the same system, and determine which partition is more effective.

Much of this thesis is devoted to the treatment of these problems. Our solutions develop an analytic framework supporting a study of partitioning. Our development highlights issues intrinsic to simulations and their distribution for parallel execution. Within this framework, we also propose a partitioning algorithm, and consider the issue of **dynamically** repartitioning a running simulation.

1.2. Context for Simulation Partitioning

There are four categories of research related to the our research. The first of these is the study of partitioning task structures for parallel execution. The vast bulk of the research in this area assumes that a number of independent communicating tasks are to be assigned to a multi-processor system. The inter-task communication traffic is known; a communication between two tasks suffers delay only if the communicating tasks are assigned to different processors. The computer system is assumed to be composed of a number of heterogeneous processors. The assignment (partition) sought minimizes the amount of communication between processors plus the sum of the execution costs, subject to memory constraints in the processors. [Bok81, DKW82, El-78, GaT84, Lo81, Lo84, Pri79, PrK84, Sto77] are examples of proposed solutions. These solutions are related to those searching for an optimal K-partition of a graph; examples of this work include [FiM82, KeL70, Kri84]. This formulation of the partitioning problem is inappropriate for simulations primarily because it ignores precedence relations which may exist between tasks. We will see that simulations define precedence relations which govern the execution order of tasks. The approach described in [ChA82] allows only a constrained form of precedence. This approach appears to be most appropriate for small systems with little concurrency. The approach in [Hol82] assumes a static collection of tasks under some precedence relation. The execution delay of each task is known, and the volume of communication between two tasks is known. A communication between two processors suffers a communication delay. The partition yielding minimal task system execution is sought. The algorithm given in [Hol82] is no more than the exhaustive enumeration of all possibilities. In [Sal83, Sip84] we find heuristics for determining a good partition for this same problem. However, the problem formulation in [Hol82, Sal83, Sip84] is not appropriate to partitioning a simulation. We will see that a simulation's task structure is dynamic, and its execution requirements are dynamic. Thus a simulation's partition should be chosen so that it is effective on all possible task structures, not just some fixed task structure. Another area related to partitioning is deterministic and probabilistic task scheduling [Cof76]. Scheduling models differ from our own in their assumption of a central scheduler and static task system structure.

A second area of research related to the partitioning of simulations is the general field of distributed simulation. The initial work in distributed simulation focused on synchronization protocols which guarantee freedom from deadlock in the running simulation [ChM79, JeS82, NiR84, PMW80, Rey82] . Recently researchers are considering how a simulation's different control and statistics gathering functions might be placed on different processors [Com82, WyS84]. The latter approaches still require that the heart of the simulation, its emulation routines, be on one processor. A hierarchical approach to defining a simulation model has been proposed by [Zei84]. This approach serves as a base for a methodology for performing distributed simulation as described in [Con85]. While this is a comprehensive treatment covering model, synchronization, and architecture, the approach in [Con85] is notably weak in its treatment of partitioning. This approach proposes that every simulation entity or software emulation module be given its own processor. Our research is based on quite a different assumption: that hardware resources are limited, and must be intelligently shared. Even if a processor were available for every simulation entity, it is arguable that the communication channels would create bottlenecks and unnecessary delays in the control of the simulation.

A third related area is that of modeling system performance: we will have to assess the performance of a distributed simulation. PERT network analysis [Elm67, HaW66, Mar65, RoT77] is not applicable as it assumes static task structures and independent activity times. A multitude of computer system models exist, for example [DuB82, RaH80, SmL82, TCB78, YYS81]. We've found no treatment which adequately describes a running simulation. Similarly, queueing network models [Kle76] are

inappropriate.

Our final technical chapter examines *dynamic* partitioning. Dynamic load balancing is currently receiving much attention [ChK79, ELZ, Ni81, RaS84, Sta84b, Sta85, TaT85]; but again the requirements of a simulation model are not met by these models. These models primarily treat queueing networks with independent tasks.

This review of the literature highlights the fact that the development of techniques to study simulation run–time behavior, and the development of static and dynamic simulation partitioning algorithms, are timely research problems. They represent the next logical research step in the field of distributed simulation, and serve as an extension to existing work in the field of partitioning and modeling.

1.3. Outline of Thesis

This thesis examines problems that must be solved before we can attempt to intelligently partition a simulation. It proposes a partitioning algorithm, and studies the problem of dynamically partitioning a simulation. We outline our treatment of these problems on a chapter by chapter basis.

Chapter 2: A Simulation Model

In Chapter 2 we lay the base for our analysis by describing a model of simulations. This model is independent of a simulation's implementation. The major result of Chapter 2 is the identification of *proper sequencing rules* which must be followed to obtain correct simulation results. Proper sequencing rules are the basis for modeling a distributed simulation's synchronization requirements.

Chapter 3: Run–Time Behavior

In Chapter 3 we look at the run–time behavior of a simulation. We propose simplifying assumptions under which the problem of finding a good partition for a simulation reduces to finding a good partition for a *cycle* of the simulation. We then show that the amount of simulation work required in one cycle of the simulation is a random variable; thus the execution time of a cycle is also a random variable which depends on the chosen partition.

Chapter 3's main contribution is to give conditions under which we can confine our analysis to *cycles* of the simulation It then shows that these cycles behave randomly. Thus, to evaluate a partition, we must evaluate its average performance over all possible cycles. These observations are important steps in defining the problems a partitioning algorithm will have to consider.

Chapter 4: Synchronization and Cycle Execution Time

Chapter 4 explores the relationship of synchronization to the cycle execution time. It defines the construction of graphs which describe a simulation's synchronization requirements. It defines a synchronization protocol related to these graphs, and shows that the performance of a simulation using this protocol is optimal over the class of "non–predictive" synchronization protocols. We then describe an efficient means of calculating the execution time of a given cycle running under this optimal protocol.

Chapter 4's significance lies in its development of modeling tools. It gives us graphical means of describing all simulation work that might be required during a cycle. These graphs form the underlying basis for much of our later analysis. Chapter 4 gives us a means of modeling synchronization behavior, and it gives us a means of calculating a given cycle's execution time. These are clearly problems that must be addressed if we are to numerically evaluate how well a partition reduces the simulation execution time over

a sequential simulation.

Chapter 5: Probabilistic Analysis

With the recognition that a partition's performance must be evaluated probabilistically, Chapter 5 develops an exact probabilistic analysis of a simulation's behavior. This development identifies assumptions that must be made to support a tractable analysis. In doing so, we discover that **general** simulations are too complicated to allow an exact analytic analysis. The methods we develop in Chapter 5 can be used to approximate a simulation's behavior. These approximations can be useful to a partitioning algorithm, as well as the statistical analysis discussed in Chapter 6.

The major contribution of Chapter 5 is to illustrate that most simulations are not amenable to an exact, static (pre-run-time) analysis. In the general case, the analysis methods given are useful for first order approximations of the simulation's behavior, but cannot be relied on for quantitative accuracy.

Chapter 6: Statistical Analysis

Given that we must evaluate a partition probabilistically, and given that most systems of interest cannot be analyzed with confidence using static probabilistic analysis, we consider the statistical analysis of a running simulation's behavior. Chapter 6 resolves certain statistical issues that must be addressed if we are to substantiate a statistical approach. It proposes a Bayesian estimation of the simulation's cycle execution time mean and standard deviation. It shows how a Bayesian framework can be employed to choose the "better" of two partitions of the same system.

The most important contribution of Chapter 6 is to show that a statistical approach to estimating a partition's performance can be justified. The underlying probabilistic basis for a statistical analysis is shown to exist; and appropriate statistical tools are identified.

Chapter 7: Qualitative Analysis

One method of comparing the performance of two partitions is to use the statistical techniques of Chapter 6 to estimate their respective cycle execution time moments. Such a comparison is by nature **quantitative.** Chapter 7 argues that certain computational savings might be realized by comparing two partitions **qualitatively.** It demonstrates conditions under which we can establish that one partition is superior to another without ever calculating their respective cycle execution time moments. It also identifies an important application for qualitative analysis. We employ time complexity arguments to show that the use of Chapter 7's results can significantly reduce the computational complexity of comparing two very similar partitions.

Chapter 8: A Partitioning Heuristic

Chapter 8 proposes a partitioning algorithm for simulations. This algorithm's decisions are based on a collection of observed simulation cycles; as such it is a *statistical partitioning algorithm.* We study the performance of the partitions produced by this algorithm on certain logic network simulations, and we argue that the partitions produced are effective.

The importance of Chapter 8 is self-evident: the development of partitioning algorithms for simulations is an important problem. While our development and testing of this algorithm is by no means complete, it is an important first step in the study of simulation partitioning algorithms.

Chapter 9: Dynamic Partitioning Decisions

This thesis proposes a statistical approach to partitioning a simulation, and to analyzing that partition's performance. Any partition which is based on statistical observations is vulnerable to a radical shift in the dynamics of the running simulation. To counter this problem, Chapter 9 develops a decision process which monitors the behavior of the running simulation. This decision process detects change in the dynamics of the run–time behavior, and decides whether a new partition is warranted. This decision process is shown to optimally balance the expected costs of repartitioning with the expected benefits from repartitioning.

Chapter 9's importance lies in its **completion** of the statistical approach to partitioning proposed by this thesis. A radical change in a simulation's behavior is the anathema of a statistically based partition, and this chapter has shown how to detect and optimally react to such a change.

Chapter 10: Conclusions

Chapter 10 presents our conclusions, and ideas for future research.

Chapter 2

A Simulation Model

2.1. Chapter Overview

This chapter models a class of simulations. Such a treatment is prerequisite for issues such as optimal partitioning of a simulation system and the performance analysis of a partitioned simulation. The model we develop usefully describes how simulation entities interact with each other at run-time. This description leads us to a natural definition of simulation correctness, and the identification of rules which ensure correctness.

Other authors have proposed models of simulations. [Sch83] proposes a graphical representation of simulations. This model allows simulation entities to affect each other in extremely general ways. This generality makes tractable analysis very difficult. Nor does this model explicitly include any information about the simulation's execution requirements. [ChM79] describes a model of *distributed* simulation. This model focuses on the synchronization behavior of the distributed processes. While this model is not suited for the type of analysis prerequisite to partitioning a simulation, our conception of simulation correctness is largely drawn from this presentation. An entirely general hierarchical model of simulations has been proposed in [Zei84]. While very powerful, we find again that the generality of the model excludes much of the quantitative analysis a partitioning algorithm must perform. Nor does this model explicitly identify the simulation's components' execution requirements.

The model proposed in this chapter is important principally in that it identifies certain **restrictions** which allow a tractable quantitative analysis not possible with the general models. Our model also clearly exposes how simulation entities affect each other; this exposure leads us to an understanding of simulation correctness and synchronization requirements.

The first subsection of this chapter formally specifies a simulation system. We discuss event scheduling in section 2.3. Section 2.4 examines simulation execution correctness, and proves that if *proper sequencing rules* are enforced within a simulation, then that simulation's execution will be correct. We discuss the implication of this result with regard to synchronization in distributed simulations.

2.2. Specifying a Simulation

Often simulation model designers think of simulations as systems of interrelated functions, e.g., a boolean logic network. Each function is given a logical delay distribution describing the input response time of the physical process modeled by the function. The specifications of the functions, their inter-dependencies, and their associated delay distributions are sufficient information for the simulation of the system. We now formalize this intuitive description of a simulation.

Simulation models are concerned with modeling physical processes by software routines or *logical processes*. A logical process is formally defined to be a function. We simulate a physical process' response to some input by evaluating its representative logical process function. This physical process' response can affect other physical processes. To support the formal definition of a simulation model, we first state the definition of a

stochastic process given in [Ros83]. A stochastic process $\underline{X} = \{ X(s), s \in S \}$ is a collection of random variables. For each $s \in S$ in the index set S, $X(s)$ is a random variable; $X(s)$ is called the *state* of $\underline{X}$ at time s. We define a simulation's input generators to be stochastic processes.

Definition 2-1: *Generator Event Functional*

Let Υ be a set of states. A *generator event functional* T_i is a stochastic process whose random variables take states from Υ. T_i's index set is the set of non-negative real numbers.

□

The collection of all generator events in a simulation serve as the simulation system's input. The simulation's response to its input is modeled with *normal event functionals.*

Definition 2-2: *Normal Event Functional*

Let Υ be a set of states, and let $n_i > 0$ be an integer. A *normal event functional* T_i is a function

$$T_i : \Upsilon^{n_i} \times \Upsilon \rightarrow \Upsilon.$$

The *state* of T_i is described by a stochastic process $\underline{T_i}$; $\underline{T_i}$'s index set is the set of non-negative real numbers.

□

We interpret Υ as a set of event functional states, and the vector Υ^{n_i} as a vector of n_i inputs; the other component of T_i's domain includes T_i's own state value at some epoch. We will describe rules that govern when and how an event functional's state changes. For the present, it is sufficient to know that T_i's state and inputs are elements of Υ, and that T_i's state will be described as a stochastic process.

Our model associates a *logical delay* probability distribution function with each event functional. This delay represents the real-time duration of the physical process modeled by the event functional. We assume that every such delay is strictly bounded from below by some constant positive number ϵ_B. Each event functional also has an *execution delay* random variable describing its evaluation's execution requirements.

We can now formally define a simulation system in terms of its event functionals, inter-functional relationships, and delay random variables.

Definition 2-3: *Simulation System*

A *simulation system* is a tuple $<\Upsilon, T, I_S, R, L, \chi>$

- Υ is a set of states;
- T is a set of event functionals $T = \{T_1, \cdots, T_n\}$; each normal event functional's domain is a subset of Υ^{n+1}, and its range is the set Υ;
- I_S is an n-vector describing the n initial states of the event functionals, so $I_S \in \Upsilon^n$;
- R is a relation $R \subset T \times T$, R defines the event functionals' inter-relationships; if $<T_i, T_j> \in R$ we say that T_j is a *successor* of T_i, and that T_i is a *predecessor* of T_j;
- L, the logical delay function, maps T into the set of all positive random variables bounded from below by $\epsilon_B > 0$;
- χ, the execution delay function, maps T into the set of all positive random variables.

□

Some components of a simulation system need further explanation. If $<T_i, T_j> \in R$, T_i's state acts as an input to T_j; we temporarily defer specifying the exact mechanics of how and when T_i's state affects T_j's. The random variable $L(T_i)$ is the logical delay associated with normal functional T_i; for a generator functional T_g, $L(T_g)$ is the random delay between T_g's state transitions. Likewise, $\chi(T_i)$ is the execution delay random variable associated with functional T_i.

The simulation system description gives rise to a naturally defined graph, the *system graph*. The nodes of the system graph are simply the event functionals T_i. A directed edge exists from node T_i to T_j if and only if T_j is a successor of T_i. Having defined the components of a simulation, we now treat the execution of a simulation.

2.3. Simulation Event Scheduling

The proper execution of a simulation program requires a careful scheduling of event functional evaluations. In this section we define an intuitive sense of correctness, and specify scheduling rules that ensure this correctness.

The following discussion uses three different time lines. The *physical* time line refers to time in the physical process being modeled; the *logical* time line is the simulation's model of actual time. The *electronic* time line is the time line of the computer running the simulation. We suppose that the simulation program treats each functional's state as a variable. Then the execution of a simulation program transforms event functionals' states at different points in electronic time; a functional's state can thus be described as a stochastic process in electronic time. We presume that a functional's evaluation is performed by an associated module of software called a logical process;

execution of a logical process begins by accepting input values for the functional evaluation, it then evaluates the functional according to its mathematical definition, assigns the result of this evaluation to the functional's state variable, and terminates. The execution of this logical process is called an *evaluation* of the functional; it is also called an *execution* of the functional. The execution starting epoch is called the *electronic starting time*; the execution termination epoch is called the *electronic completion time*. We note that the difference between an evaluation's electronic starting and its electronic completion time is a random variable. We assume that a functional's state can be modified only by an evaluation of the functional; the execution completion times are thus the state transformation epochs of the functional's state stochastic process in electronic time. There are also logical times associated with a functional's evaluation. A *logical initiation time* is associated with the presentation of inputs to the functional's software process. This time models the physical time at which the physical process modeled by the functional is stimulated by some input. A random *logical delay* is associated with the action of evaluating the functional; the logical delay models the response delay of the modeled physical process. The *logical completion time* is equal to the sum of the logical initiation time and the logical delay. The logical completion time models the time at which the modeled physical process' state is transformed in response to input.

An *event* is the evaluation of a functional. We describe an event *generated* by functional T_i by a vector $<T_i, v, t, t + d, l_e, u_e>$; T_i is the evaluated functional, v is the result of the functional evaluation, t is the logical initiation time, d is the logical delay, l_e is the execution starting time, and u_e is the execution completion time. An *execution* of a simulation program during electronic interval $[0, Z]$ is a set of events whose electronic starting times are less than Z, and the scheduling activity which causes these events to occur. We will also call this set the execution's *event set*. For convenience, we also assume that every event in the event set has an electronic completion time less than or equal to Z.

While we have noted that a functional state representation within a running simulation program is a stochastic process in electronic time, we have not precisely defined the stochastic behavior of a functional's state in logical time. We will define this latter stochastic process in terms of the electronic time stochastic process. We note that the electronic time stochastic process is governed in part by the rules constraining when (in logical and electronic time) and if a functional is evaluated. We have not yet specified these *sequencing rules*; we will develop a set of such rules, but note here that sequencing rules less intuitive than our own have been proposed [JeS82]. We let Γ denote an arbitrary set of sequencing rules.

The execution of a simulation program under sequencing rules Γ during the electronic interval $[0, Z]$ consists of an event set, presumed finite. We define a functional state's stochastic process in logical time in terms of its stochastic process in electronic time, and the event set.

Definition 2-4: *State Value at Electronic Time*

Let T_i be an event functional, Γ be a set of sequencing rules, $[0, Z]$ be an interval of electronic time, and $\underline{T_i^e}$ be the T_i state stochastic process in electronic time. The *state of T_i at electronic time t* is the state of the stochastic process $\underline{T_i^e}$ at t, and is denoted by $ES(T_i, t)$.

□

A functional transforms its state whenever it is evaluated; a logical completion time is associated with the completion of each functional evaluation. A functional's state remains constant in any period of electronic time during which the functional is not evaluated. We can then meaningfully define the state of a functional at logical time t:

Definition 2-5: *State Value at Logical Time*

Let T_i be an event functional, Γ be a set of sequencing rules, $[0, Z]$ be an interval of electronic time, and Ω be the event set of an execution of the simulation program using Γ during $[0, Z]$. Let $u_1, \cdots, u_k$ denote the monotonically increasing sequence of electronic completion times of T_i evaluations in Ω; let $v_1, \cdots, v_k$ denote the corresponding sequence of those events' logical completion times. The *state of T_i at logical time t* is given by

$$LS(T_i, t) = ES(T_i, u_G)$$

where G is the greatest integer such that v_G is less than or equal to t. We define the *last state of T_i before t* as

$$LS(T_i, t^-) = ES(T_i, u_{G^-})$$

where G^- is defined as G with strict inequality.

□

The definitions above allow an arbitrary set of sequencing rules Γ. We next develop a particular set of sequencing rules called *proper sequencing rules*. One aspect of these rules governs the scheduling of a functional's evaluation. To say that one evaluation *initiates* another at logical time t is to say that it schedules another evaluation to eventually occur, and that the inputs to the scheduled evaluation are event functional states at logical time t. We assume that at electronic time 0, every generator functionals' first evaluation is scheduled to occur. Thereafter, functional evaluations initiate other functional evaluations in accordance with the sequencing rules. Our development of sequencing rules depends on the concept of a *proper* functional evaluation.

Definition 2-6: *Proper Generator Functional Evaluation*

Let T_g be a generator event functional. A *proper evaluation* of T_g at logical time t started at electronic time l_e consists of the following actions:

- A state v is randomly chosen;
- A logical delay to the next transition, d, is randomly chosen in accordance with the distribution of the random variable $L(T_g)$;
- An execution delay e is chosen; e is randomly distributed in accordance with the distribution of $\chi(T_g)$;
- At electronic time $l_e + e$, T_g's state variable in the simulation program is assigned the state v.

□

Definition 2-7: *Proper Normal Functional Evaluation*

Let T_i be a normal event functional. A *proper evaluation* of T_i initiated at logical time t and started at electronic time l_e consists of the following actions:

- The value $v = T_i(P_i(t), LS(T_i, t^-))$ is calculated, where $P_i(t)$ is a (n_i component) vector of T_i's predecessors' states at logical time t;
- A random logical delay d is chosen in accordance with the distribution of $L(T_i)$;
- A random execution delay e is chosen; e is distributed in accordance with the distribution of $\chi(T_i)$;
- At electronic time $l_e + e$, T_i's state variable in the simulation program is assigned the value v.

□

We note that a proper normal functional evaluation as defined above is an event $<T_i, v, t, t + d, l_e, l_e + e>$.

The definition of a proper normal functional evaluation of T_i initiated at logical time t specifies that the state values used as input for T_i's evaluation are the state values of T_i's predecessors at logical time t. Our intuition suggests that such an evaluation must be initiated at logical time t if and only if one of T_i's predecessors completes an evaluation at logical time t which changes that predecessor's state. We define evaluations which change their functionals' states to be *significant* evaluations.

Definition 2-8: *Significant Evaluation*

Let T_g be a generator event functional. An evaluation of T_i at logical time t_t is said to be *significant* if $LS(T_i,t) \neq LS(T_i, t^-)$. If T_i is a normal event functional, an evaluation of T_i initiated at logical time t is said to be significant if $LS(T_i, t + d) \neq LS(T_i, (t + d)^-)$, d being the selected logical delay.

□

The definition of a proper normal functional evaluation exposes a particular assumption used by our model: if an evaluation of T_i is initiated at logical time t, the result of that evaluation is a function only of states at logical time t. Any functional evaluation (T_i's or otherwise) completed later in logical time cannot affect T_i's ultimate evaluation result. We could generalize our approach by defining a functional's state *history* to be a function of itself and its input *history*; Zeiglers [Zei84] DEVS formalism uses such an approach. Such a definition is not analytically tractable. Our constrained assumption was chosen to support the analysis of this simulation model.

Our definition of proper sequencing rules follows from the definitions of proper functional evaluations. There are two aspects to these rules. First, the rules specify conditions which cause a functional evaluation to occur. The proper evaluation of a generator functional selects a logical delay until the next evaluation of that functional. We say then that the evaluation of a generator functional initiates the next evaluation of that functional. It is also intuitively clear that the completion of a significant evaluation of functional T_i at logical time t should initiate evaluations of all of T_i's successors at logical time t. A second aspect of proper sequencing rules specifies precedence in electronic time among evaluations. The proper evaluation of normal functional T_i initiated at logical time t requires T_i's own state value at logical time t, and the state values of T_i's predecessors at logical time t. We must therefore constrain the proper evaluation of T_i from occurring in electronic time before the electronic completion times of evaluations producing the inputs used in this evaluation of T_i. These observations lead us to define *proper sequencing rules.*

Definition 2-9: *Proper Sequencing Rules*

Let $<Y,T,I_S,M,R,L,\chi>$ be a simulation system, suppose the simulation program is executed on some computer system during an electronic time interval $[0,Z]$, and let Ω be the event set of this execution. The execution is said to have followed *proper sequencing rules* if and only if

(i) A significant evaluation of T_i completing at logical time t initiates proper evaluations of all successors of T_i at logical time t;

(ii) If T_g is a generator event functional, a proper evaluation of T_g at logical time t initiates a proper evaluation of T_g at logical time $t + d$, where d is the logical delay selected by the former proper evaluation of T_g;

(iii) If $<T_i, v, t, t + d, l, u>$ and $<T_i, v_t, t_t, t_t + d_t, l_t, u_t>$ are events in Ω generated by a normal functional T_i and $t + d$ is the largest logical completion time of an event in Ω generated by T_i such that $t + d < t_t$, then $u \leqslant l_t$;

(iv) If T_i is a predecessor of T_j, and if $<T_i, v, t, t + d, l, u>$ and $<T_j, v_t, t_t, t_t + d_t, l_t, u_t>$ are events in Ω where $t + d$ is the largest logical completion time of an event in Ω generated by T_i such that $t + d \leqslant t_t$, then $u \leqslant l_t$.

□

The first proper sequencing rule states that a change in one functional's state requires the re-evaluation of its successors' states. The second rule states that the evaluation of a generator functional initiates another evaluation of the functional later in logical time. The third rule ensures that if a functional is evaluated (at least) twice, then the (later) functional evaluation can use the proper previous state of the functional. The final proper sequencing rule ensures that the proper state of T_i can be used in an evaluation of T_j. Next we show that these rules are sufficient for the correct execution of the simulation.

2.4. Simulation Execution Correctness

This section considers the correctness of a simulation and proves that adherence to proper sequencing rules ensures simulation correctness.

To say that a simulation *model* is correct is to say that it correctly captures the behavior of its modeled object. Establishing the correctness of a simulation model is a difficult problem which is often ignored [HoP84] . We will always assume that the event functionals correctly capture the behavior of their modeled objects. Now suppose we could execute the simulation program on an "ideal" (probably multi-processor) computer system where there is no distinction between electronic and logical time; we expect the ideal computer system to present the "correct" inputs to each event functional, and so arrive at the "correct" result. We then **define** the correctness of an arbitrary simulation execution (of a correct model) to be the equivalence between its functionals' state values

in logical time with the ideal system's functionals' state values in electronic time.

Definition 2-10: *Simulation Correctness*

Let $<Y,T,I_S,M,R,L,\chi>$ be a simulation system, suppose the simulation program is executed on some computer system during an electronic time interval $[0,Z]$, and let Ω be the event set of this execution. Let B denote the largest logical completion time of any event in Ω, and consider the execution of the same simulation program on an ideal system during the electronic interval $[0,B]$. Let $\hat{T}_i$ represent the ideal system's version of event functional T_i. We say that T_i's state is *correct* at logical time t if

$$LS(T_i, t) = ES(\hat{T}_i, t).$$

We say that a simulation is correct if the state of every event functional is correct at every logical time $t \in [0, B]$.

□

We will show that enforcement of proper sequencing rules ensures correctness in the sense of definition 2-10. Our correctness proof uses a graph describing the simulation's behavior during an electronic interval. Suppose a simulation system is executed on some computer (possibly multi-processor) system during the electronic time interval $[0,Z]$, and we suppose that this simulation execution enforces proper sequencing rules. Ω is the event set of this execution; thus the events in Ω satisfy the restrictions defined by the proper sequencing rules. We create a graph from the events in Ω to illustrate the simulation execution and the precedence imposed by proper sequencing rules.

Definition 2-11: *Execution Graph*

Let $<Y,T,I_S,M,R,L,\chi>$ be a simulation system, suppose the simulation program is executed on some computer system during an electronic time interval $[0,Z]$, and let Ω be the event set of this execution. We define an *execution graph* for this execution as follows.

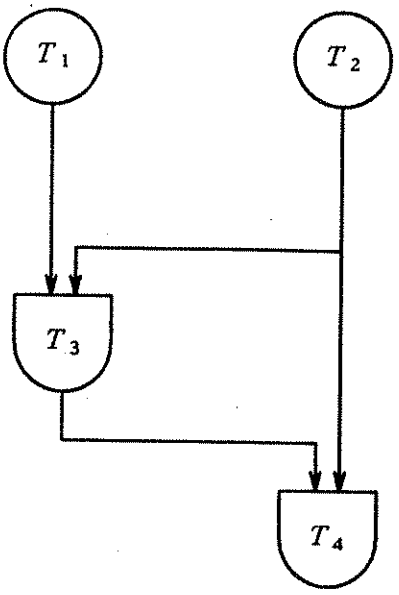
- Every event in Ω is a node in the execution graph;
- Let E be an event generated by a normal functional T_i with logical initiation time s, and let $T_1,\cdots,T_k$ denote T_i's predecessors. For every $1 \leqslant j \leqslant k$, let E_j denote the event (if it exists) in Ω generated by T_j with the largest logical completion time less than or equal to s. We define an arc from each E_j to E.
- Let E be an event generated by a normal functional T_i, and let E_p be the event generated by T_i with the greatest logical completion time less than E's logical initiation time. If E_p exists, then an arc is drawn from E_p to E.

□

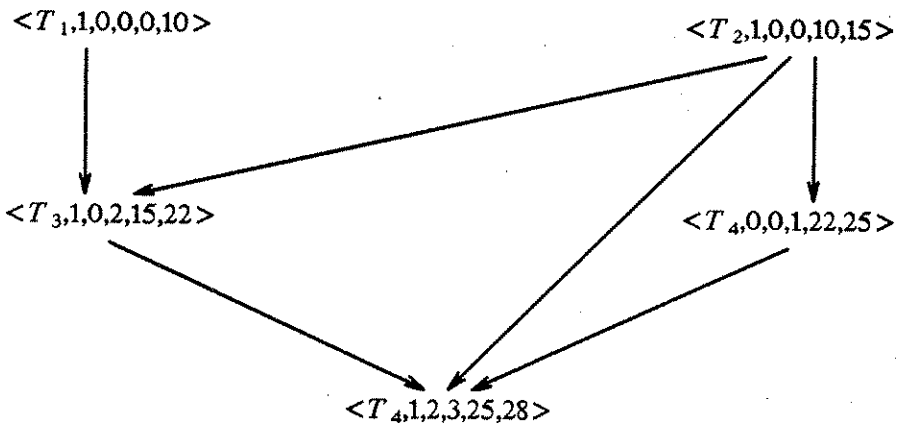
The construction of an execution graph is intimately related to proper sequencing rules. The events which appear in Ω occur as a result of the event initiation aspect of the proper sequencing rules: the execution graph's nodes are events in Ω. Each arc in the execution graph expresses a precedence required by the proper sequencing rules; there is a one-to-one correspondence between an execution graph arc and a proper sequencing rule constraining the execution starting time of the event receiving the arc. Given an event set Ω of a simulation execution which used proper sequencing rules, we can express the precedence among completed evaluations imposed by proper sequencing rules by an execution graph.

Figure 2-1 illustrates a simple system graph suggestive of a logic network, and a possible execution graph. The electronic interval described by this graph is [0,28]. Recall that the formal definition of an event (as represented in this execution graph) is a vector whose first component identifies the functional, the second component is the state resulting from the evaluation, the next two components identify the logical initiation and completion times, the last two components identify the electronic starting and completion times.

In figure 2-1, functionals T_1 and T_2 are generators creating binary values; T_3 and T_4 are binary functionals. T_3 performs the logical **and** function, and the value of T_4 is 1 if and only if its inputs are identical. The initial states of T_1 and T_2 are 0, causing T_3's initial state to be 0, and T_4's initial state to be 1. Figure 2-1 illustrates significant evaluations of T_1 and T_2 at logical time 0, so that evaluations of both T_3 and T_4 are initiated at logical time 0. We presume that T_3 takes a logical delay of 2 units, and T_4 takes a logical delay of 1 unit. At logical time 1, T_4's state then changes to 0, since the state of T_3 at logical time 0 is 0 while the state of T_2 at logical time 0 is 1. T_3's state changes to a 1 at logical time 2, initiating another evaluation of T_4 at logical time 2. This last evaluation of T_4



System Graph



Execution Graph

Figure 2–1

changes its state back to 1. The execution starting and completion time numbers shown in figure 2–1 reflect a sequential simulation of this system, where T_1 is first evaluated, then T_2, T_3, the first evaluation of T_4 and then the second evaluation of T_4.

Theorem 2–1 below inducts on a topological sorting of the execution graph's nodes. To prove the existence of such a sorting, we demonstrate that an execution graph is acyclic.

Lemma 2-1: An execution graph is acyclic.

Proof: If E_i and E_j are events such that E_i sends an arc to E_j, then E_i's logical completion time must be less than or equal to E_j's logical initiation time. Since logical delays are strictly bounded from below by ϵ_B, any arc out of E_j must be directed to an event whose logical initiation time is strictly greater than E_i's logical completion time. Hence no cycles can exist.

□

We have finally developed the terminology to demonstrate the sufficiency of the proper sequencing rules for producing correct results.

Theorem 2-1: Let $<Y,T,I_S,M,R,L,\chi>$ be a simulation system, suppose the simulation program is executed in the physical interval $[0, Z]$, and let Ω be the event set of the execution. If the simulation execution follows proper sequencing rules, then the simulation is correct.

Proof: We have argued that any execution of a simulation using proper sequencing rules can be expressed as an execution graph. We prove the theorem by inducting on a topological sorting of the execution graph's nodes to show that for every event in Ω generated by T_i, $LS(T_i,t) = ES(\hat{T}_i,t)$ (definition 2–10) where t is the event's logical completion time. The induction base is easily satisfied, as any event without predecessors in the execution graph is generated by a generator event functional whose state stochastically behaves in the desired fashion by assumption.

Let E be an event generated by event functional T_E, suppose that E's logical initiation time is s, and its logical completion time is t. Let $E_1,\cdots,E_k$ be E's predecessors in the execution graph, let $t_1,\cdots,t_k$ be their respective logical completion times, and let $T_1,\cdots,T_k$ be their respective associated event functionals. For the induction hypothesis, suppose that $LS(T_i,t_i) = ES(\hat{T}_i,t_i)$ for $i = 1,\cdots,k$. We must show that $LS(T_E,t) = ES(\hat{T}_E,t)$. That is, we inductively show that functionals' states are correct at their evaluations' logical completion times. Again, if E is generated by a generator functional, the conclusion is trivially satisfied. If E is generated by a normal event functional then (by definition of the execution graph arcs) for each i, t_i is the largest logical completion time less than or equal to s of an event in Ω generated by T_i. The induction hypothesis assumes that each T_i's logical state is correct at E_i's logical completion time. Since T_i's state can change only at logical completion times, then T_i's state does not

change in the logical time interval $[t_i, s]$ so that $LS(T_i,s) = ES(\hat{T}_i,s)$ for each $1 \leqslant i \leqslant k$. The inputs presented to the functional generating E at logical time s are consequently the same values presented to the ideal system's version of the functional at electronic time s. The conclusion follows immediately, as we assume that the two functionals produce the same result given the same input. This completes the induction.

Having established that the states of functionals are correct at their logical times of transition, it follows immediately from definition 2–10 that states of functionals are correct for all logical $t \in [0,B]$.

□

Theorem 2-1 tells us that the enforcement of proper sequencing rules leads to correct simulation results. This theorem applies to all simulations (modeled as proposed), not just sequential simulations or just distributed simulations. Recalling the motivation for proper sequencing rules, we could argue that their enforcement is *necessary* for correct simulation; to prove this rigorously we would have to explicitly assume that a functional's evaluation cannot anticipate its input values before they are computed. This is a reasonable assumption; we see then that correct simulation results depend centrally on the enforcement of proper sequencing rules. This recognition leads us to an understanding of synchronization requirements in a distributed simulation. Enforcing proper sequencing rules during the execution of a **sequential** simulation is simply a matter of making sure the evaluations are sequentially ordered properly; enforcing proper sequencing rules during the execution of a **distributed** simulation may require a processor to refrain from execution and wait for the completion of evaluations elsewhere.

2.5. Chapter Summary

We have developed a model of simulations. This model provides a base for further analysis of partitioning. Using our model, we showed that enforcement of *proper sequencing rules* is sufficient to ensure the correctness of the the simulation. These rules also lead us to an understanding of a distributed simulation's synchronization requirements.

Chapter 3

Run-Time Behavior

3.1. Chapter Overview

Before we can attempt to partition a simulation we need to understand its run–time behavior. Any model of a simulation's behavior must somehow capture the effect of synchronization, and must deal with the dynamic variation in the simulation workload.

This chapter examines a simulation's run–time behavior. Our treatment begins with a description of some simplifying assumptions. These assumptions allow us to restrict our attention to a single "cycle" of the simulation, and to model the running simulation with a Markov chain. We then illustrate how the run–time behavior may vary from cycle to cycle. These observations place a system's performance analysis in a probabilistic context: the analysis is centered on the probability distribution of a single cycle's execution delay. We describe means of estimating the cycle execution time under any partition, given that the cycle behavior is completely known.

The analysis presented in this chapter is quite important to our understanding of the partitioning problem. It also lays the foundation for the statistical analysis described in Chapter 6. It shows conclusively that partitioning a simulation is quite a different problem than partitioning statically defined task structures. We deal with this added complexity by modeling the running simulation as a Markov chain. The modeling of systems with Markov chains is standard practice; however, our presentation is different in its focus. Markov (or semi–Markov) chain models of systems define their transitions at times when a component changes state [IgS83] , e.g., when a significant evaluation occurs. Our Markov chain embeds many such changes in its state, focusing instead on times at which the system "starts over". This difference is critical to our analysis of a simulation's run–time behavior.

3.2. A Simpler Model

In Chapter 2 we presented a general model of simulations, and showed that the enforcement of proper sequencing rules leads to a correct simulation execution. However, the generality of this model excludes any further quantitative analysis. We thus make some simplifying assumptions which restrict our attention to somewhat less general simulations. We first describe assumptions removing some non–determinism from definition 2–3. We next assume non–interference properties, and then examine assumptions causing the simulation to behave cyclically. Finally, we present assumptions giving a simulation system nice stochastic properties.

3.2.1. Deterministic Delays

Consider a normal event functional evaluation initiated at logical time s, and completed at logical time t. Definition 2–3 allows the logical delay $t - s$ to be a random variable. We suppose instead that every evaluation of an normal event functional T_i introduces a constant logical delay, $L(T_i)$. This assumption is critical to the development in Chapter 4, and is satisfied by many systems we might simulate.

Definition 2–3 also allows a general execution delay for every event. We suppose that every evaluation of T_i requires a fixed execution delay $X(T_i)$. The removal of variation in the delays allows us to concentrate instead on the variation due to the rules initiating functional evaluation.

3.2.2. Non–Interference

If T_i is initiated at logical times s_1 and s_2 where $s_1 < s_2 < s_1 + L(T_i)$, we assume that the two different initiations do not interfere with each other: the evaluation result at $s_j + L(T_i)$ depends solely on state values at time s_j, $j = 1,2$. This assumption assures us that the functional's post–initiation behavior is quite predictable: it waits for all its inputs to become available, and then re–evaluates its state. We do not see how to relax this assumption and still do any sort of static analysis of the simulation.

3.2.3. Cyclic Behavior

We are interested in simulation systems where the logical times between generator functional evaluations follow a repeating pattern. Such systems may exhibit cyclic behavior, allowing us to focus our analysis on the behavior of the simulation during a single, representative cycle.

A cyclic simulation is illustrated by the example of a *CPU* circuit with one major and several minor clocks. Figure 3–1 depicts the time pattern created by the rising edges of these clocks. Some instruction is presented to the *CPU* circuit every major clock cycle; the minor clock pulses control the instruction's interpretation. The simulation activity required depends on the state of the circuit before a major clock, and the instruction presented for interpretation.

The repeating nature of generator functional evaluations is captured in the generators' logical inter–evaluation times. We let t_1 denote the first logical time at which any generator is evaluated; we identify the index of that generator by i_1. The second logical time (possibly equal to the first) at which any generator is evaluated is denoted t_2, and that generator's index is identified as i_2. This denotation leads to a sequence of pairs

$$(t_1, i_1), \cdots, (t_j, i_j), \cdots .$$

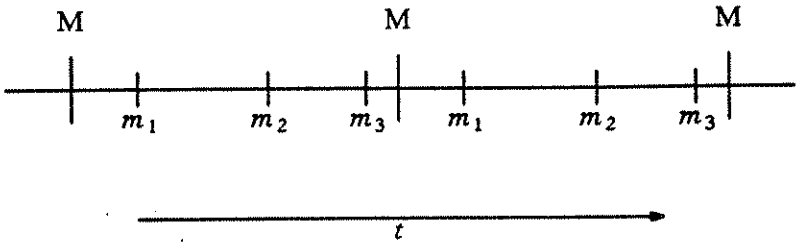
If two or more generator functionals are evaluated at the same time, we assume that they are ordered in this sequence by their indices. A given generator functional may appear in this sequence any number of times. From this sequence we construct another, which identifies the system–wide generator inter–evaluation times. Let $\Delta_i = t_{i+1} - t_i$; we construct the sequence

$$(\Delta_1, i_1), \cdots, (\Delta_j, i_j), \cdots .$$

We assume that that this latter sequence is cyclic: there exists a finite (minimal) integer Y such that every Y elements the sequence repeats itself. We let Y_i denote the sequence's *ith* repetition's logical completion time. Clearly,

$$Y_i \doteq i \cdot \sum_{j=1}^{Y} \Delta_j .$$

We define one *cycle* of the simulation to be all of the simulation activity generated as a consequence of generator evaluations at logical times t, $Y_i \leqslant t < Y_{i+1}$. We assume that each normal event functional is evaluated a bounded number of times during a cycle, and that all of the simulation work belonging to one cycle must be completed before the next cycle is begun.



M: Major Clock m_i*: Minor Clock*

CPU Clocks Cycles

Figure 3–1

3.2.4. Stochastic Assumptions

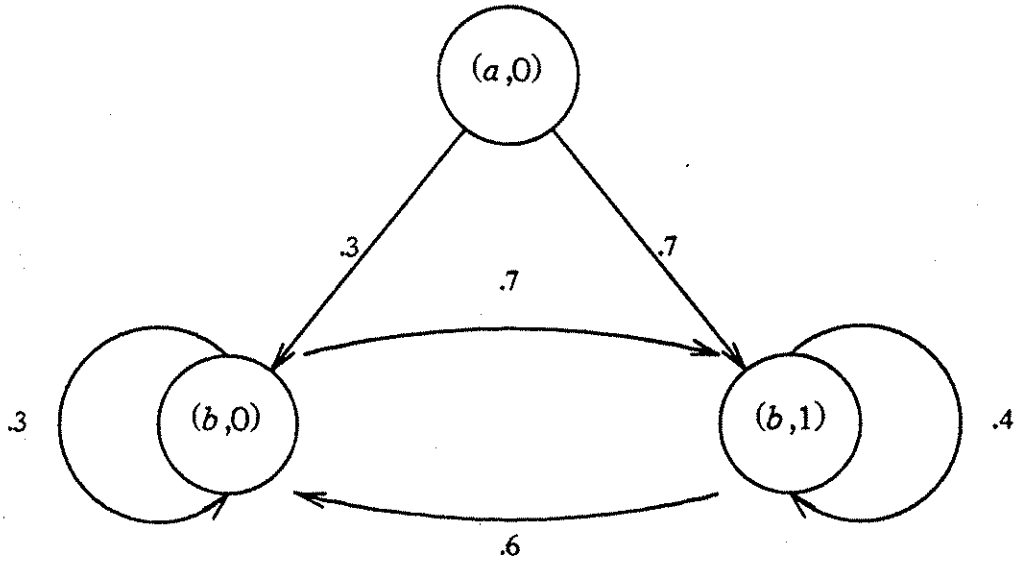
A cyclic simulation's nature is not enough to ensure the sufficiency of a single cycle analysis. We must also consider restrictions on the behavior of the generators' state transitions. We will demonstrate that if the generator transitions can be described as a Markov chain, then the simulation system as a whole will be a Markov chain. We then observe that the system chain is eventually ergodic: the cycles of an ergodic system chain in equilibrium are probabilistically identical.

Every cycle, exactly Y generator evaluations occur. Whenever a generator is evaluated, a new (possibly the same) state is chosen for the generator. Let O^k denote the vector $(v_1^k, v_2^k, \cdots, v_Y^k)$ where state v_j^k is the state assumed by generator T_{i_j} at time $t_{(k-1)\cdot Y+j}$ during the *kth* cycle. O^k is a random vector, as the states v_j^k are chosen randomly. Following standard nomenclature, we denote this stochastic process by $\{O^n\}$. We assume that the set Υ of symbols is finite; O^k is thus drawn from a finite set of vectors. We also assume that the sequence $O^1, O^2, \cdots$ is an ergodic discrete time Markov chain [Ros83], so that every state O^k is eventually visited again. This assumption is much weaker than the assumption of independence among generator state transitions. Within a cycle, dependencies may exist between the random variables describing the generator state transitions; these random variables may also depend on the values assumed in the previous cycle. The Markov chain assumption is strong enough though to yield some desirable characteristics.

The Markovian nature of the inputs is preserved by the normal event functionals. Let N^k be the vector defined by the collection of normal functional state values at the beginning of the *kth* cycle. We define the *system cycle* process $\{\Theta^n\}$ whose state space is the cross product of all possible vectors N^k with all possible vectors O^k. A change in a normal functional's state is completely determined by its current state and the states of its predecessors. Consequently for a given N^k, N^{k+1} is completely determined by the vector O^k observed during the *kth* cycle. Now consider any possible state vector $N^k O^k$ for the process $\{\Theta^n\}$. The transition probabilities associated with such a vector are precisely the transition probabilities associated with its input subvector O^k: the N^{k+1} portion of $N^{k+1}O^{k+1}$ is determined by N^k and O^k, while O^{k+1} may vary according to the structure of $\{O^n\}$. Since the process $\{O^n\}$ is a Markov chain, it follows immediately that the process $\{\Theta^n\}$ is a Markov chain. This is extremely useful, since a vector $\Theta^k = N^k O^k$ completely encodes the behavior of the simulation system during the *kth* cycle.

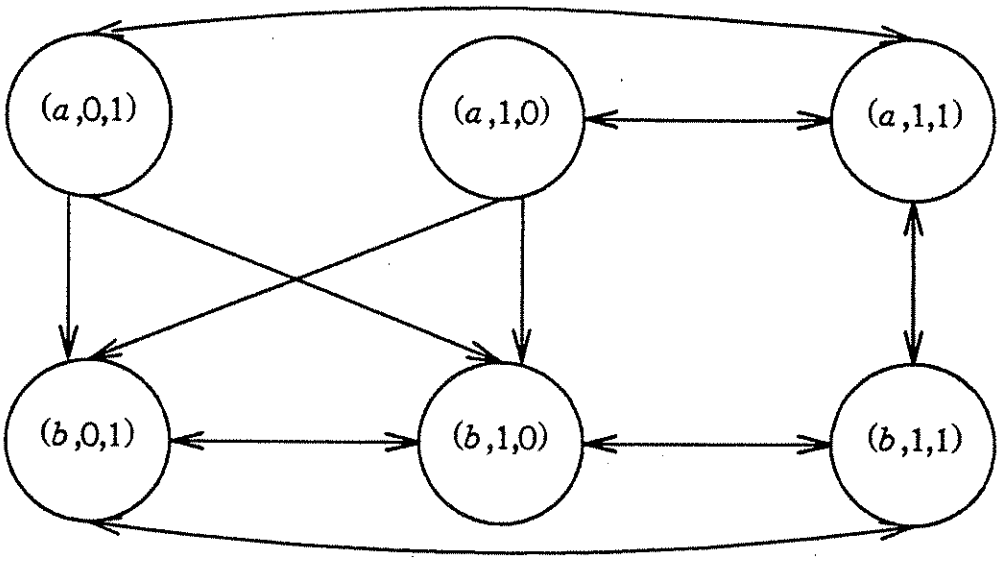
An ergodic Markov chain in equilibrium has the attractive property that the chain's state probability distribution does not change from time step to time step. This property is quite important to some of our later analysis. If we can show that $\{\Theta^n\}$ is ergodic, then we know that in equilibrium every cycle is probabilistically identical to another. However, while $\{\Theta^n\}$ inherits the Markovian nature of $\{O^n\}$ it need not inherit the *ergodicity*. This is easily shown by example. Figure 3–2 illustrates a Markov transition diagram for a single functional system with one binary input. The functional's state is either a or b, the input is either 0 or 1. Letting p_{ij} be the probability that the input chain passes from state i to j, we have $p_{00} = .3$, $p_{01} = .7$, $p_{10} = .6$, $p_{11} = .4$. The system's transitions are represented by arcs weighted by their transition probabilities. This example shows that the functional with an initial value of a will never again achieve the value a; the process $\{\Theta^n\}$ is not ergodic.

The example above might suggest that ergodicity could be preserved if $\{\Theta^n\}$'s transition diagram always provides a path between any two states. Figure 3–3 contradicts this hypothesis, depicting the transition diagram for another single functional process, with two binary inputs. Again the functional's state is either a or b, and each input is either 1 or 0. A cycle consists of a transition by each input, and we suppose that the pattern of input values is fixed, alternating between (1,0) and (0,1). The functional's state transition



Non Ergodic Markov Chain

Figure 3–2



Apparently Ergodic Transition Diagram

Figure 3-3

diagram allows for the input vector (1,1), which we have excluded from the input process. This diagram *apparently* describes an ergodic Markov chain, since a path exists between any two nodes. However, the functional with an initial value of a will never again attain the value of a as the first input vector causes a transition to state b. The only way for the functional to reattain state a is for the input system to present two successive (1,1) input vectors, which is impossible. Even though the functional's state transition diagram appears ergodic, the overall system is not ergodic.

This last example shows that a demonstration of ergodicity must consider the *joint* behavior of the system's inputs. This sort of calculation is theoretically possible, but computationally intractable. In the general case we have to consider each of $|Y|^Y$ input vectors. We cannot generally expect to be able to prove $\{\Theta^n\}$ is ergodic.

Even if $\{\Theta^n\}$ is not an ergodic chain, it will exhibit ergodic tendencies in finite time. Every state in a finite Markov chain is either "transient" or "recurrent". Transient states are visited a finite number of times with probability 1. The set of recurrent states can be partitioned into a number of classes, or subchains. A chain whose state is in a recurrent class will never leave that class. Thus, in finite time, $\{\Theta^n\}$ will enter an ergodic subchain, and for all practical purposes be ergodic.

It is convenient for us to assume that the Markov chain is fundamentally ergodic. The examples illustrating the difficulties in *proving* ergodicity are somewhat contrived. They depend on a bad choice of an initial state which can never again be achieved. The type of simulation motivating our research is the class of functional logic network simulations. These networks consist largely of "memoryless" gates whose states are entirely determined by their inputs. A simulation whose normal event functional transition functions F_i do not depend on the functionals past state will be ergodic if the process $\{O^n\}$ is ergodic. Consider any one of $\{\Theta^n\}$'s achievable states NO, and let O_N be an input vector such that application of O_N places process $\{N^n\}$ in state N, and $\{O^n\}$'s transition probability p_t from O_N to O is non-zero (the existence of O_N is guareenteed by the fact that NO is an achievable state of process $\{\Theta^n\}$, and the memoryless nature of the normal event functionals). Since $\{O^n\}$ is ergodic, the mean number of steps between visits to state O_N is finite, say μ. Every time $\{O^n\}$ returns to O_N, it passes next to state O with probability p_t, $p_t \neq 0$. The number of times it returns to O_N before next passing to O is a geometric random variable with mean $\frac{1}{p_t}$. It follows directly that the mean number of steps between visits to state NO in process $\{\Theta^n\}$ is no greater than $\frac{\mu}{p_t}$, which is finite. This is equivalent to saying that $\{\Theta^n\}$ is ergodic. We will assume then that in general $\{\Theta^n\}$ is an ergodic Markov chain, in equilibrium.

3.3. General Cycle Behavior

This section considers a simulation cycle's run-time behavior. We first observe that one cycle's simulation work may differ from another cycle's simulation work. We show how to augment the execution graph from Chapter 2 in order to calculate a lower bound on the cycle execution time; we also sketch a method for deriving better cycle execution time approximations. Finally, we argue that any performance analysis of a distributed simulation must treat that performance in a probabilistic framework.

3.3.1. Variant Cycle Behavior

The proper sequencing rules described in Chapter 2 specify that a normal event functional evaluation must be initiated at logical time t if any one of its predecessors changes state at logical time t. Adherence to these rules causes the amount of simulation work performed in a cycle to vary from cycle to cycle. As a simple example, consider the

simulation of a single logic gate. If either of its inputs changes from one cycle to another, the gate's new state must be evaluated. Identical inputs may be presented during two successive cycles; the gate is not simulated during the second cycle. Even in the simplest of simulations, the simulation workload can vary from cycle to cycle.

3.3.2. The Execution Graph Revisited

The execution graph's topology is defined without explicit reference to the simulation's implementation. The arc definitions reflect precedence relations necessary for correct simulation; these arcs are defined independently of a simulation's partitioning for parallel execution. We can modify the execution graph to reflect the assignment of the evaluated functionals to a distributed system. If node T in the execution graph has predecessors $P_1, \cdots, P_k$, each P_i evaluation must be completed before T is simulated. The assignment of events to processors leads to further precedence; within a processor, functional evaluations are customarily ordered in their logical completion times. We further suppose that co-resident events with identical logical completion times are ordered in some fixed way. Within each processor we can impose a total precedence ordering on the event evaluations. To reflect the assignment of events to processors, we augment the execution graph with these additional *execution precedence arcs*. We refer to the resulting graph as the *cycle execution graph*.

Figure 3–4 gives an example of a cycle execution graph. This graph is derived from the execution graph of figure 2–1. Figure 3–4 assumes that functionals T_1 and T_3 are assigned to the same processor, and that functionals T_2 and T_4 are assigned to a second processor. The physical initiation and completion times have been omitted from figure 3–4's event representations.

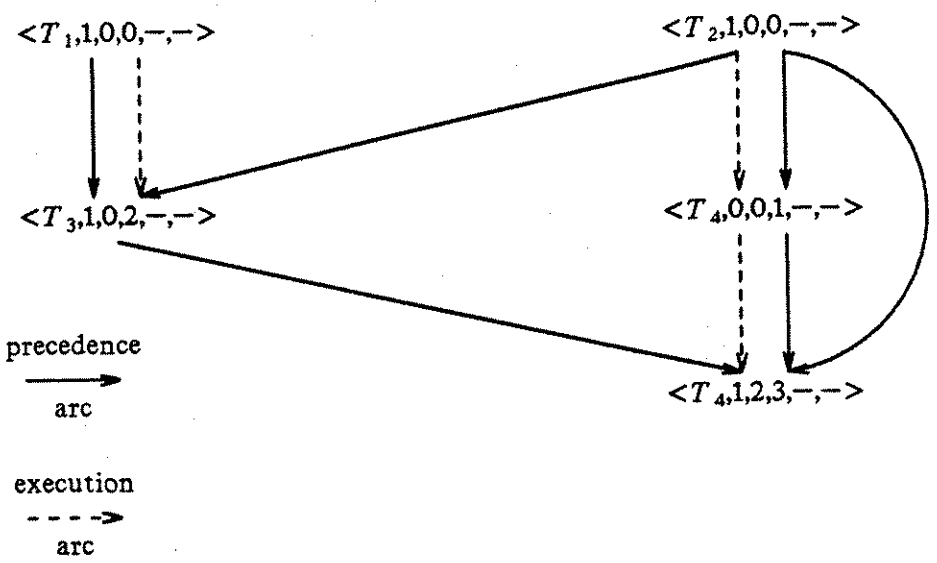
The proof of a lemma in the next subsection assumes a topological sorting of the cycle execution graph's nodes. To support the validity of this assumption, we remark that the cycle execution graph is acyclic. A formal proof of this is postponed until Chapter 4, where we show that a larger graph is acyclic; the cycle execution graph is a subgraph of this larger graph, and so is itself acyclic.

3.3.3. Cycle Execution Time Estimation

We would like to determine a cycle's execution time from its cycle execution graph. In this section we show how a lower bound on the execution time can be inferred from this graph.

Chapter 2 demonstrates that proper sequencing rules create precedence relations between functional evaluations; we will call these *logical* dependencies. The execution graph illustrates logical dependencies between evaluations with arcs; a cycle execution graph illustrates these same dependencies, as well as *execution* dependencies created by partitioning. Distributed simulations enforce these precedence relations with synchronization protocols. Most of these protocols' run-time behavior is quite complex, and cannot be conveniently modeled. Recognizing that the precedence arcs in a cycle execution graph expose the essential synchronization requirements, we will model synchronization in a distributed simulation with a protocol inspired by the cycle execution graph, *oracle synchronization*.

Under oracle synchronization, a functional completing an evaluation reports its new state to each of its successors. This message includes a bit saying whether the evaluation was significant (meaning it changed the functional's state, definition 2–8). The completed evaluation also sends a message to its execution successor, releasing the processor. A functional receiving notice of a significant evaluation must be evaluated, but must wait until its other logical predecessors have achieved the correct states. It must furthermore wait for its execution predecessor to release the processor. The functional retains a history of



Cycle Execution Graph

Figure 3–4

messages sent to it by each predecessor. We assume that an oracle exists which discerns for an initiated functional whether each message required for the evaluation has been received. The functional's evaluation will not occur until all appropriate predecessor messages have arrived. We assume that the oracle introduces no further execution delay.

The oracle synchronization protocol's activity during a given cycle is directly related to that cycle's execution graph. For every node the oracle discerns when the precedence relations described by the graph have been satisfied. It is possible then to analyze the cycle execution graph to determine the cycle execution time under the oracle synchronization method. We augment the graph with weights on the nodes and weights on the arcs. Every node is weighted with its execution delay; the initiation node is given a delay of zero. Every execution arc is given a weight of zero, and the arcs out of the initiation node are weighted by zero. Every other arc is given a weight of C, the communication delay. The length of a path through this augmented graph is the sum of node and arc weights on the path. Then

Lemma 3-1: Let c be a cycle, and let *WCG* be the weighted cycle execution graph for c, as defined above. Then the longest path through *WCG* is equal to the cycle execution time under oracle synchronization, and is a lower bound on the true cycle execution time.

Proof: Let W be a *WCG* graph node. We could induct on a topological sorting of *WCG* nodes to show that the length of the longest path to W defines W's completion time under the oracle synchronization, and is a lower bound on W's true execution completion time. This induction argument follows easily from the observations that the cycle execution graph's arcs define precedence relations. The claim that the length of the longest path to W defines W's completion time is established since W does not execute until each predecessor has completed and the graph's weights are conveniently defined. The claim that this completion is a lower bound follows from the reasonable assumption that it is not possible to know a priori the result of a predecessor's functional evaluation.

□

The cycle execution graph is useful for roughly estimating the cycle execution time, but we may desire better accuracy. We show in Chapter 4 that the oracle synchronization protocol ignores some additional synchronization needs. Furthermore, the cycle execution graph ignores some potentially important details. For example, a real synchronization protocol requires additional messages and may introduce additional computational delay; nor is event list processing overhead considered in the execution graph. It would be computationally convenient if we could further augment the cycle execution graph to reflect this overhead, but such an approach does not seem feasible. A processor's events list processing overhead can depend on queue size, and is thus time dependent. This sort of load dependent delay is not conveniently expressed as a weight on a graph arc. The most direct and accurate way of calculating the cycle's execution delay is to simulate the execution of that cycle. The synchronization and events list overhead can then be introduced as required. Given the full knowledge of the cycle's behavior, the simulation contains no random variation, and will thus give a determinate answer.

3.4. Chapter Summary

This chapter identifies simplifying assumptions under which the running behavior of a simulation can be described as a sequence of cycles whose behavior is Markovian. This allows us to restrict our analysis of the system to a single, representative cycle. We show that even in the simplest of systems, a simulation's computational workload can vary from cycle to cycle. A cycle's execution time is thus a random variable; furthermore, this random variable is a function of our choice of partition. A natural way to judge a partition is to calculate or estimate its corresponding **mean** cycle execution time. Towards this end, we have also shown how, given a description of a particular cycle's behavior, we can bound from below the cycle execution time in a distributed system.

Chapter 4

Synchronization and Cycle Execution Time

4.1. Chapter Overview

In Chapter 3 we examined the run-time behavior of a simulation, identifying important "cycles" whose behavior is randomly distributed. These cycles are intrinsic to the simulation structure, and are independent of the way the simulation is executed. In this chapter we address a problem motivated by a physically distributed simulation: how can we quantitatively model the distributed simulation's run-time behavior? We need a model to evaluate the effectiveness of a partition. The Markov chain model is too large to numerically analyze, and it does not explicitly consider synchronization. Furthermore, except for [Nic84], no one has proposed a distributed simulation run-time model; the model in [Nic84] is not suited for detailed quantitative analysis. This chapter shows how synchronization requirements can be graphically represented, and how the cycle execution time can be derived from the graph. Section 4.2 develops the *work graph*, which describes all possible simulation work during a cycle. Section 4.3 defines a *partition* of the work graph; it also shows how the work graph is transformed as a function of a partition into a *work assignment graph* which expresses necessary precedence on functional evaluations. The work assignment graph leads us to model synchronization with an ideal synchronization protocol, the *hyper-message* synchronization protocol. Section 4.4 shows that a cycle's execution time under hyper-message synchronization is optimal over the class of non-predictive synchronization protocols. Section 4.4 also shows that hyper-message synchronization is a better synchronization model than oracle synchronization (Chapter 3), and shows how to efficiently calculate an observed cycle's execution time under hyper-message synchronization.

4.2. The Work Graph

The Markov chain model of a simulation is useful in demonstrating that simulation cycles are probabilistically identical. The state space of this model while finite, is enormous, and not amenable to numerical treatment. This section shows how to represent the simulation by a substantially smaller structure, the work graph. Unlike the Markov chain model, the work graph focuses on the logical times at which simulation work might occur during a cycle, and does not explicitly consider the functionals' state values. The exclusion of state information significantly reduces the size of the representing structure.

The work graph expresses all of the work that **might** occur during a cycle. Its nodes are pairs: a logical time, and an event functional. The pair (t, T_j) is a *work node* if it is possible that T_j can have an evaluation initiated exactly t logical time units after the beginning of a cycle. We refer to T_j's state symbol at logical time $t + L(T_j)$ as (t, T_j)'s *evaluation result*. We note that the evaluation result is defined even if T_i is not initiated for evaluation t time units into the cycle. We will often refer to the evaluation or execution of a work node $W = (t, T_i)$. This is intuitively understood as an actual evaluation of the functional T_i, initiated at logical time t.

A directed edge is defined from (s, T_i) to (t, T_j) if T_j is a successor of T_i, $t = s + L(T_i)$, and $T_i \neq T_j$. The base node of the edge thus identifies a potential

evaluation of T_i which (by changing T_i's state) can cause an evaluation of T_j. (s, T_i) is said to be an *initiation predecessor* of (t, T_j) and (t, T_j) is said to be a *initiation successor* of (s, T_i).

A work graph is constructed from the system graph and a specification of the cycle's generator transition times. We show that every system graph path from a generator to T_j defines a potential evaluation of T_j. Suppose a generator functional is evaluated at logical time s, and changes state. We may suppose that normal functional T_1 is a successor of the generator, and so potentially changes its own state. This in turn may cause another functional T_2 to be evaluated, and so on. In general, if $T_1, T_2, \cdots, T_{j-1}$ is a sequence of normal functionals such that T_1 is a generator's successor, T_i is a successor of T_{i-1} for $i = 2, \cdots, j$, and the generator changes state at logical time s (into the cycle), then T_j might be evaluated exactly $s + \sum_{i=1}^{j-1} L(T_i)$ logical time units into the cycle. Furthermore, T_j can be evaluated only if there is such a path through the system graph. Creating the work graph is equivalent to finding all paths to each functional in the system graph.

We must take additional care with cyclic system graphs. We assume that for every functional T_j, the user supplies an upper bound $B(T_j)$ on T_j's evaluation initiation times. If the system graph is acyclic, we do not require these bounds, and so define $B(T_j) = \infty$ for each T_j.

Our work graph creation algorithm considers the cascading effect of a single generator transition. This procedure is then applied to each generator in turn. The effect of a generators' transition on normal event functionals can be directly found using a breadth-first traversal of the system graph [SaH76] . This traversal uses a list of pairs (t, T_i) called the *search* list; initially the search list contains only $(0, T_g)$ where T_g is a generator. As the algorithm iterates, it removes the top search list node (t, T_i) and inserts the time t into a sorted list of T_i's potential initiation times. If t already appears in the list, it is not reentered and no further processing of (t, T_i) is performed. If T_j is a successor of T_i and if $t + L(T_i) < B(T_j)$, then the entry $(t+L(T_i), T_j)$ is placed at the end of the search list. This reflects that T_i's potential state change at time $t + L(T_i)$ can initiate an evaluation of T_j. The search list will be empty once all potential evaluations caused (transitively) by T_g's transition have been generated. This termination is assured since the algorithm essentially finds all paths to each T_i with length less than $B(T_i)$. Then each functionals' list of offsets is expanded. We suppose that generator T_g makes state transitions at logical times $s_1, \cdots, s_k$. Each functional T_i's list of offsets is expanded k-fold by adding each s_j to each offset to reflect the effect of T_g's transition at s_j. We may discard duplicate entries and any entries exceeding the bound $B(T_i)$. The resulting list is again sorted, stored, and emptied. When this task is completed for each functional, the T_g *graph traversal* pass of the algorithm is considered terminated, and another generator's effect on the system is calculated. The *graph traversal phase* of the algorithm terminates upon the completion of the graph traversal pass for every generator T_g.

When the graph traversal phase terminates, each normal functional has sorted lists of potential evaluation times, one list for each generator functional. Each functional's lists are then merged, discarding duplicate entries. The merged list holds all of the functional's potential evaluation initiation times, in sorted order. The pairing of these times with the functional produces all of the functional's work nodes.

The work graph arcs are directly defined. If (t,T_j) is a work node, we need to find all work nodes sending arcs into (t,T_j). (s,T_i) sends an arc to (t,T_j) if T_i is a predecessor of T_j and if $t = s + L(T_i)$. For each predecessor T_i of T_j, this s is easily found if it exists, as T_i's list of potential initiation times is sorted.

A work graph represents all evaluations which might happen during a cycle. Our algorithm employs a breadth first search through the system graph initiated at a generator

functional. This search will thus follow all paths from the generator to a functional T_i, so all of T_i's potential evaluations are discovered.

Figure 4–1 illustrates a simple simulation system and its corresponding work graph. T_1 and T_2 are generator event functionals for this system; T_3 and T_4 are normal event functionals. Both T_3 and T_4 are assumed to have a logical delay of 1. A cycle in this system consists of the simulation work which occurs as a result of three generator transitions: both T_1 and T_2 are evaluated at time 0, only T_2 is evaluated at time 2. Figure 4–1 illustrates that a work graph might not be connected.

Later analysis relies on the fact that the work graph is acyclic. We demonstrate that this assumption is correct.

Lemma 4-1: A work graph is acyclic.

Proof: Suppose not. Let $(s_1,T_1), \cdots, (s_k,T_k), (s_1,T_1)$ be a cycle in a work graph. By construction, if an arc exists from (s_i,T_i) to (s_{i+1},T_{i+1}), then $s_i = s_{i+1} - L(T_i)$. We have assumed (in Chapter 3) that for every T_i, $L(T_i)$ is some positive constant. Thus, for every i, $s_i < s_{i+1}$. This implies that $s_1 < s_2 < \cdots < s_k < s_1$, an impossibility. Therefore no cycle exists in a work graph.

□

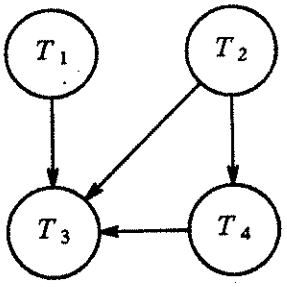
Finding all work nodes is equivalent to finding all paths through a directed graph. A simple example shows that the number of unique paths through a graph may explode combinatorially with the number of graph nodes. The complete directed graph is a graph with an arc from every node to every other node; if there are n nodes, there are $n!$ unique paths touching every node exactly once. We can define a system graph which is essentially a complete graph. We take a complete graph, add a source (generator) node, and define an edge from the source to every node in the complete graph. Likewise, we define a sink node, and an arc from every node in the complete graph to the sink node. We suppose that node j in this graph has a logical delay of 2^j. It is easy to see that there are an intractably large number of work nodes associated with this graph: every unique path to a node has a unique length.

The example above shows that work graph analysis must be restricted to simulation systems that are in a certain sense well–behaved. In well–behaved simulation systems it should not be possible for a huge amount of activity to be generated by a single input. For example, we have been able to use work graphs derived from logic network simulations. We conjecture that work graphs will be tractable for the simulation of many physical systems.

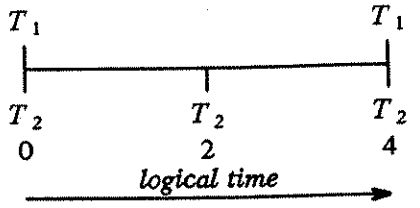
4.3. The Work Assignment Graph

The work graph defines an arc from (s,T_i) to (t,T_j) if an evaluation of T_i initiated at logical time s can initiate an evaluation of T_j at logical time t. Clearly the evaluation of (s,T_i) must be completed before the evaluation of (t,T_j) can begin. We must also consider other precedence concerns. These concerns are captured by the *work assignment graph.*

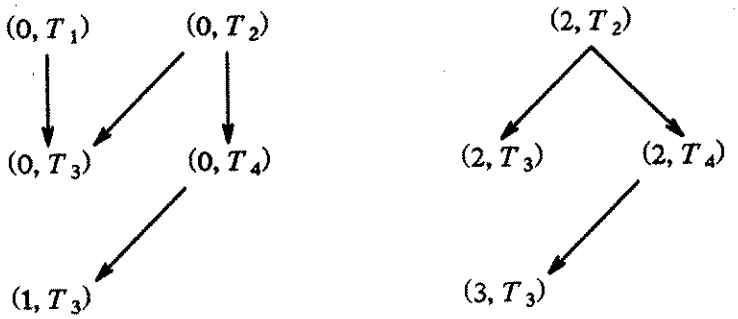
We create a work assignment graph by adding arcs to the work graph. Suppose the work graph defines an arc from (s, T_i) to (t, T_j). The evaluation of T_j initiated by T_i reads the states of **all** of T_j's predecessors at logical time t as input. T_k can be a predecessor of T_j and not be able to change its state at logical time t. The work graph will not then define an arc from a T_k work node into (t, T_j). The work assignment graph differs



System Graph



Generator Functionals Transition Cycle



Work Graph

System Graph, Generators Cycle, and Work Graph

Figure 4–1

from the work graph in that it will express (t, T_j)'s dependence on T_k. For every work node (t, T_j) and functional T_k, let $\psi_k[(t, T_j)]$ be the largest T_k work node time such that $t \geq \psi_k[(t, T_j)] + L(T_k)$. Then $(\psi_k[(t, T_j)], T_k)$ is the T_k work node with maximal time whose resulting state can affect the evaluation of (t, T_j). An evaluation of T_k initiated at logical time $\psi_k[(t, T_j)]$, must be completed before (t, T_j) can be evaluated. We thus define a *logical precedence arc* from $(\psi_k[(t, T_j)], T_k)$ to (t, T_j). Every initiation arc is also considered to be a logical precedence arc.

The work graph creation algorithm produces a sorted list of potential evaluation initiation times for each event functional. These lists are used to efficiently define a work assignment graph's logical precedence arcs. We find all logical precedence arcs into each of T_j's work nodes as follows. We first identify all of T_j's predecessors $T_1^j, \cdots, T_k^j$. For every work node (t, T_j) and every predecessor T_i, we define a *logical precedence arc* from work node $(\psi_i[(t, T_j)], T_i)$ to (t, T_j). The base node of this arc is efficiently found as T_i's work nodes are sorted in their initiation times. $(\psi_i[(t, T_j)], T_i)$ is said to be a *logical predecessor* of (t, T_j), and (t, T_j) is said to be a *logical successor* of $(\psi_i[(t, T_j)], T_i)$. Logical precedence arcs are also defined among a functional's own work nodes. The state transformation represented by (t, T_j) depends on the state of T_j at logical time t. The state of T_j at logical time t depends on the result of the potential transformation of T_j initiated at time $\psi_j[(t, T_j)]$. We therefore define a logical precedence arc from $(\psi_j[(t, T_j)], T_j)$ to (t, T_j).

The work assignment graph inherits the work graph's initiation arcs and additionally defines other logical precedence arcs. The precedence described by these arcs is independent of the simulation's partitioning. Additional precedence is induced by the assignment of work nodes to processors; analysis of a distributed simulation's run-time behavior requires consideration of the partition's effect on performance. We now formally define a partition to be a function mapping a work node W onto a pair (P_W, k_W). P_W identifies another work node whose execution will always immediately precede W's; k_W identifies the processor W is executed on. This defintion encapsulates both the assignment of work nodes to processors and the execution order of co-assigned work nodes. Definition 4-1 symbolically states five conditions on the partition function; we then explain each condition.

Definition 4-1: *Partition*

Let WG be a work graph, and let $N(WG)$ be its node set. Let $\mathbf{K} = \{1,2,\cdots,m\}$ be a set of processors. A *partition* of WG is a function $\Psi: N(WG) \rightarrow N(WG) \cup \{0\} \times \mathbf{K}$ such that

(i) If $\Psi(W) = (P_W, k_W)$ and $\Psi(V) = (W, k_V)$, then $k_V = k_W$;

(ii) For every $W \in N(WG)$, there exists at most one $V \in N(WG)$ and at most one $k \in \mathbf{K}$ such that $\Psi(V) = (W, k)$;

(iii) For every $k \in \mathbf{K}$, there exists exactly one $W_k \in N(WG)$ such that $\Psi(W_k) = (0, k)$;

(iv) If $W = (s, T_i)$ and $V = (t, T_i)$ are work nodes generated by the same functional T_i, then for some P_W, $S_V \in N(WG) \cup \{0\}$ and some processor $k \in \mathbf{K}$, $\Psi(W) = (P_W, k)$ and $\Psi(V) = (S_V, k)$;

(v) If $W = (s, T_i)$, $P_W = (t, T_j)$, and $\Psi(W) = (P_W, k)$, then $s + L(T_i) \leqslant t + L(T_j)$.

□

A partition function maps a work node W into a pair (P_W, k). k identifies W's processor, and P_W identifies a work node whose evaluation always immediately precedes W's own; we call P_W W's *execution predecessor*, and we call W P_W's *execution successor*. Condition (i) simply says that if W is V's execution predecessor, then Ψ maps W and V onto the same processor. Condition (ii) ensures that no other work node will identify W as its execution predecessor; condition (iii) ensures that within a processor, exactly one work node will not have an execution predecessor. Work nodes are ordered linearly by Ψ within a processor: exactly one node has no predecessor, every node's successor is unique. Condition (iv) says that all of a functional's work nodes must be assigned to the same processor. Finally, condition (v) says that a processor's linear ordering of work nodes is ordered in the work nodes logical completion times. We see that a partition Ψ both assigns event functionals to processors, and determines a static execution order for co-resident functionals. This static ordering ensures that functional evaluations are ordered in the evaluations' logical completion times. A static execution ordering is a useful modeling assumption, and closely resembles the technique used by most sequential simulations to order event evaluations in their logical completion times.

A processor's linear ordering of work nodes induces precedence among those nodes. Whenever node W is the execution predecessor of node V, we define an *execution precedence* arc from W to V in the work assignment graph.

We've shown how to create the *work assignment graph* $WAG(j)$ from work graph WG and partition Ψ_j by augmenting WG with additional logical and execution precedence arcs. Finally, we add a *system sink node* P_S which detects the cycle's termination. We can identify the work assignment nodes $(s_1, S_1), \cdots, (s_k, S_k)$ having no logical or execution successors. We direct a *termination arc* from each (s_i, S_i) to P_S to reflect that the cycle is not terminated at least until all simulation activity ceases.

ure 4–2 illustrates a work assignment graph for the work graph shown in Figure 4–1. Initiation arcs are labeled with an *i*. Table 4–1 gives the partition function Ψ defining the work assignment graph of figure 4–2.

Later analysis inducts on a topological sorting of the work assignment graph's nodes. The validity of this induction requires the work assignment graph to be acyclic.

Lemma 4-2: Let $WAG(j)$ be a work assignment graph for work graph WG and partition Ψ_j. Then $WAG(j)$ is acyclic.

Proof: Suppose not. Let $(s_1,T_1),\cdots,(s_k,T_k),(s_1,T_1)$ be a cycle of work assignment nodes in $WAG(j)$. We can easily adapt the argument given in the proof of lemma 4–1 to show that if the arc from P_i to P_{i+1} is a logical precedence arc, then we must infer that $s_1 < s_1$, an impossibility. Therefore, every arc in this cycle is an execution precedence arc. But this is a contradiction, the execution arcs within a processor impose a strictly linear (ie., non–cyclic) ordering on the nodes within the processor. Thus $WAG(j)$ is acyclic.

□

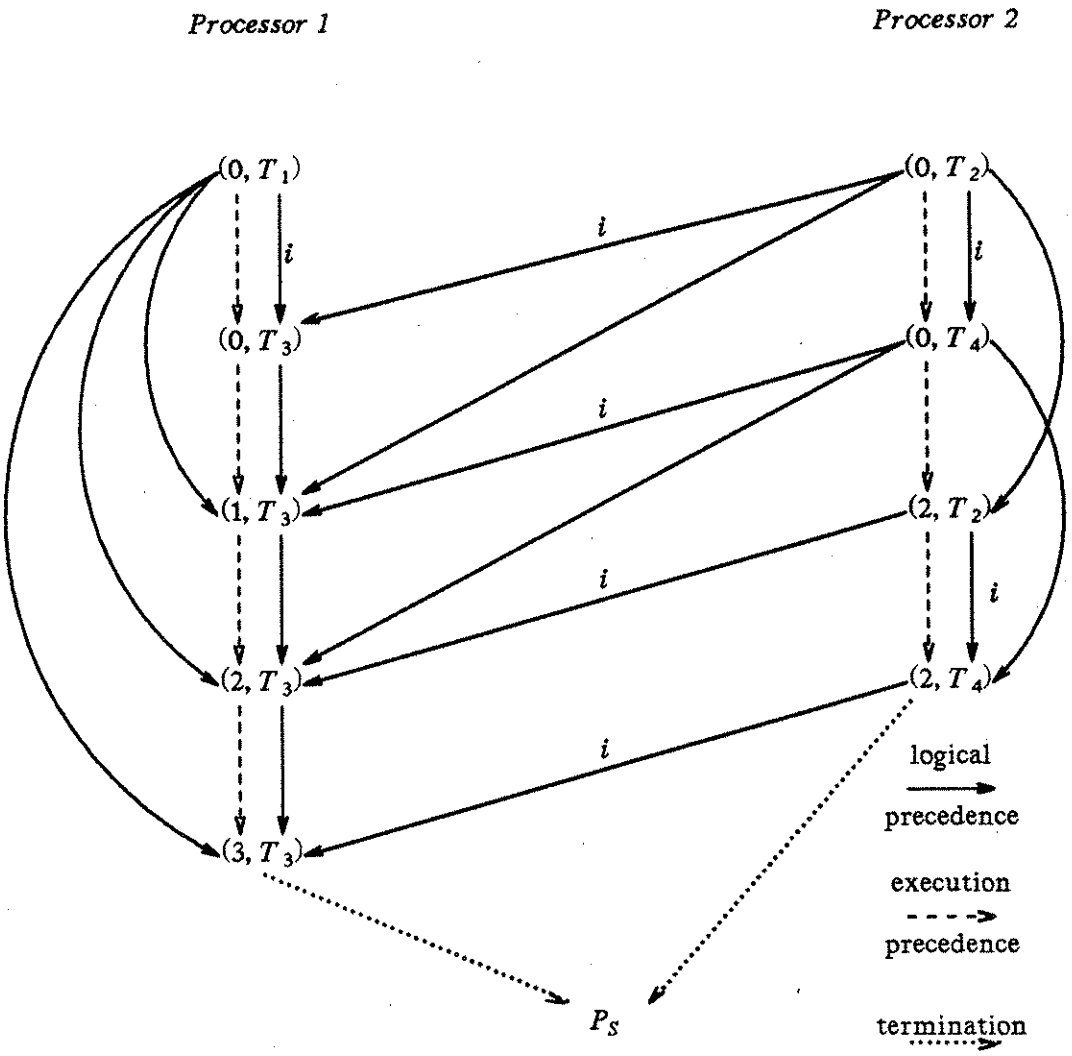
4.4. Synchronization and Cycle Execution Time

In this section we show how the work assignment graph naturally defines an ideal synchronization protocol, *hyper–message synchronization*. We show that a distributed simulation's performance under this protocol is optimal over the class of non–predictive synchronization protocols. We then show that hyper–message synchronization identifies synchronization requirements missed by the oracle synchronization scheme of Chapter 3. Finally, we show how to modify the work assignment graph with information from an observed cycle to create a directed acyclic graph whose longest path has length equal to the cycle execution time of that cycle under hyper–message synchronization.

4.4.1. Hyper–Message Synchronization

Chapter 3 describes a synchronization scheme related to the cycle execution graph. A cycle's execution time under this scheme was shown to be a lower bound on an achievable cycle execution time. We now define a synchronization scheme based on the work assignment graph. This scheme too is shown to bound any real synchronization method's cycle execution time from below. Our proposed synchronization scheme requires **every** work assignment node to be executed and to send messages to its successors. Non–zero queueing, execution and communication delays are suffered only by **evaluated** work assignment nodes. Non–evaluated work assignment nodes perform their execution and communication on a so–called "hyper" system where the "hyper–executions" and "hyper–messages" suffer no delay. We call this scheme hyper–message synchronization.

Under hyper–message synchronization, **every** work assignment node receives messages from its work assignment predecessors. A *completion* message from a logical predecessor reports the result of the predecessor's evaluation or hyper–execution; a *release* message from an execution predecessor releases the processor for use. Work assignment node W is evaluated if and only if W receives at least one completion message indicating a state change **from an initiation predecessor.** W's evaluation is initiated after W receives a message from each of its predecessors (logical and execution). This evaluation suffers a computational delay, and initiates the dispatch of completion messages to W's logical successors. A completion message suffers a communication delay of C when sent to



Work Assignment Graph
Figure 4–2

Partition Ψ			
WG node *W*	*PSI*(*W*)	*WG* node *W*	*PSI*(*W*)
(0, T_1)	(0, 1)	(0, T_2)	(0, 2)
(0, T_3)	((0, T_1), 1)	(0, T_4)	((0, T_2), 2)
(1, T_3)	((0, T_3), 1)	(2, T_2)	((0, T_4), 2)
(2, T_3)	((1, T_3), 1)	(2, T_4)	((2, T_4), 2)
(3, T_3)	((2, T_3), 1)		

Table 4–1

a different processor. If W's initiation predecessors send only "no state change" messages, W hyper–executes instantaneously with the arrival of the last no–change message. W simultaneously sends a "hyper" no–change message to each of its logical successors. A hyper–message's transition does not suffer delay.

Special rules apply to the system sink node predecessors. Every node S_i directing a termination arc to P_S sends a hyper "termination" message to P_S (P_S is not assigned to any processor) at the time S_i would issue a release message. The cycle terminates when P_S has a termination message from each of its predecessors.

When W hyper–executes, its hyper–execution initiation time does not depend on messages from non–initiation logical predecessors, nor the release message from its execution predecessor. This rule is designed so that the knowledge of W's non–evaluation (and non–change of state) is distributed as quickly as possible. W's hyper–execution might occur while W's processor is busy executing some other work assignment node. We need to take additional care to ensure the proper handling of evaluated nodes' access to the *CPU*. When W hyper–executes, it dispatchs a release message to its execution successor S_e, provided that W has received the release message from its own execution predecessor P_e. Otherwise, W sends the release message to S_e instantaneously with the arrival of the release message from P_e. A release message suffers no delay.

A cycle's execution time under hyper–message synchronization is a lower bound on any non–predictive synchronization protocol's cycle execution time. Before establishing this result, we define the concept of a *non–predictive* synchronization protocol.

Definition 4-2: *Non–Predictive Synchronization Protocol*

Let $WAG(j)$ be a work assignment graph for some simulation system under partition Ψ_j. A *synchronization protocol* is a decision policy which governs when a work assignment node to be evaluated is executed by its processor. A protocol is said to be *non–predictive* if for every work assignment graph node W with initiation predecessors $P_1, \cdots, P_k$

(i) The protocol does not anticipate that W is not evaluated before it determines that each initiation predecessor P_i does not change state;

(ii) If W is evaluated with an electronic completion time of t, the result of that evaluation is not known in W's processor before time t, and is not known in any other processor until time $t + C$, C being the communication delay;

(iii) The protocol enforces the static sequencing of evaluations defined by Ψ_j within a processor.

□

Less formally, a non–predictive protocol can ensure the simulation's correctness (see Chapter 2) only if it bases its scheduling decisions on the results of completed functional evaluations. It cannot anticipate a potential evaluation's occurrence nor its result; it cannot learn of an evaluation's result faster than the limits imposed by the system. It is easily seen that hyper–message synchronization is a non–predictive protocol. Furthermore,

we expect implemented realizable protocols to be non–predictive.

We can show that hyper–message synchronization is optimal over the class of non–predictive synchronization protocols. This demonstration is aided by some definitions. Let c be some cycle. For every work node W we define $XI(W)$ to be W's electronic (see Chapter 2) execution starting time under hyper–message synchronization. If V is a work assignment graph predecessor of W, we define $\Lambda(V, W)$ to be the electronic arrival time of V's message to W under hyper–message synchronization. $xmin(W)$ is defined to be W's minimal possible electronic execution starting time under a non–predictive protocol; $gmin(W)$ is defined to be the minimal possible electronic time at which, under a non–predictive protocol, W is granted the processor. If $W = (t, T_j)$, and $P_i = (s, T_i)$ is a logical predecessor of W, we define $kmin(W, P_i)$ to be the minimal possible electronic time at which, under a non–predictive protocol, W receives the state of functional T_i at logical time t. If W is not evaluated, $kmin(W)$ denotes the minimal electronic time at which, under any non–predictive protocol, W has all of its initiation predecessors' state values at logical time t. All of the values defined above depend on the particular cycle c. Our use of these definitions lets the dependence on the cycle be understood. Then we have

Theorem 4-1: Let $WAG(j)$ be a work assignment graph for work graph WG and partition Ψ_j. Then for every cycle of the simulation, the cycle completion time of a simulation using hyper–message synchronization is less than or equal to the cycle completion time of the simulation under any non–predictive synchronization protocol.

Proof: We denote the subgraph of $WAG(j)$ which excludes P_S and its incoming arcs by $WAG(j) - \{P_S\}$. We induct on a topological sorting of the $WAG(j) - \{P_S\}$ nodes to show

(i) If W is hyper–executed then $XI(W) \leqslant kmin(W)$.

(ii) If W is to be evaluated then $\Lambda(P_e, W) \leqslant gmin(W)$ where P_e is W's execution predecessor.

(iii) If W is to be evaluated then $XI(W) \leqslant xmin(W)$.

To establish the induction base, let W be any node without predecessors in $WAG(j)$. Conditions (i) and (ii) are vacuously satisfied; condition (iii) is trivially satisfied as W begins its execution at time 0. The induction base is thus established.

As the induction hypothesis, we suppose that all nodes which topologically precede a node W satisfy the induction statement. Let $P_1, \cdots, P_I$ be W's initiation predecessors in $WAG(j)$, let $P_{I+1}, \ldots, P_{m-1}$ be W's logical non–initiation predecessors, and let P_m be W's execution predecessor. We first establish **condition (i)** of the induction statement. Suppose that W hyper–executes. If predecessor P_i

hyper–executes then

$$\Lambda(P_i, W) = XI(P_i) \leqslant kmin(P_i) = kmin(W, P_i)$$

by the induction hypothesis. Now suppose P_i is evaluated. Then

$$\begin{aligned}\Lambda(P_i,W) &= XI(P_i) + X(P_i) + D(P_i, W)\\ &\leqslant xmin(P_i) + X(P_i) + D(P_i, W) \text{ by the induction hypothesis}\\ &= kmin(W, P_i)\end{aligned}$$

where $X(P_i)$ is P_i's execution delay and $D(P_i, W)$ is the communication delay between P_i's processor and W's processor ($D(P_i, W) \neq 0$ only if these processors are distinct). Thus for every P_i we have $\Lambda(P_i, W) \leqslant kmin(W, P_i)$. W is hyper–executed so that

$$\begin{aligned}XI(W) &= \max_{i \leqslant I}\{\Lambda(P_i, W)\}\\ &\leqslant \max_{i \leqslant I}\{kmin(W, P_i)\}\\ &\leqslant kmin(W)\end{aligned}$$

completing the induction on condition (i).

We establish **condition (ii)**. We assume that W is to be evaluated and has an execution predecessor, otherwise condition (ii) is vacuously satisfied. Let $P_{e1}, \cdots, P_{ek}$ be the sequence of nodes discovered by following execution arcs backwards from W (noting that $P_{e1} = P_m$) and let P_{ef} denote the first node in this sequence whose outgoing release message dispatch time is strictly greater than the arrival time of its incoming release message. It follows that

$$\Lambda(P_{e(i+1)}, P_{ei}) = \Lambda(P_{ei}, P_{e(i-1)}) = \Lambda(P_{e1}, W)$$

for $i = 2, \cdots, f-1$. If P_{ef} is evaluated then $xmin(P_{ef}) + X(P_{ef}) \leqslant gmin(W)$. The induction hypothesis states that $XI(P_{ef}) \leqslant xmin(P_{ef})$ so that

$$\Lambda(P_{e1}, W) = XI(P_{ef}) + X(P_{ef}) \leqslant gmin(W).$$

Suppose now that P_{ef} is hyper–executed. P_{ef} was chosen so that $xmin(P_{ef}) \leqslant gmin(W)$. By the induction hypothesis, $XI(P_{ef}) \leqslant kmin(P_{ef})$, so that $XI(P_{ef}) \leqslant gmin(W)$. But

$$\Lambda(P_{e1}, W) = \Lambda(P_m, W) = XI(P_{ef}) \leqslant gmin(W).$$

W therefore satisfies condition (ii).

We now establish **condition (iii)**. The proof of condition (i) showed that $\Lambda(P_i, W) \leqslant kmin(W, P_i)$ for $i \leqslant I$. The same arguments apply to logical prede–

cessors

$P_{I+1}, \cdots, P_{m-1}$. We proved condition (ii), showing that $\Lambda(P_m, W) \leqslant gmin(W)$. W's execution initiation time is given by

$$XI(W) = \max_{i \leqslant m}\{\Lambda(P_i, W)\}$$

$$\leqslant \max\left\{\max_{i<m}\{kmin(W, P_i)\}, gmin(W)\right\}$$

$$\leqslant xmin(W)$$

since the evaluation cannot begin until each predecessors state is known, and the processor is available. The induction on condition (iii) is thus established.

We assume that the sink node P_S has predecessors $S_1, \cdots, S_m$. Each predecessor treats P_S as its execution successor. We adapt our notation slightly to define $gmin(P_S, S_i) = gmin(P_S)$ with the understanding that P_S is S_i's execution successor. The cycle execution time is bounded by applying condition (ii):

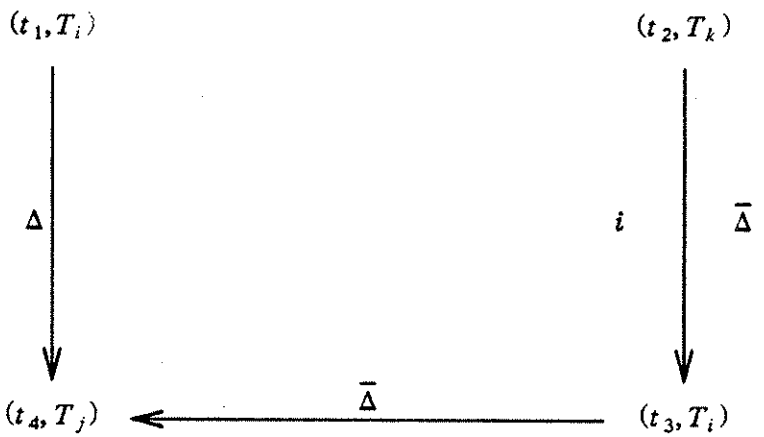
$$\max_{i \leqslant m}\{\Lambda(S_i, P_S)\} \leqslant \max_{i \leqslant m}\{gmin(P_S, S_i)\}$$

Clearly the cycle execution time must be greater than or equal to $gmin(P_S, S_i)$ for each S_i. It follows that the right hand side of the relation above is a lower bound on the cycle execution time under a non-predictive synchronization protocol, completing the theorem's proof.

□

4.4.2. Synchronization Requirements

The synchronization requirements imposed by hyper-message synchronization are more stringent than the requirements imposed by the oracle synchronization described in Chapter 3. This is directly shown by example. Figure 4-3 illustrates part of a work assignment graph. Again, initiation arcs are identified by an i. Arcs representing completion messages are labeled by Δ if a change is reported, they are otherwise labeled by $\underline{\Delta}$. Some arcs which would appear in the full work assignment graph are not relevant to our discussion, and have been omitted from figure 4-3. For example, the initiation arcs into (t_1, T_i), (t_2, T_k), and (t_4, T_j) are not shown; it is understood that these nodes have their evaluations initiated. Nor do we illustrate other logical arcs which might be rooted in the shown nodes. Consider some cycle where all nodes in figure 4-3 except (t_3, T_i) are evaluated; (t_3, T_i) and its associated incoming and outgoing arcs are not represented in the cycle execution graph, while all other nodes and arcs in figure 4-3 are. Under oracle synchronization an evaluation of (t, T_j) cannot begin until the completion of every functional evaluation whose result is an input to (t, T_j)'s evaluation. In figure 4-3, the



Comparison of Hyper-Message and Oracle Synchronization
Figure 4-3

evaluation of (t_4, T_j) must wait for (t_1, T_i)'s evaluation completion. Assuming no other precedence (save its initiation predecessor), the oracle synchronization protocol allows (t_4, T_j) to be evaluated as soon as (t_1, T_i)'s evaluation has completed. We now demonstrate that the hyper–message protocol imposes additional constraints on (t_4, T_j)'s evaluation.

An initiation arc is directed from (t_2, T_k) to (t_3, T_i). Furthermore, the work assignment graph places a logical (non–initiation) arc from (t_3, T_i) to (t_4, T_j). Under hyper–message synchronization, (t_4, T_j)'s evaluation cannot begin before it receives a completion message from (t_3, T_i). (t_3, T_i) cannot dispatch that completion message before it receives the no–change message from (t_2, T_k). This is reasonable; if the state of T_k **had** been changed by (t_2, T_k)'s evaluation, then (t_3, T_i) would be evaluated, and could change T_i's state. (t_4, T_j)'s correct evaluation would depend on the result of (t_3, T_i)'s evaluation, not the result of (t_1, T_i)'s evaluation.

We may assume (t_1, T_i)'s evaluation completes at physical time t. We may also assume that (t_4, T_j) receives a completion message from its initiation predecessor before physical time t. Under oracle synchronization, (t_4, T_j)'s evaluation may begin execution at time t (assuming that T_i is assigned to the same processor as T_j). We may also assume that (t_2, T_k)'s evaluation is **not** completed by physical time t. Under hyper–message synchronization, (t_4, T_j)'s evaluation cannot begin at or before physical time t. This discrepancy arises from the oracle's ability to discern **when (t_1, T_i)'s evaluation is completed** that (t_4, T_j)'s evaluation can begin without regard to the potential evaluation represented by (t_3, T_i). The oracle then has the power to predict that (t_2, T_k)'s evaluation will not result in a change of state, a somewhat unrealistic assumption. We see that the oracle synchronization method is not non–predictive in the sense of definition 4–2, and that hyper–message synchronization better models the synchronization that a distributed simulation must contend with.

4.4.3. Hyper–Execution Graph

An observed cycle's execution time under oracle synchronization was shown to be equal to the length of the longest path through the cycle execution graph. We can similarly modify and weight a partition's work assignment graph as a function of an observed cycle so that the cycle execution time under hyper–message synchronization is equal to the length of the longest path through this graph. The modified graph is called a *hyper–execution graph*. We next describe the construction of a hyper–execution graph and show that it defines the cycle execution time.

A cycle execution node represents an evaluated event; the length of the longest path through the node is equal to the node's execution completion time. The "action" represented by the node, namely execution, is initiated once the node has messages from each predecessor. A hyper–executed work assignment node may send its release message at a different time than it sends its completion messages. Such a node performs two actions: hyper–execution (and the issuance of subsequent completion messages), and the issuance of a release message. We are thus led to modify the work assignment graph by splitting each hyper–executed node W into two nodes $IP(W)$ and $EP(W)$. $IP(W)$ is W's *initiation precedence node*, and represents W's hyper–execution responsibilities. The arcs from W's initiation predecessors are directed to $IP(W)$; the arcs to W's logical successors are rooted in $IP(W)$. $IP(W)$ performs its action, hyper–execution, once every initiation predecessor has reported completion. The node $EP(W)$ is W's *execution precedence node*, and represents W's release message responsibilities. The arc from W's execution predecessor is directed to $EP(W)$, the arc to W's execution successor is rooted in $EP(W)$. An arc is directed to $EP(W)$ from $IP(W)$ to reflect the precedence on the release message's issuance. If W is evaluated during the cycle, it is represented by a single node $R(W)$ which is understood to be both an execution and logical precedence node for W. A single node

$R(P_S)$ represents $WAG(j)$'s system sink node P_S.

This intuitive description of a hyper–execution graph leads us to a more formal definition. Let WG be a work graph, Ψ_j some partition of that graph, and c an observed cycle of the simulation. Let $WAG(j)$ be WG's work assignment graph for Ψ_j. Let $D(W, V)$ be the communication delay between work node W's and work node V's processors. The *hyper–execution graph* $HEG(WG,j,c)$ is defined as follows:

(i) If W is evaluated during c, there is a node $R(W)$ in $HEG(WG,j,c)$; $R(W)$ is weighted by W's execution delay. If W is not evaluated in c, there are nodes $IP(W)$ and $EP(W)$ in $HEG(WG,j,c)$; both nodes are weighted by zero.

(ii) If W is a logical non–initiation predecessor of V and V is evaluated, there is an arc in $HEG(WG,j,c)$ from W's initiation precedence node to $R(V)$. If W is evaluated in cycle c, this arc is weighted by $D(W, V)$; the arc is otherwise weighted by zero.

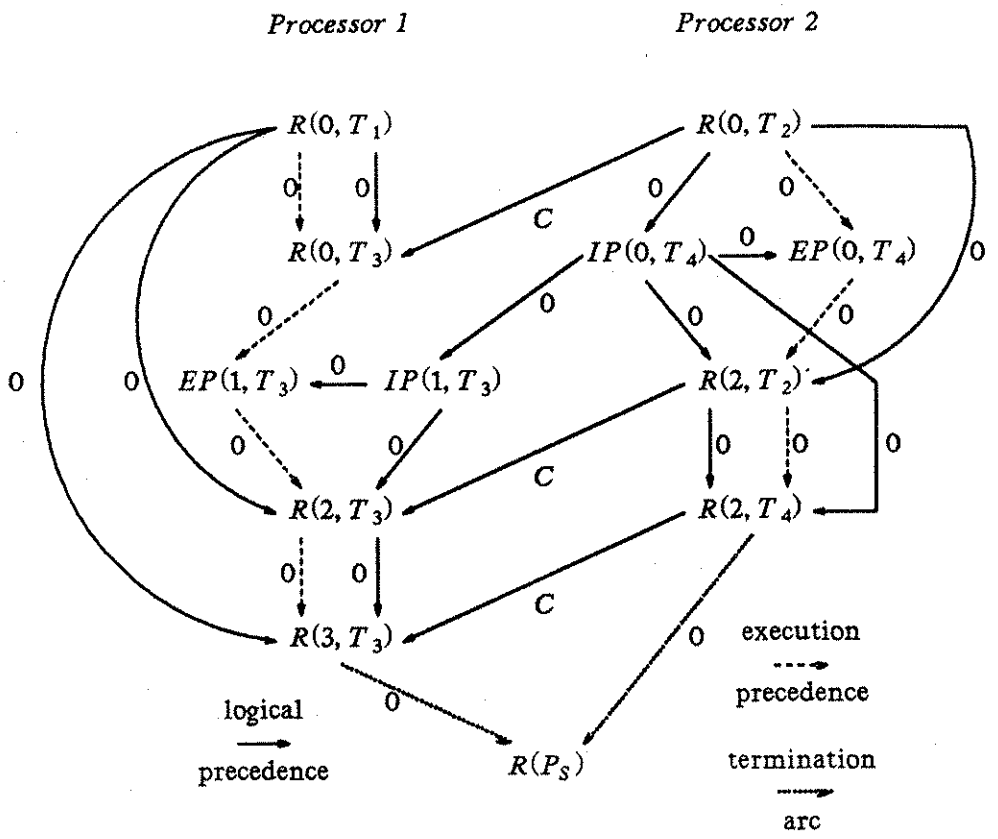
(iii) If W is an initiation predecessor of V there is an arc in $HEG(WG,j,c)$ from W's initiation precedence node to V's initiation precedence node. If W is evaluated in cycle c, this arc is weighted by $D(W, V)$; the arc is otherwise weighted by zero.

(iv) If W is an execution predecessor of V in $WAG(j)$, there is an arc from W's execution precedence node to V's execution precedence node in $HEG(WG,j,c)$. This arc is weighted by zero.

(v) If W is a termination predecessor of P_S in $WAG(j)$ there is an arc from W's execution precedence node to to $R(P_S)$. This arc is weighted by zero.

(vi) If W is not evaluated in cycle c, there is an arc from $IP(W)$ to $EP(W)$ in $HEG(WG,j,c)$. This arc is weighted by zero.

Figure 4–4 shows how the work assignment graph of Figure 4–2 is transformed into a hyper–execution graph after the observed cycle given by table 4–2. This table states whether each work node was evaluated and whether it changed its state. Figure 4–4 does not explicitly identify the hyper–execution graph's node weights; however, each arc is weighted. The non–zero arc weights are seen to represent the delays suffered by real (non–hyper) messages. If W is not evaluated, we omit arcs describing completion messages from W's logical, but non–initiation predecessors; these messages are irrelevant to W's behavior. For example, there is a logical arc from $(0, T_1)$ to $(1, T_3)$ in the work assignment graph (figure 4–2), but no arc from $R(0, T_1)$ to $IP(1, T_3)$ in the hyper–execution graph.



Hyper–Execution Graph
Figure 4–4

Observed Cycle		
Node	Evaluated?	Change State?
$(0, T_1)$	yes	yes
$(0, T_2)$	yes	no
$(0, T_3)$	yes	yes
$(0, T_4)$	no	–
$(1, T_3)$	no	–
$(2, T_2)$	yes	yes
$(2, T_3)$	yes	yes
$(2, T_4)$	yes	yes
$(3, T_3)$	yes	yes

Table 4–2

We will show that the length of the longest path through a hyper–execution graph is equal to the observed cycle's execution time under the assumed partition and hyper–message synchronization. As this argument inducts on a topological sorting of the hyper–execution graph's nodes, we must first prove that a hyper–execution graph is acyclic.

Lemma 4-3: Let $HEG(WG,j,c)$ be a hyper–execution graph. Then $HEG(WG,j,c)$ is acyclic.

Proof: For the sake of contradiction, we suppose that a cycle $P_1,\cdots,P_k,P_1$ of hyper–execution graph nodes exists in $HEG(WG,j,c)$. The same reasoning given in the proof of lemma 4–2 shows that no arc in this cycle can be a precedence arc from some $R(W)$ or $IP(W)$ to some $R(V)$ or $IP(V)$; every arc in this cycle must either be an execution precedence arc, or an arc from some $IP(W)$ to $EP(W)$. However, the only arcs directed to nodes of the form $IP(W)$ are logical precedence arcs, so all arcs in the cycle are execution precedence arcs. This is a contradiction, as all nodes in a processor are ordered linearly (and non–cyclically). Thus $HEG(WG,j,c)$ is acyclic.

□

We now demonstrate that the length of the longest path through $HEG(WG,j,c)$ is equal to the execution time of cycle c under partition Ψ_j, assuming hyper–message synchronization.

Lemma 4-4: Let $HEG(WG,j,c)$ be a hyper–execution graph, let W be a node in the work graph WG, and suppose that hyper–message synchronization is used. Then,

(i) If W is evaluated in cycle c, the length of the longest path through $R(W)$ is equal to W's execution completion time.

(ii) If W is hyper–executed in cycle c, the length of the longest path through $IP(W)$ is equal to W's hyper–execution time.

(iii) If W is hyper–executed in cycle c, the length of the longest path through $EP(W)$ is equal to the time at which W sends a release (or termination) message.

Proof: All of the assertions above follow fairly directly from an induction on a topological sorting of $HEG(WG,j,c)$'s nodes. The salient observations supporting this proof are that the edge and node weights correctly model the behavior of communication and execution delays, and that the arcs correctly model precedence.

□

The central result follows directly from this last lemma.

Theorem 4-2: Let $HEG(WG,j,c)$ be a hyper–execution graph, and let P_S be the sink node for $WAG(j)$. Then the length of the longest path to $R(P_S)$ is equal to the cycle execution time under hyper–message synchronization.

Proof: Let $S_1,\cdots,S_k$ be P_S's predecessors in $WAG(j)$. By conditions (i) and (iii) of lemma 4–4, the length of the longest path through the execution precedence node of S_i in $HEG(WG,j,c)$ is S_i's termination message dispatch time. The cycle is terminated when P_S receives the last message; it follows that the length of longest path to $R(P_S)$ in $HEG(WG,j,c)$ is the cycle execution time.

□

4.5. Chapter Summary

The results developed in this chapter are fundamental to most of the remainder of this dissertation; we highlight these results again. Before we can reasonably partition a simulation, we have to understand how different partitions affect the distributed simulation's performance. The synchronization protocol critically affects performance. To model the effect of a partition on performance we must first model synchronization. To model synchronization, we must first identify the simulation's synchronization needs; these needs are expressed by the work graph and the work assignment graph. Using the work assignment graph as a base, we describe the hyper–message synchronization protocol. This protocol is useful in two major ways. First, we prove that a distributed simulation's performance (assuming our simulation and partition model) under hyper–message synchronization is optimal over the class of non–predictive protocols. Secondly, we can calculate any given cycle's execution time under a particular partition assuming hyper–message synchronization. Our development of hyper–message synchronization is thus an important modeling result; we can model the performance of a distributed simulation by assuming that the simulation uses hyper–message synchronization. If we can model the distributed simulation, then we can begin to address the problem of choosing a partitic of the simulation.

Chapter 5

Probabilistic Analysis

5.1. Chapter Overview

In Chapter 3 we showed that a simulation cycle's run-time behavior varies randomly. A natural way of analyzing the effectiveness of a chosen partition is to consider the mean and standard deviation of the cycle execution time distribution. In this chapter we consider the problem of estimating these quantities statically, given a description of the simulation and its input behavior. In section 5.2 we show how to determine each work node's probability of evaluation during a cycle. We find that this computation is feasible only if we restrict the topology of the simulation's system graph to be an in-tree. However, the methodology used to calculate these probabilities can be used in the general case to **estimate** the probabilities of evaluation. In section 5.3 we use the probabilities of section 5.2 and the hyper-message synchronization protocol of Chapter 4 to develop an algorithm for calculating lower bounds on a partition's cycle execution time mean and second moment. The derivation of these bounds assumes that a work node's incoming message arrival times under hyper-message synchronization are probabilistically independent. This assumption will not always be valid; but again, the same calculations can be made in the general case by simply assuming this independence, to yield estimated lower bounds.

The analysis developed in this chapter is straightforward, using standard probabilistic techniques and inequalities. Yet, nothing of its kind exists in the simulation literature. Researchers have not considered modeling a distributed simulation's run-time performance. The work most closely resembling our own studies the estimation of PERT network finishing time moments [Elm67, HaW66, Mar65, RoT77]. This work is not directly applicable to our problem, as a PERT network's topology is static, and its activity times are assumed to be independently distributed. Other work on modeling system run-time performance include [DuB82, Kle76, RaH80, SmL82, TCB78, YYS81]; these various models all use assumptions which vary significantly from our own.

The results of this chapter are useful in three major ways. A partitioning algorithm needs to consider both the frequency and execution delay of a work node's evaluation. Our technique for calculating probabilities of evaluation is thus useful to a partitioning algorithm. Secondly, in Chapter 6 we will show that a partition's cycle execution time moments can be estimated using Bayesian statistics. A Bayesian estimation uses prior knowledge of the cycle mean and standard deviation; our bounds estimation technique can be used to provide that knowledge. Finally, it is important to appreciate the magnitude of the restrictions supporting our analysis. Most simulations will not satisfy these restrictions; while we can approximate the system's behavior with our analysis, we have no assurance that these approximations are accurate. This realization suggests that a **static** probabilistic analysis should not be the pillar of our approach to partitioning.

5.2. Equilibrium Probability of Evaluation

A cycle's execution time is largely determined by the amount of simulation work performed during that cycle, which varies randomly. A study of the system's average

behavior must then be probabilistic. One measure of a cycle's simulation activity is obtained by calculating each work graph node's probability of evaluation. These probabilities in no way depend on the distributed simulation's synchronization protocol or partition; they depend only on the structure of the work graph and the probabilistic nature of the inputs presented to the simulation. Clearly, knowledge of each work node's evaluation probability is potentially useful information for a partitioning algorithm. In this section we identify conditions allowing the calculation of these probabilities, and specify the necessary computations.

5.2.1. System Restrictions

We encounter some difficulties when we attempt to calculate a work node's exact probability of evaluation, and are so led to restrictions which may allow this calculation.

The work graph WG identifies a work node's initiation predecessors; a work assignment graph $WAG(j)$ additionally identifies its execution and logical predecessors. Let $W \in WAG(j)$; W's logical predecessors in $WAG(j)$ identify the evaluation results used as input to W's evaluation (recall that a node V's evaluation result is defined even if the potential evaluation represented by V does not occur). Unless W's execution predecessor is also a logical predecessor, its behavior is not directly considered in the evaluation of W. We therefore transform the work assignment graph $WAG(j)$ into the *work precedence* graph WPG by simply removing all execution arcs from $WAG(j)$. A work node (s, T_i)'s evaluation result is then completely determined by the functional specification of T_i, and the evaluation results of (s, T_i)'s predecessors in WPG. The behavior of (s, T_i)'s initiation predecessors in WPG (inherited from WG by way of $WAG(j)$) determines whether (s, T_i) is actually evaluated.

Work node W's probability of evaluation during a cycle depends on the probabilities that W's initiation predecessors $P_1, \cdots, P_I$ change state during that cycle. Virtually the only way to calculate these predecessors' joint probabilities of change is to find circumstances where the initiation predecessors' behavior are conditionally independent of each other. This amounts to finding a set U of work nodes such that no path from $u \in U$ to W's initiation predecessor P_i can contain a node (other than u) which lies on a path to another initiation predecessor P_j. The analysis proceeds by conditioning on the joint behavior of U's nodes; given that the nodes in U behave a particular way during a cycle, the predecessors' behavior are independent of each other, making it easy to calculate the probability that at least one of them changes state (causing W to be evaluated). To find the unconditional probability that W is evaluated, it is necessary to consider every possible joint behavior of the nodes in U. This computation is exponentially complex in the size of the set U. U can consist of all work nodes topologically preceding (in WG) W's initiation predecessors, so that this sort of analysis can be intractable. However, if W's initiation predecessors are known to behave (unconditionally) independently of each other, then W's evaluation probability is directly found. We must restrict the system graph's topology to achieve independence among initiation predecessors.

A directed *in-tree* is a connected, directed acyclic graph such that every node has at most one successor. By assuming that the system graph is an in-tree, each functional has *path independent* predecessors.

Definition 5-1: *Path Independence*

Let P_i and P_j be nodes in a directed acyclic graph. P_i and P_j are said to be *path independent* if no path exists between them, and if P_i and P_j have no common ancestors.

□

It is easily seen that all of a functional's predecessors in an in–tree system graph are path independent. Furthermore, if T_W is an in–tree node with predecessors T_1 and T_2, any ancestor of T_1 and any ancestor of T_2 are path independent.

We can show that if a system graph is an in–tree, then the work precedence graph acquires useful independence properties.

Lemma 5-1: Let SG be a system graph with an in–tree topology, and let WPG be its work precedence graph with respect to some input cycle vector. Let T_1 and T_2 be path independent functionals in SG, and let (s_1, T_1) and (s_2, T_2) be work nodes in WPG. Then (s_1, T_1) and (s_2, T_2) are path independent in WPG.

Proof: For the sake of contradiction we suppose that (s_1, T_1) and (s_2, T_2) are not path independent in WPG. Then there exists a work node (s_c, T_c) in WPG with a path to both (s_1, T_1) and (s_2, T_2). Now consider any two work nodes (s, T_i) and (t, T_j) which share an arc in WPG; by definition, one work node must be a logical predecessor of the other. It follows that either $T_i = T_j$, or T_i and T_j share an arc in the system graph. By repeating this argument to the nodes on the paths from (s_c, T_c) to (s_1, T_1) and (s_2, T_2), we see that either $T_c = T_1$ or there is a path through the system graph from T_c to T_1. Likewise, either $T_c = T_2$ or there is a path through the system graph from T_c to T_2. We suppose first that $T_c = T_1$. Since T_1 and T_2 are distinct, $T_c \neq T_2$, so there is a path from T_c to T_2 in the system graph. $T_c = T_1$, so that there is a path from T_1 to T_2 in the system graph. However, T_1 and T_2 are path independent, a contradiction. Clearly, the same sort of contradiction is found by assuming that $T_c = T_2$. If T_c is distinct from both T_1 and T_2, then there is a path through the system graph from T_c to T_1 **and** a path from T_c to T_2. This again is a contradiction, as T_1 and T_2 are path independent in the system graph.

We have explored the logical consequences of assuming that (s_1, T_1) and (s_2, T_2) are **not** path independent, and have found a contradiction in every case. We conclude then that (s_1, T_1) and (s_2, T_2) **are** path independent.

□

A useful, and immediate consequence of lemma 5–1 is stated as a corollary.

Corollary 5-1: Let SG be a system graph with an in–tree topology, and let WPG be its work precedence graph with respect to some input cycle vector. Let W be a work node with initiation predecessors $P_1, \cdots, P_I$ in WPG. Then for every $i \neq j$, P_i and P_j are path independent in WPG.

□

Corollary 5–1 states that a work node W's initiation predecessors are path independent in WPG. We would like these predecessors to behave independently. However, correlation in a functional's initiation predecessors' behavior may be introduced through correlations in different generators' transformations. We will therefore assume that every generator's behavior during a cycle is independent of any other generator's behavior. If T_g is a generator, we let H_g^k be the vector of transition values taken by T_g at its specified times during the *kth* cycle. We assume that the process $\{H_g^n\}$ is an ergodic Markov chain, independent of any other process $\{H_g^n\}$, and independent of all other functionals' states. A system with such generators is said to have *independent inputs.*

Even if a system's graph is an in–tree and the system has independent inputs, it is still possible for a work node's initiation predecessors to inherit correlation from the initial state assignments. We will thus assume that each functional's initial state is assigned (probabilistically) independently of any other functional's initial state. A deterministic assignment of initial states satisfies this assumption, albeit in a degenerate way. A system whose functionals' initial states are assigned independently of each other and independently of the system's input processes is said to have an *independent initialization.*

One further bit of notation aids our analysis. If W is a work node in WPG, we let $W(n)$ denote W's evaluation result during the *nth* cycle of the simulation. Now we can show that during the *nth* cycle, W's initiation predecessors behave independently of each other.

Lemma 5-2: Let SG be an in–tree system with independent inputs and independent initialization. Let WPG be its work graph with respect to some input cycle vector. Let T_1 and T_2 be path independent functionals in the system graph SG, and let $P_1 = (s_1, T_1)$ and $P_2 = (s_2, T_2)$ be work nodes in WPG. Then for every n, $P_1(n)$ and $P_2(n)$ are are probabilistically independent.

Proof: We induct on the cycle, n. For the base case, we consider $n = 1$. Let $A(P_i)$ denote the set of all P_i's ancestors in WPG, $i = 1,2$. Since (by lemma 5–1) P_1 and P_2 are path independent, it follows that $A(P_1) \cap A(P_2) = \emptyset$. For $i = 1,2$, P_i's evaluation result $P_i(1)$ is entirely determined by

(i) The initial states of functionals with work nodes in $A(P_i)$; and

(ii) The transitions chosen by generator work nodes $(s_g, T_g) \in A(P_i)$ during the first cycle.

$P_1(1)$ and $P_2(1)$ are thus seen to be random variables, as they are functions of other random variables. However, the random variables from which $P_1(1)$ is determined are independent of the random variables from which $P_2(1)$ is determined. It follows immediately that $P_1(1)$ and $P_2(1)$ are independent.

For the induction hypothesis, we suppose that whenever T_1 and T_2 are path independent functionals in the work graph with work nodes $P_1 = (s_1, T_1)$ and $P_2 = (s_2, T_2)$ in WPG, then $P_1(n-1)$ is independent of $P_2(n-1)$. Consider cycle n and let T_1 and T_2 be any two path independent functionals in the system graph. Suppose $P_1 = (s_1,T_1)$ and $P_2 = (s_2,T_2)$ are work nodes in WPG. For $i = 1,2$, P_i's evaluation result $P_i(n)$ is entirely determined by

(i) The states of functionals with work nodes in $A(P_i)$ at the beginning of cycle n; and

(ii) The transitions chosen by generator work nodes $(s_g,T_g) \in A(P_i)$ during the *nth* cycle.

Suppose now that functional T_{k1} has a work node in $A(P_1)$, and that functional T_{k2} has a work node in $A(P_2)$. It follows from the definition of WPG that T_{k1} is an ancestor of T_1 and T_{k2} is an ancestor of T_2 in the system graph. Since the system graph is an in-tree, T_{k1} and T_{k2} are path independent in the system graph. Let s_{k1} be the latest time associated with T_{k1} in a work node; likewise define s_{k2}. Then the state of T_{k1} at the beginning of cycle n is equal to the evaluation result $(s_{k1},T_{k1})(n-1)$. Likewise, the state of T_{k2} at the beginning of cycle n is equal to the evaluation result $(s_{k2},T_{k2})(n-1)$. By the induction hypothesis then, the states of T_{k1} and T_{k2} at the beginning of cycle n are independent. Furthermore, we have assumed that the generator functionals' transitions are independent of each other and the state of the simulation system. Thus $P_1(n)$ is a function of random variables which are independent of the random variables determining $P_2(n)$; it follows that $P_1(n)$ and $P_2(n)$ are independent, completing the induction argument.

□

Corollary 5-2: Let SG be a system graph with an in-tree topology, and let WPG be its work precedence graph with respect to some input cycle vector. Let W be a work node with initiation predecessors $P_1,\cdots,P_I$ in WPG. Then for every i,j, $i \neq j$, the limiting (as $n \to \infty$) probability that P_i changes its functional's state during a cycle is independent of the limiting probability that P_j changes its functional's state during a cycle.

□

The probabilistic independence of path independent functionals in an in-tree system graph is our key to an exact probabilistic treatment of evaluation probabilities.

5.2.2. Exact Evaluation Probabilities

The last subsection constrained our attention to in–tree systems with independent inputs, and independent initialization. It **may** then be computationally possible to calculate a work node's exact equilibrium evaluation probability. In this section we show how to compute that probability, and discover that the computation is still exponentially complex in the product of the number of initiation predecessors, the number of possible functional states, and the number of a functional's work nodes. However, for systems where this product is small, we can use this method to learn a good deal about how the system will behave.

The solution approach uses a Markov chain model of the system, very much like Chapter 3's. However, Chapter 3's general treatment uses a chain whose state vector encompasses the entire simulation system. If a system has an in–tree topology, independent inputs, and independent initialization, we can drastically reduce the size of the chains we have to deal with. Instead of treating the whole system collectively, we are able to restrict our attention to a functional's local environment. The state vectors of the chains we create encode a functional's transition behavior, and its (system graph) predecessors' input behavior.

We assume that the system graph has been topologically sorted, and will process the functionals with respect to this order. It is necessary to jointly treat all of a functional's different possible evaluations, since each evaluation affects the same system variable. We encapsulate all possible evaluations with a *cycle history vector*. The *mth* component of T_i's cycle history vector records the evaluation result of T_i's *mth* possible evaluation during a cycle. H_i^k denotes T_i's cycle history vector for the *kth* cycle. H_i^k describes what happened to T_i's state during the *kth* cycle; the last component of H_i^k gives T_i's state at the beginning of the $k+1st$ cycle. As a result of T_i's state dependency on its previous state and the states of its (system graph) predecessors, H_i^{k+1} is completely determined by the last component of H_i^k, the cycle history vectors of T_i's (system graph) predecessors during the $k+1st$ cycle, and T_i's functional definition. T_i's cycle history vector defines a state in a stochastic process $\{H_i^n\}$.

We suppose that T_i has k system graph predecessors and denote the *jth* predecessor's cycle history process by $\{I_{ij}^n\}$. We describe T_i's total input process by concatenating T_i's predecessors' input processes $\{I_i^n\} = \{I_{i1}^n I_{i2}^n \cdots I_{ik}^n\}$. Lemma 5–2 implies that the processes $\{I_{ij}^n\}$ $j = 1,2,\cdots,k$ are mutually independent. As demonstrated in Chapter 3, if the input process $\{I_i^n\}$ is a Markov chain, then $\{H_i^n\}$ will be a Markov chain. Since we analyze the event functionals in topological order, we may assume that the equilibrium state probabilities of each process $\{I_{ij}^n\}$ are known. Likewise, we may assume that each (system graph) predecessors' work nodes' equilibrium probabilities of changing state are known. Having made these assumptions, we must demonstrate how to calculate $\{H_i^n\}$'s equilibrium state probabilities, and T_i's work nodes' probabilities of changing state.

T_i's work nodes' equilibrium evaluation probabilities are easily calculated from its (system graph) predecessors' work nodes' probabilities of changing state. Let W be a T_i work node, and let $(s_1,T_1),\cdots,(s_I,T_I)$ be its initiation predecessors in WPG. For each $j = 1,\cdots,I$, let Q_j be the equilibrium probability that functional T_j changes its state at logical time $s_j + L(T_j)$. W is evaluated if and only if at least one of its initiation predecessors changes state. Since W's initiation predecessors behave independently, W's equilibrium evaluation probability is given by

$$Prob\{W \text{ is evaluated}\} = 1 - \prod_{j=1}^{I}(1 - Q_j); \tag{5–1}$$

which is one minus the probability that W is not evaluated.

To analyze other aspects of $\{H_i^n\}$'s behavior, we analyze the behavior of a stochastic process whose states are the concatenation of $\{H_i^n\}$'s states with $\{I_i^n\}$'s states: $\{H_i^n I_i^n\}$. The H_i component of the state is interpreted as T_i's cycle history vector from the **last** cycle, while the I_i component represents the input presented to T_i during the **current** cycle. It is useful to treat a state of $\{H_i^n I_i^n\}$ as a vector of its constituent cycle history vectors: $<H_i, I_{i1}, \cdots, I_{ik}>$. Then $\{H_i^n I_i^n\}$'s transition probability from $<H_i, I_{i1}, \cdots, I_{ik}>$ to $<\hat{H}_i, \hat{I}_{i1}, \cdots, \hat{I}_{ik}>$ is 0 if $\hat{H}_i$ is **not** the result of applying T_i's functional definition to H_i's last component and the input vectors I_{ij}, $j = 1, \cdots, k$. If $\hat{H}_i$ is the proper result, the transition probability is just the probability that the collective input vector $<\hat{I}_{i1}, \cdots, \hat{I}_{ik}>$ occurs in the next cycle. By independence, this probability is just the product of the constituent processes' equilibrium state probabilities: $\prod_{j=1}^{k} Prob\{\hat{I}_{ij}\}$. This specification of process states and transition probabilities defines $\{H_i^n I_i^n\}$; we assume that $\{H_i^n I_i^n\}$ is ergodic, and so solve for its equilibrium state probabilities.

We still need to find the T_i work nodes' probabilities of changing T_i's state. We can simplify our analysis by collapsing $\{H_i^n I_i^n\}$'s state space; we partition its states so that states with common H_i components are grouped together. For each vector H_i, we sum the equilibrium probabilities of all states in H_i's partition set to find the equilibrium probability of observing H_i in a cycle. We sum all the transition probabilities of states in H_i's partition set to states in $\hat{H}_i$'s partition set to find the transition probability of $\{H_i^n\}$ passing from H_i to $\hat{H}_i$. Likewise, to find the equilibrium probability of $\{H_i^n\}$ passing from H_i back into H_i, we sum the transition probabilities between states in H_i's partition set. T_i's work nodes' probabilities of change are easily found from this collapsed chain. Consider a state vector $H_i = <h_i^1, \cdots, h_i^w>$. If h_i^m is a different value than h_i^{m-1}, we know that the work node associated with the *mth* component of T_i's history vector changes T_i's state in all cycles corresponding to states in H_i's partition set. We find this occurrence's equilibrium probability by examining every one of $\{H_i^n\}$'s state vectors, summing the equilibrium probabilities of those showing a difference between the $m-1st$ and *mth* components. This procedure is applied for $m = 2, \cdots, w$. It is only slightly more complicated to determine the the probability of the first possible T_i evaluation changing T_i's state. We examine each vector H_i, summing the transition probabilities from H_i to vectors H_j whose first component is different from H_i's last component. Any such transition represents a way for the first possible evaluation of T_i to change T_i's state. We multiply H_i's equilibrium state probability with the sum of these transition probabilities; we perform this analysis for each state of $\{H_i^n\}$, and sum the resulting products. This final sum is the equilibrium probability that the first possible evaluation of T_i results in a state change.

This whole procedure is based on the foreknowledge of T_i's (system graph) predecessors' behavior. To begin the construction we must consider functionals without predecessors, the generators. We assumed that a generator's cycle history vector process is an ergodic Markov chain. These processes' equilibrium state probabilities can be determined, and the probabilities of the history vector's components changing state can be found in the fashion described above. We also note that generators are always evaluated at their scheduled times, so that their evaluation probabilities are always one.

5.3. Lower Bounds on Cycle Time Moments

The previous section showed how to calculate a work node's evaluation probability and the probability of changing its functional's state. We now use these probabilities to bound the cycle execution time first and second moments from below. Our derivation depends on the assumption that every work assignment node's predecessors are probabilistically independent. We first show how these bounds are derived given this assumption, and then examine the generality of that assumption.

5.3.1. Derivation of Lower Bounds

In Chapter 4 we showed that a cycle's execution time under hyper–message synchronization is optimal over the class of non–predictive synchronization protocols; the cycle execution time moments are thus minimized under hyper–message synchronization. However, these minimized moments are too difficult to determine exactly, this determination is a generalization of finding PERT network moments [Elm67, HaW66, Mar65, RoT77]. We can bound these moments from below by assuming independence among each work assignment graph node's predecessors.

Under hyper–message synchronization, a work assignment node must wait for messages from certain predecessors. Its execution (or hyper–execution) is initiated as soon as the last of these messages arrive. The node's execution initiation time is $\max\{X_1, \cdots, X_n\}$ where the X_i are the appropriate predecessors' message arrival times. Noting that the max function is convex, we see that Jensen's inequality [MaO79] might help us bound the mean execution initiation time from below.

Lemma 5-3: (Jensen's Inequality) Let $X_1, X_2, \cdots, X_n$ be independent random variables, and let g be a convex function from R^n into R. Then

$$E[g(X_1, \cdots, X_n)] \geq g(E[X_1], \cdots, E[X_n]).$$

□

We can use this inequality to bound a work assignment graph node's mean execution initiation time from below if the node's predecessors' message arrival times are independent. We suppose that a partitioned system's work node evaluation probabilities and probabilities of changing state are known. We assume that the work assignment nodes have been topologically sorted, and will process the nodes with respect to this order. Let W be a node with predecessors $P_1, \cdots, P_k$. We will assume that $P_1, \cdots, P_I$ are W's initiation predecessors, that P_{I+1} is W's execution predecessor, and that $P_{I+2}, \cdots, P_k$ are W's logical non–initiation predecessors. Let S_W be a successor of W in the work assignment graph. We develop a lower bound on the mean arrival time of W's message to S_W.

Let $\Lambda(1, P_i, W)$ be the arrival time of P_i's message to W under hyper–message synchronization given that P_i is evaluated, and let $\Lambda(0, P_i, W)$ be the arrival time of P_i's message to W given that P_i is not evaluated. The unconditional arrival time of P_i's message to W is denoted $\Lambda(P_i, W)$. Let $\lambda(1, P_i, W)$ be a lower bound on $E[\Lambda(1, P_i, W)]$, and $\xi(1, P_i, W)$ be a lower bound on $E[\Lambda(1, P_i, W)^2]$. Let $\lambda(0, P_i, W)$ be a lower bound on $E[\Lambda(0, P_i, W)]$, and $\xi(0, P_i, W)$ be a lower bound on $E[\Lambda(0, P_i, W)^2]$. Let $\lambda(P_i, W)$ be a lower bound on $E[\Lambda(P_i, W)]$, and $\xi(P_i, W)$ be a lower bound on $E[\Lambda(P_i, W)^2]$. If W has no predecessors in the work assignment graph, we take each of these bounds to be zero. Since we are processing nodes in topological order, we may assume that these bounds have been previously determined for each of W's predecessors P_i. Define $XI(1, W)$ to be W's execution initiation time given that W is evaluated, and let $XI(0, W)$ be its execution initiation time given that it is hyper–executed. $X(W)$ denotes W's execution delay when evaluated.

We show how to derive lower bounds on the first two moments of $\Lambda(W, S_W)$, the arrival time of W's message to a successor S_W. This derivation requires the consideration of four cases, depending on whether S_W is a logical or execution successor of W, and depending on whether W is evaluated. We consider each of these four cases in turn.

W Evaluated, S_W Logical Successor

If W is evaluated and S_W is a logical successor, the arrival time of W's completion message to S_W is equal to W's execution initiation time, plus its execution delay $X(W)$, plus the communication delay $D(W, S_W)$ between W's processor and S_W's processor (note that $D(W, S_W) = 0$ if these processors are identical). Then the mean arrival time of W's message to S_W is given by

$$E[\Lambda(1, W, S_W)] = E[XI(1, W)] + X(W) + D(W, S_W). \tag{5-2}$$

We thus derive a lower bound on the arrival time of W's message to S_W by deriving a lower bound on $E[XI(1, W)]$.

W is evaluated if and only if at least one of its initiation predecessors changes state; an initiation predecessor can change state only if it is evaluated. Given that W has I initiation predecessors, there are $2^I - 1$ different ways in which each of W's initiation predecessors can either be evaluated or not, with the proviso that at least one initiation predecessor is. We let **J** be the collection of all binary vectors of length I with at least one "1" component. We can describe the particular evaluation occurrences of W's initiation predecessors with a vector $J = <j_1, \cdots, j_I> \in \mathbf{J}$: the *mth* component j_m is a 1 if initiation predecessor P_m is evaluated, otherwise j_m is zero. For a particular vector J, the probability that the evaluation occurrences described by the vector actually occur is the product

$$Prob\{J\} = \prod_{i=1}^{I} Prob\{j_i\} \tag{5-3}$$

where, when $j_i = 1$, $Prob\{j_i\}$ is the probability that predecessor P_i is evaluated; when $j_i = 0$ $Prob\{j_i\}$ is the probability that P_i is not evaluated. We are interested first in the probability that the evaluation combination described by J occurs, and that at least one initiation predecessor changes state. By the definition of conditional probability, we know that this probability is equal to the probability that at least one initiation predecessor changes state given the vector J, times $Prob\{J\}$ given above. Suppose then we are given J. The probability that at least one initiation predecessor changes state is one minus the probability that every initiation predecessor does not change state. Given that an initiation predecessor P_m is evaluated , we need to know the probability that it will not change state. We may assume that we know the **unconditional** probability that P_m changes state, and the probability that P_m is evaluated. Let Q_m be the unconditional probability of P_m changing state, and let p_m be the probability that P_m is evaluated. We know that

$$Q_m = Prob\{P_m \textit{ changes state} \mid P_m \textit{ is evaluated}\} \cdot p_m$$

$$+ Prob\{P_m \textit{ changes state} \mid P_m \textit{ is not evaluated}\} \cdot (1 - p_m).$$

The second term in this sum is zero since no functional changes state without being evaluated. Then the conditional probability that P_m does not change state, given that it is evaluated is $1 - \frac{Q_m}{p_m}$. We can now determine the probability that given J, at least one initiation predecessor changes state. Let $V(J)$ be the set of indices of J's components which are 1. Given J, the probability that at least one of the predecessors identified by $V(P)$ changes state is equal to

$$1 - \prod_{i \in V(J)} (1 - \frac{Q_i}{p_i}). \tag{5-4}$$

Then the probability that the evaluation combination described by J occurs, and at least

one evaluated predecessor changes state is given by

$$\left[1 - \prod_{i \in V(J)} (1 - \frac{Q_i}{p_i})\right] \cdot \prod_{i=1}^{I} Prob\{j_i\}. \tag{5-5}$$

We need to determine the probability that the predecessor evaluation behavior described by a vector J occurs given that at least one initiation predecessor changes state. By the definition of conditional probability, this probability is equal to the probability that J occurs **and** at least one initiation predecessor changes state (given above) divided by the unconditional probability that at least one initiation predecessor changes state. We denote this quotient by $Prob\{J_c\}$, and observe that

$$Prob\{J_c\} = \frac{\left[1 - \prod_{i \in V(J)} (1 - \frac{Q_i}{p_i})\right] \cdot \prod_{i=1}^{I} Prob\{j_i\}}{1 - \prod_{i=1}^{I}(1 - Q_i)} \tag{5-6}$$

W's execution is not initiated until W receives a message from each of its predecessors. Given that W is evaluated, and given a vector $J \in \mathbf{J}$ describing W's initiation predecessors' evaluation behavior, we have

$$XI(1, W) = \max\left\{\max_{i \leq I}\{\Lambda(j_i, P_i, W)\}, \max_{I < i \leq k}\{\Lambda(P_i, W)\}\right\}. \tag{5-7}$$

By the assumed independence of W's predecessors, we may apply Jensen's inequality to discover

$$E[XI(1, W)] \geq \max\left\{\max_{i \leq I}\{E[\Lambda(j_i, P_i, W)]\}, \max_{I < i \leq k}\{E[\Lambda(P_i, W)]\}\right\}. \tag{5-8}$$

$$\geq \max\left\{\max_{i \leq I}\{\lambda(j_i, P_i, W)\}, \max_{I < i \leq k}\{\lambda(P_i, W)\}\right\}$$

since $\lambda(j_i, P_i, W) \leq E[\Lambda(j_i, P_i, W)]$, for each $i = 1,2,\cdots,I$, and $\lambda(P_i, W) \leq E[\Lambda(P_i, W)]$ for $i = I+1,\cdots,k$.

We can use the result above to bound $E[XI(1, W)]$ from below. We condition on each possible state change occurrence combination described by the set $\mathbf{J}$:

$$E[XI(1, W)] = E_{J_c}\left[E[XI(1, W) \mid J]\right] \tag{5-9}$$

$$\geq E_{J_c}\left[\max\left\{\max_{i \leq I}\{\lambda(j_i, P_i, W)\}, \max_{I < i \leq k}\{\lambda(P_i, W)\}\right\}\right]$$

$$= \sum_{J \in \mathbf{J}} Prob\{J_c\} \cdot \left[\max\left\{\max_{i \leq I}\{\lambda(j_i, P_i, W)\}, \max_{I < i \leq k}\{\lambda(P_i, W)\}\right\}\right]$$

where E_{J_c} is the expectation operator with respect to the distribution of J conditioned on at least one of W's predecessors changing state. It follows from relation 5–2 that the mean arrival time of W's message to S_W is bounded from below by

$$\lambda(1, W, S_W) = \sum_{J \in \mathbf{J}} Prob\{J_c\} \cdot \left| \max\left\{ \max_{i \leqslant I}\{\lambda(j_i, P_i, W)\}, \max_{I < i \leqslant k}\{\lambda(P_i, W)\} \right\} \right| \tag{5-10}$$

$$+ X(W) + D(W, S_W).$$

We now consider the second moment of $\Lambda(1, W, S_W)$. We have

$$E[\Lambda(1,W,S_W)^2] = \tag{5-11}$$

$$E[XI(1,W)^2] + 2 \cdot E[XI(1,W)] \cdot (X(W)+D(W, S_W)) + (X(W)+D(W, S_W))^2.$$

We have already bounded $E[XI(1, W)]$ from below, we next bound $E[XI(1, W)^2]$ from below in much the same manner. Given a vector J describing W's predecessors' evaluation behavior, an immediate parallel to equation 5–7 is given by

$$XI(1, W)^2 = \max\left\{ \max_{i \leqslant I}\{\Lambda(j_i, P_i, W)\}, \max_{I < i \leqslant k}\{\Lambda(P_i, W)\} \right\}^2 \tag{5-12}$$

$$= \max\left\{ \max_{i \leqslant I}\{\Lambda(j_i, P_i, W)^2\}, \max_{I < i \leqslant k}\{\Lambda(P_i, W)^2\} \right\}.$$

where the second equality follows from the fact that all quantities involved are positive. It follows that

$$E[XI(1, W)^2] \geqslant \max\left\{ \max_{i \leqslant I}\{E[\Lambda(j_i, P_i, W)]^2\}, \max_{I < i \leqslant k}\{E[\Lambda(P_i, W)]^2\} \right\} \tag{5-13}$$

$$\geqslant \max\left\{ \max_{i \leqslant I}\{\xi(j_i, P_i, W)\}, \max_{I < i \leqslant k}\{\xi(P_i, W)\} \right\}$$

where the first inequality follows from Jensen's inequality, and the second inequality follows from the use of known lower bounds. A lower bound on $E[\Lambda(1, W, S_W)^2]$ is derived as before by conditioning on the evaluation behavior of W's initiation predecessors, and by using relation 5–11 and the lower bound $\lambda(1, W, S_W)$.

$$\xi(1, W, S_W) = \sum_{J \in \mathbf{J}} Prob\{J_c\} \cdot \left| \max\left\{ \max_{i \leqslant I}\{\xi(j_i, P_i, W)\}, \max_{I < i \leqslant k}\{\xi(P_i, W)\} \right\} \right| \tag{5-14}$$

$$+ 2 \cdot \lambda(1, W, S_W) \cdot (X(W) + D(W, S_W)) + (X(W) + D(W, S_W))^2.$$

W Hyper-Executed, S_W Logical Successor

W's behavior is somewhat different when it is hyper-executed. The arrival time of W's message at S_W is equal to W's execution initiation time, as W's hyper-execution and its hyper-message suffer no delay. W is evaluated as soon as the last of its no-change messages is received from initiation predecessors. Given a vector J describing the pattern of W's initiation predecessors' evaluations, we have

$$XI(0, W) = \max_{i \leqslant I}\{\Lambda(j_i, P_i, W)\}. \tag{5-15}$$

Jensen's inequality implies that

$$E[XI(0, W)] \geqslant \max_{i \leqslant I}\{E[\Lambda(j_i, P_i, W)]\} \tag{5-16}$$

$$\geqslant \max_{i \leqslant I}\{\lambda(j_i, P_i, W)\}.$$

We need to calculate the probability that the vector J occurs given that none of W's predecessors changed state. We know that

$$Prob\{J\} = Prob\{J \mid W \text{ is evaluated}\}\cdot Prob\{W \text{ is evaluated}\}$$

$$+ Prob\{J \mid W \text{ is not evaluated}\}\cdot Prob\{W \text{ is not evaluated}\}.$$

We have already calculated $Prob\{J\}$ (equation 5-3), $Prob\{W \text{ is evaluated}\}$ (equation 5-1), and $Prob\{J \mid W \text{ is evaluated}\} = Prob\{J_c\}$ (equation 5-6) so that we can solve for the probability of interest, denoted $Prob\{P_{\bar{c}}\}$. Then letting p_W denote the probability that W is evaluated, we have

$$Prob\{J_{\bar{c}}\} = \frac{Prob\{J\} - Prob\{J_c\}\cdot p_W}{(1 - p_W)}. \tag{5-17}$$

Recalling that the set **J** excluded the zero vector, we denote the set of all binary vectors with I components by **J**$\cup$**0**. Just as before, we condition on the evaluation behavior of W's initiation predecessors, and define

$$\lambda(0, W, S_W) = \sum_{J \in \mathbf{J}\cup\mathbf{0}} Prob\{J_{\bar{c}}\}\cdot\max_{i \leqslant I}\{\lambda(j_i, P_i, W)\}. \tag{5-18}$$

By the same argument that established $\lambda(1, W, S_W)$ as a lower bound on $E[\Lambda(1, W, S_W)]$, we see that $\lambda(0, W, S_W)$ bounds from below the mean arrival time of W's message to S_W given that W is not evaluated.

An entirely similar derivation shows that a lower bound on the second moment $E[XI(0, W)^2]$ is given by

$$\xi(0, W, S_W) = \sum_{J \in \mathbf{J}\cup\mathbf{0}} Prob\{J_{\bar{c}}\}\cdot\max_{i \leqslant I}\{\xi(j_i, P_i, W)\}. \tag{5-19}$$

W Overall Moments, S_W Logical Successor

The previous two sections derive bounds on the moments of the arrival time of W's completion message to S_W, conditioned on whether W is evaluated. To get bounds on W's unconditional message arrival time at S_W we appeal to the law of conditional expectation which states that

$$E[\Lambda(W, S_W)^n] = p_W\cdot E[\Lambda(1, W, S_W)^n] + (1 - p_W)\cdot E[\Lambda(0, W, S_W)^n]$$

where p_W is the probability that W is evaluated. p_W can be calculated as described by equation 5–1. Applying the results of the previous subsections, we define

$$\lambda(W, S_W) = p_W \cdot \lambda(1, W, S_W) + (1 - p_W) \cdot \lambda(0, W, S_W) \tag{5–20}$$

and

$$\xi(W, S_W) = p_W \cdot \xi(1, W, S_W) + (1 - p_W) \cdot \xi(0, W, S_W); \tag{5–21}$$

we observe that $E[\Lambda(W, S_W)] \geq \lambda(W, S_W)$ and $E[\Lambda(W, S_W)^2] \geq \xi(W, S_W)$.

W Evaluated, S_e Execution Successor

The case where W is evaluated and S_e is its execution successor is essentially the same as when S_W is a logical successor. W dispatches a release message simultaneously with its completion messages. The arrival time of W's message to its execution successor is the same as the arrival time of its completion message to a logical successor in the same processor. Thus we have the lower bounds

$$\lambda(1, W, S_e) \tag{5–22}$$

$$= \sum_{J \in \mathbf{J}} Prob\{J_c\} \cdot \left| \max \left\{ \max_{i \leq I} \{\lambda(j_i, P_i, W)\}, \max_{I < i \leq k} \{\lambda(P_i, W)\} \right\} \right| + X(W)$$

and

$$\xi(1, W, S_e) \tag{5–23}$$

$$= \sum_{J \in \mathbf{J}} Prob\{J_c\} \cdot \left| \max \left\{ \max_{i \leq I} \{\xi(j_i, P_i, W)\}, \max_{I < i \leq k} \{\xi(P_i, W)\} \right\} \right|$$

$$+ 2 \cdot \lambda(0, W, S_e) \cdot X(W) + X(W)^2.$$

W Hyper–Executed, S_e Execution Successor

We now consider W's behavior with respect to its execution successor S_e when W is hyper–executed. W may dispatch its completion messages before it dispatches its release message; however, its release message is **never** sent before its completion message goes out. W sends the release message as soon as it receives the last of the messages from its initiation and execution predecessors $P_1, \cdots P_{I+1}$. The release message arrives at the same time as it is sent. Then given a vector J describing W's initiation predecessors' evaluation behavior, we have

$$\Lambda(0, W, S_e) = \max \left| \max_{i \leq I} \{\Lambda(j_i, P_i, W)\}, \Lambda(P_{I+1}, W) \right|. \tag{5–24}$$

Then by Jensen's inequality,

$$E\left|\max\left\{\max_{i\leqslant I}\{\Lambda(j_i, P_i, W)\}, \Lambda(P_{I+1}, W)\right\}\right| \tag{5-25}$$

$$\geqslant \max\left\{\max_{i\leqslant I}\{E[\Lambda(j_i, P_i, W)]\}, E[\Lambda(P_{I+1}, W)]\right\}.$$

$$\geqslant \max\left\{\max_{i\leqslant I}\{\lambda(j_i, P_i, W)\}, \lambda(P_{I+1}, W)\right\}.$$

Then we define

$$\lambda(0, W, S_e) \tag{5-26}$$

$$= \sum_{J\,\in\,\mathbf{J}\cup O} Prob\{J_{\bar{c}}\}\cdot\max\left\{\max_{i\leqslant I}\{\lambda(j_i, P_i, W)\}, \lambda(P_{I+1}, W)\right\}$$

and observe as before that $\lambda(0, W, S_e)$ bounds from below the mean arrival time of W's message to S_e. Similarly, a lower bound on $E[\Lambda(0, W, S_e)^2]$ is given by

$$\xi(0, W, S_e) \tag{5-27}$$

$$= \sum_{J\,\in\,\mathbf{J}\cup O} Prob\{J_{\bar{c}}\}\cdot\max\left\{\max_{i\leqslant I}\{\xi(j_i, P_i, W)\}, \xi(P_{I+1}, W)\right\}$$

W Overall Moments, S_e Execution Successor

Using reasoning entirely similar to that leading to the overall bounds for a logical successor, a lower bound on W's mean message arrival time at its execution successor is given by

$$\lambda(W, S_e) = p_W\cdot\lambda(1, W, S_e) + (1 - p_W)\cdot\lambda(0, W, S_e).$$

Likewise, a bound on the unconditional second moment is given by

$$\xi(W, S_e) = p_W\cdot\xi(1, W, S_e) + (1 - p_W)\cdot\xi(0, W, S_e).$$

Lower Bounds on Cycle Execution Time Moments

The sections above show how to bound the arrival time moments of a work node's message to a successor. The cycle execution time moment bounds are derived from the sink node's predecessors' termination messages moment bounds. Recall that a termination message is sent when a release message would be sent. Let $S_1, \cdots, S_p$ be the sink node's termination predecessors. Then a lower bound on the mean cycle execution time is given by

$$\max_{i\leqslant p}\{\lambda(S_i, P_S)\}$$

where $\lambda(S_i, P_S)$ is derived under the assumption that P_S is the execution successor of S_i. A lower bound on the second moment of the cycle execution time is given

$$\max_{i \leq p} \{ \xi(S_i, P_S) \}$$

where $\xi(S_i, P_S)$ too is derived under the assumption that P_S is the execution successor of S_i.

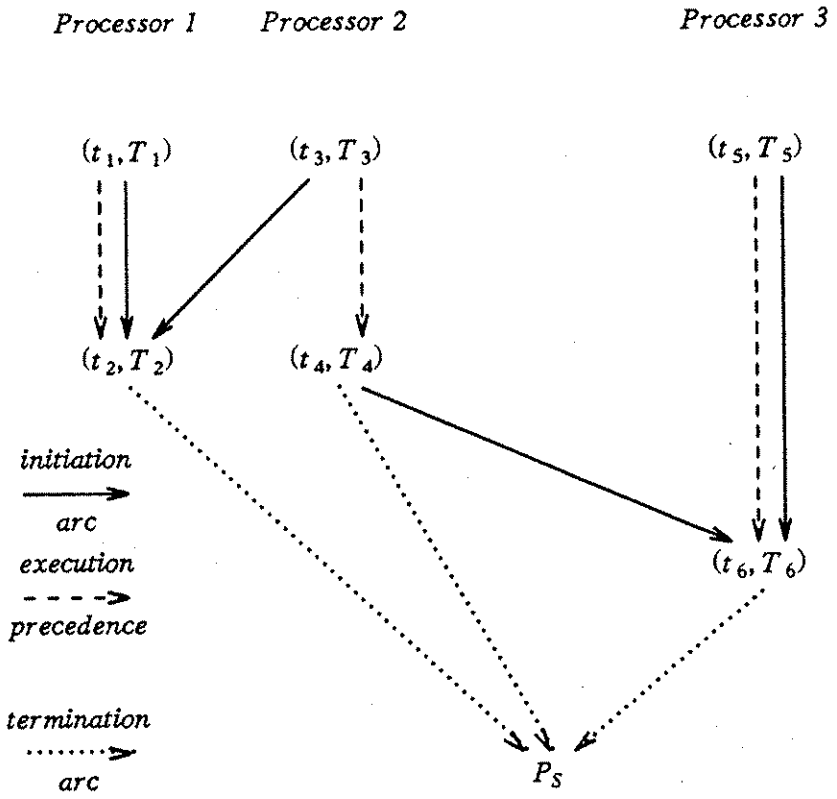
The more usual second order moment is the second central moment, or variance. If CX denotes the cycle execution time, then $Var[CX] = E[CX^2] - E[CX]^2$. We have only lower bounds on $E[CX^2]$ and $E[CX]^2$ so that we cannot make a quantitative statement about $Var[CX]$. We could however use these bounds and the variance formula to estimate the true variance (provided that this estimate is positive). The square root of this variance estimation estimates the standard deviation of the cycle execution time.

5.4. Generality of Assumptions

The derivation of lower bounds assumed that every work assignment node's predecessors behaved independently. In this section we briefly consider the generality of this assumption. We show by example that such work assignment graphs exist, but then suggest that this assumption of independence is rarely valid in practice. Despite this, we argue that there is some use for the bounds produced by simply assuming this independence, even when this assumption is unsupportable.

Consider the work assignment graph illustrated in figure 5–1. We show that each of a node predecessors' behave independently. We first observe that work nodes (t_1, T_1), (t_3, T_3), (t_4, T_4) and (t_5, T_5) have no initiation predecessors, and so represent generator event evaluations (which are evaluated every cycle). Since (t_1, T_1) is always evaluated, the arrival times of its release and its completion messages to (t_2, T_2) are identical, constant, and therefore probabilistically independent. Similarly, the arrival time of (t_3, T_3)'s completion message at (t_2, T_2) is constant. It follows that the arrival times of (t_2, T_2)'s predecessors' messages are probabilistically independent. For identical reasons, the arrival times of (t_6, T_6)'s predecessors' messages are all constant, and are thus probabilistically independent. Finally, we show that the arrival times of the terminal messages at P_S are all independent. We observe that the execution initiation times of both (t_2, T_2) and (t_6, T_6) are constant. Whether (t_2, T_2) is evaluated depends entirely on whether (t_1, T_1) or (t_3, T_3) changes state. Likewise, whether (t_6, T_6) is evaluated depends entirely on whether (t_4, T_4) or (t_5, T_5) changes state. Assuming that all generators' transitions are independent of each other, we conclude that (t_2, T_2)'s evaluation result is independent of (t_6, T_6)'s evaluation result. Given that their respective execution initiation times are constant, we conclude that the arrival time of their respective termination messages at P_S are independent. The arrival time of (t_4, T_4)'s termination message at P_S is seen to be constant; whence the arrival times of termination messages at P_S are all probabilistically independent.

The assumption of independence among a node's predecessors is more restrictive than the example above might suggest. For example, suppose that work node W represents a normal functional's evaluation, and that W is both a logical and an execution predecessor of work node V. Suppose also that it is possible for W to be either evaluated or hyper-executed, and that it is possible that W's incoming release message (when W is hyper-executed) arrives either before or after W's hyper-execution. W's execution initiation time is thus variable, so that the arrival times of W's completion and release messages at V are both variable. However, these two arrival times are not independent: they are correlated on W's (variable) execution completion time and the (variable) arrival time at W of its incoming release message. As we will see in our chapter on partitioning (Chapter 8), we will want to construct partitions where a logical successor is placed in the same processor as one of its logical predecessors. Thus the assumption of predecessor independence is generally not valid. However, our bounds derivation may still be useful. The relation expressed by Jensen's inequality may be satisfied even when the arrival times are



Work Assignment Graph Preserving Independence

Figure 5–1

correlated; we just can't prove that this relation holds. Nevertheless, we expect the numbers to give us some "order of magnitude" information about the cycle execution time moments; this information can be determined statically from the description of the simulation. While we should not trust this information's accuracy, there is a place for this information in a Bayesian moment estimation context. We explore this idea further in the following chapter.

5.5. Chapter Summary

Our investigation of a simulation's run-time behavior shows that a work node may or may not be evaluated during a cycle. We are thus led to consider a work node's **evaluation probability**. In this chapter we showed how we can analyze a static description of a simulation and its inputs to determine each work node's evaluation probability. We showed that this probability's exact derivation is possible only under a set of restricting assumptions; the system graph should be an in-tree, the input processes and initialization of functional states should preserve probabilistic independence. Even under these assumptions the computation may be intractable due to its exponential complexity. We also observed that the calculation might be used to estimate these probabilities even if these restrictions are violated. A partitioning algorithm should consider how frequently an event functional is evaluated as well as how long it takes to evaluate it; our evaluation probability calculation algorithm is thus an important tool for intelligent partitioning algorithms.

A natural way to measure the performance of a given partition is to calculate the cycle execution time distribution's moments. These moments are a function of the execution work required intrinsically by the simulation system, the partition chosen, and the synchronization protocol. We have shown how to calculate lower bounds on the cycle execution time mean and second moment, recognizing that (i) the probabilities of work node evaluation represent the simulation's intrinsic behavior; (ii) the chosen partition can be represented by the work assignment graph; (iii) we can model synchronization behavior with the hyper-message protocol given in Chapter 4. Our derivation assumed probabilistic independence among each work node's work assignment predecessors. We showed by example that this assumption is not vacuous, but then showed why most partitioned simulations will not support this independence. Finally, we argued that our bounds calculation procedure may be useful in a general setting, as it gives us a statically determinable estimate of the cycle execution moments. We will find these estimates quite useful in the following chapter on statistical cycle execution moment estimation.

Chapter 6

Statistical Analysis

6.1. Chapter Overview

Variant simulation run-time behavior is one of the critical features distinguishing the problem of partitioning a simulation from other partitioning problems. We must evaluate a partition's performance by looking at its resulting cycle execution time moments. In Chapter 5 we derived lower bounds on these moments from a static description of the simulation and its inputs. We found that the mathematical soundness of these bounds rests by necessity on assumptions which are often unsatisfied in practice. In view of this difficulty, a **statistical** estimation of a partition's cycle execution time moments is appropriate. We propose to **observe** the running simulation's behavior, and use these observations to estimate the cycle execution time moments. Given statistical moment estimates for two different simulation partitions we can decide which is most effective.

This chapter is divided into three sections. The first section gives an overview of problems encountered in statistical moment estimation. The second section examines statistical moment estimation in a Bayesian framework. We justify the distributional assumptions of this framework, and argue that the analytically derived lower bounds can be used to construct the Bayesian "prior" distributions. The third section examines a decision theoretic comparison of two partitions. We then show how the most effective partition is readily identified.

The discussions in this chapter do not advance the study of statistics. This chapter's main contribution is the demonstration that formal statistical analysis can be applied to a partition's performance evaluation, and to the comparison of two partitions. This demonstration enhances our ability to analyze a simulation for the purposes of partitioning.

6.2. Statistical Moment Estimation

In Chapter 3 we argued that a cycle's behavior is random and that our evaluation of a partition should be based on its average performance over **all** possible cycles. This observation led to the investigation of analytical bounds on the cycle execution time moments. We discovered that an exact calculation of these bounds is not generally tractable and requires restrictive assumptions. We now explore the possibility of estimating cycle execution time moments statistically. A statistical approach raises certain issues we need to resolve.

Statistical moment estimation implicitly assumes there is an underlying cycle execution time probability distribution. We first observe that the system cycle Markov chain model of Chapter 3 provides the basis for this assumption. We recall that the states of the system cycle Markov chain $\{\Theta^n\}$ encode all normal functionals' states at the beginning of the cycle, and all inputs presented to the simulation during the cycle; this information suffices for the derivation of that cycle's execution time under hyper-message synchronization. The process $\{\Theta^n\}$ eventually becomes ergodic, and advances into equilibrium. Once in equilibrium, if we observe a cycle at random there is a probability Q_c of observing state c of $\{\Theta^n\}$; Q_c is state c's equilibrium state probability. If we **independently** observe

another cycle, the probability of observing state c is again Q_c. So long as the cycle observations are independent the probability distribution underlying these observations is $\{\Theta^n\}$'s equilibrium state probability distribution. Since the cycle execution time distribution is completely determined by $\{\Theta^n\}$, it follows that there is an underlying common probability distribution for independently observed cycle execution times.

We could conceivably calculate a partition's mean cycle execution time from $\{\Theta^n\}$'s equilibrium cycle state probabilities and the hyper-execution graph technique of Chapter 4. We let $F(WG,j,c)$ be the length of the longest path though the hyper-execution graph $HEG(WG,j,c)$ defined by work graph WG, partition Ψ_j and cycle c. Then if we assume that $\{\Theta^n\}$ has N states $1, 2, \cdots, N$, the mean cycle time (under hyper-message synchronization) is simply $\sum_{c=1}^{N} Q_c \cdot F(WG,j,c)$. However, $\{\Theta^n\}$'s state space is potentially huge. It is much more reasonable to statistically estimate the cycle execution time moments. Standard statistical practice and theory require that we have a number of *independent* cycle execution time observations; however, successive states assumed by $\{\Theta^n\}$ are correlated. Simulation practitioners face the same problem in the analysis of correlated simulation output data. The two major ways of achieving independent observations are the method of *independent replications* [Law84] , and the method of ***batch means*** [Fis78] .

The method of independent replications uses samples from independently generated runs of the simulation. Each restarted simulation may not be accomplishing any further useful work; the additional simulation computation exists solely to ensure the probabilistic independence of the cycles sampled from different runs. A batch means treatment transforms a sequence of observations from a single run into a shorter sequence of approximately independent and normally distributed observations. It partitions the observation sequence $X_1, X_2, \cdots, X_j, \cdots$ into adjacent groups, each with n observations. The batch mean observation for group $X_{jn+1}, \cdots, X_{(j+1)n}$ is simply the sample mean $\frac{1}{n}\sum_{i=1}^{n} X_{jn+i}$. A batch means treatment is more consistent with our goal of high performance than is independent replications: useful work can be accomplished while the data is being gathered for transformation. A batch means transformation changes the distribution of the data we deal with. If the orginal distribution's mean and variance is μ and σ^2, the transformed distribution's mean and variance are μ and $\frac{\sigma^2}{n}$.

The approximate normality and independence of batch mean observations makes this method an attractive option; many statistical tests are based on the assumptions of independent, normally distributed observations. Henceforth we will assume that a "cycle execution time" is really a batch means observation, normally distributed with mean μ and variance $\frac{\sigma^2}{n}$. Our estimation of "cycle execution time moments" is really the estimation of μ and $\frac{\sigma^2}{n}$. To estimate the moments of the underlying cycle execution time distribution, we will estimate the *parameters* of this normal distribution. Mapping back to the true moments is trivially accomplished.

Given a set of independent cycle execution time observations, we can use standard classical techniques [HoT83] to estimate the mean and variance. These techniques use only the observed data to construct their estimates; but there is more information available about the system than just the data, e.g. the system graph. A Bayesian estimation of distribution parameters may then be appropriate. In the following section, we consider a Bayesian estimation of the cycle execution time parameters.

6.3. Bayesian Parameter Estimation

A Bayesian parameter estimation associates "belief" distributions with the parameters mean and standard deviation. Loosely speaking, a belief distribution places probability weights on potential values of a parameter. These weights reflect our beliefs about which value the true parameter is likely to have. The Bayesian approach assumes the existence of one "prior" belief distribution for the mean, and one for the standard deviation. These prior distributions encode what we know about the system before we collect cycle execution time observations. The prior distributions and a set of independent observations are combined using Bayes' Theorem [Sch69], yielding "posterior" belief distributions for the parameters. The posterior distributions describe how the additional information provided by the observations update what we (ought to) believe about the true values of the cycle execution time parameters. In contrast with classical point estimates, a Bayesian approach estimates cycle execution parameters with probability distributions.

A tractable Bayesian treatment requires certain assumptions about the data and parameter distributions. In this section we argue that these assumptions can be justified, and then examine construction of the prior distributions.

6.3.1. Distributional Assumptions

A full treatment of a Bayesian approach's computational details is given in [NoJ74](chapter 7), and is not appropriate here. However, we should justify the treatment's assumptions.

The Bayesian technique assumes that the data is normally distributed; data derived from a batch means transformation is approximately normal. The cycle execution time mean μ and standard deviation σ distributions are assumed to have certain forms so that the application of Bayes' theorem is tractable. The mean distribution is based on a Student's t distribution and is symmetric and uni-modal. The standard deviation distribution is based on a χ^2 distribution; it too is unimodal, and has a relatively high variance. Uni-modality for our parameter belief distributions does not violate anything we know about the cycle execution time, and so is a reasonable assumption. Our limited simulation experience with logic networks has shown the cycle execution mean to be relatively stable from run to run, while the cycle standard deviation is somewhat more variable. These observations correspond well with the symmetric nature of the mean distribution, and the relatively high variance of the standard deviation distribution. Thus the necessary distributional assumptions can be justified.

The final assumption of the Bayesian treatment presumes that the marginal prior distributions are independent. This assumption is harder to justify. Any non-trivial prior distributions we derive will be based on static information about the simulation system, and will at best be approximate. If an oracle gave us the exact value of one parameter, our understanding of the other parameter is not affected. Since the change in one parameter's value does not affect our understanding of the other's, we conclude that our initial understandings are independent.

Thus we see that the assumptions required by a Bayesian treatment can be heuristically justified. We turn next to the determination of the prior distributions.

6.3.2. Prior Distributions

One of the attractive features of a Bayesian parameter estimation approach is its ability to incorporate "prior" information about the parameters. Determining the prior distributions is a fundamental problem in any Bayesian treatment. In Chapter 5 we developed lower bounds on the cycle execution time parameters. These bounds are determined statically, but require restrictive assumptions. We can use these techniques to obtain approximations when the assumptions are not satisfied. These approximations

should yield some (inexact) information about the system, and can be used to create prior distributions.

We first reconsider the derivation of each work node's evaluation probability. This derivation depended on the node's predecessors' independent behavior. This assumption will not be satisfied in most simulations. To test this assumption's robustness, we performed some experiments on small, randomly generated logic networks to determine the correlation in a node's predecessors' behavior. The results were very encouraging: significant correlation between two gates evaluation behavior was observed **only** when those gates shared an input. Of course, this observation should be treated with some skepticism as the networks studied were randomly generated. Nevertheless, these observations do provide some hope that the assumption of independent behavior is not unreasonable.

The derivation of lower bounds on the cycle execution time moments requires independence among all of a node's work assignment predecessors; we may simply assume independence in the general case. We can use the first moment bound as an estimator of the mean, and the second moment minus the first moment squared as an estimator of the variance. This variance estimator is then divided by n to reflect our batch means treatment of the cycle execution time observations. The square root of this variance estimator estimates the standard deviation (of the batch means cycle execution time data).

The prior distribution for the mean μ is assumed to be distributed such that $\frac{\nu_c^{1/2}\cdot(\mu-\zeta)}{\kappa^{1/2}}$ has a t distribution with ν_c degrees of freedom; ν_c, ζ and κ are the parameters of this distribution. Following [NoJ74], we refer to this distribution as a $t(\nu_c, \zeta, \kappa)$ distribution. This distribution is uni-modal; its mean, median and mode are all equal to ζ. Its variance (when it exists) is equal to $\frac{\kappa}{\nu_c - 2}$. The (batch means) cycle execution time standard deviation $\bar{\sigma}$ is assumed to distributed such that $\frac{\lambda}{\bar{\sigma}^2}$ is a χ^2 random variable with ν_s degrees of freedom. We will say that the standard deviation $\bar{\sigma}$ has a $\chi^{-1}(\nu_s, \lambda)$ distribution. The mean of such a random variable is (approximately)

$$\left(\frac{\lambda}{\nu_s - \frac{2}{3}}\right)^{1/2};$$

the mode is

$$\left(\frac{\lambda}{\nu_s + 1}\right)^{1/2};$$

and the variance is approximately

$$\frac{\lambda}{2\cdot(\nu_s - 2)\cdot(\nu_s - \frac{5}{3})}.$$

The $t(\nu_c, \zeta, \kappa)$ and $\chi^{-1}(\nu_s, \lambda)$ distributions are completely specified by their respective parameters ν_c, ζ, κ and ν_s, λ; prior construction requires quantification of these parameters. We briefly sketch the role each parameter plays in shaping the cycle execution mean distribution. ν_c is a "degrees of freedom" parameter. Standard t distributions have only one parameter, the degrees of freedom, and can be constructed by taking the sum of certain normalized random variables. The number of terms in this sum defines the standard t's degrees of freedom. [NoJ74] argues that in a Bayesian prior distribution, the parameter ν_c can be thought of as the number of observations our prior information is "worth". The larger the value of ν_c, the more weight is given to the prior distribution

when combined with observations in Bayes' Theorem. The parameter ζ plays the role of an additive offset: standard t distributions are symmetric and centered at 0; $t(\nu_c, \zeta, \kappa)$ is centered at ζ. Finally, ν_c and κ together form a scale factor whose effects are seen in the variance: $\frac{\kappa}{\nu_c - 2}$. The variance for a standard t with ν_c degrees of freedom is $\frac{\nu_c}{\nu_c - 2}$. An intuitive incorporation of our analytically derived estimations is to assume that $t(\nu_c, \zeta, \kappa)$ is centered at our mean estimate, ie., set ζ equal to our derived mean. We still must determine ν_c and κ. [NoJ74] discusses guidelines in terms of credibility intervals. ν_c's and κ's selection defines the spread of the distribution; the wider the spread, the less weight will be placed on the prior information.

We similarly determine the standard deviation prior distribution. The parameter ν_s in $\chi^{-1}(\nu_s, \lambda)$ identifies degrees of freedom, and λ is a scale factor. We can equate $\chi^{-1}(\nu_s, \lambda)$'s mode (point of maximum likelihood) $\left[\frac{\lambda}{\nu_s + 1}\right]^{1/2}$ with our analytically determined standard deviation estimate; this again centers the distribution. The spread of the distribution is determined by a selection of ν_s and λ subject to the centering constraint. Again, these values might be determined using [NoJ74]'s guidelines.

In addition to [NoJ74]'s guidelines we suggest that our analytically derived estimates' accuracy be considered. Uncertainty in our point estimates arises from assumption violations and the fact that the point estimates are constructed to be lower bounds. For example, we consider how a disparity between the lower bound and the true parameter value can occur as a result of applying Jensen's inequality. Suppose that $X_1, \cdots, X_n$ are independent random variables. Jensen's inequality states that

$$E[\max\{X_1, \cdots, X_n\}] \geq \max\{E[X_1], \cdots, E[X_n]\}$$

If the $E[X_i]$ are all equal, the difference between the left hand side and right hand side of this inequality will tend to be significant: the likelihood that one of the X_i exceeds its mean is fairly large. On the other hand, if one particular $E[X_j]$ is substantially larger than all the rest, that X_j will almost always define the maximum, so that the difference between $E[\max\{X_1, \cdots, X_n\}]$ and $\max\{E[X_1], \cdots, E[X_n]\}$ will tend to be small. As the calculation of the bounds proceeds, we could compare the magnitudes of the $E[X_i]$ to get a sense of how accurate the substitution of $\max\{E[X_1], \cdots, E[X_n]\}$ for $E[\max\{X_1, \cdots, X_n\}]$ is, and define our priors' variance accordingly.

We have not developed quantitative techniques for defining the priors' variance; this problem is worthy of future attention. In any event, the uninformed priors [NoJ74] can be used. An uninformed prior assumes that **any** value for the parameter is as likely as another, and so does not encode any information about the system. As we will see in section 6.4, we may still benefit from a Bayesian estimation of the cycle execution time parameters even if the uninformed priors are used.

6.3.3. Section Summary

We summarize our thinking about Bayesian parameter estimation. A Bayesian approach is particularly useful when we have "prior" information about the parameters. A Bayesian approach encodes this information with a "prior" distribution. A collection of independent observations is combined with a prior distribution by means of Bayes' theorem to produce "posterior" distribution for a parameter. This application of Bayes' theorem requires certain distributional assumptions: we have justified those assumptions. We have also considered the construction of prior parameter distributions, and have argued that our probabilistic techniques developed in Chapter 5 can be utilized.

The Bayesian approach may seem overly elaborate for the simple estimation of cycle execution parameters. While the mathematics behind the approach is elaborate, the computational details (given prior distributions) are quite simple. It is not necessary to expend any great energy constructing prior distributions, the so-called "uninformed" priors [NoJ74] work very easily into the model. The Bayesian framework for parameter estimation is very useful when we compare two different partitions of the same system. As we will next see, the identification of the "best" partition can be placed in a useful decision theoretic framework supported by the Bayesian model.

6.4. Bayesian Partition Decision

We now consider the comparison of two partitions Ψ_1 and Ψ_2. We suppose there are posterior parameter belief distributions associated with both partitions' cycle execution time parameters. We would like to decide which of the two partitions is more effective. We state this problem in Bayesian decision theoretic terms, and then show how the posterior distributions are used to identify the partition with minimized expected "cost".

Decision theory considers the choice of some action on the basis of incomplete knowledge of the true, unknown, world state. The action taken exacts some cost whose value depends on the chosen action and on the true world state. As applied to our problem, the true world state consists of the true values of the cycle execution time parameters; the action is choosing one of two partitions. We have flexibility in defining an action's cost. We will consider several appropriate cost functions.

The simpliest cost of choosing Ψ_j is just Ψ_j's true world cycle execution time mean μ_j. This cost function is appropriate for selecting the partition yielding the lowest mean cycle execution time. A useful variant is the sum $\mu_j + w \cdot \bar{\sigma}_j$ of the chosen partition's true mean and its weighted standard deviation. We use this cost if we are concerned about the mean cycle execution time, but are willing to accept a slightly higher mean in exchange for much reduced variation.

More sophisticated cost functions reflect a chosen partition's use. For example, the system may be running with partition Ψ_1, and considers adopting Ψ_2. The choice to run under Ψ_2 requires us to first stop the system and perform the reassignment. Will this change decrease the system's overall finishing time? That depends on how much longer the system will run. We suppose that the system will run for M more *batch cycles*, a batch cycle being the n system cycles defining a batch observation. The total remaining system execution time under partition Ψ_j is the sum of $M \cdot n$ cycle execution times. If we let $CX_j(1), \cdots, CX_j(M \cdot n)$ denote the remaining $M \cdot n$ cycle execution times under Ψ_j, and let $BX_j(1), \cdots, BX_j(M)$ be the M corresponding batch means, we readily see that

$$\sum_{i=1}^{M \cdot n} CX_j(i) = n \cdot \sum_{i=1}^{M} BX_j(i).$$

The remaining system execution time under Ψ_j is the random variable $n \cdot \sum_{i=1}^{M} BX_j(i)$. The $BX_j(i)$ are approximately independent and normally distributed with mean μ_j and variance $\bar{\sigma}_j^2$, where μ_j is the true cycle execution time mean under Ψ_j, σ^2 is the true cycle execution time variance under Ψ_j, and $\bar{\sigma}_j^2 = \frac{\sigma_j^2}{n}$. Sums of independent normally distributed random variables are themselves normally distributed; the remaining system execution time under Ψ_j is thus normally distributed with mean $M \cdot n \cdot \mu_j$ and standard deviation $\sqrt{M \cdot n} \cdot \bar{\sigma}_j$.

The partition should be chosen on the basis of its performance over the remaining system cycles. As before, we may wish to measure this performance in terms of the mean

remaining system execution time or in terms of the mean and standard deviation of the remaining system execution time. There is an additional overhead cost of changing from one partition to another: if we are running under Ψ_1 and choose Ψ_2, we suffer an overhead delay U to actually perform the change in partitions.

Consideration of the partition replacement cost function raises an interesting issue. If the system is running under partition Ψ_1, how can we obtain the necessary Ψ_2 cycle execution time observations? While it is possible to measure Ψ_1's performance, it would seem that we can only estimate Ψ_2's. It is desirable that the partitions' performance be measured using the same method. In Chapter 4 we showed how an observed cycle's execution time under hyper–message synchronization can be calculated. In order to compare Ψ_1 and Ψ_2 on the same footing, we suggest that the cycle execution times under both partitions be estimated using hyper–message synchronization. Even though measurements of Ψ_1's performance are potentially available, estimated measurements are more meaningfully compared to Ψ_2's estimated measurements. We must then hope that these estimates reflect the relative effectiveness of Ψ_1 and Ψ_2.

We now show how the partition with the minimal expected cost of the two choices is selected. A Bayesian decision theoretic approach uses the posterior parameter distributions as probability distributions of the true world state: the true cycle execution time mean and standard deviation. The cost of a decision depends on the true world state; if the cost function is reasonably simple and the posterior distributions are known, we can then conceivably calculate the **expected** cost of a decision by taking the expectation of the cost function with respect to the posterior parameter distributions. The optimal decision is the one with minimized expected cost.

We show by example that the expected decision costs for our problem are easily found. Consider the decision to replace Ψ_1 with Ψ_2 in the remaining $M \cdot n$ system cycles. Assuming the $M \cdot n \cdot \mu_2$ cost function, this decision's expected cost is

$$E[cost\ of\ \ \Psi_2] = \int_0^\infty Prob\{\mu\} \cdot (U + M \cdot n \cdot \mu)\ d\mu$$

$$= U + M \cdot n \cdot E[\mu_2]$$

where $Prob\{\mu\}$ denotes the posterior density function value associated with the possibility that the world state of μ_2 is μ. $E[\mu_2]$ is the mean of μ_2's posterior distribution. Similarly, the cost of retaining Ψ_1 for the remaining M cycles is $M \cdot n \cdot E[\mu_1]$. Both of these expected decision costs are trivially calculated, as $E[\mu_1]$ and $E[\mu_2]$ are easily calculated (again, see [NoJ74] for details). The decision yielding the lowest expected cost is thus readily identified. We observe that it is possible for $E[\mu_2] < E[\mu_1]$, and still have $U + M \cdot n \cdot E[\mu_2] > M \cdot n \cdot E[\mu_1]$; the per cycle speedup available by using Ψ_2 may not be enough to overcome the overhead cost U suffered by changing the partition.

The other proposed cost functions share this simplicity of identifying the best partition with respect to the chosen cost function.

6.5. Chapter Summary

In this thesis we continually observe that we must consider a partition's *average* performance over all simulation cycles. In general, we cannot predict a proposed partition's performance with a static analysis. This chapter proposes a statistical estimation of a partition's performance. In doing so, it both brings together much of our analysis from earlier chapters, and identifies statistical tools that we can apply in this estimation. In section 6.2, we showed there is an underlying probabilistic base for

statistical estimation, and showed how to transform cycle execution time observations for eventual analysis. In section 6.3, we argued that a Bayesian estimation of the cycle execution time distribution parameters is appropriate. We argued that the assumptions of the Bayesian model are justified, and examined the construction of prior distributions. In section 6.4 we show how the posterior cycle execution time parameter distributions can be used to select the better of two partitions. We identified several relevant decision cost functions, and argued that the selection of the partition achieving minimized expected cost is computationally trivial.

Chapter 7

Qualitative Analysis

7.1. Chapter Overview

We must ultimately compare two partitions by comparing the moments of their resulting cycle execution time distributions. In Chapter 6 we propose that these moments be estimated statistically. We can then **quantitatively** compare the partitions by numerically computing and comparing their execution time moments. We now show that it is sometimes possible to compare these moments **qualitatively** without actually computing them. We use time complexity arguments to show that such a comparison can be computationally more efficient than a quantitative comparison.

In section 7.2 we motivate our development of qualitative partition comparison. In section 7.3, we describe certain *stochastic order relations*, our basic tools for qualitative comparison. A stochastic order relation between random variables states that one is probabilistically larger than the other. We give conditions for random variables to be stochastically related. Sections 7.4 and 7.5 illustrate two different scenarios in which we might use qualitative partition comparison. Section 7.4 shows how stochastic order relations might be of use to a partitioning algorithm which sequentially assigns work nodes to processors. In section 7.5 we consider a general qualitative comparison of two partitions, present a numerical example of general qualitative analysis, and identify an important application for a qualitative comparison.

The analysis developed in this chapter presented with two disclaimers. First, stochastic order relations make statements about probability distributions. Our results are generally of the form "*If random variable X is stochastically related to random variable Y, then* **this** *partition is better than* **that** *partition* ". The X and Y we consider have distribution functions which we cannot realistically quantify. However, it may be possible to deal with stochastic order relations statistically. Our work lays the necessary probabilistic base for statistical qualitative comparison, illustrated by example in section 7.5. Our second warning is that qualitative comparison is not always possible. The conditions for stochastic relations are often restrictive. Nevertheless, the power of stochastic order relations is attractive enough to warrant a careful consideration of their potential application.

7.2. Motivation for Qualitative Comparison

A statistical analysis of a simulation's behavior observes a certain number of cycles, calculating the cycle execution of each from its hyper–execution graph. This calculation has linear time complexity in the size of the work assignment graph; however, the work assignment graph might be very large. The time complexity of estimating a partition's cycle execution time parameters is $O(\mathbf{Z} \cdot \mathbf{d})$, where $\mathbf{Z}$ is the sum of the number of nodes and edges in the work assignment graph, and $\mathbf{d}$ is the number of cycle observations.

For large $\mathbf{Z}$ and $\mathbf{d}$, determining the partition with smaller moments can be computationally expensive, prohibiting much of this kind of comparison. For example, many design automation heuristics [BrF76] incorporate a perturbation phase where many small perturbations of the solution are considered, and any perturbation improving performance

is adopted. A quantitative comparison of moments is just as expensive when the partitions are very much alike as it is when they are very different. We are discouraged from using a perturbation phase to improve a partition. In this chapter we show that there is a more efficient approach. We identify circumstances where we may conclude that one partition is superior to another, without explicitly estimating their respective cycle execution moments.

7.3. Three Stochastic Relations

In this section we define three stochastic order relations, and identify conditions where these relations hold.

Stochastic order relations relate random variables X and Y by considering $E[g(X)]$ and $E[g(Y)]$, where g is a function of a certain class. The first stochastic order relation we consider is that of *stochastic variation* [Ros83].

Definition 7-1: *Stochastically More Variable*

Let X and Y be random variables. X is said to be ***stochastically more variable*** than Y, written $X \geq_v Y$, if and only if $E[g(X)] \geq E[g(Y)]$ for every increasing convex function g.

□

The functions $g(x) = x$ and $h(x) = x^2$ are both increasing and convex, so that if $X \geq_v Y$, then X's mean ($E[X] = E[g(X)]$) and second moment ($E[X^2] = E[h(X)]$) are greater than or equal to Y's mean and second moment. If X and Y have equal means, the relation $Var(X) = E[X^2] - E[X]^2$ implies that $Var(X) \geq Var(Y)$. $X \geq_v Y$ is then a strong statement of X's being more variable than Y (when their means are equal). An equivalent characterization of the $\geq_v$ relation is given in [Ros83].

Lemma 7-1: [Ros83] Let X and Y be non-negative random variables. $X \geq_v Y$ if and only if for every $a \geq 0$

$$\int_0^a Prob\{X \geq t\}\, dt \quad \geq \quad \int_0^a Prob\{Y \geq t\}\, dt.$$

□

Another useful result concerning stochastic variability is also given in [Ros83];

Lemma 7-2: [Ros83] If $X_1, \cdots, X_n$ are independent, and $Y_1, \cdots, Y_n$ are independent, and $X_i \geq_v Y_i$, $i = 1,2, \cdots, n$, then

$$g(X_1, \cdots, X_n) \geq_v g(Y_1, \cdots, Y_n)$$

for all increasing convex functions $g: R^n \rightarrow R$.

□

The second stochastic order relation we consider is stronger than stochastic variability.

Definition 7-2: *Stochastically Larger*

Let X and Y be random variables. X is said to be *stochastically larger* than Y, written $X \geq_{st} Y$, if and only if $E[f(X)] \geq E[f(Y)]$ for all increasing functions f.

□

Clearly, when $X \geq_{st} Y$, then $X \geq_{v} Y$. An equivalent definition of the $\geq_{st}$ relation exists when X and Y are non-negative.

Lemma 7-3: [Ros83] Let X and Y be non-negative random variables. Then $X \geq_{st} Y$ if and only if

$$Prob\{X \geq t\} \geq Prob\{Y \geq t\}$$

for all $t \geq 0$.

□

A result entirely analogous to lemma 7-2 exists for the $\geq_{st}$ relation. This result, lemma 7-3, and the definition of the $\geq_{st}$ relation are all found in [Ros83].

The $\geq_{st}$ relation is naturally extended to arbitrary vectors of random variables [Ros83]. This extension defines the *jointly larger* stochastic relation.

Definition 7-3: *Jointly Larger*

Let $\mathbf{X} = <X_1, \cdots, X_n>$ and $\mathbf{Y} = <Y_1, \cdots, Y_n>$ be arbitrary vectors of random variables. We say that $\mathbf{X}$ is *jointly larger* than $\mathbf{Y}$, written $\mathbf{X} \geq_{jl} \mathbf{Y}$, if

$$E[g(X_1, \cdots, X_n)] \geq E[g(Y_1, \cdots, Y_n)]$$

for all increasing functions $g: R^n \rightarrow R$.

□

This relation will be most useful to us if we can find conditions on the respective joint densities of $\mathbf{X}$ and $\mathbf{Y}$ which imply that $\mathbf{X} \geq_{jl} \mathbf{Y}$. We have discovered both a necessary condition, and (stronger) sufficient conditions for joint dominance.

Lemma 7-4: Let $\mathbf{X} = \langle X_1, \cdots, X_n \rangle$ and $\mathbf{Y} = \langle Y_1, \cdots, Y_n \rangle$ be arbitrary vectors of non-negative random variables. Then

(i) If $\mathbf{X} \geqslant_{jl} \mathbf{Y}$, then for every vector $\langle t_1, \cdots, t_n \rangle$ with non-negative components,

$$Prob\{X_1 \geqslant t_1, \cdots, X_n \geqslant t_n\} \geqslant Prob\{Y_1 \geqslant t_1, \cdots, Y_n \geqslant t_n\}. \tag{7-1}$$

(ii) Suppose that for every subset D of vector component indices and every non-negative vector $\langle t_1, \cdots, t_n \rangle$ we have

$$Prob\{X_i \geqslant t_i \text{ for all } i \in D \mid X_j = t_j \text{ for all } j \in \bar{D}\}$$

$$\geqslant Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}\}.$$

Furthermore suppose that the right hand side of this inequality is an increasing function of every t_j, $j \in \bar{D}$. Then $\mathbf{X} \geqslant_{jl} \mathbf{Y}$.

Proof: We show first that $\mathbf{X} \geqslant_{jl} \mathbf{Y}$ implies the inequality 7-1. For every vector $\mathbf{T} = \langle t_1, \cdots, t_n \rangle$ we define the function $f_{\mathbf{T}}: R^n \to R$ by

$$f_{\mathbf{T}}(x_1, \cdots, x_n) = \begin{cases} 1 & \text{if } x_j \geqslant t_j \text{ for } i=1,\ldots,n \\ 0 & \text{otherwise} \end{cases}$$

This function is easily seen to be increasing in each argument. Furthermore,

$$E[f_{\mathbf{T}}(X_1, \cdots, X_n)] = Prob\{X_1 \geqslant t_1, \cdots, X_n \geqslant t_n\}.$$

Since $E[f_{\mathbf{T}}(\mathbf{X})] \geqslant E[f_{\mathbf{T}}(\mathbf{Y})]$, inequality 7-1 follows directly.

Now we show conditions (ii) imply the definition of the $\geqslant_{jl}$ relation. We induct on the number of vector components. The base of the induction argument, namely $n = 1$, is easily satisfied, since this case reduces to the relation $\geqslant_{st}$. As the induction hypothesis, we suppose that this implication is true for all random vectors of length $k-1$. Let $\langle X_1, \cdots, X_k \rangle$ and $\langle Y_1, \cdots, Y_k \rangle$ be vectors of random variables satisfying conditions (ii). Let $g: R^k \to R$ be an increasing function. Now

$$E[g(X_1, \cdots, X_k)] = E_{X_k}\left[E^X_{t_k}[g(X_1, \cdots, X_{k-1}, t_k)]\right]$$

where E_{X_k} is the expectation operator for X_k, and $E^X_{t_k}$ is the expectation operator for the joint density of $X_1, \cdots, X_{k-1}$ conditioned on $X_k = t_k$. We let $\mathbf{X}_x^{k-1}$ denote the vector of $\mathbf{X}$'s first $k-1$ components conditioned on $X_k = x$; we likewise define $\mathbf{Y}_y^{k-1}$ to be $\mathbf{Y}$'s first $k-1$ components conditioned on $Y_k = y$. We first demonstrate that $\mathbf{X}_{t_k}^{k-1} \geqslant_{jl} \mathbf{Y}_{t_k}^{k-1}$. Let $\langle t_1, \cdots, t_{k-1} \rangle$ be a non-negative vector, and let D be any subset of $\{1, \cdots, k-1\}$. Define $\bar{D} = \{1, \cdots, k-1\} - D$, and $\bar{D}_k = \bar{D} \cup \{k\}$. Then

$$Prob\{X_i \geqslant t_i \text{ for all } i \in D \mid X_j = t_j \text{ for all } j \in \bar{D}; X_k = t_k\}$$

$$= Prob\{X_i \geqslant t_i \text{ for all } i \in D \mid X_j = t_j \text{ for all } j \in \bar{D}_k\}$$

$$\geqslant Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}_k\} \tag{7-2}$$

$$= Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}; Y_k = t_k\} \tag{7-3}$$

where the establishment of the inequality follows from our assumption that $\mathbf{X}$ and $\mathbf{Y}$ satisfy the first of (ii)'s conditions. Thus vectors $\mathbf{X}_{t_k}^{k-1}$ and $\mathbf{Y}_{t_k}^{k-1}$ satisfy the first of (ii)'s conditions. Furthermore, the equivalence established between lines 7–2 and 7–3 establishes that the second of (ii)'s conditions is satisfied as well. The probability of line 7–2 is assumed to be increasing in any t_j, $j \in \bar{D}_k$. In particular, it is increasing in any t_j, $j \in \bar{D}$. The probabilities of lines 7–2 and 7–3 are equal, so that the second of (ii)'s conditions is met. $\mathbf{X}_{t_k}^{k-1} \geqslant_{jl} \mathbf{Y}_{t_k}^{k-1}$ then, by the induction hypothesis.

For $\hat{s} > s$, we show that $\mathbf{Y}_{\hat{s}}^{k-1} \geqslant_{jl} \mathbf{Y}_{s}^{k-1}$. We have

$$Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}; Y_k = \hat{s}\}$$

$$= Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}_k\} \text{ when } t_k = \hat{s}$$

$$\geqslant Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}_k\} \text{ when } t_k = s \tag{7-4}$$

$$= Prob\{Y_i \geqslant t_i \text{ for all } i \in D \mid Y_j = t_j \text{ for all } j \in \bar{D}; Y_k = s\} \tag{7-5}$$

where the inequality is established because we have assumed the vector $\mathbf{Y}$ satisfies the second of (ii)'s conditions. Vectors $\mathbf{Y}_{\hat{s}}^{k-1}$ and $\mathbf{Y}_{s}^{k-1}$ therefore meet (ii)'s first condition. (ii)'s second condition is established by the equivalence of the probabilities of lines 7–4 and 7–5. By the induction hypothesis, we have $\mathbf{Y}_{\hat{s}}^{k-1} \geqslant_{jl} \mathbf{Y}_{s}^{k-1}$.

Having established these stochastic relationships, we exploit their properties to show the desired result. Since $\mathbf{X}_{t_k}^{k-1} \geqslant_{jl} \mathbf{Y}_{t_k}^{k-1}$, it follows by definition that

$$E_{t_k}^{X}[g(X_1, \cdots, X_{k-1}, t_k)] \geqslant E_{t_k}^{Y}[g(Y_1, \cdots, Y_{k-1}, t_k)] \tag{7-6}$$

since this restriction of g is still increasing in its first $k-1$ arguments. Now let $h_x(s)$ be X_k's marginal density function. As inequality 7–6 is true for every t_k, we have

$$E_{X_k}\left[E_{t_k}^{X}[g(X_1, \cdots, X_{k-1}, t_k)]\right] = \int_0^\infty h_x(s) \cdot E_s^X[g(X_1, \cdots, X_{k-1}, s)]\, ds$$

$$\geqslant \int_0^\infty h_x(s) \cdot E_s^Y[g(Y_1,\cdots,Y_{k-1},s)]\,ds. \tag{7-7}$$

Now $\mathbf{Y}_{\hat{s}}^{k-1} \geqslant_{jl} \mathbf{Y}_s^{k-1}$ whenever $\hat{s} > s$; it follows by definition that

$$E_{\hat{s}}^Y[g(Y_1,\cdots,Y_{k-1},\hat{s})] \geqslant E_s^Y[g(Y_1,\cdots,Y_{k-1},s)] \tag{7-8}$$

whenever $\hat{s} > s$. This expectation is therefore an increasing function of s. Under our assumptions about $\mathbf{X}$ and $\mathbf{Y}$, it is not hard to show that $X_k \geqslant_{st} Y_k$: choose $D = \{1,\cdots,k\}$ and $t_j = 0$ for $j = 1,2,\cdots,k-1$. Then (ii)'s first condition implies that $Prob\{X_k \geqslant t\} \geqslant Prob\{Y_k \geqslant t\}$ for every non-negative t, so that $E[f(X_k)] \geqslant E[f(Y_k)]$ for any increasing function f. Let $h_y(s)$ be Y_k's marginal density function. Since the expectation on the right hand side of 7-8 is increasing in s, we have

$$\int_0^\infty h_x(s) \cdot E_s^Y[g(Y_1,\cdots,Y_{k-1},s)]\,ds \geqslant \int_0^\infty h_y(s) \cdot E_s^Y[g(Y_1,\cdots,Y_{k-1},s)]\,ds$$

$$= E[g(Y_1,\cdots,Y_k)]. \tag{7-9}$$

The chain of inequalities from 7-6 to 7-9 establishes

$$E[g(X_1,\cdots,X_k)] \geqslant E[g(Y_1,\cdots,Y_k)],$$

whence $\mathbf{X} \geqslant_{jl} \mathbf{Y}$ by definition.

□

The presented sufficient conditions for joint domination are stringent. In the special case where the X_i are independent and the Y_i are independent, the necessary conditions given by lemma 7-4 are also sufficient. It is easily seen that the necessary conditions imply that $X_i \geqslant_{st} Y_i$ for $i = 1,2,\cdots,n$; for any t,

$$Prob\{X_i \geqslant t\} = Prob\{X_i \geqslant t;\ X_j \geqslant 0 \text{ for all } j \neq i\}$$

$$\geqslant Prob\{Y_i \geqslant t;\ Y_j \geqslant 0 \text{ for all } j \neq i\}$$

$$= Prob\{Y_i \geqslant t\}.$$

Then we know that the independence of the X_i and the independence of the Y_i imply that

$$g(X_1,\cdots,X_n) \geqslant_{st} g(Y_1,\cdots,Y_n)$$

for all increasing functions g. Thus

$$E[g(X_1,\cdots,X_n)] \geqslant E[g(Y_1,\cdots,Y_n)]$$

for all increasing g, hence $\mathbf{X} \geqslant_{jl} \mathbf{Y}$.

7.4. Sequential Assignment

We now consider how the $\geqslant_v$ and $\geqslant_{st}$ relations can be used by a partitioning algorithm. One possible partitioning approach is to assign the work nodes sequentially,

attempting at every step to find the best assignment for the node. This might be accomplished by first sorting the work graph nodes topologically, and then by sequentially assigning each node with respect to this order. We suppose that some number of work graph nodes has already been assigned. We assume there are K processors, and that L_i denotes the last node assigned to processor i. To avoid explanation of special cases, we assume that each processor is initially assigned a placebo node with no associated precedence, and zero execution cost.

Our problem is to assign a node W to some processor. This assignment can be viewed as the attachment of W to some L_i with an execution precedence arc rooted in L_i. We generally assume that all of a functional's work nodes must be assigned to the same processor. Attaching W to L_i also involves moving any previously assigned work nodes associated with W's functional to processor i as well. All other assigned work is assumed to be fixed. We presume that some algorithm determines the resulting execution order within processor i. Let $F(L_i)$ denote the finishing time of L_i; $F(L_i)$ is the time at which processor i completes all of the work assigned to it. We denote the attachment of W to L_j by $W@L_j$. $F(W@L_j)$ is the finishing time of processor j with W attached. We would like to attach W to a processor so that all moments of the assigned work's finishing time are minimized. Formally, we search for that L_j such that

$$E[\max\{F(L_1),\cdots,F(W@L_j),\cdots,F(L_K)\}^n],$$

the *nth* moment, is minimized for all n. Since the $\max\{\ \}^n$ function is both increasing and convex, stochastic order relations may help us identify this L_j.

We can use both the $\geqslant_v$ and the $\geqslant_{st}$ relations to compare the assignment of W to L_i with the assignment of W to L_j. Without loss of generality, we will use $i = 1$ and $j = 2$ in the following discussion. Also without loss of generality, we assume that all of W's ancestors reside in processors $1,2,\cdots,p$. We find it necessary to jointly treat all processors holding W's ancestors; this necessity stems from the observation that these processors' behavior are correlated. Suppose we can group the processors so that the finishing times of $1,2,\cdots,p$ are all independent of the finishing times of $p+1,\cdots,K$. We can then compare the assignment of W to L_1 with the assignment to L_2 without consideration for the finishing times $F(L_{p+1}),\cdots,F(L_K)$. This is not surprising, as W is in no way connected with the processors $p+1,\cdots,K$. Our ability to make this comparison is formalized by the next lemma.

Lemma 7-5: Let

$$M_1 = \max\{F(W@L_1),F(L_2),\cdots,F(L_p)\}$$

and

$$M_2 = \max\{F(L_1),F(W@L_2),\cdots,F(L_p)\}.$$

Suppose there exists a p such that all of W's ancestors reside in processors $1,2,\cdots,p$, and all finishing times $F(L_j)$, $j > p$ are independent of $F(L_i)$, $1 \leqslant i \leqslant p$. Then

(i) If

$$\int_0^a Prob\{M_1 \geqslant t\}\,dt \geqslant \int_0^a Prob\{M_2 \geqslant t\}\,dt$$

for all $a \geqslant 0$, then

$$\max\{F(W@L_1),F(L_2),\cdots,F(L_K)\} \geqslant_v \max\{F(L_1),F(W@L_2),\cdots,F(L_K)\}.$$

(ii) If

$$Prob\{M_1 \geqslant t\} \geqslant Prob\{M_2 \geqslant t\}$$

for all t, then

$$\max\{F(W@L_1),F(L_2),\cdots,F(L_K)\} \geqslant_{st} \max\{F(L_1),F(W@L_2),\cdots,F(L_K)\}.$$

Proof: We first demonstrate (i). Let $U = \max\{F(L_{p+1}),\cdots,F(L_K)\}$. Frivolously, we have $U \geqslant_v U$. By lemma 7–1, we have $M_1 \geqslant_v M_2$. Now

$$\begin{aligned}
&\max\{F(W@L_1),F(L_2),\cdots,F(L_K)\} \\
&\quad = \max\{\max\{F(W@L_1),\cdots,F(L_p)\},\max\{F(W@L_{p+1}),\cdots,F(L_K)\}\} \\
&\quad = \max\{M_1,U\} \\
&\quad \geqslant_v \max\{M_2,U\} \text{ by lemma 7–2} \\
&\quad = \max\{F(L_1),F(W@L_2),\cdots,F(L_K)\}.
\end{aligned}$$

Item (ii) is shown in a similar fashion.

□

The preceding lemma identifies circumstances under which we can compare two potential assignments of W. However, we should stress that it might not be possible to make this comparison. We have no assurances that the lemma's hypothesis will be (non-trivially) met; p could be equal to K. In fact, we must be able to compare each pair of

assignments if we are to **prove** that a particular assignment of W minimizes $E[\max\{F(L_1),\cdots,F(W@L_j),\cdots,F(L_K)\}]$; this is not generally possible. Nevertheless, this approach holds promise. Even if the stochastic relations induce only a partial order on the possible assignments of W, it is useful to know that one assignment is provably better (on the average) than another. Some computational savings might be achieved using this approach; lemma 7–5 says that we need not consider the work nodes in processors p, ..., K when we choose between placing W in processor 1 or 2 .

To say that $X \geqslant_v Y$ or $X \geqslant_{st} Y$ is a much stronger statement than to simply say $E[X] \geqslant E[Y]$. If assumptions of the form $E[X] \geqslant E[Y]$ are strong enough to produce lemmas of the form given in 7–5, it might well be argued that consideration of $\geqslant_v$ and $\geqslant_{st}$ relations is unnecessary. However, we can show by example that assumptions of the form $E[X] \geqslant E[Y]$ are not strong enough. An analog to lemma 7–5 would say : If

$$E[\max\{F(W@L_1),F(L_2),\cdots,F(L_p)\}] \geqslant E[\max\{F(L_1),F(W@L_2),\cdots,F(L_p)\}]$$

then

$$E[\max\{F(W@L_1),F(L_2),\cdots,F(L_K)\}] \geqslant E[\max\{F(L_1),F(W@L_2),\cdots,F(L_K)\}].$$

We discredit such a statement by giving an example of independent X, Y, and Z such that $E[X] > E[Y]$, but $E[\max\{X,Z\}] < E[\max\{Y,Z\}]$. Let X be the constant .6 , let Y be uniform on [0, 1], and let Z be the constant .75. We assume that Y is independent of X and Z. Now $E[X] = .6 > .5 = E[Y]$. Furthermore,

$$E[\max\{Y,Z\}] = Prob\{Y \leqslant .75\}\cdot(.75) + Prob\{Y > .75\}\cdot E[Y \mid Y > .75]$$

$$> .75$$

$$= E[\max\{X,Z\}]$$

since $E[Y \mid Y > .75] > .75$.

Given that the assumption $E[X] \geqslant E[Y]$ is not strong enough to be useful while the assumption $X \geqslant_v Y$ is, we defend consideration of the stronger statement $X \geqslant_{st} Y$. First of all, $\geqslant_{st}$ is easier to verify. The mathematical statement of lemma 7–3 is easier to check than the statement of lemma 7–1. Furthermore, some interesting results about $\geqslant_{st}$ are known. For example, [Ros83] states that a normal random variable increases stochastically in its mean (it does not increase in its variance). If we can reasonably assume that two random variables being compared are normal and have the same variance, then we need only check their means to identify $\geqslant_{st}$. While comparing X and Y, we may therefore save some computational effort by testing for $X \geqslant_{st} Y$ first. Many other results concerning the $\geqslant_{st}$ relation are given in [MaO79] .

The statements of $\geqslant_v$ and $\geqslant_{st}$ are statements about probability distributions. In view of the difficulty encountered simply trying to calculate the moments of the cycle execution time, it may seem futile to develop a theory based on the entire distribution. In answer to this objection, we appeal again to the necessity of treating partitioning in a statistical context. While we are not able to precisely ascertain that one random variable stochastically dominates another, we might be able to examine observations of those random variables and develop a reasonable confidence of this domination.

7.5. General Comparison

The previous section showed how a partitioning algorithm might employ the $\geqslant_v$ and $\geqslant_{st}$ relations to qualitatively compare two potential "subpartitions" under

consideration. In this section we show how any two partitions might be qualitatively compared. This approach focuses on a vector of work nodes whose behavior reflects all differences between two partitions. In subsection 7.5.1 we define this *border vector* and show how to efficiently discover a border vector. We derive a *border graph* from this vector, and illustrate the graph's important properties. In subsection 7.5.2 we discuss a transformation of the border graph into a *hyper–border* graph created as a function of an observed cycle. We prove that the length of the longest path through a hyper–border graph is equal to the cycle execution time of the observed cycle under hyper–message synchronization. We then show how the border vector and hyper–border graph are used to qualitatively compare two partitions. Subsection 7.5.3 gives a metric for the comparison of two border vectors related by $\geqslant_{jl}$. In subsection 7.5.4 we present a numerical example of how qualitative comparison can be statistically applied. Subsection 7.5.5 concludes the development of qualitative analysis by proving that qualitative analysis can be more computationally efficient than quantitative moment estimation.

7.5.1. The Border Vector

We begin our analysis with a definition.

Definition 7-4: *Fixed, Stable Work Nodes*

Let WG be a work graph, and let $WAG(1)$ and $WAG(2)$ be work assignment graphs for WG under partitions Ψ_1 and Ψ_2, respectively. Let W be a work node. W is said to be *fixed* if W resides in the same processor under Ψ_1 and Ψ_2, and has the same execution successor under Ψ_1 and Ψ_2. The system sink node is defined to be *stable*. A general work node W is said to be stable if it is fixed, and has stable work assignment graph successors under Ψ_1 and Ψ_2.

□

We use the concept of fixed nodes to define a *border* vector which reflects differences between $WAG(1)$ and $WAG(2)$.

Definition 7-5: *Border Vector*

Let WG be a work graph, and let Ψ_1 and Ψ_2 be two partitions of WG. Let $WAG(1)$ and $WAG(2)$ be their respective work assignment graphs, and let $\mathbf{B} = <B_1, \cdots, B_k>$ be a vector of work nodes. $\mathbf{B}$ is said to be a *border vector* for $WAG(1)$ and $WAG(2)$ if

(i) Each B_i is stable;

(ii) For $j = 1,2$ every path through $WAG(j)$ from a non–fixed node to the system sink node must include some B_i.

□

A border vector captures the topological differences between Ψ_1 and Ψ_2. The topological differences between partitions are embodied by non–fixed work nodes, which conceptually pass the effects of these differences along work assignment graph paths. Since

every path out of a non-fixed node to the sink includes a border node, the message timing differences between two partitions is reflected in the difference in the border nodes' joint behavior under Ψ_1, and their joint behavior under Ψ_2.

There are partition pairs Ψ_1 and Ψ_2 such that no border exists for $WAG(1)$ and $WAG(2)$: if work node S_t is a predecessor of the sink P_S in $WAG(1)$ and is not fixed, then S_t is not a border node, so that the path from S_t to P_S defies condition (ii) of definition 7-5. This observation leads us to necessary and sufficient conditions for the existence of a border vector for work assignment graphs $WAG(1)$ and $WAG(2)$.

Lemma 7-6: Let WG be a work graph, Ψ_1 and Ψ_2 be partitions of WG, and $WAG(1)$ and $WAG(2)$ be their respective work assignment graphs. Let $S_1, \cdots, S_k$ be the sink node's predecessors in $WAG(1)$. Then a border vector **B** exists for $WAG(1)$ and $WAG(2)$ if and only if for $i = 1,2, \cdots, k$, S_i is stable.

Proof: The argument of the last paragraph showed that every S_i must be stable. The sufficiency of this condition is easily seen. The vector $<S_1, \cdots, S_k>$ satisfies condition (i) of definition 7-5 by assumption. Any path to the sink must pass through one of the S_i, so that $<S_1, \cdots, S_k>$ also satisfies condition (ii).

□

We will always implicitly assume that the work assignment graphs under discussion do have a border vector. The border vector identified by lemma 7-6 is not a particularly interesting or useful one; we would like to trap the differences between partitions "far back" from the sink as we can. We sketch here an efficient procedure for finding a better border vector. This procedure requires the identification of stable work nodes. The definition of stability suggests that the work nodes' stability be determined by processing nodes in reverse topological order (with respect to the work assignment graph). Then to check for a node's stability, we need to check that it is fixed and that each successor is stable. The time complexity of determining all work nodes' stability is seen to be $O(N \cdot (E+1))$, where there are N work nodes, and the maximum number of node successors in the work assignment graph is E.

Let **B** be the vector of all stable successors of non-stable nodes. We claim that **B** is a border. Clearly the requirement that border nodes be stable is met. Let W be a non-fixed node, and let $W, P_1, \cdots, P_S$ be any path through a work assignment graph $WAG(j)$ from W to the sink node. Let P_i be the first stable node in this sequence. P_i is thus the descendent of a non-stable node, and hence is in **B**. This path includes the border node P_i; since the work assignment graph $WAG(j)$ was arbitrarily chosen, the argument works equally well for $WAG(1)$ or $WAG(2)$. Consequently, condition (ii) of definition 7-5 is met; it follows that **B** is a border vector. This **B** can be found while the nodes' stabilities are determined, with no additional time complexity. Thus, given two work assignment graphs of the same simulation, we can efficiently find a border vector for those graphs.

Given a border vector **B** and work assignment graph $WAG(j)$, we define a subgraph of $WAG(j)$ called the *border graph.*

Definition 7-6: *Border Graph*

Let $\mathbf{B} = <B_1, \cdots, B_k>$ be a border vector for $WAG(1)$ and $WAG(2)$. **B**'s *border graph* for $WAG(j)$ is the graph $BG(j)$ composed of

(i) All of **B**'s nodes ;

(ii) All fixed work nodes having a path through $WAG(j)$ to the sink node which does not include some node in **B**;

(iii) The sink node P_S;

(iv) All arcs in $WAG(j)$ shared by the nodes defined above;

□

Figure 7–1 illustrates two different work assignment graphs for the same system. The vector $<(1, T_5), (1, T_6)>$ is a border vector; figure 7–1 also illustrates the associated border graph. We observe that the border graph is a common subgraph of both work assignment graphs. We will soon prove that the border graph is always a common subgraph of its underlying work assignment graphs.

The following lemmas establish important properties of $BG(j)$.

Lemma 7-7: Let $BG(j)$ be a border graph for border **B** and work assignment graph $WAG(j)$, and let W be a node in $WAG(j)$. If W does not appear in $BG(j)$, then every path in $WAG(j)$ from W to the system sink must include some border node.

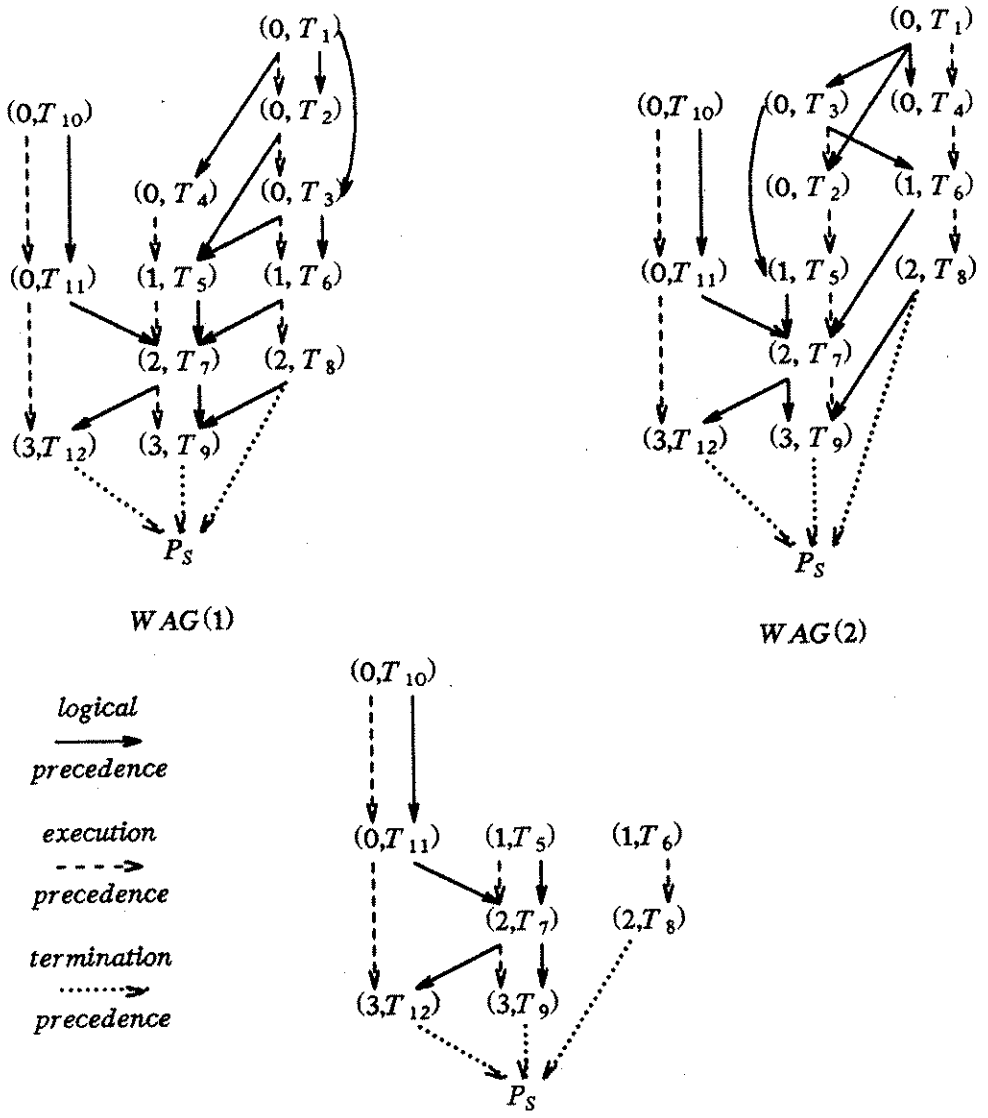
Proof: W is either fixed or not fixed. If W is not fixed, then every path from W through the system sink in $WAG(j)$ includes some border node, by definition of **B**. Suppose then that W is fixed. Since W does not appear in $BG(j)$, all paths from W to the sink in $WAG(j)$ must include some border node, otherwise condition (ii) of definition 7–6 implies that W is in $BG(j)$.

□

One use of lemma 7–7 is to aid in the proof of the next lemma.

Lemma 7-8: Let $BG(j)$ be a border graph for border **B** and work assignment graph $WAG(j)$, and let W be a work node. If W appears in $BG(j)$ and is not a border node , then every one of W's $WAG(j)$ predecessors appear in $BG(j)$.

Proof: Suppose W appears in $BG(j)$ and is not a border node. For the sake of contradiction we suppose that W has a $WAG(j)$ predecessor P not in $BG(j)$. We may assume that P is not a border node, otherwise it would certainly be in $BG(j)$. By lemma 7–7, all paths from P to the sink in $WAG(j)$ must include some border node. Since W is in $BG(j)$ but is not a border node, W is in $BG(j)$



Border Graph for $<(1,T_5),(1,T_6)>$

Border Graph

Figure 7–1

by means of condition (ii) of definition 7–6. Thus, a path π exists in $WAG(j)$ from W to the system sink which does not include a border node. But the path which goes from P to W, and then follows path π does not include a border node, a contradiction. P is consequently in $BG(j)$.

□

The utility of a border graph lies in the fact that if **B** is a border for two work assignment graphs $WAG(1)$ and $WAG(2)$, then the border graph $BG(1)$ is identical to the border graph $BG(2)$.

Theorem 7-1: Let **B** be a border vector for $WAG(1)$ and $WAG(2)$. Then $BG(1) = BG(2)$.

Proof: We induct on a reverse topological sorting of $BG(1)$'s nodes to show that

(i) If W is a work node in $BG(1)$, then W is a work node in $BG(2)$;

(ii) If W is a predecessor of V in $BG(1)$, then W is a predecessor of V in $BG(2)$.

For the induction base, consider the sink node P_S, and its predecessors $S_1, \cdots, S_k$. P_S is in both $BG(1)$ and $BG(2)$ by condition (iii) of definition 7–6. Any S_i which is a border node is in both $BG(1)$ and $BG(2)$. Otherwise S_i has a trivial path to P_S which does not include a border node; S_i is assumed stable (hence fixed) so that S_i is in $BG(1)$ and $BG(2)$ by definition. The only arcs out of the S_i are directed to P_S, whence the induction base is satisfied.

For the induction hypothesis let W be in $BG(1)$ and suppose that all of W's $BG(1)$ successors are in $BG(2)$. We first establish **condition (i).** The case where W is a border node is trivial; we suppose that W is not a border node. By definition 7–6, W must have at least one $WAG(1)$ successor S_W which is not a border node and which has a path in $WAG(1)$ to the sink not including a border node. S_W is consequently in $BG(1)$, otherwise lemma 7–7 is contradicted. The induction hypothesis states that S_W is also in $BG(2)$. We now claim that S_W is a successor of W in $WAG(2)$. If S_W is the execution successor of W in $WAG(1)$, then it is in $WAG(2)$ since W is fixed. W's logical successors in $WAG(1)$ and $WAG(2)$ are identical, so that S_W must be a successor of W in $WAG(2)$. Recalling that S_W is not a border node, lemma 7–8 implies that W is in $BG(2)$. We have thus completed the induction on condition (i).

We now establish **condition (ii).** Let S_W be any successor of W in $BG(1)$. By the induction hypothesis, S_W is also in $BG(2)$. We showed above that W's successors are identical in $WAG(2)$ and $WAG(1)$. S_W is then a $BG(2)$ successor of W since $BG(2)$ takes its arcs from $WAG(2)$. This completes the induction on condition (ii), and the entire induction statement.

The induction statement we have proven shows that every node and every edge in $BG(1)$ appears in $BG(2)$. Thus $BG(1)$ is a subgraph of $BG(2)$. But we can apply the same form of argument to show that $BG(2)$ is a subgraph of $BG(1)$. Thus $BG(1) = BG(2)$.

□

7.5.2. The Hyper–Border Graph

In Chapter 4 we defined the hyper–execution graph, a graph describing an observed cycle's behavior under hyper–message synchronization. A hyper–execution graph $HEG(WG,j,c)$ is created by modifying the topology of the work assignment graph $WAG(j)$, and then by weighting nodes and arcs. The longest path through $HEG(WG,j,c)$ is identical to the finishing time of the cycle c under partition Ψ_j. In an similar fashion, we will modify a border graph as a function of a cycle c and partition Ψ_j to create a graph $HB(\mathbf{B},j,c)$. The length of the longest path through $HB(\mathbf{B},j,c)$ is c's cycle execution time under Ψ_j and hyper–message synchronization. We will call $HB(\mathbf{B},j,c)$ the *hyper–border* graph.

The border graph is a common subgraph of two work assignment graphs. We create a hyper–border graph by first applying the work assignment graph to hyper–execution graph transformation to the border graph. An evaluated work node W is represented in the hyper–execution graph by a node $R(W)$; a hyper–evaluated node W is represented by two nodes $IP(W)$ and $EP(W)$. If work graph node W appears in the border graph, we represent W in the hyper–border graph by either $R(W)$, or $IP(W)$ and $EP(W)$ in exactly the same way. Thus any node H appearing in the hyper–border graph also appears in the hyper–execution graph. As done with the hyper–execution graph, we define hyper–border graph arcs and weights entirely as a function of the set of evaluated work nodes.

At this stage of a hyper–border graph's construction, we have a hyper–execution graph induced by the transformation of the work assignment graph nodes appearing in the border graph. We now change this topology slightly by removing all arcs directed to nodes representing border nodes. Because the border graph is a common subgraph of the two work assignment graphs, the hyper–border graph in its present state of construction does not reflect any differences between the two partitions. To distinguish between partitions, we add rootless weighted arcs, directed to hyper–border nodes representing border nodes. The weights on these arcs will reflect the difference between the two partitions. If a border node W is evaluated, we define two rootless arcs directed to $R(W)$ in $HB(\mathbf{B},j,c)$. One arc is the *initiation* arc, and is weighted by the time at which W begins its execution in cycle c, under partition Ψ_j and hyper–message synchronization. A *release* arc is also directed to $R(W)$. The release arc is weighted by the time at which W's execution predecessor reports its completion to W under Ψ_j in cycle c. If a border node W is not evaluated, the nodes $IP(W)$ and $EP(W)$ represent it in $HB(\mathbf{B},j,c)$. We define a rootless initiation arc directed to $IP(W)$, weighted by the time at which W hyper–executes in cycle c under partition Ψ_j and hyper–message synchronization. We define a rootless release arc directed to $EP(W)$, weighted by the arrival time of W's execution predecessor's release message. We see then that the only difference between $HB(\mathbf{B},1,c)$ and $HB(\mathbf{B},2,c)$ is the weight given to the rootless arcs.

We next show that $HB(\mathbf{B},j,c)$ can be used to calculate c's cycle execution time under Ψ_j and hyper–message synchronization. The only difference between $HB(\mathbf{B},1,c)$ and $HB(\mathbf{B},2,c)$ lies in the rootless arcs' weights; this may allow us to compare their cycle execution times by comparing these arcs' weights. We are aided by further definitions. If H is a $HEG(WG,j,c)$ node, we let $l_x(H)$ be the length of the longest path through H in

$HEG(WG,j,c)$, and we let $l_x(P_i, H)$ be the length of the longest path through H in $HEG(WG,j,c)$ whose last arc passes from P_i to H. We correspondingly define $l_b(H)$ and $l_b(P_i, H)$ for H and P_i in $HB(\mathbf{B},j,c)$. $w(H)$ is the weight assigned to node H (in both graphs) and $e(P_i, H)$ is the weight on the edge between P_i and H (in both graphs). Our result is estabished with two lemmas.

Lemma 7-9: Let $\mathbf{B}$ be a border vector with respect to work assignment graphs $WAG(1)$ and $WAG(2)$, and let $HB(\mathbf{B},j,c)$ be hyper-border graphs, $j = 1,2$. If node H has no predecessors in $HB(\mathbf{B},j,c)$ then $l_x(H) = l_b(H)$.

Proof: Suppose that H has no predecessors in $HEG(WB,j,c)$, so that $l_x(H) = w(H)$. If H has incoming arcs in $HB(\mathbf{B},j,c)$ then H represents a border node. These arcs are weighted by zero since H is a root in $HEG(WB,j,c)$; thus $l_b(H) = w(H)$, so $l_b(H) = l_x(H)$. Likewise, if H has no incoming arcs in $HB(\mathbf{B},j,c)$ then $l_b(H) = w(H) = l_x(H)$; thus if H is a root in $HEG(WB,j,c)$ the conclusion is established.

Suppose then that H has predecessors in $HEG(WG,j,c)$. Let W be the work node represented by H; we claim that W is a border node. If W has incoming arcs in $BG(j)$ then those arcs were removed in the transformation to $HB(\mathbf{B},j,c)$: targets of removed arcs always represent border nodes. But if W is a root in in $BG(j)$, then W is a border node, otherwise lemma 7–8 is contradicted. Thus H represents a border node.

H has one of three forms. Suppose that $H = EP(W)$. Let u be the weight on the rootless arc into H, and let v be the weight on the rootless arc into $IP(W)$. Since $w(H) = w(IP(W)) = 0$ we have $l_b(H) = \max\{u, v\}$. But u and v are defined so that $\max\{u, v\}$ is the time that W releases the processor under hyper–message synchronization. By lemma 4–4, $l_x(H)$ is identically this time. Thus $l_x(H) = l_b(H)$.

The arguments for cases $H = IP(W)$ and $H = R(W)$ are quite similar. The rootless arc weights are purposely defined so that $l_x(H) = l_b(H)$. This completes the lemma's proof.

□

Lemma 7–9 establishes the induction base for the following lemma.

Lemma 7-10: Let $\mathbf{B}$ be a border vector with respect to work assignment graphs $WAG(1)$ and $WAG(2)$, and let $HB(\mathbf{B},j,c)$ be hyper–border graphs, $j = 1,2$. For every node H in $HB(\mathbf{B},j,c)$, $l_x(H) = l_b(H)$.

Proof: We induct on a topological sorting of $HB(\mathbf{B},j,c)$'s nodes. Lemma 7–9 es–tablishes the induction base. For the induction hypothesis, let H be a node in $HB(\mathbf{B},j,c)$ with a non–empty collection of predecessors $P_1,\cdots,P_k$ such that each P_i satisfies the induction statement. Suppose that H represents work node

W. W is not a border node, otherwise H would have no predecessors in $HB(\mathbf{B},j,c)$. Then lemma 7–8 states that all of W's predecessors in $WAG(j)$ appear in $BG(j)$; since the transformation of $BG(j) \rightarrow HB(\mathbf{B},j,c)$ is identical with transformation $WAG(j) \rightarrow HEG(WG,j,c)$ with regard to non–border nodes, H has exactly the same predecessors in $HEG(WG,j,c)$ as it does in $HB(\mathbf{B},j,c)$.

Consider H's predecessor P_i. We have

$$l_x(P_i, H) = l_x(P_i) + e(P_i, W) + w(H).$$

By the induction hypothesis, $l_x(P_i) = l_b(P_i)$. Since the arcs and weights are identical in $HEG(WG,j,c)$ and $HB(\mathbf{B},j,c)$, we have $l_x(P_i, H) = l_b(P_i, H)$. Then

$$\begin{aligned} l_x(H) &= \max_{i \leqslant k}\{l_x(P_i, H)\} \\ &= \max_{i \leqslant k}\{l_b(P_i, H)\} \\ &= l_b(H). \end{aligned}$$

This completes the induction.

□

Corollary 7-1: The length of the longest path through $HB(\mathbf{B},j,c)$ is equal to the cycle execution time of cycle c under partition Ψ_j and hyper–message synchronization.

□

Every observed cycle's execution time under hyper–message synchronization can be found as the length of the longest path through a hyper–border graph. The length of this path depends on the weights placed on the rootless initiation and release arcs. It is easily shown that the longest path through a directed acyclic graph is an increasing function of the node and arc weights. We state this as a lemma.

Lemma 7-11: Let G be a directed acyclic graph with weighted arcs and nodes. The longest path through G is an increasing function of all the arc and node weights.

Proof: It is directly shown by induction on a topological sorting of nodes that the longest path to a node is an increasing function of the arc and node weights of all its ancestors.

□

In particular, the longest path through a hyper–border graph is an increasing function of the weights on the rootless arcs. By construction, there are two rootless arcs associated

with every border node. One arc is weighted by the border node's execution (or hyper-execution) initiation time, the other is weighted by the arrival time of the message releasing the processor to the node. This collection of rootless arc weights is conveniently expressed as a vector of real values.

Definition 7-7: *Timing Vector*

Let WG be a work graph, and let Ψ_1, Ψ_2 be two partitions of WG. Let $\mathbf{B} = <B_1,\cdots,B_k>$ be a border vector for $WAG(1)$ and $WAG(2)$, and let c be a cycle. We define the *timing vector* $\mathbf{BT}(j,c)$, $j = 1, 2$ as follows. For $i = 1,2,\cdots,k$, the ith component of $\mathbf{BT}(j,c)$ is equal to the weight on the rootless initiation arc directed to either $R(B_i)$ or $IP(B_i)$ in $HB(\mathbf{B},j,c)$. The $(k+i)th$ component is equal to the weight on the rootless release arc directed to either $R(B_i)$ or $EP(B_i)$ in $HB(\mathbf{B},j,c)$.

□

We can now show that the performance of partitions Ψ_1 and Ψ_2 on cycle c might be compared by analyzing their respective timing vectors $\mathbf{BT}(1,c)$ and $\mathbf{BT}(2,c)$.

Lemma 7-12: Let

$$\mathbf{BT}(1,c) = <b_{11}(c),\cdots,b_{1(2k)}(c)>$$

and

$$\mathbf{BT}(2,c) = <b_{21}(c),\cdots,b_{2(2k)}(c)>$$

be timing vectors for partitions Ψ_1 and Ψ_2 on some cycle c. If $b_{1i}(c) \geqslant b_{2i}(c)$ for all $1 \leqslant i \leqslant 2k$, then the (hyper-message synchronization) cycle execution time for c under Ψ_1 is greater than or equal to the cycle execution time under Ψ_2.

Proof: Lemma 7-11 implies that the length of the longest path through $HB(\mathbf{B},j,c)$ is an increasing function of its arc and node weights. The only difference between $HB(\mathbf{B},1,c)$ and $HB(\mathbf{B},2,c)$ is the weights given to the rootless arcs. By assumption, the weight given to a particular rootless arc in $HB(\mathbf{B},1,c)$ is greater than or equal to the weight given to that arc in $HB(\mathbf{B},2,c)$. It follows that the length of the longest path through $HB(\mathbf{B},1,c)$ is greater than or equal to the length of the longest path through $HB(\mathbf{B},2,c)$. By corollary 7-1, the length of the longest path through $HB(\mathbf{B},j,c)$ is equal to the cycle execution time of cycle c under partition Ψ_j, using hyper-message synchronization. Our conclusion follows directly.

□

If $b_{1i}(c) \geqslant b_{2i}(c)$ for all $1 \leqslant i \leqslant 2k$, we say that $\mathbf{BT}(1,c)$ *dominates* $\mathbf{BT}(2,c)$. It is intuitively obvious that if $\mathbf{BT}(1,c)$ dominates $\mathbf{BT}(2,c)$ on every cycle c, then the mean cycle execution time under Ψ_1 is larger than the mean cycle execution time under Ψ_2. In fact, Ψ_1's cycle execution time is $\geqslant_{st}$ than Ψ_2's.

Lemma 7-13: Let $CX(j)$ be Ψ_j's cycle execution time random variable for $j = 1,2$, and suppose that $\mathbf{BT}(1,c)$ dominates $\mathbf{BT}(2,c)$ on every cycle c. Then $CX(1) \geq_{st} CX(2)$.

Proof: In Chapter 3 we developed a Markov chain model of the simulation system whereby the states of the model completely encode the behavior of the system during a cycle. The size of this chain is finite, with states $1,2,\cdots,N$; the chain is assumed to be ergodic, with the equilibrium probability of state i denoted by Q_i. Now let $g:R \rightarrow R$ be any increasing function. Then

$$E[g(CX(1))] = E_c\left[E_{CX|c}[g(CX(1)) \mid \text{system cycle } c]\right]$$

where E_c is the expectation with respect to the system states' equilibrium probabilities, and $E_{CX|c}$ is the expectation of the cycle execution time given the cycle specification. However, if the cycle is specified, the cycle execution time is specified. That is,

$$E_{CX|c}[g(CX(1)) \mid \text{system cycle } c] = g(CX(1,c))$$

where $CX(1,c)$ denotes the execution time of cycle c under Ψ_1. Since $\mathbf{BT}(1,c)$ dominates $\mathbf{BT}(2,c)$ on every cycle c, we conclude by lemma 7-11 that $CX(1,c) \geq CX(2,c)$ for every cycle c. Since g is increasing, we must then have that $g(CX(1,c)) \geq g(CX(2,c))$ for every cycle c. Thus,

$$\begin{aligned} E[g(CX(1))] &= E_c\left[E_{CX|c}[g(CX(1)) \mid \text{system cycle } c]\right] \\ &= \sum_{c=1}^{N} Q_c \cdot g(CX(1,c)) \\ &\geq \sum_{c=1}^{N} Q_c \cdot g(CX(2,c)) \\ &= E[g(CX(2))]. \end{aligned}$$

□

The hypothesis that $\mathbf{BT}(1,c)$ dominates $\mathbf{BT}(2,c)$ on every cycle c is rather strong. However, this result is useful in a statistical context. If we have a large number of observations and find $\mathbf{BT}(1,c)$ dominating $\mathbf{BT}(2,c)$ for all or most of the observed cycles c, we are then reasonably confident that Ψ_2 is the better partition.

7.5.3. Another Look at $\geq_{jl}$

We now consider what inferences can be made when $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$, where $\mathbf{BT}(j)$ is Ψ_j's timing vector considered as a random vector. We would ideally prove that this relation implies that the mean cycle execution time under Ψ_1 exceeds that under Ψ_2. While this may be true in practice, we immediately run into serious difficulties trying to prove such a result. This difficulty's central cause lies in the correlation of the timing

vector's values and the hyper–border graph's topology. The translation of differences between timing vectors into differences in cycle execution times depends intimately on the structure of the hyper–border graph. When $\mathbf{BT}(1,c)$'s component values tend to be larger than $\mathbf{BT}(2,c)$'s, it is conceptually possible that the hyper–border structure will diminish the effect of these inequalities on the cycle execution times, so that Ψ_1's cycle execution times aren't very much larger than Ψ_2's. To compliment this, suppose that when $\mathbf{BT}(2,c)$'s component values tend to be larger than $\mathbf{BT}(1,c)$'s, the associated hyper–border graph accentuates these differences, making Ψ_2's cycle execution times much larger than Ψ_1's. The combination of these situations could cause Ψ_2's cycle time mean to exceed Ψ_1's. However, this worst case scenario is counter–intuitive. We understand that partitions' timing vectors differ because one partition is more effective than another. We expect that this effectiveness transcends the hyper–border graph topology.

We have just noted that timing vector values and hyper–border topologies are correlated. We nevertheless consider a function which assumes these entities are independent. This function chooses a system cycle at random in accordance with the system cycle process equilibrium probabilities. The function independently selects a timing vector at random, in accordance with the partition's timing vector components joint probability distribution. The chosen cycle defines a hyper–border graph topology; the chosen time vector defines the weights on that graph's rootless arcs. The length of the longest path through the resulting hyper–border graph is the cycle execution time if the chosen timing vector had been observed with the chosen hyper–border topology. Clearly, this function allows unrealizable combinations of timing vector and cycles. However, we can still use this function to compare two partitions by redefining our standard of comparison. We let $F(c, BT)$ denote the length of the longest path through the hyper–border graph defined by cycle c and timing vector BT. We let $\mathbf{V}$ be the random variable of the chosen cycle. Then $F(\mathbf{V}, \mathbf{BT}(j))$ is the random "execution" time of this function using the timing vector distribution caused by partition Ψ_j. Our modified standard of comparison states that Ψ_2 is a better partition than Ψ_1 if $E[F(\mathbf{V}, \mathbf{BT}(1))] \geq E[F(\mathbf{V}, \mathbf{BT}(2))]$. We can show that this inequality holds whenever $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$.

Lemma 7-14: If $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$, then $E[F(\mathbf{V}, \mathbf{BT}(1))] \geq E[F(\mathbf{V}, \mathbf{BT}(2))]$.

Proof: By the laws of conditional expectation,

$$E[F(\mathbf{BT}(1))] = E_c \left| E[F(c, \mathbf{BT}(1)) \mid \text{cycle is } c \ ] \right|$$

where E_c is the expectation with respect to the system cycle. By assumption, the selection of the cycle (and hence the hyper–border topology) are independent. Thus

$$E_c \left| E_{F|c}[F(\mathbf{BT}(1)) \mid \text{cycle is } c \ ] \right| = E_c \left| E[F(c, \mathbf{BT}(1))] \right|$$

$$= \sum_{c=1}^{N} Q_c \cdot E[F(c\ , \mathbf{BT}(1))]$$

where F_c denotes the function calculating the longest path through the hyper–border graph whose topology is established by cycle c. However, lemma 7–11 implies that $F(c,BT)$ is an increasing function of the components of the timing vector BT. Since $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$, we must have

$$E[F(c,\mathbf{BT}(1))] \geq E[F(c,\mathbf{BT}(2))]$$

for every c. Thus

$$\sum_{c=1}^{N} Q_c \cdot E[F(c,\mathbf{BT}(1))] \geq \sum_{c=1}^{N} Q_c \cdot E[F(c,\mathbf{BT}(2))]$$

$$= E[F(\mathbf{BT}(2))].$$

□

While the knowledge that $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$ does not prove that partition Ψ_2 yields smaller cycle execution times, we do have a context in which Ψ_2 yields smaller expectations of measurements very much like cycle execution times. Even so, we must face the fact that $\mathbf{BT}(1) \geq_{jl} \mathbf{BT}(2)$ is a very strong mathematical statement whose veracity we cannot generally hope to establish. But again, we must consider our application's context. If statistical observations tend to support the sufficient conditions for $\geq_{jl}$ established in lemma 7–4 , we can reasonably expect Ψ_2 to be better than Ψ_1. We might extend this argument to situations where the observations support only the **necessary** condition for $\geq_{jl}$ given by lemma 7–4. This condition is quite strong in itself. Our inability to prove its sufficiency in the general case stems again from the possibility of counter–intuitive correlation between components of the timing vector.

7.5.4. A Numerical Example

We now illustrate by example how our proposed qualitative approach can be used. We suppose that a border vector of interest has two border nodes, $\mathbf{B} = <B_1, B_2>$. The timing vectors $\mathbf{BT}(1)$ and $\mathbf{BT}(2)$ thus have four components. Let I_{ji} denote B_i's execution initiation time under Ψ_j, and let R_{ji} denote B_i's release signal arrival time under Ψ_j. We will assume that the random variables associated with a partition are independent; for example, we suppose that I_{11}, I_{12}, R_{11} and R_{22} are all independent. In section 7.3, we proved that if the components of a vector $\mathbf{X} = <X_1, \cdots, X_n>$ are independent and the components of a vector $\mathbf{Y} = <Y_1, \cdots, Y_n>$ are independent, then the necessary condition for $\geq_{jl}$ given in lemma 7–4 is also sufficient. Under this assumption of independence, the necessary and sufficient condition is equivalent to saying that $X_i \geq_{st} Y_i$ for each $i = 1, \cdots, n$. We present a numerical example in which we test for the conditions $I_{1i} \geq_{st} I_{2i}$ and $R_{1i} \geq_{st} R_{2i}, i = 1,2$.

To illustrate qualitative analysis, we randomly generated values for the border nodes' execution initiation and processor acquisition times under two partitions. Figure 7–2 illustrates two tables. The first table illustrates the simulated times; the data for random variables we compare are enclosed in double barred columns. We associated two probability distributions with each random variable. One distribution governs the random variable when its associated border node is evaluated, the other governs when the border node is hyper–executed. The decision whether the node was evaluated is also random. The randomly generated data were adjusted to be consistent with the role they play. If B_i was assumed to be evaluated, then the I_{1i}, R_{1i}, I_{2i}, and R_{2i} observations were generated from the distributions governing evaluated node observations. Also, if B_i is evaluated, then we ensured that R_{ji} is less than or equal to I_{ji}. However, if B_i is not evaluated, its hyper–execution can precede the arrival of its incoming release message, so that R_{ji} can be either greater than or less than I_{ji}.

The 20 simulated observations shown are actually a subset of 50 generated observations. The second table in figure 7–2 reports the results of some simple analysis of these 50 observations. In lemma 7–3 we saw that $X \geq_{st} Y$ if and only if $Prob\{X > t\} \geq Prob\{Y > t\}$ for all $t > 0$. For each pair of compared random variables, we attempted to determine whether the compared variables observations support

20 Border Vector Observations								
obs	I_{11}	I_{21}	I_{12}	I_{22}	R_{11}	R_{21}	R_{12}	R_{22}
1	3.485	3.221	4.751	4.435	3.485	3.097	4.751	3.621
2	3.388	1.969	4.528	4.544	3.388	1.787	4.528	3.854
3	3.292	2.978	4.058	3.590	2.992	2.623	3.170	3.302
4	0.864	0.896	1.799	1.726	1.619	1.200	1.739	1.586
5	1.091	0.582	4.958	4.769	1.265	1.508	4.303	2.976
6	0.963	0.841	4.167	3.838	1.465	1.161	3.483	3.838
7	2.927	2.987	1.803	1.660	2.927	2.987	1.799	1.102
8	1.024	0.916	4.519	3.922	1.384	1.418	4.519	3.922
9	1.048	0.907	1.938	1.550	1.233	1.385	1.524	1.411
10	3.901	3.253	1.821	2.034	2.143	3.253	1.752	1.351
11	3.462	3.641	4.380	3.962	3.389	2.472	4.380	3.962
12	3.257	3.068	4.856	3.213	3.257	1.797	4.044	3.213
13	3.843	3.638	4.009	3.676	2.745	3.638	3.013	3.676
14	1.221	1.050	2.234	1.620	1.277	1.326	1.784	1.007
15	0.963	0.808	2.107	1.752	1.554	1.569	1.215	1.286
16	3.317	2.953	1.789	1.786	2.706	2.148	1.745	1.441
17	3.132	3.196	3.849	4.460	2.061	3.196	3.639	4.460
18	2.883	2.503	2.058	1.939	2.380	2.066	1.584	1.489
19	1.223	0.862	2.111	1.917	1.279	1.553	1.494	1.117
20	3.012	2.761	4.662	4.725	3.001	2.668	4.662	4.486

Probability of Exceeding *kth* Fixed Threshold								
k	I_{11}	I_{21}	I_{12}	I_{22}	R_{11}	R_{21}	R_{12}	R_{22}
1	0.86	0.68	0.86	0.70	0.88	0.80	0.90	0.74
2	0.56	0.56	0.60	0.58	0.76	0.56	0.62	0.58
3	0.56	0.56	0.58	0.58	0.56	0.48	0.58	0.58
4	0.56	0.56	0.58	0.58	0.48	0.38	0.58	0.58
5	0.54	0.52	0.58	0.56	0.44	0.34	0.58	0.56
6	0.54	0.46	0.58	0.50	0.36	0.24	0.50	0.48
7	0.44	0.40	0.50	0.30	0.20	0.18	0.38	0.38
8	0.36	0.18	0.36	0.20	0.18	0.08	0.30	0.20
9	0.16	0.12	0.18	0.10	0.04	0.06	0.16	0.08

Example Data

Figure 7–2

this relation. We describe our procedure in terms of I_{11} and I_{21}. We pooled all observations of I_{11} and I_{21}, and identified the maximum value M and the minimum value m. We let $\Delta = \frac{(M - m)}{10}$; then for $k = 1,2,\cdots,9$, we call the value $m + k\cdot\Delta$ the *kth fixed threshold* for I_{11} and I_{21}. The fixed threshold values place the role of t in lemma 7–3 : we ask whether the data supports the hypothesis that

$$Prob\{I_{11} > m + k\cdot\Delta\} \geqslant Prob\{I_{21} > m + k\cdot\Delta\}$$

for each fixed threshold $m + k\cdot\Delta$. For each k and random variable X, we estimate $Prob\{X > m + k\cdot\Delta\}$ with the fraction of X's observations which do exceed $m + k\cdot\Delta$.

The results of this computation are reported in the second table shown in figure 7–2. It appears that our observations support the hypothesis that $I_{11} \geqslant_{st} I_{21}$, $I_{12} \geqslant_{st} I_{22}$, $R_{11} \geqslant_{st} R_{21}$, and $R_{12} \geqslant_{st} R_{22}$. Assuming independence, we surmise that $\mathbf{BT}(1) \geqslant_{jl} \mathbf{BT}(2)$; we appeal to lemma 7–14 to conclude that partition Ψ_2 is superior to partition Ψ_1.

This analysis procedure is relatively efficient. The values of M and m are obtained in one scan of the data; the fixed threshold values are then readily calculated. For each observation, the greatest k such that the observation exceeds $m + k\cdot\Delta$ can be determined in $O(\log_2(9))$ time with a binary search. Given this k, at most 10 estimations of the probabilities $Prob\{X > k\cdot\Delta\}$ will then need to be adjusted. With $\mathbf{d}$ data observations for each random variable, we can judge whether the data supports $X \geqslant_{st} Y$ in $O(2\cdot\mathbf{d}\cdot 10\cdot\log_2(9))$ time. Ignoring the constants, we may say that this computation is linear in the number of data observations. If the border vector has n components, then the complexity of determining whether each random variable pair satisfies $\geqslant_{st}$ is linear in $n\cdot\mathbf{d}$.

The proposed procedure is merely a heuristic. It would perhaps be better to use a more formal statistical test for $X \geqslant_{st} Y$, if such a test were computationally efficient. We have not discovered any formal statistical means of testing the hypothesis $X \geqslant_{st} Y$ in the literature. Our general qualitative comparison scheme in a statistical setting might well benefit from such a test.

7.5.5. Use of General Qualitative Comparison

We conclude our development of general qualitative partition comparison by considering appropriate circumstances for its use. Our ability to compare two partitions through timing vectors depends on the border graph being essentially a subgraph of both partitions' work assignment graphs. The two partitions are identical "beyond" the border, so that an analysis of the border's behavior may indicate that one partition is superior. Of course, to quantify the border timing vectors we must perform the same analysis as performed in the moment estimation procedure, up to the border vector. The general qualitative approach is useful when the analysis at the border is less time consuming than the more straightforward estimation of cycle execution time parameters. When the difference between two partitions is small, the border vector may be small. For example, the difference between partitions in Figure 7–1 is reasonably small, and the border vector has only two nodes.

The general moment estimation procedure carries its analysis into a region of the work assignment graph not considered in the border analysis. This region is encompassed by the border graph. The analysis is just "longest path" analysis, known to have time complexity equal to the sum of the graph's nodes and edges. If we let $\mathbf{S}_B$ be the sum of nodes and edges in the border graph, then the analysis performed by the general moment estimation procedure which is **not** performed by the general qualitative approach has time complexity $O(\mathbf{S}_B\cdot\mathbf{d})$. We showed in the last section that the time complexity of the illustrated analysis procedure was $O(n\cdot\mathbf{d})$, n being the length of the border vector. We

generally expect that $n << S_B$. Comparing the two procedures' complexity orders is not always fair; however this comparison does suggest that computational savings are possible using general qualitative comparison. The most promising situation for achieving these savings is when we analyze a small change in a large work assignment graph. This is exactly the situation we might encounter trying to improve a constructed partition by considering small perturbations in the partition.

7.6. Chapter Summary

Throughout this thesis we have developed a understanding of the problems facing a partitioning algorithm. We first observed that the behavior of the simulation varies from cycle to cycle; we then saw that distributed simulations pose special synchronization problems, and we developed graphical techniques for describing synchronization. We studied analytic methods of determining a partition's performance, and found that in general we must make restrictive assumptions if we are to progress with such analysis. We are left then with statistical methods of determining a partition's performance. These statistical methods estimate a partition's cycle execution time moments; as such, they are **quantitative** in nature. In this chapter we developed a **qualitative** analysis technique. We motivated this investigation with the realization that the quantitative techniques can be computationally prohibitive in a situation where many minor perturbations of a partition are being considered. We showed how stochastic order relations provide us with the qualitative tools we need. We then developed a theory of border vectors, vectors of work nodes which capture all message timing differences between two partitions. We showed how two partitions are compared by analyzing the behavior of the border vector. We illustrated the application of this technique with a numerical example, and further strengthed the case for qualitative comparison by showing that the time complexity of a qualitative analysis can be significantly less than the time complexity of general cycle execution time moment estimation.

Chapter 8

A Partitioning Heuristic

8.1. Chapter Overview

Previous chapters have examined issues in modeling and analyzing a distributed simulation system. The analysis developed in those chapters is intended to support the partitioning of a simulation. In this chapter we propose a partitioning algorithm; in doing so, we show how some of our analytic tools can be used. Finding an **optimal** partition is an intractable problem; the algorithm we present is a heuristic. We report the results of tests performed on the partitions produced by this algorithm, and argue that the algorithm produces effective partitions.

This chapter is divided into 3 additional sections. The next section provides an overview of the partitioning problem, and examines important concerns any reasonable partitioning algorithm will have to address. Following that, we propose a partitioning algorithm based on the behavior of observed cycles. This heuristic addresses the concerns identified by section 8.2. The final section discusses an empirical study of partitions produced by this algorithm.

8.2. Problem Overview

In Chapter 2 we developed a formal model of simulations, defined in terms of event functionals. The event functionals are evaluated to produce the simulation's results. An event functional may then be thought of as a software module, and its evaluation may be thought of as that module's execution. Definition 4-1 formally defines a partition. A partition is intuitively understood to be an assignment of each event functional to a processor in a multi-processor computer system. Every evaluation of a functional is performed in the functional's assigned processor. Our goal is to find a partition of a simulation which effectively reduces the mean execution time of a simulation cycle over that of a sequential simulation. This problem is an extension of a well-known NP-complete multi-processor scheduling problem: find a multi-processor schedule for a task system which minimizes the system execution time [Cof76] . It follows that finding an optimal partition for a simulation is at least as computationally intractable as this scheduling problem.

This thesis has shown that there are a number of concerns a partitioning algorithm will have to consider; including the following.

(i) The amount of execution required during a cycle will vary from cycle to cycle. To construct a partition which reduces the cycle execution time, we will thus have to consider the **frequency** of a functional's evaluation, as well as its execution delay.

(ii) A distributed simulation needs to continually synchronize its processors. A partitioning algorithm will have to consider the effect of this syn-

chronization on the cycle execution time.

(iii) There is a non-zero communication delay between processors. The partitioning algorithm must consider the effect of placing two communicating functionals in different processors.

An important observation in connection with concern (iii) is that some communications are more critical than others. We illustrate this point by example.

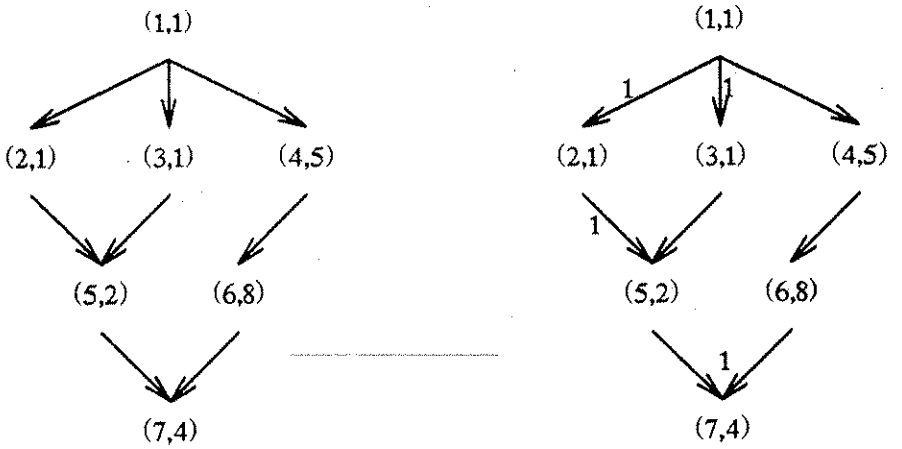
Figure 8-1*a* illustrates a directed acyclic graph with weighted nodes; this graph is interpreted as a task graph whose arcs define precedence, and whose weights define execution delays. (i,j) denotes node i with weight j. The path with the longest length through this graph is the sequence 1, 4, 6, 7, with length 18. Now suppose we partition this graph into three sets of nodes. Whenever node i sends an arc to node k, and i and k are assigned to different sets, we weight the arc with 1, a communication delay. We will also say that i and k are *separated*. The length of the longest path through the resulting graph is equal to the finishing time of the partitioned task system. Note that we cannot separate nodes on the critical path without increasing the longest path length. However, we can separate nodes 1 and 2 even though they communicate and still not increase the resulting graph's critical path length over the original graph. Figure 8-1*b* depicts an assignment which does not increase the critical path length beyond 18; figure 8-1c depicts an assignment which does increase this path length. The overall finishing time of the system is apparently more sensitive to the separation of some communicating nodes than it is to others.

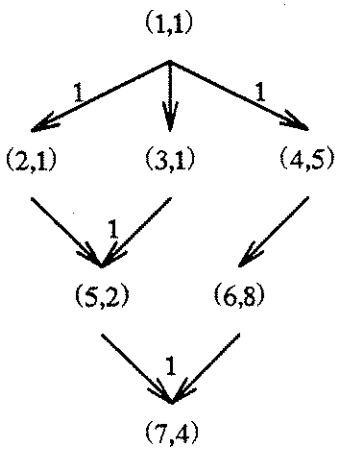
This example illustrates the importance of considering how **critical** a potential inter-processor communication is to the timely execution of the task system. We will see that our proposed partitioning heuristic considers the importance of particular communications, the effect of synchronization, and the frequency of a functional's evaluation.

8.3. A Partitioning Algorithm

In this section we describe an algorithm for constructing partitions. This algorithm is motivated by a number of the observations made in this thesis. It is a *statistical partitioning algorithm*, the partitions it constructs are based on a number of cycle observations. Our algorithm considers the effect of communication delays, potential concurrency, synchronization, and frequency of a functional's evaluation.

Suppose that we have M independent cycle observations of some simulation system. The first phase of our partitioning algorithm considers each observed cycle separately. In Chapter 4 we defined a simulation's work precedence graph; a work precedence graph is a work assignment graph with the execution arcs removed. The work precedence graph thus expresses only the logical precedence inherent in the simulation. Every potential functional evaluation has a representing work node in the work predecence graph WPG. An observed cycle can be thought of as a collection of functional evaluations and their logical initiation times. We weight each work precedence graph node which corresponds to an observed evaluation with the evaluation's execution delay. All other nodes are weighted by zero. We then find the critical path length through this weighted work precedence graph. Given the critical path length, the *earliest starting time* and the *latest starting time* [SaH76] of each node is determined. The earliest starting time of a node W is the earliest time at which W's execution can be initiated, subject to the precedence expressed by the graph. W's latest starting time is the latest time at which W's execution can begin without increasing the length of the critical path. Figure 8-2 illustrates the graph of figure 8-1a with the calculated earliest and latest starting times. The pair $<s,t>$ next to a node reflects that the node's earliest starting time is s, and its latest





Partitions of a Task System

Figure 8–1

starting time is t. The time complexity of determining the critical path length and each node's earliest and latest starting times is $O(\mathbf{S} + \mathbf{E})$ where $\mathbf{S}$ is the number of nodes and $\mathbf{E}$ is the number of edges in the work precedence graph.

The length of a critical path through the weighted work precedence graph is the absolute minimum execution time for the observed cycle. Suppose that we wanted to partition the simulation so that every time this cycle occurred, the cycle execution time is equal to this critical path length. The node's earliest and latest starting times can be used to derive **necessary** (but not sufficient) conditions on any partition which is optimal on the observed cycle. Given every node's earliest and latest starting times, we calculate *minimal overlaps*, *maximal overlaps*, and *minimal delays*. Let W and V be any two evaluated nodes in WPG, and suppose that neither node constrains the execution of the other (which is equivalent to saying that there is no path in WPG between W and V). Let $e(W)$ and $e(V)$ be the nodes' respective earliest starting times; let $l(W)$ and $l(V)$ be their latest starting times. We calculate the quantity

$$mno(W, V) = \min_{s \in [e(W), l(W)],\, t \in [e(V), l(V)]} \left\{ \min\{s + X(W), t + X(V)\} - \max\{s, t\} \right\}^+$$

where $\{x\}^+$ is defined to be the maximum of x and 0, and $X(W)$ and $X(V)$ denote the execution delays of W and V. $mno(W, V)$ is called the *minimal overlap* of W and V. Given execution starting times s and t, the quantity inside of $\{\ \}^+$ above is the overlap in W and V's executions. The minimal overlap is simply the minimal possible (non-negative) overlap. If $mno(W, V) > 0$, we know that W and V are assigned to different processors in any partition yielding the WPG critical path length as the execution time for the observed cycle.

We also consider the *maximal overlap* $mxo(W, V)$ between two evaluated work nodes. The maximal overlap is equal to the maximum amount of time that nodes W and V **can be** concurrently executed, given the earliest and latest starting time constraints. $mxo(W, V)$ is thus defined as

$$mxo(W, V) = \max_{s \in [e(W), l(W)],\, t \in [e(V), l(V)]} \left\{ \min\{s + X(W), t + X(V)\} - \max\{s, t\} \right\}^+$$

It is quite possible for $mno(W, V) = 0$ and $mxo(W, V) > 0$. This happens when one or both of the nodes have enough slack (difference between earliest and latest starting times) so that their executions don't have to overlap, even though they can overlap. To quantify **both** of these overlap possibilities, we define the *weighted overlap* to be the average of the minimal and maximal overlaps:

$$wo(W, V) = \tfrac{1}{2}(\, mno(W, V) + mxo(W, V)\,).$$

The concept of a *minimal delay* is related to the minimal overlap. Suppose that W and V are evaluated work nodes such that V's evaluation depends directly on the result of W's evaluation. We define the *minimal delay* function $md(W, V)$ by

$$md(W, V) = \{e(W) + X(W) + C - l(V)\}^+.$$

$md(W, V)$ measures the overlap between the earliest possible arrival time of a signal sent from W to V, and V's latest possible starting time. This calculation considers the effect of placing W and V in different processors, so that a communication delay C is incurred by the signal. If $md(W, V) > 0$, we know that W and V are assigned to the same processor in any partition yielding the WPG critical path length as the cycle execution time.

The values $mno(W, V)$ or $md(W, V)$ indicate whether it is necessary to assign W and V to the same processor to achieve the critical path as the observed cycle's execution

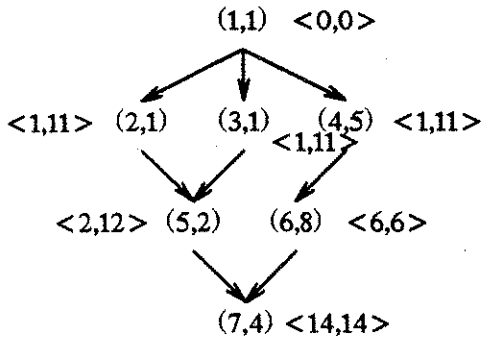
time. Furthermore, the **magnitude** of $wo(W, V)$ and $md(W, V)$ are a measure of how strongly we should consider placing W and V in different processors (in the case of $wo(W, V)$) or how strongly we should consider placing W and V in the same processor (in the case of $md(W, V)$). We use this observation to modify the work precedence graph yet again, creating the *overlap/delay* graph. The overlap/delay graph's nodes are identical to the work precedence graph's nodes. Then for every node pair W and V, we place an undirected edge between W and V weighted by $md(W, V)$ if $md(W, V)$ is defined and is greater than 0. Similarly, we place an undirected edge between W and V weighted by $-wo(W, V)$ if no path exists in WPG between W and V, and if $wo(W, V) > 0$. The negative sign is used to distinguish this weight from a minimal delay weight. Figure 8–3 illustrates the transformation of the graph in figure 8–2 into an overlap/delay graph.

Finally, we transform the overlap/delay graph into a *concurrency/dependency* graph, or *CD* graph. The nodes of a *CD* graph are the event functionals. The edges of a *CD* graph are a function of the overlap/delay graph's edges. An edge in the *CD* graph exists between functionals T_i and T_j if in the overlap/delay graph, some work node associated with T_i shares an edge with some work node associated with T_j. This edge is weighted by the sum of all edge weights between work nodes associated with T_i and work nodes associated with T_j in the overlap/delay graph.

The next stage of the algorithm is to create a single graph which represents all of the observed cycles. We assume that a *CD* graph has been constructed for every observed cycle. Given M cycle observations, the M different *CD* graphs can be merged into a single *collective CD* graph. The nodes of this collective graph are again the event functionals. An edge will exist between T_i and T_j in the collective *CD* graph if any one of the *CD* graphs defines an edge between T_i and T_j. The weight on an edge between T_i and T_j is equal to the average weight placed on this edge in the *CD* graphs. That is, if $W(T_i,T_j,k)$ denotes the weight on the edge between T_i and T_j in the *kth* cycle observation's *CD* graph, and if $W(T_i,T_j,k)$ is defined to be zero if no such edge exists, then an edge from T_i to T_j in the collective *CD* graph is weighted by $\frac{1}{M}\sum_{k=1}^{M} W(T_i,T_j,k)$.

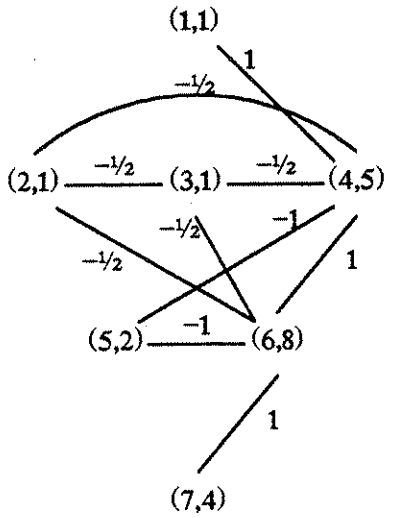
The construction of the collective *CD* graph does consider each of the points raised by section 8.2. The work precedence graph defines logical precedence which is intrinsic to the simulation. The synchronization required by a distributed simulation is a function of this precedence. By using critical path analysis of the work precedence graph, we are considering the synchronization constraints of a distributed simulation. The construction of the collective *CD* graph also considers the relative importance of communication delays; the $md(W, V)$ metric measures the criticality of communication between evaluated work nodes. The frequency of a functional's evaluation also plays a role in the construction of the collective *CD* graph. The $-wo(W, V)$ and $md(W, V)$ edge weights are defined only for evaluated W and V. Since the weights on the collective *CD* graph are ultimately a function of the measured overlaps and delays, we see that the collective *CD* graph does implicitly encode the functionals' frequency of evaluation.

Our next step is to partition the collective *CD* graph's nodes into as many sets as we have processors. All the functionals within a set are assigned to the same processor. The partitioning decision is based on the edge weights found in the collective *CD* graph. Consider again the fact that the overlap/delay graph edge weights defined by the minimal overlap function are negative, while the weights defined by the minimal delay function are positive. Thus, an edge between functionals in the collective *CD* graph gives us some evidence indicating whether the functionals should be placed in the same processor, or in separate processors. An edge weighted with a large negative number implies that the functionals should be separated; an edge weighted with a large positive number implies that the functionals should be placed in the same processor. This observation suggests that we partition the collective *CD* graph so that the sum of the weights on cut edges is



Earliest and Latest Starting Times

Figure 8–2



An Overlap/Delay Graph

Figure 8–3

minimized. An edge is cut if its two functionals are placed in different processors.

The K–partition problem in graph theory is to partition the nodes of an edge–weighted graph into K sets such that the sum of cut edges is minimized. This problem is known to be NP–complete; many heuristics have been suggested in the context of design automation [FiM82, KeL70, Kri84]. We attempted to use such heuristics to partition the collective *CD* graph, but did not achieve good partitions. We then realized that instead of trying to minimize the sum of cut edges, we should use the edge weights to guide the partitioning process in exploiting what we know about this problem. In particular, we know that nodes which tend to be on a critical path should be placed in the same processor. To apply this knowledge, we modified a K–partitioning heuristic studied in [Los85].

We define a *cluster* of functionals to be any collection of functionals already assigned to a particular processor. Given a cluster, and an unassigned functional T_i, we define the *attraction* between the cluster and functional to be the sum of the weights on edges shared by T_i with functionals in the cluster. Let $W(T_i,T_j)$ be the weight on an edge between T_i and T_j, and let C_k be a cluster. Then the attraction of C_k and T_i, denoted $A(C_k,T_i)$, is given by

$$A(C_k, T_i) = \sum_{T_j \in C_k} W(T_i,T_j).$$

Our heuristic's fundamental idea is to merge the cluster–functional pair with the highest attraction. Suppose first that we have somehow selected K functionals as *cluster seeds*; we briefly defer describing the selection of these seeds. Every cluster seed is defined to be a cluster. The attraction of every unassigned functional to every cluster is calculated. Now the algorithm iterates; at each step, the cluster–functional pair with highest attraction is selected, and the functional is assigned to that cluster. Suppose that functional T_i is assigned to cluster C_k. If T_j is any unassigned functional which shares an edge with T_i, then T_j's attraction with cluster C_k is augmented with the weight of the edge shared by T_i and T_j. The algorithm then chooses another cluster–functional pair to merge. The algorithm terminates when every functional has been assigned to some cluster. Since clusters and processors are synonymous, the clusters define our partition.

The initial selection of the K seed nodes can be performed in an analogous manner. First, the two functionals with the **lowest** attraction are chosen to form an anti–cluster. The attraction of every functional with this anti–cluster is calculated. The anti–cluster is augmented successively with the functional having lowest attraction until K functionals reside in the anti–cluster. These K functionals are taken to be the cluster seeds.

This heuristic does exploit the knowledge that functionals on a critical path should be placed in the same processor. Such *critical functionals* will tend to share edges with relatively large positive weights. These weights will boost the attraction between a critical functional and a cluster which contains other critical functionals. Any negative weights on edges shared by a critical functional with functionals in a different cluster will decrease the attraction between the critical functional and the cluster. We have observed our algorithm's step by step construction of partitions, and have seen that nodes which tend to be on critical paths are merged together first.

Before discussing an empirical study of our algorithm's performance, we consider its overall time complexity. As stated before, the complexity of finding a critical path and all earliest and latest starting times is $O(\mathbf{S} + \mathbf{E})$, where $\mathbf{S}$ is the number of work nodes, and $\mathbf{E}$ is the number of graph edges. Given M cycle observations, the complexity of this step over all observed cycles is $O(M \cdot (\mathbf{S} + \mathbf{E}))$. We create the overlap/delay graph by determining the minimal delay and minimal overlap values from a node–weighted work precedence graph. This computation essentially requires the examination of each pair of evaluated work nodes, and has time complexity $O(\mathbf{S}^2)$; the overall complexity of this

step is $O(M \cdot \mathbf{S}^2)$. The transformation of an overlap/delay graph into a *CD* graph can be done in $O(\mathbf{S} + \mathbf{E})$ time; a single scan of the nodes determines their functional associations, as does a single scan of the edges. It follows that the construction of the collective *CD* graph from the individual *CD* graphs has time complexity $O(M \cdot (\mathbf{S} + \mathbf{E}))$. The step dominating the collective *CD* graph's construction is the edge weight calculation. We showed that this step has complexity $O(M \cdot \mathbf{S}^2)$.

We turn next to the complexity of our K-partitioning algorithm. Finding the two functionals sharing the most negative edge in the collective *CD* graph has complexity **e**, where **e** is the number of edges in the collective *CD* graph. The attraction of every other functional to this initial anti-cluster is determined in linear time $O(N)$, N being the number of functionals in the collective *CD* graph. These attraction values can be placed in a priority queue in $O(N)$ time. At every step, the functional having the lowest attraction is picked off the head of the priority queue, and is merged into the anti-cluster. We observed that this merger will affect the attraction value of every unassigned functional sharing an edge with the merged functional. We may then be required to calculate as many as **e** new attraction values, **e** being the maximal degree of a collective *CD* graph node. $O(\log N)$ time is required to insert a new attraction value into the priority queue, so that the overall complexity of inserting the new attraction values is $O(\mathbf{e} \cdot \log N)$. The anti-cluster construction phase has K iterations, so that the overall complexity of finding the K seed nodes is $O(K \cdot \mathbf{e} \cdot \log N)$. The analysis of the rest of the K-partitioning algorithm follows much the same form. Calculating the attraction of the the $N-K$ unassigned functionals to the K initial clusters requires $O(K \cdot (N-K))$ time. Let $N_K = K \cdot (N-K)$. The collection of cluster-functional attractions can be arranged in a priority queue in $O(N_K \cdot \log N_K)$ time. The algorithm then iterates $N-K$ times. Each iteration, the cluster-functional pair with highest attraction is found and merged, affecting as many as **e** attraction values. Updating the priority queue of attractions has complexity $O(\mathbf{e} \cdot \log N_K)$. Thus, the final stage of the K-partitioning algorithm has time complexity $O((N-K) \cdot \mathbf{e} \cdot \log N_K)$. For $K << N$, this complexity clearly dominates the time complexity of finding the initial K cluster seeds.

We showed that the time complexity of creating the collective *CD* graph is $O(M \cdot \mathbf{S}^2)$, and that the time complexity of partitioning the collective *CD* graph is $O((N-K) \cdot \mathbf{e} \cdot \log (K \cdot (N-K)))$. It is not possible to determine a priori which of these complexities dominates the other. In our experience, **S** tends to be less than 10 times as large as N, and $K << N$. In practice, the number of evaluated functionals is substantially less than than **S**; while the worst case behavior of creating the *CD* graph is $O(M \cdot \mathbf{S}^2)$, the average case will be substantially better. The value of **e** can be as large as $\frac{N^2}{2}$. We have used large values of M, on the order of $M = 100$. In our experience with this algorithm, the collective *CD* graph construction phase tends to dominate the time complexity.

8.4. An Empirical Study

In this section we report the results of an empirical study of partitions produced by our partitioning algorithm. We first discuss the parameters of this study, and then describe the tools used to implement the study. We outline our means of analyzing the partitions' performance; we finally present and analyze our results.

8.4.1. Study Parameters

Our study focused on two parameters for each of two simulation systems. One parameter is the communication cost C. The other is the number of processors targeted by the partition. Two values of C were used. One value is simply 0. The other value is ½ of the mean execution delay for an event functional. We created partitions for 2, 3, 4, 5, 10, 15, and 20 processor systems.

We examined two different logic network simulations. System 1 is a network describing an adder with carry look–ahead logic; the network is described in [Bae80], page 96. System 2 is an insertion permutation network described in [DaR85], page 566. Both networks are reasonably complex; System 1 has 69 event functionals, and generates 99 work nodes. System 2 has 87 event functionals, and generates 527 work nodes. While these systems are small by logic network standards, they are fairly large if viewed simply as a group of interacting simulation entities.

8.4.2. Study Tools

The performance of partitions produced by our algorithm were studied by simulating the execution of the simulation systems on a distributed system. Parameters to the simulation include logic network description, the number of processors, the partition of that network, and the communication cost between processors. Another program analyzes output from this simulator, and creates the collective concurrency/dependency graph. A final program accepts the collective concurrency/dependency graph and partitions it. These programs were all run on the University of Virginia Department of Computer Science's VAX 11/780.

8.4.3. Performance Analysis

Heuristics are often judged by the closeness of their solutions to the optimal solution. We were prohibited from such a comparison by the size of the systems we studied. Nor are there other simulation partitioning algorithms to compare ours with. We choose to compare our partitions' performance with the performance of randomly generated partitions. In addition, we calculate a lower bound on the achievable mean cycle execution time. The lower bound is quite simple. We measured the mean critical path length μ_C through the work precedence graph, and we measured the mean volume μ_V of simulation work that was executed in a cycle. A lower bound on the achievable mean cycle execution time (for the cycles from which these measurements were taken) is $\max\{ \mu_C, \frac{\mu_V}{n} \}$, where n is the number of processors. Note that this lower bound does not take into account any communication delay.

In Chapter 6 we present a statistical means of estimating the cycle execution time parameters, and of determining which of two partitions is more effective. We will use this method to compare the performance of partitions generated by our algorithm with the performance of (uniformly) random partitions. The cost function we adopt is the sum of the cycle mean and standard deviation. The prior distributions used in this analysis are the uninformed priors. According to the decision theory referred to by Chapter 6, the better partition is the one whose *expected* sum of mean and standard deviation is smallest. The expectation is taken with respect to the posterior distributions of the mean and standard deviation.

8.4.4. Study Results

The tables given below summarize the results of our simulation study. The automatically generated partitions were created from 100 cycle observations. The cycle execution time statistics were taken from a different collection of 100 cycles. In the tables below, LB denotes the lower bound on the mean cycle execution time; $E[\mu_r]$, $E[\sigma_r]$ and $E[\mu_p]$, $E[\sigma_p]$ denote the mean posterior mean and standard deviation of the random partition and our partition, respectively. We have also calculated the relative difference between the expected decision costs.

Simulation System 1							
Processors	C	LB	$E[\mu_r]$	$E[\mu_p]$	$E[\sigma_r]$	$E[\sigma_p]$	relative difference
1	–	50	–	64.5	–	19.24	–
2	0	25	31.59	30.09	6.35	6.56	0.035
3	0	16.67	27.31	21.53	5.75	3.83	0.303
4	0	12.5	19.22	18.13	3.66	3.47	0.059
5	0	10.0	18.71	15.45	3.10	3.15	0.172
10	0	6.6	16.90	10.52	2.61	2.65	0.481
15	0	6.6	11.09	8.68	1.87	2.33	0.177
20	0	6.6	11.18	8.14	2.04	1.96	0.308
2	0.5	25	37.00	35.00	6.32	6.00	0.057
3	0.5	16.67	27.11	25.65	5.48	3.60	0.114
4	0.5	12.5	25.84	23.03	5.29	3.81	0.160
5	0.5	10.0	23.53	19.88	3.86	3.17	0.188
10	0.5	6.6	18.38	16.04	2.41	1.68	0.173
15	0.5	6.6	16.90	14.32	1.76	1.73	0.163
20	0.5	6.6	15.22	13.51	1.70	2.17	0.079

The first row of the table above shows that the average simulation workload is 50, while the average running time for a sequential simulation is 64.5. This seeming discrepency is caused by the simulation's modeling of event list overhead. This overhead is not considered in the average workload figure. It would also appear that we achieve super–linear speedup with 2 and 3 processors: $30.09 < 64.5/2$ and $21.53 < 64.5/3$. This too is an artifact of modeled event list overhead. The cost of maintaining an events list is proportional to its length, so that in our model, the sequential simulation is suffering more overhead. This phenomenon is discussed in [Dav84, Rey83].

The table below reports our measurements of simulation system 2. The dynamics of system 2 are different than those of system 1 because its space of possible cycles is much larger.

Simulation System 2							
Processors	C	LB	$E[\mu_r]$	$E[\mu_p]$	$E[\sigma_r]$	$E[\sigma_p]$	relative difference
1	–	60	–	61.32	–	19.02	–
2	0	30	44.90	44.60	13.76	11.57	0.044
3	0	20	35.86	32.93	9.12	8.04	0.098
4	0	15	32.51	30.34	7.95	8.71	0.036
5	0	12	29.59	27.58	8.26	6.93	0.097
10	0	8.5	24.93	22.80	5.54	6.15	0.052
15	0	8.5	23.56	21.01	5.67	5.40	0.107
20	0	8.5	20.31	17.95	5.24	5.00	0.113
2	0.5	30	62.21	54.91	13.45	10.80	0.151
3	0.5	20	57.38	55.13	8.56	8.58	0.035
4	0.5	15	47.15	46.86	7.57	7.90	-0.001
5	0.5	12	45.42	40.58	7.61	7.69	0.099
10	0.5	8.5	40.39	38.73	6.41	7.48	0.013
15	0.5	8.5	37.48	32.16	5.05	5.51	0.129
20	0.5	8.5	35.11	33.05	4.01	3.78	0.062

The following data was measured after altering system 2's input characteristics to increase the amount of simulation activity. We also raised the communication cost to 2.

Simulation System 2: High Volume, High Communication Cost							
Processors	C	LB	$E[\mu_r]$	$E[\mu_p]$	$E[\sigma_r]$	$E[\sigma_p]$	relative difference
1	–	110	–	118.76	–	9.9	
2	2.0	55	147.72	129.17	20.89	11.21	0.201
3	2.0	36	140.85	128.73	5.56	6.61	0.082
4	2.0	27.5	132.76	91.88	3.34	4.80	0.408
5	2.0	22	130.95	85.21	3.64	4.29	0.504

8.4.5. Analysis of Results

It is difficult to judge the performance of our automatically generated partitions. The lower bound on performance is quite weak as it does not consider communication delays nor precedence (when the bound is equal to $\frac{\mu_v}{n}$). We can't expect any partition to closely follow this bound as a function of the number of processors. We do observe that in the case of system 1 with zero communication costs, our partitions' mean cycle execution time approaches this minimum well before we use as many processors as we have functionals. This shows that the critical paths are being found and are being placed in common processors. The non-critical functionals are being assigned so that the critical path length is not increased much.

The figures represented do show that our partitions are doing something right. Given enough processors and no communication costs, the absolute minimum cycle execution time mean is approached. In system 2, we also observed that under high communication costs our partitions are far superior to random partitions. However, the execution cycle times under such high communication costs were worse than a sequential simulation until we had at least 4 processors. Under the lower communication costs shown, our partitions

are, on the average, 12% better than random partitions. This is evidence that our partitioning algorithm is in some ways effective.

If we compare the relative differences achieved by our partitions with system 2 (its first table) with those achieved with system 1, we note that we performed better on system 1. One explanation is that system 2's behavior during a cycle is very much more variable than system 1's. It may be that 100 cycle observations were not enough to capture any relatively dominant patterns of behavior. It may also be that there aren't any dominant patterns of behavior. A statically defined partition may simply not be good enough when the simulation's behavior varies wildly. An open field of inquiry is the characterization of simulations that are amenable to a static partitioning for parallel execution.

8.4.6. Caveats

The study we report is too small to be conclusive. It does provide us with evidence that some of the ideas motivating our heuristic are well-founded. At present, we cannot look at performance figures and know whether our partitions are achieving close to optimal performance. Before we can lay claim to a truly effective partitioning algorithm we need to resolve a number of critical issues.

(i) *Insight into System Structure:* Perhaps the most critical issue is finding means of analyzing simulations to determine whether or not they are amenable to distributed simulation. No partition will perform well if the system should not be distributed.

(ii) *Lower Bounds:* Better lower bounds on achievable performance are needed. Such bounds might well come out of a successful treatment of item (i) above.

(iii) *Adaptive Number of Processors:* Under significant communication delays, it may be better to use fewer processors than are available. A partitioning algorithm should be able to detect and react to this situation.

The work developed in previous chapters of this thesis may provide the basis for treating these problems.

8.5. Chapter Summary

In this chapter we identified important concerns that any partitioning algorithm will have to consider. We then described a partitioning algorithm which takes each of this concerns into account. We implemented this algorithm and studied the performance of partitions it created on logic networks taken from the literature. The results of our empirical study indicate that the partitions created by our algorithm can be effective. In addition, this algorithm and its analysis show how some of the tools developed earlier in this thesis can be applied to a partitioning algorithm.

The work reported in this chapter should be viewed as a first step in the design of partitioning algorithms. The study of our algorithm does validate the importance of our ideas about partitioning. But our understanding of this problem is woefully incomplete. We need to develop better insights into when a simulation can be effectively partitioned for parallel execution, how well we can expect a partitioned simulation to perform, and how many processors should be used in a parallel execution. These are all open problems

which must be treated before we can be confident that we are partitioning simulations as well as can be expected.

Chapter 9

Dynamic Partitioning Decisions

9.1. Chapter Overview

The dynamics of a distributed simulation system are governed by the distributions of the values assumed by the generator event functionals. If those distributions were to change in the middle of a simulation run, the running behavior of the simulation may drastically change. As a consequence, a once good partitioning of the simulation may become bad. In this chapter we describe a decision process that anticipates, detects, and reacts to such a change. We first model the repartitioning problem as a Markov decision process. Within this model, we are able to develop a strong result: the identification of a decision policy which **optimally** balances the risks and benefits of repartitioning. Two of the parameters in the decision model represent quantities which are not known until a behavioral change is first detected. We demonstrate that the optimal policy does not need values for these parameters before change is first detected.

Dynamic load balancing in distributed systems has recently received some attention. [ChK79] treats a system with a central scheduler and a number of heterogeneous processors. Arriving jobs are assumed to be independent, and all load balancing is centralized in the scheduler. [Ni81] demonstrates optimal probabilistic load balancing strategies for a number of queueing networks; [TaT85] describes means of determining an optimal static load balancing policy in a system where each processor receives its own job stream. The jobs in this system are assumed to be independent. [ELZ] argues that simple load balancing heuristics perform well. In [RaS84, Sta84b] we find heuristics for distributed load balancing using bidding algorithms. Again, the jobs in the system are assumed to be independent of each other. [Sta85] describes decentralized load balancing based on Bayesian decision theory; again the jobs are assumed to be independent. None of these models fit our needs. We assume no central scheduler, and we know that the "jobs" in our system are heavily influenced by precedence relations. We approach dynamic partitioning differently. We do not consider letting the functionals migrate dynamically. Instead, we wait until we have evidence that the current partition is no longer working well, and then decide whether to change it.

9.2. Problem Statement

We suppose that a simulation description has already been distributed across a computer system, and consider the running behavior of that system. We expect that the system is running optimally when the sum of processor utilizations (excluding system overhead) is maximized. The sum of processor utilizations during a cycle is thus a good measure of system performance during that cycle; there may be other measures of performance. Let Z_i denote the sum of processor utilizations over the *ith* cycle. Since the cycles form a stationary process [Ros83], and each processor utilization during a cycle is completely a function of that cycle, the process of utilization sums is itself a stationary process. We now consider the possibility that at some unknown time t, the sequence Z_t, $Z_{t+1}, \cdots$ forms a stochastic process that is different from $Z_0, Z_1, \cdots, Z_{t-1}$. If s is a time greater than or equal to t, we will say that a *change* has occurred by time s. If a change occurs and the mean measure Z_j is observed to decrease, we can conclude that the system

is not running at the level of performance it once had. This drop in performance can be caused by a change in the dynamics of the running simulation; a change which causes undue synchronization blocking as a result of the partitioning. The system performance might be improved then by repartitioning. This chapter examines the issue of detecting such change, and deciding whether to repartition. To this end, we model the problem as a decision process.

A *decision process* can be thought of as a sequence of actions; time is divided into intervals, and an action is chosen at the end of each interval as a function of the *decision policy*. Each possible action has an associated cost. The decision process we envision for our problem must determine at each interval whether a change has occurred. The process has the option of either calculating a new partition (we presume that some statistical partitioning algorithm is employed), or continuing with the old one. If a new partition is calculated, it is tested against the old partition on the most recent cycle profiles; this test determines if a substantial enough improvement in the finishing time can be expected by using the new partition; Chapter 6 showed how this could be done in a Bayesian decision framework. The cost of the decision to repartition has two components. The system suffers delay due to the calculation and testing of the new partition; the second component is a benefit, or negative cost. This benefit is achieved if the new partition is found to be superior to the old one; the benefit is the expected differential in system finishing time between the system running under the old partition and the system running under the new partition. If the new partition is adopted, the decision process is considered to have *stopped*. Otherwise we conclude that the change did not occur, and continue the decision procedure. Once the decision process decides to stop, no further change is tested for; our decision model assumes that at most one new partition will be adopted during the simulation session. In practice, we expect that the decision process will actually be restarted after a stop decision. However, this more general model proved to be much less tractable, and was not pursued.

When the decision process does not choose to calculate a new partition, the system is run for the next interval of time under the old partition. This decision exacts a "lost opportunity" cost if the change has already occurred and future repartitioning benefits are achievable. The decision to continue foregoes the benefit of the new partition for the next interval.

The cost of a decision policy is the sum of the costs incurred at each decision dictated by that policy. An optimal decision policy is one with minimal expected cost. We intuitively see that the optimal policy must balance the costs and benefits of repartitioning with the costs of not repartitioning. We are able to characterize the structure of an optimal decision policy.

The similar change detecting problems have been treated in depth in [RSB79] ; our work differs principally in that we require the repartitioning benefit to vary as a function of time. This difference does have a significant impact on the problem solution. The general statistical field of sequential analysis [Gov75] treats the problem of deciding when to stop sampling; however, this sort of analysis generally assumes distributional knowledge of the quantities involved, and focuses on the minimization of the number of samples taken. This minimization metric gives equal cost to each sample; we will observe that a precise model of our problem will give observations unequal expected costs.

9.3. Problem Formulation

In this section we model the dynamic repartitioning problem as a Markov decision process. This requires a precise definition of Markov decision processes, and the formulation of dynamic partitioning within the confines of this definition. This formulation depends on our ability to calculate the probability of the anticipated change having

already occurred. This probability is affected by a statistical procedure which examines recent system behavior and decides whether the change has occurred. We will treat the statistical issues of detecting change first, and then proceed to the larger task of formulating dynamic partitioning in terms of a Markov decision process.

9.3.1. Statistical Issues in Detecting Change

We must address two issues. Most statistical tests assume that observations are independent; many assume that observations are normally distributed. To use such tests we must first transform our measurements. The second issue is simply detecting change. We propose solutions to both of these problems.

9.3.1.1. Measurement Transformation

We treated this problem in Chapter 6 and proposed using the batch means transformation. We will assume that b cycle measurements are batched into a single transformed observation. We recall that observations transformed by this method are approximately normal and independent.

9.3.1.2. A Clustering Decision Procedure

Our task is to examine transformed utilization sum measurements as they become available, and determine if and when a significant difference occurs in these observations' distribution. We can use solutions to the so-called model identification problem [BoS83] to detect this change. Applied to our situation, the model identification problem decides whether two groups of independent normal observations are best described as being drawn from the same distribution, or from two different distributions.

Using a model identification approach, we detect distributional change in the sequence of normal random variables by first creating adjacent groups of equal size c. We assume the existence of a base group taken before a change could have occurred. A test group is statistically compared with the base group for distributional equivalency. Positive indication of a distributional difference is evidence for the change having already occurred.

A new model selection procedure is particularly well suited for our problem. [BoS83] describes the AIC model selection criterion. This criterion attempts to describe data with a probabilistic model having as few parameters as possible; at the same time, it attempts to choose the model that best fits the observed data. We are to decide whether the base and test groups are best described with two parameters (the common mean and variance), or four parameters (separate means and variance). For each of these two models, we calculate the statistic

$$AIC = -2 \cdot l(K) + 2 \cdot K$$

where K is the number of model parameters, and $l(K)$ is the maximized log-likelihood function [Kal79] using K model parameters. The model with the smallest AIC statistic is chosen. To apply this technique to our problem, let $\hat{\sigma}_B^2$, $\hat{\sigma}_C^2$, and $\hat{\sigma}_f^2$ respectively denote the sample variances from the base group, the test group, and the pooled test and base groups. Then calculate

$$AIC_s = \frac{c}{2} \cdot \ln(\sigma_B^2 \cdot \sigma_C^2) + 8$$

and

$$AIC_j = c \cdot \ln(\sigma_f^2) + 4.$$

If $AIC_s < AIC_j$, then the two groups are considered to be distributionally distinct. The

advantage of this approach is its simplicity. The significance level of the statistic is effectively chosen by the derivation of the statistic, and no statistical tables need be stored. Furthermore, the calculations take linear time in the size of the groups.

The model selection criterion provides a *statistical* means of testing for change. While giving us information about a potential change, there is a non-zero probability that it fails to correctly identify whether the change occurred. Given a prior probability of the change having already occurred and an *AIC* test result, we can use Bayes' theorem to calculate a posterior probability of the change having already occurred. To use Bayes' theorem we need to know the accuracy of the *AIC* procedure. We denote the probability that the model selection statistic falsely indicates a change by α (type I error); we denote the probability that a true change is not indicated by β (type II error). Later analysis will find it necessary to assume that α and β are small enough so that $1 - \alpha - \beta \geq 0$. It seems entirely reasonable to expect this much accuracy from the statistical test.

9.3.2. The Decision Process Model

We begin our development of the decision model with a precise definition of Markov decision processes. Then we formulate dynamic partitioning in terms of such a process.

9.3.2.1. Markov Decision Models

This section briefly defines Markov decision processes, and presents an important result about such processes. Our notation is taken largely from [Ros70].

We consider a stochastic process whose state we observe at each of a sequence of times $t = 0, 1, 2, \cdots, N$. Let I be the set of all possible states. At each time j, the state of the process is discerned to be some $s \in I$. Then a decision is made, choosing some action a from a finite set A of actions; action a in state s incurs a cost $C(s,a)$. $C(s,a)$ may be random; we assume that $E[C(s,a)] < \infty$ for all states s and actions a. The process then passes into another state; the transition probabilities $p_{sq}(a)$ are conditioned both on the previous state, s, and the action taken, a. The expected total cost of a decision policy is the expected sum of all the costs incurred at each decision step.

In our search for an optimal policy, we restrict our attention to the class of *stationary* policies, those policies which are deterministic functions of the discerned state. A useful body of results exist for the identification of optimal stationary policies. Perhaps most useful is the following statement [Ros70] of a functional relationship satisfied by an optimal policy's cost function.

Theorem 9-1: [Ros70] Let $V(s)$ be the expected cost of the process which starts in state s and which is governed by the optimal stationary policy. Then,

$$V(s) = \min_{a \in A} \left\{ E[C(s,a)] + \int_{q \in I} p_{sq}(a) \cdot V(q) \right\}$$

Furthermore, the policy which, when in state s chooses the action minimizing the right hand side of the above is optimal.

□

The function $V(s)$ is known as the *optimal cost function.*

Another intuitive meaning of $V(s)$ is that it denotes the expected future costs of the process in state s, given the process is governed by the optimal decision policy.

9.3.2.2. Model Formulation

To model repartitioning as a Markov decision process we must define the decision epochs, the process states, the set of actions, the state dependent action costs, and the state and action dependent transition probabilities. Each of these issues is addressed in turn.

9.3.2.2.1. Time Steps and Process States

We can test for change every $b \cdot c$ system cycles. We consider time to be divided into a sequence of *decision steps*, with $b \cdot c$ system cycles defining the period between two adjacent decision steps. Time n is considered to be the time corresponding to the *nth* decision step. We assume that the entire execution session constitutes N decision steps.

Our decision model revolves around the probability that the anticipated change has already occurred. We let p_n denote the probability that the system has changed by time n. The state of the decision process at time n is defined to be the pair $<p_n, n>$. The central activity of the decision process is maintaining p_n as a function of time.

9.3.2.2.2. Maintaining the Probability of Change

We suppose that the decision process is in state $<p, n>$. A number of probabilities are of interest to us. Let $p^*(p_n)$ denote the probability that the system *will have* changed by time $n+1$, conditioned on the value of p_n. This probability depends on prior knowledge of the distribution of the time of change. Supposing that such information is not precisely known, it is reasonable (and convenient) to assume that the failure rate of this distribution is some constant ϕ. Using simple conditioning arguments we have for any $p \in [0, 1]$

$$p^*(p) = p + \phi \cdot (1 - p) \tag{9-1}$$

$$= (1 - \phi) \cdot p + \phi.$$

Given p_n, the probability of change by time $n+1$ is enhanced by a test for change at time $n+1$. This posterior probability is calculated directly by Bayes' Theorem [Sch69]. If the last observation indicated change, the posterior probability is

$$p^c(p) = \frac{p^*(p) \cdot (1 - \beta)}{p^*(p) \cdot (1 - \beta) + (1 - p^*(p)) \cdot \alpha} \tag{9-2}$$

for any $p = p_n$. Likewise, given a negative indication of change, the posterior probability is

$$p^{\bar{c}}(p) = \frac{p^*(p) \cdot \beta}{p^*(p) \cdot \beta + (1 - p^*(p)) \cdot (1 - \alpha)}. \tag{9-3}$$

These equations maintain p_n for each n as a function of the prior probability p_{n-1} and a change test result.

9.3.2.2.3. Decision Actions and Costs

The decision process has two options in state $<p_n, n>$; it may choose to *continue*, or it may choose to *test*. The test decision causes the calculation and possible adoption of a new partition. The Markov decision model requires us to define state dependent action costs. We define these costs by considering the effects of the chosen action.

Test Decision Consequences and Costs

The decision to test initiates a two step process. We assume that the system is halted, and a new partition is calculated. Given a new partition and performance traces of recent

system behavior we predict the performance of the system under the new partition. We then compare the performance of the system under the old and new partitions, and choose the better partition. We will assume that the new partition is chosen if and only if a change occurred and is significant enough to warrant repartitioning. As a consequence, if we retain the old partition we redefine p_n to be 0. The system is restarted once the appropriate partition is selected. The decision process is considered to have stopped if the new partition is chosen. If the old partition is chosen, the decision process resumes with $p_n = 0$ as its initial prior probability.

We assume that the computational delay of calculating and testing a new partition is some constant D_d. If the new partition is chosen, an additional delay of D_r is incurred to physically effect the repartitioning. Under a new partition, we expect a resultant speedup over the time to finish the session under the old partition. This differential speedup is estimated when the system initially indicates change; the speedup, or gain over one decision step's worth of time is denoted by G. Given a finite number of N decision steps, the overall speedup resulting from a new partition chosen at time n is $G \cdot (N - n + 1)$.

Integrating all of these observations, we see that the expected cost of the test decision in state $< p_n, n >$ is

$$E[C(< p_n, n >, test)] = D_d - p_n \cdot \left| G \cdot (N - n + 1) - D_r \right|$$

Continue Consequences and Costs

The consequences of the continue decision are substantially simpler. A continue decision simply allows the system to run for one more time period under the old partition. The system is then tested for change and a new posterior probability of change is calculated. A "lost opportunity" cost is associated with the continue decision, as the process foregoes any possible benefit from repartitioning in the next decision step's worth of system time. Presuming that we use a statistical partitioning algorithm, we expect that a new partition (based on the same performance metric distribution as the old partition) will not perform better than the one in use. We therefore expect to benefit from a new (statistically generated) partition only if the change has already occurred: the probability of this is p_n. We incur no cost by continuing if the change has not yet occurred. Consequently, the expected cost of the continue decision in state $< p_n, n >$ is simply

$$E[C(< p_n, n >, continue)] = p_n \cdot G$$

9.3.2.2.4. State Transition Probabilities

After an action in state $< p_n, n >$, the process makes a state transition; we first consider the transition after a decision to continue. By continuing, the system runs under the old partition until time $n+1$, at which point the system is again tested for change. The posterior probability p_{n+1} defines the next state $< p_{n+1}, n+1 >$; depending on the outcome of the change test at time $n+1$, p_{n+1} is equal either to $p^c(p_n)$ or $p^{\bar{c}}(p_n)$. The state transition probabilities from $< p_n, n >$ after a continue decision are thus defined by the probability of observing a change at time $n+1$.

Let $q^c(p_n)$ denote the probability that the change test will report a change at time $n+1$, conditioned on p_n. We recall that α and β denote the true type I and type II errors of the test procedure; again, direct conditioning arguments establish that for any $p \in [0,1]$

$$q^c(p) = p^*(p) \cdot (1 - \beta) + (1 - p^*(p)) \cdot \alpha \qquad (9\text{–}4)$$

and the probability of observing no change at time $n+1$ is

$$q^{\bar{c}}(p) = 1 - q^{c}(p)$$

$$= p^{*}(p)\cdot\beta + (1 - p^{*}(p))\cdot(1 - \alpha).$$

Consider the transition probabilities out of $<p,n>$ after a test decision. The new partition is adopted with probability p; in this event the decision process stops so that there is no next state. Conversely, the probability of rejecting the new partition is $1 - p$. After rejection, we infer that $p = 0$; the state of the process at time $n+1$ is thus one of the two states reachable from state $<0,n>$, and depends again on the outcome of the change test at time $n+1$. Thus for any state $<\hat{p},n+1>$, the probability of passing from $<p,n>$ into $<\hat{p},n+1>$ is $1 - p$ times the probability of reaching $<\hat{p},n+1>$ from $<0,n>$.

9.3.2.2.5. Process Optimal Cost Function

We may now restate the functional relationship satisfied by the optimal cost function. A further definition will enhance the readability of this relationship. Recall that $V(<p,n>)$ denotes the minimal expected future costs of the decision process in state $<p,n>$. Theorem 9–1 states a relationship the optimal cost function satisfies in terms of the minimal expected future costs of the process at time step $n+1$, given $p_n = p$. We denote this expectation by $E_v[<p,n>]$, and observe

Definition 9-1: *Next Step Minimized Cost Function*

$$E_v[<p,n>] = q^{c}(p)\cdot V(<p^{c}(p),n+1>) + q^{\bar{c}}(p)\cdot V(<p^{\bar{c}}(p),n+1>)$$

denotes the expected minimal expected future costs of the decision process at time $n+1$, given the state at time n.

□

Theorem 9–1 may thus be restated in terms of our problem's notation as follows.

$$V(<p,n>) = \min\begin{cases} D_d - p\cdot\left|G\cdot(N - n + 1) - D_r\right| + (1 - p)\cdot E_v[<0,n>] \\ p\cdot G + E_v[<p,n>] \end{cases} \qquad (9\text{–}5)$$

The optimal decision in state $<p,n>$ is the action associated with the cost function defining this minimum.

Since the decision process will always stop at or before time N, we define we define $V(<p,N+1>) = 0$ for every $p \in [0,1]$.

9.4. Optimal Decision Policy

We can now demonstrate that the optimal decision policy is characterized by a sequence $\pi_1,\cdots,\pi_N$ of constants from the interval [0, 1]. The optimal decision in state $<p,n>$ is to test if and only if $p > \pi_n$. The optimal policy is implicitly given in the statements of Theorem 9–1 and equation 9–5: given state $<p,n>$, the optimal decision is to choose the action with minimal expected future cost. For a given time n, if we can characterize each action's expected future cost as a function of p, we can then compare the actions' expected future costs over the range $p \in [0,1]$ and determine the optimal decision as a function of p. To show that the suggested policy is optimal, we need to demonstrate

the existence of $\pi_n \in [0,1]$ such that $p > \pi_n$ if and only if the expected future cost of the test decision in $<p,n>$ is less than the expected future cost of the continue decision in $<p,n>$. We next present a number of lemmas which establish the existence of such π_n.

9.4.1. Properties of $V(<p,n>)$.

The optimality of the proposed policy is shown by demonstrating certain properties of the optimal cost function. We show that for any given time step n the minimal expected future test decision cost is linear in p_n, while the minimal expected future continue decision cost is concave in p_n. We also demonstrate important inequalities.

The optimal cost function's value at $<p,n>$ is the minimum value of two functions evaluated at $<p,n>$. One function expresses the minimal expected future costs after a decision to test, and is denoted by

$$ECT(<p,n>) = D_d - p\cdot\left|G\cdot(N - n + 1) - D_r\right| + (1 - p)\cdot E_v[<0,n>].$$

The second function expresses the minimal expected future costs after a decision to continue and is denoted by

$$ECC(<p,n>) = p\cdot G + E_v[<p,n>].$$

We first demonstrate that for fixed n, $ECT(<\cdot,n>)$ is a linear function in p.

Lemma 9-1: For fixed $1 \leqslant n \leqslant N$, $ECT(<\cdot,n>)$ is a linear function of p.

Proof: For fixed n, the value of $E_v[<0,n>]$ is independent of the value of p in $ECT(<p,n>)$, and is thus constant. The degree of p in ECT is one, and all other components of ECT are constant.

□

The concavity of $ECC(<\cdot,n>)$ in p for fixed n follows principally from the following lemma reported in [RSB79], and stated in terms of our notation.

Lemma 9-2: [RSB79] If $V(<\cdot,n+1>)$ is a piecewise linear and concave function of p, then $E_v[<\cdot,n>]$ is a piecewise linear and concave function of p.

□

For notational convenience, we say that a real-valued function is *plcc* if it is piece-wise linear, continuous, and concave. Using this, we have

Lemma 9-3: For every $1 \leqslant n \leqslant N$, $V(<\cdot,n>)$ and $ECC(<\cdot,n>)$ are *plcc* functions of p.

Proof: We induct on n; we show that for every $1 \leqslant n \leqslant N$, both $V(<\cdot,n>)$ and $ECC(<\cdot,n>)$ are *plcc* functions of p. For the base case we consider $n = N$. Now for any $p \in [0,1]$,

$$ECC(<p,N>) = p\cdot G + E_v[<p,N>]$$

$$= p\cdot G$$

since $V(<p,N+1>) = 0$ for all p. Thus $ECT(<\cdot,N>)$ is *plcc*. The concavity and continuity of $ECT(<\cdot,N>)$ follows directly from its linear nature.

Furthermore, V is defined as the pointwise minimum of ECT and ECC, and piecewise linearity, continuity, and concavity are all preserved under the pointwise minimum operator; hence $V(<\cdot,N>)$ is a *plcc* function of p, and the induction base is satisfied. For the induction hypothesis, we suppose that both $ECC(<\cdot,n+1>)$ and $V(<\cdot,n+1>)$ are *plcc* functions of p; and again consider

$$ECC(<p,n>) = p \cdot G + E_v[<p,n>].$$

$E_v[<\cdot,n>]$ is a *plcc* function of p: the induction hypothesis satisfies the hypothesis of the lemma 9–2. The class of *plcc* functions is closed under addition and pointwise minimum, thus it follows that both $V(<\cdot,n>)$ and $ECC(<\cdot,n>)$ are *plcc* functions, completing the induction.

□

It is easy to compare the values of ECT and ECC at domain points $<1,n>$. Once we have established these values at such points we can infer the relationship between ECT and ECC at all domain points; this relationship is established with arguments concerning global functional behavior, e.g., concavity. The following two lemmas establish that ECC is everywhere less than ECT once the process has crossed a certain threshold in time.

Lemma 9-4: There exists a non-negative integer K_0 such that $V(<1,n>) = ECC(<1,n>) = G \cdot (N - n + 1)$ for all $N - K_0 + 1 \leqslant n \leqslant N$.

Proof: Let K_0 be the largest integer such that $2 \cdot G \cdot K_0 \leqslant D_d + D_r$. We demonstrate inductively that K_0 has the required property. Consider

$$V(<1,N>) = \min \begin{cases} D_d - G + D_r \\ G \end{cases}$$

If $K_0 > 0$, the value for $V(<1,N>)$ is easily seen to be given by $ECC(<1,N>) = G$. For the induction hypothesis, suppose there is an $n \geqslant N - K_0 + 1$ such that

$$V(<1,n+1>) = ECC(<1,n+1>) = G \cdot (N - (n+1) + 1).$$

Then

$$V(<1,n>) = \min \begin{cases} D_d - G \cdot (N - n + 1) + D_r \\ G + E_v[<1,n>] \end{cases}$$

$$= \min \begin{cases} D_d - G \cdot (N - n + 1) + D_r \\ G + V(<1,n+1>) \end{cases}$$

$$= \min \begin{cases} D_d - G \cdot (N - n + 1) + D_r \\ G \cdot (N - n + 1) \end{cases}$$

where the second step follows from the definition of $E_v[<1,n>]$ and the obser-

vation that given $p_n = 1$, then $p_{n+1} = 1$ regardless of the change observation at time $n+1$. The third step follows by the induction hypothesis. Since $n \geq N - K_0 + 1$, it follows that $K_0 \geq N - n + 1$. Thus

$$ECT(<1,n>) - ECC(<1,n>) = D_d + D_r - 2 \cdot G \cdot (N - k + 1)$$

$$\geq D_d + D_r - 2 \cdot G \cdot K_0$$

$$\geq 0$$

by definition of K_0. Thus $V(<1,n>)$ is defined by $ECC(<1,n>)$, and has the required form.

□

Consider the function graphs representing $ECT(<\cdot,n>)$ and $ECC(<\cdot,n>)$ as functions of p for some n greater than $N - K_0$. Lemma 9–4 proves that ECT exceeds ECC at $p = 1$. We next endeavor to show that on such graphs, the value of ECC is less than the value of ECT for *all* p. We know that ECT is a decreasing (linear) function of p, and we know that ECT exceeds ECC at $p = 1$. We simply establish that ECC is an *increasing* function of p to prove that ECT exceeds ECC for all p.

Lemma 9-5: For all n such that $N \geq n > N - K_0$, $ECC(<p,n>)$ is an increasing function of p, and $V(<p,n>)$ is defined by $ECC(<p,n>)$ for all $p \in [0,1]$.

Proof: We first establish the following proposition: if $V(<\cdot,n+1>)$ is increasing in p, then $ECC(<\cdot,n>)$ is increasing in p. The proof of this is achieved by taking the derivative of ECC with respect to p, and observing that it is positive wherever it is defined. As a first step in taking the derivative, we refer to equations 9–1 and 9–4, noting that

$$\frac{d}{dp} q^c(p) = (1 - \phi) \cdot (1 - \alpha - \beta).$$

$$\geq 0$$

since $0 < \phi < 1$ and $1 - \alpha - \beta \geq 0$ by general assumption. Recalling the definition of $E_v[<p,n>]$, the full derivative of $ECC(<p,n>)$ with respect to p is given by the chain rule, and can be expressed as

$$\frac{d}{dp} ECC(<p,n>) = G + \frac{d}{dp} q^c(p) \cdot \left| V(<p^c(p),n+1>) - V(<p^{\bar{c}}(p),n+1>) \right|$$

$$+ q^c(p) \cdot \frac{d}{dp} V(<p^c(p),n+1>) + q^{\bar{c}}(p) \cdot \frac{d}{dp} V(<p^{\bar{c}}(p),n+1>).$$

The non-negativity of this expression follows directly from the increasing nature of $V(<\cdot,n+1>)$ and the observation that $p^c(p) > p^{\bar{c}}(p)$. Since $ECC(<\cdot,n>)$ is piecewise linear, this derivative exists almost everywhere; since this derivative is non-negative and ECC is continuous, we have that ECC is increasing (non-decreasing) in p.

This last result allows us to inductively show that $ECT(<\cdot,n>)$ everywhere defines $V(<\cdot,n>)$ and is increasing for all n such that $N \geqslant n > N - K_0$. Consider the base case of $n = N$. $ECC(<p,N>) = p \cdot G$ for all p, so that ECC is increasing in p. ECT exceeds ECC at $p = 1$, and ECT is decreasing in p. It follows that ECT exceeds ECC for all p, whence V is defined by ECC for all p. For the induction hypothesis, let $n > N - K_0$ be such that $ECC(<\cdot,n+1>)$ is increasing in p, and $V(<p,n+1>) = ECC(<p,n+1>)$ for all p. By lemma 9-4, ECT exceeds ECC at $p = 1$. Since $V(<\cdot,n+1>)$ increases in p, we have demonstrated that $ECC(<\cdot,n>)$ increases in p. Exactly as before, we observe that $ECT(<\cdot,n>)$ decreases in p, from which it follows that $ECT(<p,n>)$ exceeds $ECC(<p,n>)$ for all p. Thus $V(<p,n>)$ is defined by $ECC(<p,n>)$ for all p, completing the induction.

□

Lemma 9-5 establishes that $N - K_0$ is a critical point in time. The continue decision is always optimal after this time. In terms of the probability thresholds π_n, we have $\pi_n = 1$ for all $N - K_0 < n \leqslant N$. This is intuitively reasonable, since the total benefit of repartitioning diminishes with time.

We have yet to consider the optimal decision policy for times less than $N - K_0$. The structure of the policy follows from the relationship between $ECT(<1,n>)$ and $ECC(<1,n>)$, and global behavior of the cost functions. The following lemma shows that $ECT(<1,n>)$ is exceeded by $ECC(<1,n>)$.

Lemma 9-6: $ECT(<1,n>) < ECC(<1,n>)$ for all n such that $0 \leqslant n \leqslant N - K_0$.

Proof: We induct again on n. Let $n = N - K_0$. Lemma 9-4 established that

$$V(<1,N-K_0+1>) = G\cdot(N - (N - K_0 + 1) + 1)$$

$$= G\cdot K_0.$$

It follows that the optimal cost function satisfies

$$V(<1,N-K_0>) = \min\begin{cases} D_d - G\cdot(N - (N-K_0)+1) + D_r \\ G + E_v[<1,N-K_0>] \end{cases}$$

$$= \min\begin{cases} D_d - G\cdot(K_0 + 1) + D_r \\ G + V(<1,N-K_0+1>) \end{cases}$$

$$= \min\begin{cases} D_d - G\cdot(K_0 + 1) + D_r \\ G + G\cdot K_0 \end{cases}$$

$$= \min\begin{cases} D_d - G\cdot(K_0 + 1) + D_r \\ G\cdot(K_0 + 1) \end{cases}.$$

Consequently,

$$ECT(<1,N-K_0>) - ECC(<1,N-K_0>) = D_d - 2\cdot G\cdot(K_0 + 1) + D_r.$$

By definition, K_0 is the largest integer such that

$$D_d - 2\cdot G\cdot K_0 + D_r \geqslant 0,$$

from which it follows that the optimal decision at $<1,N - K_0>$ is to test.

We suppose for the induction hypothesis that

$$V(<1,N-K_0-(k-1)>) = ECT(<1,N-K_0-(k-1)>)$$

$$= D_d - G\cdot(N - (N-K_0-(k-1)) + 1) + D_r$$

for some $k \geqslant 1$. Then

$$V(<1,N-K_0-k>) = \min\begin{cases} D_d - G\cdot(N - (N-K_0-k) + 1) + D_r \\ G + V(<1,N-K_0-(k-1)>) \end{cases}$$

$$= \min\begin{cases} D_d - G\cdot(K_0 + k + 1) + D_r \\ G + D_d - G\cdot(N - (N-K_0-(k-1)) + 1) + D_r \end{cases}$$

$$= \min\begin{cases} D_d - G\cdot(K_0 + k + 1) + D_r \\ D_d - G\cdot(K_0 + k - 1) + D_r \end{cases}$$

$$= D_d - G\cdot(K_0 + k + 1) + D_r$$

$$= ECT(<1,N-K_0-k>)$$

which completes the induction argument.

□

We now consider the global behavior of ECT and ECC. Lemma 9–1 proves that $ECT(<\cdot,n>)$ is decreasing and linear in p; Lemma 9–3 proves that $ECC(<\cdot,n>)$ is concave in p. These facts suffice to show the existence of a number π_n such that

$ECC(<p,n>) < ECT(<p,n>)$ if and only if $p < \pi_n$.

Lemma 9-7: Let $n \leq N - K_0$. There exists $\pi_n \in [0,1]$ such that $ECC(<p,n>) < ECT(<p,n>)$ if and only if $p < \pi_n$. Furthermore, either $\pi_n = 0$ or π_n is the single point in [0,1] at which $ECT(<\pi_n,n>) = ECC(<\pi_n,n>)$.

Proof: We adapt a solution given in [Ros70] for sequential analysis. We first note that $V(<\cdot,n>)$ must be convex in p, as it is the pointwise minimum of two convex functions. By definition of convexity, for every p_1, p_2 and λ in [0,1] we have

$$V(<(\lambda \cdot p_1 + (1-\lambda) \cdot p_2), n>) \geq \lambda \cdot V(<p_1,n>) + (1-\lambda) \cdot V(<p_2,n>).$$

We now show that the set of all p such that $V(<p,n>) = ECT(<p,n>)$ must be an interval $[\pi_n,1]$. Let m and b be those constants such that

$$ECT(<p,n>) = m \cdot p + b,$$

and suppose that p_1 and p_2, $p_1 \leq p_2$, are points such that

$$V(<p_i,n>) = m \cdot p_i + b \qquad i = 1, 2.$$

Consider any p_m such that $p_1 \leq p_m \leq p_2$. There exists a $\lambda \in [0,1]$ such that $p = \lambda \cdot p_1 + (1-\lambda) \cdot p_2$. By the convexity of V, we must have that

$$V(<p_m,n>) \geq \lambda \cdot V(<p_1,n>) + (1-\lambda) \cdot V(<p_2,n>).$$

$$= m \cdot p_m + b.$$

However, by definition of $V(<p_m,n>$ we must have

$$V((<p_m,n>) \leq ECT(<p_m,n>) = m \cdot p_m + b,$$

from which it follows that

$$V(<p_m,n>) = m \cdot p_m + b.$$

Consequently, the set of all p such that $V(<p,n>) = ECT(<p,n>)$ is an interval subset of [0,1]. Let π_n be the infimum all such p; since $V(<1,n>) = ECT(<1,n>)$ by lemma 9–6, the form of this interval is $[\pi_n,1]$.

Finally, we show that π_n is either 0 or is the single point of intersection between *ECT* and *ECC*. If *ECT* and *ECC* do not intersect, then π_n is clearly 0. Now suppose that *ECT* and *ECC* are equal across an interval $[p_1, p_2]$. Since *ECC* is piecewise linear and concave, the slopes on its graph's sequence of line segments must be strictly decreasing as p increases. In particular, the slopes of all its segments to the right of the one on which *ECT* coincides are less than *ECT*'s slope. The continuity of *ECC* then ensures that $ECC(<1,n>) < ECT(<1,n>)$, a contradiction of lemma 9–6. Thus, if *ECT* and *ECC* intersect, they intersect in at most one point.

□

9.4.2. Optimal Policy Structure

The previous section provides us with all the tools needed to determine the optimal policy structure. The principal result determining this policy is stated in Theorem 9–1: the action minimizing the expected future process costs is optimal. For every state $<p,n>$, the optimal decision is to test if and only if $ECT(<p,n>)$ is less than $ECC(<p,n>)$. Lemmas 9–4 and 9–6 prove that if K_0 is the largest integer such that $2 \cdot G \cdot K_0 \leq D_d + D_r$, and if n is greater than $N - K_0$, then $V(<p,n>) = ECC(<p,n>)$ for all p. The optimal policy for any process reaching time $N - K_0 + 1$ without stopping is simply to always continue through time N. Lemma 9–7 in turn shows that if $n \leq N - K_0$, there exists a threshold π_n such that $V(<p,n>) = ECC(<p,n>)$ for all $p \leq \pi_n$, and $V(<p,n>) = ECT(<p,n>)$ for all $p > \pi_n$. Consequently, the optimal decision for the process in state $<p,n>$ to make is to test if and only if $p > \pi_n$.

The preceding paragraph proves the following theorem.

Theorem 9-2: There exist constants $\pi_0, \cdots, \pi_N$ such that the optimal decision decision in state $<p,n>$ is to test if and only if $p > \pi_n$. Furthermore, there exists an integer K_0 such that $\pi_n = 1$ for all $n > N - K_0$.

□

The π_n defining the optimal decision policy cannot generally be analytically derived. The π_n are obtained through explicit solution of the optimal cost functions. In Appendix A we consider how these π_n can be obtained numerically. We show that their exact solution is computationally intractable, and propose an approximation method which is in some sense optimal.

9.5. Waiting for Change

The derivation of the optimal decision policy assumes that all decision model parameters are known a priori. By contrast, our real world assumptions state that we do *not* have prior knowledge of the nature of the change; we are prohibited from finding the values of β and G before the change occurs. In this section, we address this seeming contraction of assumptions by deriving conditions on the model parameters ϕ, α, and β such that under the optimal policy, the decision process will always wait for at least one change to be detected before deciding to repartition. Under these conditions we can derive an upper bound on p_n given that no change has yet been detected. When the optimal threshold π_n is greater than this bound, the policy of waiting for change is optimal.

We may assume that $p_0 = 0$. We are interested in calculating p_n given that the change test has not detected a change by time n. This probability is expressed recursively using equation 9–3.

Definition 9-2: *Probability of No Change by Time n*

Let $\bar{C}(n)$ denote p_n given that no change has been detected by time n. Then

$$\bar{C}(0) = 0$$

and

$$\bar{C}(n+1) = \frac{p^*(\bar{C}(n))\cdot\beta}{p^*(\bar{C}(n))\cdot\beta + (1 - p^*(\bar{C}(n)))\cdot(1-\alpha)}$$

□

For any n, we could use the equation above to numerically compute $\bar{C}(n)$. This approach gives us no understanding though of how that probability behaves as a function of n. We are led to consider an easily calculated upper bound on all $\bar{C}(n)$. This bound is derived from the observation that $p^{\bar{c}}(p)$ is a *contraction mapping.*

Definition 9-3: *Contraction Mapping*

Let g be a function on $[a,b]$. g is said to be a *contraction mapping* on $[a,b]$ if

(i) $p \in [a,b] \implies g(p) \in [a,b]$,

(ii) g's derivative $\dot{g}$ exists, is continuous, and $|\dot{g}(p)| \leq L < 1$ for some L and for all p in $[a,b]$.

□

The following theorem is taken from [PhT73] , and illustrates our interest in $p^{\bar{c}}(p)$ as a contraction mapping.

Theorem 9-3: [PhT73] Let g be a contraction mapping on $[a,b]$. Then

(i) The equation $p = g(p)$ has a unique solution in $[a,b]$, say p_s.

(ii) Given any $p_0 \in [a,b]$, the sequence p_r defined by

$$p_{r+1} = g(p_r)$$

converges from one side to p_s.

□

The definition of $\bar{C}(n)$ defines a sequence in the form of conclusion (ii) in theorem 9–3. If we can show that $p^{\bar{c}}(\cdot)$ is a contraction mapping on some interval $[0,s]$, and if we can show that the sequence's one sided convergence is from the left, then the solution p_s to the equation $p = p^{\bar{c}}(p)$ will bound $\bar{C}(n)$ from above for all n. We first present conditions under which $p^{\bar{c}}(\cdot)$ is a contraction mapping.

Lemma 9-8: Let c be the smallest positive solution to the equation

$$\Big|(1-\phi)\cdot p + \phi)\cdot\beta + (1-(1-\phi)\cdot p - \phi)\cdot(1-\alpha))\Big|^2 - (1-\phi)\cdot(1-\alpha)\cdot\beta = 0$$

and let p_s be the smallest positive solution to the equation $p = p^{\bar{c}}(p)$. If $p_s < c$, then $p^{\bar{c}}(\cdot)$ is a contraction mapping on $[0,p_s]$.

Proof: Recalling equations 9–1 and 9–3, we have

$$p^{\bar{c}}(p) = \frac{((1-\phi)\cdot p + \phi)\cdot\beta}{((1-\phi)\cdot p + \phi)\cdot\beta + (1-(1-\phi)\cdot p - \phi)\cdot(1-\alpha)}.$$

The derivative with respect to p of the numerator is $(1-\phi)\cdot\beta$; and the derivative of the denominator is $(1-\phi)\cdot(\alpha+\beta-1)$. We apply the quotient rule for derivatives, and after much algebra, find that

$$\frac{d}{dp}p^{\bar{c}}(p) = \frac{(1-\phi)\cdot(1-\alpha)\cdot\beta}{\Big|((1-\phi)\cdot p + \phi)\cdot\beta + (1-(1-\phi)\cdot p - \phi)\cdot(1-\alpha)\Big|^2}. \tag{9–6}$$

$$> 0.$$

When $p = 0$, the denominator of equation 9–6 is

$$\Big|\phi\cdot\beta + (1-\phi)\cdot(1-\alpha)\Big|^2 = \phi^2\cdot\beta^2 + 2\cdot\phi\cdot\beta\cdot(1-\phi)\cdot(1-\alpha) + (1-\phi)^2\cdot(1-\alpha)^2.$$

We now demonstrate that the derivative given in equation 9–6 is less than one at $p = 0$. By general assumption, $1-\alpha-\beta \geq 0$. It follows directly that

$$0 < \phi\cdot\beta + (1-\phi)\cdot(1-\alpha-\beta)$$

so that

$$(1-\phi)\cdot\beta < \phi\cdot\beta + (1-\phi)\cdot(1-\alpha).$$

Then

$$\beta < 2\cdot\phi\cdot\beta + (1-\phi)\cdot(1-\alpha)$$

so that by multiplying everything by $(1-\phi)\cdot(1-\alpha)$,

$$(1-\phi)\cdot(1-\alpha)\cdot\beta < 2\cdot\phi\cdot\beta\cdot(1-\phi)\cdot(1-\alpha) + \Big|(1-\phi)\cdot(1-\alpha)\Big|^2$$

$$< \Big|\phi\cdot\beta + (1-\phi)\cdot(1-\alpha)\Big|^2.$$

This shows that the numerator in equation 9–6 is strictly less than the denominator; hence the derivative is less than one at $p = 0$. Comparison of equation 9–6's denominator with equation 9–4 shows that this denominator is simply $q^{\bar{c}}(p)$ squared. $q^{\bar{c}}(\cdot)$ is easily seen to be strictly decreasing (and less than 1) in p on $[0,1]$; it follows that the derivative in equation 9–6 is strictly increasing in p. Thus, if c is the minimal positive solution to the equation equating the numerator and denominator of equation 9–6, then the derivative is strictly less than 1 on the interval $[0,c)$. Then if we let L be the the value of this derivative at p_s, the derivative is less than or equal to L on $[0,p_s]$. Furthermore, $p^{\bar{c}}(\cdot)$ is increasing on

on $[0,p_s]$ since its derivative is always positive on this interval; it follows that $p^{\bar{c}}(p) \in [0,p_s]$ for all p in $[0,p_s]$. By definition then, $p^{\bar{c}}(\cdot)$ is a contraction mapping on $[0,p_s]$.

□

The requirement that the solutions c and p_s exist and are related by $p_s < c$ may seem stringent. In practice, we have observed that this is always the case for all α, β, and ϕ under our operational assumptions. Table 9–1 gives the values of p_s and c for several assignments of α, β, and ϕ.

α	β	ϕ	c	p_s
.01	.1	.001	.75	.000125
.1	.01	.001	.90	.000011
.1	.1	.001	.75	.000112
.1	.1	.1	.74	.0138
.1	.1	.5	.72	.125
.2	.2	.5	.72	.33
.2	.3	.5	.81	.6
.3	.2	.5	.7	.4
.3	.2	.001	.65	.0004

Values of p_s

Table 9–1

This table strongly suggests that $p_s < c$ in general; however, we have not yet been able to prove this. The remaining discussion implicitly assumes that the hypothesis of lemma 9–8 is satisfied.

When $p^{\bar{c}}(\cdot)$ is a contraction mapping on $[0,p_s]$ we can bound the values of $\bar{C}(n)$ from above.

Lemma 9-9: Suppose that $p^{\bar{c}}(\cdot)$ is a contraction mapping, and let p_s be the smallest positive solution to the equation $p^{\bar{c}}(p) = p$. Then $\bar{C}(n) \leqslant p_s$ for every n.

Proof: Theorem 9–3 states that the sequence defining the values of $\bar{C}(n)$ converges to p_s from one side. Since $\bar{C}(1) > \bar{C}(0)$, it follows that the convergence is from the left. The conclusion follows immediately.

□

If no change has yet been observed by time n, then $p_n < p_s$. Table 9–1 shows that for small ϕ, p_s is small. If no change has been observed by time n, and if $p_s < \pi_n$, then it follows that $p_n < \pi_n$; hence choosing to continue if no change has ever been detected is the optimal decision. This proves theorem 9–4.

Theorem 9-4: Suppose that no change has ever been detected by time n. If $p_s < \pi_n$, then the optimal decision at time n is to continue.

□

We believe that the values of the optimal decision thresholds π_n are likely to be quite high. This belief stems from the observation that when p_n is not close to 1, one or two (consecutive) positive change indications will greatly increase the probability of change. When substantial assurance that the change occurred is so quickly gained, we expect that the thresholds π_n will be close to 1. Under these circumstances, we expect that $p_s < \pi_n$ so that the hypothesis of Theorem 9–4 is satisfied.

9.6. Model Parameters Reconsidered

Solving for the optimal decision thresholds requires that we quantify each model parameter. In this section we consider each parameter and suggest how it might be estimated.

α, β: The type I error α is the probability that the *AIC* test falsely indicates a change. α would seem then to be a function of the base group's distribution and the size of the test clusters. We could estimate α as a function of these parameters by performing a simulation study which measures the type I misclassifications. The type II error β is the probability that the *AIC* test falsely indicates no change. β would seem to depend on the difference between the base cluster distribution and the changed distribution. β's behavior as a function of this difference could also be established through simulation. The π_n solution procedure would have access to the tables created by these simulation studies.

ϕ: The change time failure rate ϕ is a critical component of our model. In the absence of any other information, we might assume that a change in the time period $[0, N]$ is as likely as not. In this case, we let ϕ be the parameter of the exponential distribution whose median is N.

G: It is not generally possible to predict the exact gain available by repartitioning. We might assume that we can achieve the old level of performance. Further research into this problem might establish upper and lower bounds on G. In this case, Appendix B shows that any point estimate of G between an upper and lower bound is a "good" guess in the sense that the resulting estimations of π_n lie in some interval with the true π_n.

D_d, D_r: It is reasonable to assume that we know the running time of our partitioning algorithm. It is also reasonable to assume that we know how long it takes to effect a new partition.

9.7. Chapter Summary

Since any good partitioning algorithm is based on the anticipated run–time behavior of the simulation, we are vulnerable to unexpected change in the nature of this behavior. This chapter treats this problem by formulating the repartitioning problem as a Markov decision process. This process continually tests for change, deciding if and when to repartition the simulation system. On the basis of observed change, the decision process maintains a probability that the change has already occurred. The decision process balances the potential benefit and risks of repartitioning in such a way that the expected cost of the decision process is minimized. The optimal decision policy is characterized by a sequence $\pi_0, \cdots, \pi_N$ of probabilities, so that repartitioning is chosen at time n if and only if the probability that the change occurred by time n exceeds the threshold π_n.

The derivation of the optimal decision policy assumed that all decision model parameters are known a priori. This assumption conflicts with the real world assumption

that the nature of any behavior change is not known until the change occurs. We have demonstrated that under real world conditions, the optimal policy will not call for a new partition before change is observed. It is not necessary then to have a priori knowledge of all decision model parameters, we may estimate the parameters at the time that change is observed.

Chapter 10

Conclusions

Simulation is one of the most important and widespread uses for computers today. As multi-processor computer systems become more common, finding an automated means of partitioning complex simulations for parallel execution emerges as an important problem. In this thesis we have laid the foundations for a treatment of this problem, proposed a simulation partitioning algorithm, and developed an optimal repartitioning decision process. In this final chapter, we summarize our results, discuss the relative strengths and weaknesses of our approach, and identify issues that should be addressed in our future research. We then outline our conclusions, and ruminate on the significance of our work.

10.1. Summary of Results

The work presented in this thesis provides a firm analytic framework in which the problem of partitioning a simulation can be studied. We provide tools for modeling and analyzing the run-time behavior of a distributed simulation, and argue that the behavior and analysis of a running distributed simulation is best treated statistically. We propose a statistical partitioning algorithm, and we formally examine the issue of dynamically partitioning a simulation.

We now highlight the contributions made by each chapter. **Chapter 2** proposed a model of simulations which, unlike previous models, lends itself to a certain amount of quantitative analysis. It also identified a set of execution sequencing rules which were seen to be the basis for synchronization in distributed simulations. **Chapter 3** made the important observation that a simulation's run-time behavior is complex. It showed how to deal with this complexity in the event that the simulation behaves cyclically. It also placed the analysis of running simulations on firm mathematical ground by modeling a running simulation as a Markov chain. It also observed that the performance of a partition has to be treated in terms of its average behavior. **Chapter 4** explored the relationship between synchronization and the cycle execution time. It defined a number of graphs which express the synchronization needs of the simulation. The importance of these graphs is reflected by their continual application throughout this thesis. Chapter 4 showed how synchronization in a distributed simulation can be modeled: this is an important result in that synchronization plays a large role in determining the overall execution time of a distributed simulation. Chapter 4 also showed how a given cycle's execution time can be estimated. **Chapter 5** analyzed the probabilistic behavior of a running simulation. The importance of this work is, paradoxically, that extremely restrictive assumptions must be made to proceed with a tractable mathematical treatment. While the analysis given has some use, the real contribution of this chapter is that static probabilistic analysis is not a reliable means of predicting a simulation's behavior under some partition. **Chapter 6** follows naturally from Chapter 5; if static analysis is not reliable, then we will have to use statistical analysis. Chapter 6 shows that a statistical approach is well-founded, and identifies an appropriate statistical technique for this analysis. **Chapter 7** develops a theory of comparing the performance of two partitions qualitatively. This theory is the basis for achieving computational savings when searching for the better of two partitions. We show that these results are applicable in an important application area. **Chapter 8** proposes a statistical partitioning algorithm. This algorithm and the analysis of the

partitions it produces show how tools developed in earlier chapters can be used. An empirical study of the partitions produced by this algorithm show it to be effective. **Chapter 9** develops an optimal dynamic partitioning decision process. This work is quite important; this thesis shows that we must treat partitioning in a statistical context, which leaves our partitions vulnerable to radical changes in the simulations behavior. We have shown how to optimally deal with this problem.

10.2. Strengths and Weaknesses

We now reflect on the relative strengths and weaknesses of our research. Chapters 2, 3 and 4 develop a model of running simulations. The outstanding strength of this model is its tractability: we can calculate any given cycle's execution time under hyper–message synchronization. This tractability has come at the cost of generality. The most prominent among our simplifications are the assumptions of non–interference among evaluations of the same functional, fixed logical delays for functional evaluations, and cyclic generator inter–evaluation times. We have not discovered any reasonable means of relaxing the assumption of non–interference. Even if a simulation's behavior does not satisfy this assumption, it is possible to **model** the behavior as though there is no interference. The assumption of fixed logical delays is quite critical to the approach developed in this thesis. If we allow the logical delays to vary, we can no longer build the work graph or any of its derivatives. We see the problem of relaxing this assumption as the major hurdle in extending our work. We will discuss some of our ideas concerning the relaxation of this assumption in the next section of this chapter. The assumption that the simulation has a cyclic nature is fairly common (e.g. regenerative simulation [Hel79, MeH82]) and there are ways of defining cycles other than the way we've proposed. Our proposed method is more in keeping with the simulation of logic networks than with, say, queuing networks.

The strength of the material developed in Chapters 2, 3, and 4 is thus seen to be its tractability. The weakness is the simplifying assumptions prerequisite to that tractability. However, these simplying assumptions can be defended as being acceptable modeling assumptions.

Chapters 5, 6 and 7 use the model developed in earlier chapters to analyze the performance of a partitioned simulation. We have already pointed out the weakness of the exact probabilistic analysis: it requires very restrictive assumptions. But this same observation is the strength of the statistical analysis; we **can** analyze the system statistically even if the underlying probability structure is extremely complex. The strength of our work in qualitative analysis is that it provides a probabilistic base for a statistical approach to qualitative comparison. The major weakness in our development of statistical analysis is that we have not addressed all of the issues that might be raised. For example, our treatment of the Bayesian priors is incomplete, and we have not addressed the issue of how many observations we should collect for analysis. Our omission of these issues can be defended on the grounds that their investigation constitutes major efforts which are tangential to the overall focus of this thesis.

Thus the strength of our developments in Chapters 5, 6, and 7 is in the provision of mathematical techniques which we can use to analyze and compare the performance of partitioned simulations. The weakness in our approach is that we have not addressed all aspects of a statistical approach. We identify the furthered investigation of these aspects as an important next step in our research.

Chapter 8 proposes a partitioning algorithm and studies the performance of partitions created by this algorithm. The strengths of the material developed in Chapter 8 are twofold. Our heuristic and its analysis demonstrate concretely how the analytic tools developed in earlier chapters can be applied to a partitioning algorithm. Furthermore, our heuristic is useful because the partitions it produces are effective. The major weakness of

this material is its incompleteness. We present the results of studying two complex simulation systems; clearly a more exhaustive study is called for. Variations on our algorithm should be considered, as should entirely different approachs to partitioning. The continuation of algorithm development and study is premiere on our list of future research efforts.

Chapter 9 completes our study of statistical partitioning by developing an optimal repartitioning decision process. The strength of this decision model lies in the protection it affords a statistically partitioned simulation. Our development of this decision process has two weaknesses. One is inherent in the model; certain parameters of the model are difficult to determine exactly. Nevertheless, these parameters can be estimated, so that the decision model is usable. The second weakness is the lack of any performance analysis of a simulation using the decision process. While we can defend this omission on the grounds that the decision process is optimal (given the modeling assumptions), we do think that a performance study would further our understanding of this decision model and the sensitivity to its parameters. Such a study joins our list of future research efforts.

10.3. Future Research

Throughout this thesis, we have identified areas that we feel merit further research. We summarize these areas in their order of importance.

(i) *Relax model assumptions.* Some of the assumptions in our model are restrictive. We will investigate ways of relaxing these assumptions, and the implications of doing so.

(ii) *Characterize simulations amenable to partitioning.* It is fruitless to distribute a simulation if its internal structure prohibits it. We need to develop ways of analyzing a simulation to determine its suitability for parallel execution.

(iii) *Algorithm development and testing.* We should continue developing and testing static and dynamic partitioning algorithms.

(iv) *Further statistical work.* We should take a closer look at means of determining Bayesian prior distributions, and the sensitivity of our model to changes in the prior. We should address the issue of how many observations are needed to produce an acceptable partition.

We sketch our ideas about each of these areas.

Many simulations' components do not have constant logical delays associated with them. An important extension to our work would consider ways in which we could treat such simulations. By relaxing the assumption of constant delays, we lose our ability to numerically analyze the work graph and its derivative graphs. We could define corresponding graphs, but they would certainly be too large for computational purposes. It may be possible to circumvent these difficulties by concentrating on the behavior of the simulation during the observed cycles. Given an observed cycle, we would define a graph corresponding somehow to the hyper-execution graph. Ideally, the length of the longest path through this graph would be equal to the cycle execution time under some extension of hyper-message synchronization to this system. Our statistical analysis could then be concentrated on this graph rather than the hyper-execution graph. One possible graph is

defined as follows. The nodes of the graph are the functional evaluations during the cycle. We suppose that every functional's **minimal** logical delay is known, so that the minimal time between an evaluation of T_i and a (transitively) subsequential evaluation of T_j can be calculated. Then a work node $W = (t, T_i)$ has an incoming arc from every node $V = (s, T_j)$ such that $s < t$, and the difference between V's logical completion time and W's logical initiation time is greater than the minimal time it takes an evaluation of T_j to affect an evaluation of T_i. Such a graph expresses precedence which must be satisfied; we hope that it expresses all of the precedence which must be satisfied.

After relaxing our model assumptions, we should be able to treat a more general class of simulations. We can then consider the problem of analyzing a simulation for the possibility of parallel execution. This analysis will still have to be statistically based. It could be based on a better theory of lower bounds on execution time. It will certainly have to take into account the synchronization needs of a distributed simulation. We will probably need a more finely tuned mechanism for specifying when a functional's state change must affect the state of another functional (this would come out of relaxing the model). We will need to look at the collection of observed cycles and consider the variablity in the cycle behavior. Too much variability probably defeats a static partitioning.

A promising different approach to partitioning is *hierarchical* partitioning. A model of simulation like DEVS [Zei84] defines the simulation in structured levels. Such structure should give a partitioning algorithm further information about the nature of the simulation. A hierarchical model of simulation might lend itself to a top-down approach to partitioning rather than the atomic-level approach we have examined. A hierarchical approach would probably be easier to implement in a simulation system than an implementation of an atomic-level approach.

We also need to examine the sensitivity of our dynamic repartitioning decision process. Determining the optimal decision policy can be computationally expensive, and it may be that we can easily approximate the optimal decision thresholds (for example, choose $\pi_n = .99$ for all n) without serious consequence.

To complete our proposed statistical approach, we should take a formal look at determining how many cycle observations are "enough". The theory of sequential analysis might be applicable to this problem. We should also determine whether any significant gain is achieved by using prior distributions other than the uninformed priors. If so, we need to automate the construction of the prior distributions.

10.4. Conclusions

The single most important conclusion we offer in this thesis is that simulations need to be **statistically** partitioned for parallel processing. Towards this end, we have developed models, algorithms, and means of analysis to support statistical partitioning. The bulk of our work is independent of any particular partitioning algorithm; we have provided an analytic framework in which the partitioning and analysis of distributed simulations can be approached. This work is prerequisite to any development and analysis of simulation partitioning algorithms. This work is also prerequisite to any analysis which determines whether or not a simulation is even ameanable to parallel execution. The automated partitioning of simulations for parallel processing is the next logical step in the study of distributed simulation; our work is thus an important step furthering this field of study. We next argue that our work is important in a more general context.

An abstracted view of our partitioning problem sees the issue as one of *placement*. We need to place "entities" in "places" so as to improve some measure of performance. The entities are related to each other in a non-static way, and the costs associated with the way we place the entities are also dynamically changing. The dynamic behavior of these relations and costs is too complicated to statically analyze, so we propose to **observe** the

dynamic system, and make our placement decisions on the basis of these observations. We note that this scenario describes systems other than distributed simulations. For example, [Sta84a] discusses the problem of program restructuring to reduce page faults; this problem fits in the described scenario (although it lacks the parallel component of our problem). Synchronization requirements are not unique to distributed simulation; some variation of our approach could conceivably be applied to partitioning some larger class of general programs.

In this thesis we have examined the issue of automatically partitioning a simulation for parallel execution. We have provided algorithms and means of analysis to deal with this problem. In one sense, our work is theoretical, as attested to by the plethora of definitions, lemmas and theorems. Still, this theory has been developed with a practical goal always in sight. The work presented here is focused entirely on problems related to modeling, analyzing and partitioning distributed simulations. While this work is important in the context of distributed simulation, it is also applicable to a larger class of problems in computer science. Further research efforts will continue our development in the context of distributed simulation, and will also investigate means of applying our approach to more general problems.

Appendix A

Numerical Solution Techniques

In Chapter 9 we demonstrated the existence of probability thresholds π_n which define the optimal stationary repartitioning decision policy. For this result to be of practical use, we must be able to determine these thresholds. A useful closed non-recursive solution is not generally possible, so we consider a numerical solution. We first discuss an exact recursive solution procedure, and note that the nature of the problem prohibits us from using this procedure for very many recursive steps. Next, we demonstrate how one *plcc* function may be optimally approximated with another for any desired degree of accuracy. We then propose a comprehensive approach to the approximation of the π_n, and show that the overall error in the optimal cost functions can be made as small as desired.

Exact Segment Endpoint Mapping

We first show how to numerically find the thresholds π_n in such a way that the only error is due to digitizing real numbers. We also show that the nature of the problem prohibits the general use of this procedure.

Lemma 9-3 shows that for every n, both $V(<\cdot,n>)$ and $ECC(<\cdot,n>)$ are piecewise linear functions of p. In theory, we can represent these functions by their sequence of (two-dimensional) linear segment endpoints $(p,f(p))$. We next develop a procedure which exactly determines $V(<\cdot,n>)$'s segment endpoints given $V(<\cdot,n+1>)$'s segment endpoints. Discussion of this procedure is aided by naming the domain component of these endpoints.

Definition A-1: *Domain Transition Point*

Let f be a piecewise linear function of p, and let p_0 be a point in f's domain such that

$$f(p_0) = m_1 \cdot p_0 + b_1$$

$$= m_2 \cdot p_0 + b_2$$

for some $m_1 \neq m_2$ and $b_1 \neq b_2$. Then p_0 is said to be a *domain transition point*, and the two-dimensional point $(p,f(p))$ is said to be a *segment endpoint* for f. 0 and 1 are considered to be domain transition points, and $(0,f(0))$ and $(1,f(1))$ are considered to be segment endpoints.

□

We are principally interested in characterizing $V(<\cdot,n>)$'s linear segments as a function of $V(<p,n+1>)$'s. We recall that

$$V(<p,n>) = \min\left\{ECC(<p,n>),\ ECT(<p,n>)\right\},$$

and that $ECT(<\cdot,n>)$ is simply linear in p. We need then to consider $ECC(<\cdot,n>)$'s linear segments as a function of $V(<\cdot,n+1>)$'s. The first step is to identify the value of $ECC(<p,n>)$ at p in terms of values of $V(<\cdot,n+1>)$.

Lemma A-1: Let $p \in [0,1]$, and suppose that m_1, b_1, m_2, b_2 are numbers such that

$$V(<p^c(p),n+1>) = m_1 \cdot p^c(p) + b_1$$

and

$$V(<p^{\bar{c}}(p),n+1>) = m_2 \cdot p^{\bar{c}}(p) + b_2.$$

Then

$$ECC(<p,n>) = m_3 \cdot p + b_3$$

where

$$m_3 = G + (1-\phi)\cdot\left[(1-\beta)\cdot m_1 + \beta \cdot m_2\right] + (1-\alpha-\beta)\cdot(b_1 - b_2) \qquad \text{(A-1)}$$

and

$$b_3 = \phi \cdot \left[(1-\beta)\cdot m_1 + \beta \cdot m_2\right] + \alpha \cdot b_1 + (1-\alpha)\cdot b_2. \qquad \text{(A-2)}$$

Proof: By the definition of $ECC(<p,n>)$ we have

$$ECC(<p,n>) = p \cdot G + q^c(p)\cdot\left[m_1 \cdot p^c(p) + b_1\right] + (1 - q^c(p))\cdot\left[m_2 \cdot p^{\bar{c}}(p) + b_2\right]. \qquad \text{(A-3)}$$

The expressions given for m_3 and b_3 follow directly by expanding p^c, $p^{\bar{c}}$, and q^c into their expressions as functions of p, and then algebraically collecting coefficients of p.

□

The nature of $ECC(<\cdot,n>)$'s linear response at p depends on the nature of the linear responses of $V(<\cdot,n+1>)$ at $p^c(p)$ and $p^{\bar{c}}(p)$. To determine $ECC(<\cdot,n>)$'s segment endpoints in terms of $V(<\cdot,n+1>)$'s segment endpoints, we need to consider consider how $p^c(\cdot)$ and $p^{\bar{c}}(\cdot)$ behave as functions of p.

Lemma A-2: $p^c(\cdot)$ and $p^{\bar{c}}(\cdot)$ are increasing continuous functions of p.

Proof: The continuity of $p^c(\cdot)$ and $p^{\bar{c}}(\cdot)$ follow from the observation that both the numerator and denominator in their respective quotient forms are continuous, and that the denominators are never 0. Equation 9–6 gives the derivative of $p^{\bar{c}}(\cdot)$ with respect to p; this derivative is positive over [0,1]. Precisely the same treatment can be made to the function $p^c(\cdot)$, yielding again a positive first derivative. Both $p^c(\cdot)$ and $p^{\bar{c}}(\cdot)$ are thus increasing and continuous.

□

By the closure of continuity under functional composition, it follows that $V(<p^c(\cdot),n+1>)$ and $V(<p^{\bar{c}}(\cdot),n+1>)$ are continuous functions of p. From this, we observe

Theorem A-1: Let $p \in [0,1]$, and let m,b be numbers such that $ECC(<p,n>) = m \cdot p + b$. If neither $p^c(p)$ nor $p^{\bar{c}}(p)$ is a domain transition point for $V(<\cdot,n+1>)$, then there exists a maximal interval $[l, u]$ containing p such that $ECC(<q,n>) = m \cdot q + b$ for all $q \in [l, u]$.

Proof: Let l^c and u^c denote, respectively, $V(<\cdot,n+1>)$'s greatest domain transition point less than $p^c(p)$, and its least domain transition point greater than $p^c(p)$. Similarly define $l^{\bar{c}}$ and $u^{\bar{c}}$ for $p^{\bar{c}}(p)$. It follows that there are numbers m_1, b_1, m_2, b_2 such that

$$V(<p,n+1>) = m_1 \cdot p + b_1 \quad \textit{for all } p \in [l^c,u^c]$$

and

$$V(<p,n+1>) = m_2 \cdot p + b_2 \quad \textit{for all } p \in [l^{\bar{c}},u^{\bar{c}}]$$

Now let δ_l^c be the solution to the equation

$$p^c(\delta_l^c) = l^c$$

and let δ_u^c be the solution to the equation

$$p^c(\delta_u^c) = u^c.$$

The increasing continuity of $p^c(\cdot)$ ensures that if $p \in [\delta_l^c, \delta_u^c]$, then $p^c(p) \in [l^c, u^c]$. Furthermore, $[\delta_l^c, \delta_u^c]$ is the largest interval such that

$$p \in [\delta_l^c, \delta_u^c] \implies V(<p^c(p),n+1>) = m_1 \cdot p^c(p) + b_1.$$

We similarly define an interval $[\delta_l^{\bar{c}},\delta_u^{\bar{c}}]$ by considering $p^{\bar{c}}(p)$, and note that it too is maximal. Then let $l = \min(\delta_l^c, \delta_l^{\bar{c}})$ and $u = \min(\delta_u^c, \delta_u^{\bar{c}})$. By construction, for any $p \in [l, u]$ we have

$$V(<p^c(p),n+1>) = m_1 \cdot p + b_1$$

and

$$V(<p^{\bar{c}}(p),n+1>) = m_2 \cdot p + b_2.$$

Thus, by lemma A–1, there exists m, b such that

$$ECC(<p,n>) = m \cdot p + b$$

for all $p \in [l, u]$. Furthermore, $[l, u]$ is the largest interval for which this is true.

□

We see that to find $ECC(<p,n>)$'s domain transition points (and hence its segment endpoints), we simply find those p that are mapped by either p^c or $p^{\bar{c}}$ onto $V(<\cdot,n+1>)$'s domain transition points. We state this observation as a corollary.

Corollary A-1: Let T be the set of all $V(<\cdot,n+1>)$'s domain transition points, and let Q be the set of all non-negative solutions to either of the following equations

$$p^c(q) = t \qquad for\ t \in T \tag{A-4}$$

$$p^{\bar{c}}(q) = t \qquad for\ t \in T. \tag{A-5}$$

Then Q is the set of all domain transition points for $ECC(<p,n>)$.

□

Once $ECC(<\cdot,n>)$'s domain transition points are identified, the piecewise linear structure is revealed. Each segment's linear coefficients is determined by lemma A-1. We can store these endpoints in a linear list, ordered by value of the domain transition points. We can achieve this sorted order by exploiting the increasing nature of p^c and $p^{\bar{c}}$. We may suppose that $V(<\cdot,n+1>)$'s domain transition points are sorted. We first calculate all of $ECC(<\cdot,n>)$'s domain transition points generated by equation A-4. We use an increasing sequence of ts in this process; the increasing nature of p^c ensures that the sequence of solutions is increasing. The same reasoning applies to the domain transition points generated by equation A-5. We can then merge the two increasing sequences to produce all of $ECC(<\cdot,n>)$'s domain transition points in increasing order. If $V(<\cdot,n+1>)$ has D domain transition points, then $ECC(<\cdot,n>)$ has as many as $2{\cdot}D$; the computational complexity of determining $ECC(<\cdot,n>)$'s domain transition points is seen to be $O(2{\cdot}D)$.

Our ultimate goal is to determine the linear segments of $V(<\cdot,n>)$. $V(<\cdot,n>)$'s linear segments are the same as $ECC(<\cdot,n>)$'s whenever $ECC(<p,n>) \leqslant ECT(<p,n>)$. $ECT(<\cdot,n>)$ is known to be linear; given the function $V(<\cdot,n+1>)$ it is easy to calculate ECT's linear coefficients by means of equation 9-5. We determine $ECC(<\cdot,n>)$'s domain transition points δ_1 and δ_2 between which ECT intersects ECC, and calculate the domain point π_n at which they intersect. All of $ECC(<\cdot,n>)$'s domain transition points less than π_n are domain transition points of $V(<\cdot,n>)$; π_n and 1 are the only other domain transition points of $V(<\cdot,n>)$.

We have shown how to numerically determine the thresholds π_n without error (other than that error inherent in digitizing real numbers). If $V(<\cdot,n>)$ has D domain transition points, the complexity of this procedure is $O(2{\cdot}D)$; while the procedure is computationally efficient in the number of domain points, the number of domain points may double at each step in the recursive solution. Finding π_n exactly thus has complexity $O(2^{N-n})$. We consider next an approximation technique to use in place of the exact solution.

Approximating *plcc* Functions

Since an exact solution of the π_n's is not always feasible, we must use approximations. We next consider approximating a *plcc* function having k segments by a *plcc* function with j segments, $j \leqslant k$. The approximation method is shown to be computationally optimal within a certain class of approximations.

Our proposed technique approximates a concave piecewise linear function with another concave piecewise linear function such that (i) the error is bounded by ϵ, and (ii) within a class of approximations, ours uses the minimal number of linear segments necessary to bound the error by ϵ. We first define the class of approximations on which it is optimal.

Definition A-2: *Interior Approximation*

Let g be a *plcc* function, and let a be a piecewise linear approximation to g such that if $(p,a(p))$ is a segment endpoint of a, then $a(p) = g(p)$. We say that a is an *interior approximation* to g.

□

One prime motivation for using an interior approximation is that it will preserve concavity. Not all approximations will, as shown by figure A–1. Our proof that interior approximations are concave uses the concepts of the *interior* and *exterior* of a functional graph.

Definition A-3: *Interior of Functional Graph*

Let g be a function on [0,1], and let $x \in [0,1]$. A two–dimensional point (x,y) is said to be in g's *interior* if $y \leqslant g(x)$; otherwise (x,y) is said to be in g's exterior.

□

Lemma A-3: Let g be a concave function on [0,1] and let a be an piecewise linear interior approximation. Then a is concave.

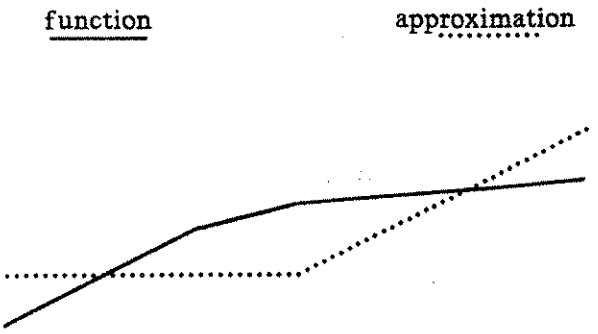
Proof: Suppose not. Then there exist p_1, p_2, and λ in [0,1] such that

$$a(\lambda \cdot p_1 + (1 - \lambda) \cdot p_2) > \lambda \cdot a(p_1) + (1 - \lambda) \cdot a(p_2).$$

Geometrically, this means that there is a chord on a's function graph that exceeds a over some interval (x_1,x_2) where a and the chord intersect at both x_1 and x_2. Since a is piecewise linear, its two intersections with this chord must lie on two different linear segments. Consequently, there is at least one of a's domain transition points p_δ in (x_1,x_2). But, $a(p_\delta) = g(p_\delta)$ by construction; hence the chord exceeds g's graph over some subinterval of (x_1,x_2) as well. The concavity of g ensures that $a(p) \leqslant g(p)$ for all p, consequently the chord endpoints $(p_1, a(p_1))$ and $(p_2, a(p_2))$ lie in the interior of g's functional graph. A straight line between any two points in the interior of a concave function's graph must remain in the interior. Thus there is a contradiction, having already shown that part of the chord lies in g's exterior. a is therefore concave. Figure A–2 illustrates this argument.

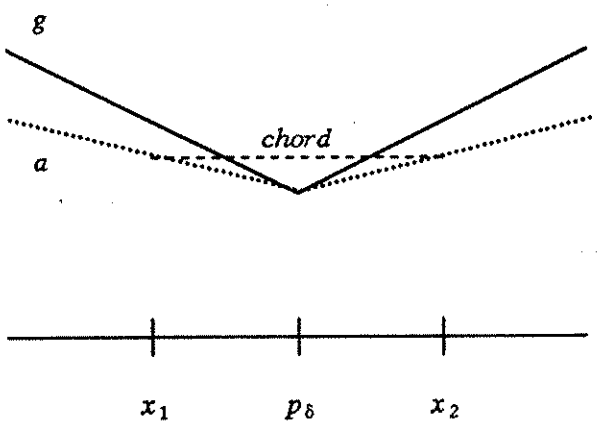
□

A piecewise linear interior approximation a of the *plcc* function g can be thought of as a sequence of intervals over which g is approximated by a straight line forming a chord to g's function graph. We call this straight line a *first order approximation*, and call its domain an *approximation interval*. We first consider where the maximal error in such an interval occurs.



Non–Concave Approximation

Figure A–1



Lemma A–3

Figure A–2

Lemma A-4: Let $g(\cdot)$ be a *plcc* function on $[l, u]$, and let

$$a(p) = m \cdot p + b$$

be a first order approximation to g on $[l, u]$. Among all of g's segments on $[l, u]$, let m_0 be the least slope greater than or equal to m, and let $(x_0, g(x_0))$ denote the right endpoint of this segment. Then the error $g(p) - a(p)$ on $[l, u]$ is maximized at x_0.

Proof: Consider any segment of g over $[l, u]$ with slope $m_s \geq m$; let $[l_s, u_s]$ be the domain interval of this segment. Let $p \in [l_s, u_s]$, and consider the difference

$$g(p) - a(p) = m_s \cdot p + b_s - m \cdot p + b$$

$$= (m_s - m) \cdot p + (b_s - b).$$

The expression above shows that the error is an increasing function of p. It follows that the error at u_s is greater than or equal to the error at l_s. An entirely similar argument shows that if $m_s < m$, then the error at l_s is greater than the error at u_s. The result follows by observing that the sequence of slopes of g's segments from l to u is strictly decreasing, a result of g's concavity.

□

Given the left endpoint of an approximation interval, our technique will find the unique right endpoint such that the maximal error of the resultant first order approximation is exactly ϵ. To show that this technique is feasible, we show that the maximal error over an approximation interval is a continuous, strictly increasing function of the right interval endpoint. The first step is a demonstration that the slope of the first order approximation is strictly decreasing in the right endpoint.

Lemma A-5: Let g be a *plcc* function, and let $a(\cdot, l, u)$ be the first order approximation over $[l, u]$. Then for fixed l and u large enough that $a \neq g$ somewhere on $[l, u]$, the slope of $a(\cdot, l, u)$ is a continuous strictly decreasing function of u.

Proof: The slope $m(u)$ of $a(\cdot, l, u)$ is given by

$$m(u) = \frac{g(u) - g(l)}{u - l}. \tag{A-6}$$

The continuity of $m(u)$ follows immediately from the continuity of its constituent numerator and denominator. Now the derivative of $m(u)$ with respect to u is

$$\frac{d}{du} m(u) = \frac{(u - l) \cdot m_a - (g(u) - g(l))}{(u - l)^2}$$

where m_a is the slope of g at $g(u)$. Whenever g and a are not identical on $[l, u]$, the concavity of g forces m_a to be strictly less than $m(u)$; it follows from equation A-6 that the numerator of the derivative above is less than 0. Since the denominator is positive, $m(u)$ is decreasing in u.

□

An immediate implication of this lemma is stated as a corollary.

Corollary A-2: For fixed l and fixed $p > l$, $a(p,l,u)$ is a continuous decreasing function of u.

□

Next we show that the maximal error over an approximation interval is a continuous, increasing function of the right endpoint.

Theorem A-2: Let g be a *plcc* function on [0,1], let $0 \leqslant l < 1$ be fixed, and let $\gamma(u)$ denote the maximal error of the interval approximation $a(\cdot,l,u)$ over $[l, u]$, $l < u$. Then $\gamma(u)$ is a continuous, increasing function of u.

Proof: For any u, the error $\gamma(u)$ is given by

$$\gamma(u) = m_0(u) \cdot x_0(u) + b_0(u) - a(x_0(u),l,u) \qquad \text{(A-7)}$$

where $m_0(u)$ and $b_0(u)$ are the parameters of g's segment over each of which the maximum error occurs, and $x_0(u)$ is its right domain endpoint. As u increases, $m_0(u)$, $b_0(u)$, and $x_0(u)$ are constant so long as $m_0(u)$ remains the least of g's segment slopes greater than $m(u)$. On intervals $[u_1,u_2]$ such that $m_0(u)$, $b_0(u)$ and $x_0(u)$ are constants, equation A–7 shows that $\gamma(u)$ is increasing, since $a(x_0(u),l,u)$ is decreasing. To complete the proof, we need only show that $\gamma(u)$ is continuous at points u_δ where $m_0(u)$ changes. $m_0(u)$ changes precisely g's segment to the immediate right of the one defining $m_0(u)$ has a slope equal to the slope of the first order approximation. The approximation and this latter segment are then parallel, so that the error is uniform across the segment. In particular, the error is equal at this segment's two endpoints, making the error continuous at u_δ.

□

We use the fact that the approximation error is a continuous increasing function of the right endpoint to show that we can pick a right endpoint so the the maximal error is *exactly* ϵ.

Lemma A-6: Let l be fixed, and let $[u_1,u_2]$ be an interval such that

(i) g is linear on $[u_1,u_2]$,

(ii) $\gamma(u_1) < \epsilon \leqslant \gamma(u_2)$.

Then there exists a point $u_t \in [u_1,u_2]$ such that $\gamma(u_t) = \epsilon$.

Proof: The existence of such a point follows directly from the continuity of $\gamma(\cdot)$. We show how to construct the point u_t. We suppose that $g(u) = m \cdot u + b$ for $u \in [u_1,u_2]$, and suppose that the segment endpoint at which the maximal error occurs is (x_0,y_0). We let (l,y_l) be the left endpoint of the first order approxima–

tion. Given u as the right endpoint, the slope of the first order approximation is given by

$$m(u) = \frac{m \cdot u + b - y_l}{u - l}$$

and the intercept by

$$b(u) = y_l - m(u) \cdot l.$$

The error is given by

$$\gamma(u) = y_0 - m(u) \cdot x_0 + b(u)$$

and through simple algebraic rearrangement, we solve the equation $\gamma(u_t) = \epsilon$ for u_t to find

$$u_t = \frac{-l \cdot (y_l + y_0 - \epsilon) - b \cdot (x_0 + l)}{(x_0 + l) \cdot m - (y_l + y_0 - \epsilon)}$$

□

We can now describe our approximation technique. Given a *plcc* function g on [0,1], and desired accuracy of ϵ, we take 0 as a left endpoint. In increasing order of i, we examine g's *ith* smallest domain transition point x_i, calculating the first order approximation's maximal error over $[0,x_i]$. We thus identify the least x_i such that the error over $[0,x_i]$ exceeds ϵ. We know then that the interval $[x_{i-1},x_i]$ contains a point x_t such that the maximal approximation error over $[0,x_t]$ is exactly ϵ : we use lemma A–5 to calculate this x_t. We choose x_t as the interval's right hand endpoint (and the next interval's left endpoint). This procedure is repeated until the entire interval [0,1] is covered by approximation intervals.

We demonstrate that the complexity of this procedure is linear in the number of g's domain transition points. Consider the behavior of the algorithm in determining one interval endpoint. We can use a pointer *maxp* to point to the segment endpoint defining the maximal error, and a pointer *nxtp* to point at one of g's domain transition endpoints. Initially, both pointers are set to the interval's left endpoint, l. Iterating on i, g's next domain transition point x_i is examined, and the slope of the first order approximation on $[l, x_i]$ is calculated. The segment defining the maximum error over this interval cannot lie to the left of the current position of *maxp*, since the slope of the first order approximation is decreasing in its right endpoint. The slopes of segments to the right of the one pointed to by *maxp* are scanned until the appropriate one is found. The maximal error on $[l,x_i]$ is calculated; we iterate again on i if this error is less than ϵ. Once the interval holding the right endpoint is found, determination of that endpoint requires constant calculation. In the course of finding x_t then, each of g's domain transition points in $[l,x_t]$ is scanned at most once by *maxp* and exactly once by *nxtp*. It follows that the complexity of covering [0,1] with approximation intervals is linear in the number of g's domain transition points.

Another useful feature of our proposed approximation technique is that it minimizes the number of approximation intervals necessary to bound the maximum error by ϵ. The demonstration of this fact is aided by the following definition.

Definition A-4: *Minimal Segments in Interior Approximation*

Let g be a *plcc* function on $[a,b]$, and let $\epsilon \geq 0$. Then $N_\epsilon([a,b])$ denotes the minimal number of segments required by an interior approximation to g on $[a,b]$ to bound the maximal error by ϵ.

□

Then we have the following.

Theorem A-3: Let g be a *plcc* function on [0,1], and let a be the interior approximation constructed by the proposed technique. Then for every ϵ, a has exactly $N_\epsilon([0,1])$ linear segments.

Proof: Let $x_1, \cdots, x_n$ be the sequence of interval endpoints chosen by the approximation technique. We inductively show that for every $k = 1, \cdots, n$,

(i) $N_\epsilon([0,x_k]) = k$;

(ii) For every $\delta > 0$, $N_\epsilon([0,x_k + \delta]) > k$.

As the induction base, we consider $k = 1$ and observe that both conclusions are immediately satisfied. For the induction hypothesis, we suppose that the conclusions are true for some $k-1$. By the induction hypothesis, $[0,x_{k-1}]$ is minimally covered by $k-1$ segments. The construction of x_k ensures that the maximal error on $[x_{k-1},x_k]$ is precisely ϵ, whence we have $N_\epsilon([0,x_k]) \leq k$. But $N_\epsilon([0,x_{k-1} + \delta]) > k-1$ for any $\delta > 0$, so we must have $N_\epsilon([0,x_k]) = k$. To show that conclusion (ii) is true as well, pick $\delta > 0$ and consider any interior approximation of g on $[0,x_k + \delta]$ whose maximal error is bounded by ϵ; let y_k be the left endpoint of the approximation segment whose right endpoint is $x_k + \delta$. We show that $y_k > x_{k-1}$: we have shown that a first order approximation's error is increasing in its right domain endpoint; entirely similar arguments establish that this error increases as the left domain endpoint decreases. The fact that $y_k > x_{k-1}$ then follows from the observation that the maximal error of the first order approximation to $[x_{k-1},x_k + \delta]$ is greater than ϵ. By the induction hypothesis, we have $N_\epsilon([0,y_k]) \geq k$; consequently, $N_\epsilon([0,x_k + \delta]) > k$, completing the induction. Conclusion (i) with $k = n$ proves the theorem.

□

Approximating π_n

We now describe a comprehensive approach to the calculation of the thresholds π_n. We first outline the overall approach. We then describe a closed form solution for $V(< \cdot, n >)$ when $\pi_n = 1$, and finally show that the error between the true optimal cost functions and their approximations can be made arbitrarily small.

Chapter 9 demonstrated that the continue decision is optimal if no change has yet been detected. We suppose that change is first detected at time t. The full decision model parameters can then be estimated, and the decision thresholds π_n for $n \geqslant t$ can be calculated. Starting with $n = N$, some initial number of thresholds $\pi_N, \cdots, \pi_{N-k}$ can be determined exactly as previously shown. The remaining thresholds $\pi_{N-k-1} \cdots \pi_t$ will be approximated. We suppose an error tolerance ϵ has been chosen; let ϵ_0 be $\frac{\epsilon}{N-k-t}$. The proposed approximation procedure is now described recursively. We consider the function $V(<\cdot, N-k>)$ to be its own approximation. Then for each i, $N-k > i \geqslant t$, the approximation technique takes the approximation to $V(<\cdot, i+1>)$ and uses the exact segment mapping technique to approximate functions $ECC(<\cdot, i>)$ and $ECT(<\cdot, i>)$. The approximation of π_i is taken to be the intersection of these two functions. Then, in order to reduce the number of domain transition points, we take the optimal interior approximation of this first approximation to $V(<\cdot, i>)$, forcing the maximal error between the first and second approximations to be precisely ϵ_0. It is important to note that ϵ_0 bounds the error between the two approximations of $V(<\cdot, i>)$: it does not bound the error between the true $V(<\cdot, i>)$ and either of these approximations. The domain transition points of the second approximation to $V(<\cdot, i>)$ are used to approximate $ECC(<\cdot, i-1>)$ and $ECT(<\cdot, i-1>)$, and so on.

Our derivation of the optimal policy structure showed that $N - K_0 + 1$ was a critical threshold in time: for all $n \geqslant N - K_0 + 1$, $\pi_n = 1$; it is equivalent to say that $V(<p,n>) = ECC(<p,n>)$ for all such n. This knowledge allows us to derive a closed form expression for $V(<\cdot, N-K_0+1>)$; the exact numerical solution phase of the calculation can then begin at $n = N - K_0$.

Lemma A-7:

$$V(<p,N-K_0+1>) = G \cdot \left| p \cdot \frac{1-(1-\phi)^{N-K_0+2}}{\phi} + N - K_0 + 1 - (1-\phi) \cdot \frac{1-(1-\phi)^{N-K_0+1}}{\phi} \right|.$$

Proof: For $p \in [0,1]$ we define the operator $(\ p\ )_n^*$ by

(i) $(\ p\ )_0^* = p$;

(ii) $(\ p\ )_n^* = p^*((\ p\ )_{n-1}^*)$ for $n > 0$.

We briefly note that for any n this operator is linear in p, a fact easily shown by induction. We now inductively prove that for $j \leqslant K_0 - 1$,

$$ECC(<p,N-j>) = G \cdot \sum_{i=0}^{j} (\ p\ )_i^*. \qquad \text{(A-8)}$$

We use $j = 0$ as the induction base, and note that the equation above reduces to $p \cdot G$ as required. We suppose that equation A-8 is true for some $j - 1 < K_0 - 1$, and consider the function

$$ECC(<p,N-j>) = p \cdot G + E_v[<p,N-(j-1)>]$$

$$= p \cdot G + q^c(p) \cdot ECC(<p^c(p),N-(j-1)>) + q^{\bar{c}}(p) \cdot ECC(<p^{\bar{c}}(p),N-(j-1)>)$$

$$= p \cdot G + q^c(p^c(p)) \cdot G \cdot \sum_{i=0}^{j-1} (p^c(p))_i^* + q^{\bar{c}}(p^c(p)) \cdot G \cdot \sum_{i=0}^{j-1} (p^{\bar{c}}(p))_i^*$$

$$= p \cdot G + G \cdot \sum_{i=0}^{j-1} p^*((p)_i^*) \qquad \text{by linearity}$$

$$= G \cdot \sum_{i=0}^{j} (p)_i^*$$

which completes the induction argument. The lemma's conclusion follows from the identity [Knu73]

$$\sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1 - x} \quad \text{when } x \neq 1$$

and some algebraic rearrangement.

□

The proposed approximation procedure asks that the user supply a desired error bound ϵ; the approximation procedure then forces the maximal error between a function and its interior approximation to be exactly $\epsilon_0 = \frac{\epsilon}{N-k-t}$. We next demonstrate that the proposed procedure does in fact bound the total error in the optimal cost function by ϵ.

In the course of the approximation process, the function $V(< \cdot ,n >)$ is approximated twice. The first approximation is denoted by $AV(< \cdot ,n >)$, and is the one used to approximate π_n. $AV(< \cdot ,n >)$ is then itself approximated by the optimal interior approximation, denoted by $AV_o(< \cdot ,n >)$. $AV_o(< \cdot ,n >)$ is used to create $AV(< \cdot ,n-1 >)$, and so on. Lemma A–8 shows that the approximation error in any optimal cost function is bounded.

Lemma A-8: For every $i = N - k, N - k - 1, \cdots, t$ and every p,

$$| V(<p,i>) - AV(<p,i>)| \leq (N - k - i) \cdot \epsilon_0.$$

Proof: We induct on i, using $i = N - k$ as a base. $AV(< \cdot ,N-k >)$ is identical to $V(< \cdot ,N-k >)$, as $AV(< \cdot ,N-k >)$ denotes the function obtained from an exact segment mapping of $V(< \cdot ,N-k+1 >)$. The induction base is thus satisfied. We suppose that the conclusion is true for some $i+1$, and consider the optimal interior approximation to $AV(< \cdot ,n+1 >)$. By construction, we have

$$| AV_o(<p,i+1>) - AV(<p,i+1>)| \leq \epsilon_0$$

for all p. Then

$$| AV_o(<p,i+1>) - V(<p,i+1>)| = \qquad \text{(A–9)}$$

$$|(AV_o(<p,i+1>) - AV(<p,i+1>)) + (AV(<p,i+1>) - V(<p,i+1>))|$$

$$\leqslant |AV_o(<p,i+1>) - AV(<p,i+1>)| + |AV(<p,i+1>) - V(<p,i+1>)|$$

$$\leqslant \epsilon_0 + (N - k - i - 1)\cdot\epsilon_0$$

$$= (N - k - i)\cdot\epsilon_0.$$

Next consider the approximation $AECC(<\cdot,i+1>)$ to the true $ECC(<\cdot,i+1>)$:

$$AECC(<p,i>) = p\cdot G + q^c(p)\cdot AV_o(<p^c(p),i+1>) + q^{\bar{c}}(p)\cdot AV_o(<p^{\bar{c}}(p),i+1>).$$

Equation A–9 implies that

$$|(q^c(p)\cdot V(<p^c(p),i+1>) + q^{\bar{c}}(p)\cdot V(<p^{\bar{c}}(p),i+1>)) -$$

$$(q^c(p)\cdot AV_o(<p^c(p),i+1>) - q^{\bar{c}}(p)\cdot AV_o(<p^{\bar{c}}(p),i+1>))|$$

$$\leqslant q^c(p)\cdot(N - k - i)\cdot\epsilon_0 + q^{\bar{c}}(p)\cdot(N - k - i)\cdot\epsilon_0$$

$$= (N - k - i)\cdot\epsilon_0$$

so that for p less than or equal to the true (unknown) π_i we have

$$|AECC(<p,i>) - V(<p,i>)| \leqslant (N - k - i)\cdot\epsilon_0.$$

We can similarly show that for p greater than the true π_i

$$|AECT(<p,i>) - V(<p,i>)| \leqslant (N - k - i)\cdot\epsilon_0.$$

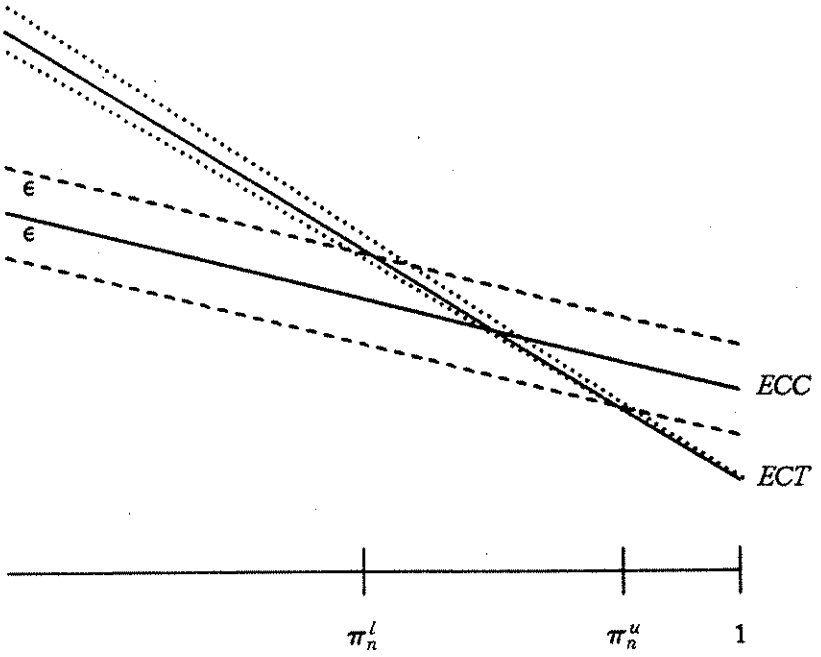
Thus we have

$$|AV(<p,i>) - V(<p,i>)| \leqslant (N - k - i)\cdot\epsilon_0$$

for all p, completing the induction.

□

The proposed approximation procedure does not attempt to bound the error on the approximated π_n. Given the approximations of $ECT(<\cdot,n>)$ and $ECT(<\cdot,n>)$ within some error bound ϵ we can construct upper and lower bounds π_n^l, π_n^u on the true π_n as illustrated by figure A–3 . Clearly, these bounds are sensitive to the slopes of the intersecting approximation segments. However, no matter what the approximated π_n are, we are assured that the expected cost of the resulting decision process is within ϵ of the true minimal expected cost. Even if the approximated π_ns are wildly different from the true ones, the bound on the cost function error ensures that



Error in Approximating π_n

Figure A–3

Appendix B

Sensitivity to G

An important parameter of our dynamic repartitioning decision process is the per time step speedup from repartitioning, G. The true value of G will not generally be known. However, given bounds on the true G, we might choose a point estimate of G between the bounds. Consideration of such a heuristic leads to a natural question: how good are the resulting estimations of π_n? In this section we treat this question, and find the intuitive result that these estimations are as good as the bounds on G are tight. Let G^u bound G from above, and denote the π_n generated by using $G = G^u$ by π_n^u; similarly define π_n^l for the lower bound on G, G^l. The major result of this appendix demonstrates general conditions under which the true π_n are tightly bounded from below by π_n^u, and are tightly bounded from above by π_n^l.

A caveat: our derivation of this section's result uses a slightly different decision model. We were not able to derive these same results for the model already described (we will later identify the missing steps which would provide the same results for this model). We first describe this model, and then embed this model into a system in which G is continuous. The optimal cost functions are shown to continuous and differentiable (almost everywhere) in the parameter G. We then show that the point of intersection (i.e., π_n) between functions $ECT(<p,n>)$ and $ECC(<p,n>)$ is a continuously decreasing function of G. Thus if we bound G from above and below, we know that the true threshold π_n lies between the π_n^l given by the system using G's lower bound, and π_n^u, given by the system using G's upper bound. Furthermore, we can predict the relative effect of using either pessimistic or optimistic point estimates of G: pessimistic estimations lead to higher estimations of π_n, while optimistic estimates lead to lower estimations of π_n.

Changing the Model

The decision model of Chapter 9 is general and powerful. It is particularly useful in its ability to model the continuation of the process after rejection of a newly calculated partition. This same feature makes further analysis of the optimal cost equations very difficult. Consequently, we now consider a slightly different model. Instead of allowing the process to continue after rejecting a new partition, we require it to stop. If a new partition is created and adopted, the same reward is earned as before. If a new partition is rejected at time n, a penalty $T \cdot G \cdot (N - n)$ is incurred; $0 < T \leq 1$. This penalty is seen to depend both on the number of remaining decision steps, and on the gain G. The penalty serves as an additional cost to help prevent premature repartition calculation decisions.

The optimal cost function equations for the new model are

$$V(<p,n>) = \min \begin{cases} D_d - p \cdot \left| G \cdot (N - n + 1) - D_r \right| + (1 - p) \cdot T \cdot G \cdot (N - n) \\ p \cdot G + E_v[<p,n>] \end{cases} \quad \text{(B-1)}$$

A complete analysis of this model could be made as before; the optimal policy for this

model is still characterized by a sequence $\pi_0, \cdots, \pi_N$. Of course, these thresholds need not be the same as the ones for the earlier model.

A Larger System

The optimal cost functions defined by equation B–1 implicitly assume that the parameter G is constant. We now consider G to be continuous. To show the system's dependence on G, we include G in the state $<p,n,G>$. Our first task is to show that the functions $V(<p,n,G>)$ are continuous and differentiable (almost everywhere) in G.

Lemma B-1: For every fixed n and every fixed $p \in [0,1]$, $V(<p,n,G>)$ is a continuous and finitely piecewise linear function of G.

Proof: We induct on n, using $n = N$ as the induction base. If $p \leqslant \pi_N$, then

$$V(<p,N,G>) = p \cdot G,$$

which is seen to be simply linear in G. If $p > \pi_N$, then

$$V(<p,N,G>) = D_d - p \cdot G + p \cdot D_r,$$

which again is linear in G. For the induction hypothesis, we suppose there is a n such that for all fixed $p \in [0,1]$, $V(<p,n+1,G>)$ is a continuous, finitely piecewise linear function of G. This assumption implies that for all fixed p the function $E_v[<p,n,G>]$ is a continuous finitely piecewise linear function of G. Thus the function

$$ECC(<p,n,G>) = p \cdot G + E_v[<p,n,G>)]$$

is continuous and finitely piecewise linear by the closure of such functions under addition. Equation B–1 shows that $ECT(<p,n,G>)$ is simply linear in G. For a given G_0, the definition of $V(<p,n,G_0>)$ depends on the relationship of p to the thresholds $\pi_n(G_0)$ that arise from the use of $G = G_0$. Both $ECC(<\cdot,n,G>)$ and $ECT(<\cdot,n,G>)$ are continuous and finitely piecewise linear in G. As functions of G then, they cannot intersect more than a finite number of times (if an interval $[G_1,G_2]$ occurs where the functions are identical everywhere on the interval, we consider only the interval endpoints to be true intersections). Thus the function $\min\{ECT(<p,n,G>), ECC(<p,n,G>)\}$ is continuous, and piecewise linear with only a finite number of linear segments. This completes the induction argument.

□

Two immediate consequences of this last observation are stated as corollaries.

Corollary B-1: For every fixed n and every fixed $p \in [0, 1]$, $V(<p,n,G>)$ is differentiable in G except at a finite number of points (a set of measure 0).

□

Corollary B-2: For every fixed n and every fixed $p \in [0, 1]$, both the left handed and right handed derivatives of $V(<p,n,G>)$ with respect to G exist.

□

The remaining analysis is concerned with the derivatives $\frac{\partial}{\partial G}V(<p,n,G>)$; the corollaries above state that these derivatives exist almost everywhere, and that the left and right-handed derivatives [Bar76] always exist. We let it be understood then that for any function F, $\frac{\partial}{\partial G}F(<p,n,G>)$ refers to the right-handed derivative, and so ensure the existence of the derivative under discussion.

Segment Parameter Derivatives

We now focus on the partial derivatives of $V(<\cdot,n,G>)$'s linear segment (in p) parameters, the slope and intercept. The main thrust of this analysis demonstrates conditions under which

$$\frac{\partial}{\partial G}ECC(<p,n,G>) \geqslant \frac{\partial}{\partial G}ECT(<p,n,G>).$$

The linear segments discussed here will always be the segments defined by considering $V(<\cdot,n,G>)$ as a function of p. We define $m_n^1(G)$ and $b_n^1(G)$ to be the parameters of $V(<\cdot,n,G>)$'s segment with $p = 0$ as the left domain endpoint. We likewise define $m_n^i(G)$ and $b_n^i(G)$ to be the parameters of $V(<\cdot,n,G>)$'s *ith* segment. We let L denote the number of segments $V(<\cdot,n,G>)$ has (letting L's dependence on n be understood). We identify $ECC(<\cdot,n,G>)$'s segment parameters in a similar way: $M_n^i(G)$ and $B_n^i(G)$; we let K enumerate $ECC(<\cdot,n,G>)$'s segments. These definitions cause a function's *ith* non-zero domain transition point to be the right endpoint for the *ith* segment, and the left endpoint for the $i+1st$ segment. Finally, we note that we may express the derivatives of $V(<\cdot,n,G>)$ and $ECC(<\cdot,n,G>)$ as

$$\frac{\partial}{\partial G}V(<p,n,G>) = \frac{\partial}{\partial G}m_n^i(G)\cdot p + \frac{\partial}{\partial G}b_n^i(G) \quad \text{for appropriate } i$$

and

$$\frac{\partial}{\partial G}ECC(<p,n,G>) = \frac{\partial}{\partial G}M_n^i(G)\cdot p + \frac{\partial}{\partial G}B_n^i(G) \quad \text{for appropriate } i.$$

Our first lemma shows that the sequence $\frac{\partial}{\partial G}m_n^i(G) + \frac{\partial}{\partial G}b_n^i(G)$, $i = 1,2,\cdots,L$ is decreasing.

Lemma B-2: For every n,G, and $i = 1,2,\cdots,L-1$

$$\frac{\partial}{\partial G}m_n^i(G) + \frac{\partial}{\partial G}b_n^i(G) \geqslant \frac{\partial}{\partial G}m_n^{i+1}(G) + \frac{\partial}{\partial G}b_n^{i+1}(G).$$

Proof: We induct on n, using $n = N$ as the base. If $\pi_N = 1$, then $V(<\cdot,N,G>)$ is strictly linear in p and the conclusion is trivially satisfied. If $\pi_N < 1$, then $V(<\cdot,N,G>)$ consists of two linear segments. For $p \leqslant \pi_N$, it is easily seen that

$$\frac{\partial}{\partial G}m_N^1(G) = 1 \qquad \frac{\partial}{\partial G}b_N^1(G) = 0.$$

For $p > \pi_N$, we have

$$\frac{\partial}{\partial G} m_N^2(G) = -1 \qquad \frac{\partial}{\partial G} b_N^2(G) = 0.$$

so that the induction base is satisfied. For the induction hypothesis we suppose that the lemma's conclusion is true for some $n+1$. We will first show that the sequence $\frac{\partial}{\partial G} M_n^i(G) + \frac{\partial}{\partial G} B_n^i(G)$ is decreasing in i. Let q_j be $ECC(< \cdot, n, G >)$'s *jth* domain transition point ($j \neq 0, K$), so that $M_n^j(G)$ and $B_n^j(G)$ define $ECC(< \cdot, n, G >)$'s linear response for p just less than (and equal to) q_j; $M_n^{j+1}(G)$ and $B_n^{j+1}(G)$ define $ECC(< \cdot, n, G >)$'s linear response at p just greater than (and equal to) q_j. Lemma A–1 shows that these parameters are themselves constructed from certain $V(< \cdot, n+1, G >)$ segment parameters: let a and z be integers such that $m_{n+1}^a(G)$ and $b_{n+1}^a(G)$ define $V(< \cdot, n+1, G >)$'s response at p arbitrarily close to $p^c(q_j)$ on the left, and $m_{n+1}^z(G)$ and $b_{n+1}^z(G)$ define $V(< \cdot, n+1, G >)$ response at p arbitrarily close to $p^{\bar{c}}(q_j)$ on the left. Equations A–1 and A–2 construct $M_n^j(G)$ and $B_n^j(G)$ in terms of these parameters. Now, $ECC(< \cdot, n, G >)$'s domain transition points are directly related to $V(< \cdot, n+1, G >)$'s domain transition points by way of equations A–4 and A–5 . If q_j is caused by a transition in $V(< \cdot, n+1, G >)$ at $p^c(q_j)$, then the resulting change in $ECC(< \cdot, n, G >)$ parameters is effected by replacing $m_{n+1}^a(G)$ with $m_{n+1}^{a+1}(G)$ and $b_{n+1}^a(G)$ with $b_{n+1}^{a+1}(G)$ in equations A–1 and A–2. Likewise, if q_j is caused by a transition in $V(< \cdot, n+1, G >)$ at $p^{\bar{c}}(q_j)$, we replace $m_{n+1}^z(G)$ with $m_{n+1}^{z+1}(G)$ and $b_{n+1}^z(G)$ with $b_{n+1}^{z+1}(G)$. If $V(< \cdot, n+1, G >)$ has transitions at both $p^c(q_j)$ and $p^{\bar{c}}(q_j)$, then both substitutions are made. By applying these observations to equations A–1 and A–2, a minor bit of algebra reveals that

(i) If $p^c(q_j)$ is a domain transition point for $V(< \cdot, n+1, G >)$ and $p^{\bar{c}}(q_j)$ is not, then

$$\left(\frac{\partial}{\partial G} M_n^j(G) + \frac{\partial}{\partial G} B_n^j(G)\right) - \left(\frac{\partial}{\partial G} M_n^{j+1}(G) + \frac{\partial}{\partial G} B_n^{j+1}(G)\right) =$$

$$(1-\beta)\cdot\left|\left(\frac{\partial}{\partial G} m_{n+1}^a(G) + \frac{\partial}{\partial G} b_{n+1}^a(G)\right) - \left(\frac{\partial}{\partial G} m_{n+1}^{a+1}(G) + \frac{\partial}{\partial G} b_{n+1}^{a+1}(G)\right)\right|$$

$$\geqslant 0 \quad \text{by the induction hypothesis;}$$

(ii) If $p^{\bar{c}}(q_j)$ is a domain transition point for $V(< \cdot, n+1, G >)$ and $p^c(q_j)$ is not, then

$$\left(\frac{\partial}{\partial G} M_n^j(G) + \frac{\partial}{\partial G} B_n^j(G)\right) - \left(\frac{\partial}{\partial G} M_n^{j+1}(G) + \frac{\partial}{\partial G} B_n^{j+1}(G)\right) =$$

$$\beta\cdot\left|\left(\frac{\partial}{\partial G}m_{n+1}^{z}(G)+\frac{\partial}{\partial G}b_{n+1}^{z}(G)\right)-\left(\frac{\partial}{\partial G}m_{n+1}^{z+1}(G)+\frac{\partial}{\partial G}b_{n+1}^{z+1}(G)\right)\right|$$

$\geqslant 0$ by the induction hypothesis;

(iii) If both $p^{c}(q_j)$ and $p^{\bar{c}}(q_j)$ are domain transition points for $V(<\cdot,n+1,G>)$ then

$$\left(\frac{\partial}{\partial G}M_{n}^{j}(G)+\frac{\partial}{\partial G}B_{n}^{j}(G)\right)-\left(\frac{\partial}{\partial G}M_{n}^{j+1}(G)+\frac{\partial}{\partial G}B_{n}^{j+1}(G)\right)=$$

$$(1-\beta)\cdot\left|\left(\frac{\partial}{\partial G}m_{n+1}^{a}(G)+\frac{\partial}{\partial G}b_{n+1}^{a}(G)\right)-\left(\frac{\partial}{\partial G}m_{n+1}^{a+1}(G)+\frac{\partial}{\partial G}b_{n+1}^{a+1}(G)\right)\right|$$

$$+\beta\cdot\left|\left(\frac{\partial}{\partial G}m_{n+1}^{z}(G)+\frac{\partial}{\partial G}b_{n+1}^{z}(G)\right)-\left(\frac{\partial}{\partial G}m_{n+1}^{z+1}(G)+\frac{\partial}{\partial G}b_{n+1}^{z+1}(G)\right)\right|$$

$\geqslant 0$ by the induction hypothesis.

Since q_j was chosen arbitrarily, the sequence $\frac{\partial}{\partial G}M_{n}^{i}(G)+\frac{\partial}{\partial G}B_{n}^{i}(G)$, $i=1,2,\cdots,K$ is decreasing. If $\pi_n=1$, then $ECC(<\cdot,n,G>)$ defines $V(<\cdot,n,G>)$ everywhere, and we are done.

We suppose that $\pi_n<1$; $V(<\cdot,n,G>)$ has only one segment whose parameters $m_n^i(G)$ and $b_n^i(G)$ are not identically $M_n^i(G)$ and $B_n^i(G)$: its last segment, defined by $ECT(<\cdot,n,G>)$. To complete the proof of the lemma, we show that the smallest possible value of $\frac{\partial}{\partial G}M_{n}^{i}(G)+\frac{\partial}{\partial G}B_{n}^{i}(G)$ exceeds $\frac{\partial}{\partial G}m_{n}^{L}(G)+\frac{\partial}{\partial G}b_{n}^{L}(G)$. If $V(<\cdot,n+1,G>)$ has exactly J linear segments, the induction hypothesis implies that

$$\frac{\partial}{\partial G}m_{n+1}^{i}(G)+\frac{\partial}{\partial G}b_{n+1}^{i}(G)\geqslant\frac{\partial}{\partial G}m_{n+1}^{J}(G)+\frac{\partial}{\partial G}b_{n+1}^{J}(G)\qquad\text{(B-2)}$$

for all $i=1,2,\cdots,J-1$. As shown before, the sum $\frac{\partial}{\partial G}M_{n}^{i}(G)+\frac{\partial}{\partial G}B_{n}^{i}(G)$ can be expressed by differentiating the sum of equations A–1 and A–2 : equation B–2 then implies that this sum is minimized by using $m_{n+1}^{J}(G)$ and $b_{n+1}^{J}(G)$ as the constituent parameters in lemma A–1:

$$\frac{\partial}{\partial G}M_{n}^{K}(G)+\frac{\partial}{\partial G}b_{n}^{K}(G)\geqslant 1+\frac{\partial}{\partial G}m_{n+1}^{J}(G)+\frac{\partial}{\partial G}b_{n+1}^{J}(G).$$

The values of $m^J_{n+1}(G)$ and $b^J_{n+1}(G)$ depend on the value of π_{n+1}. If $\pi_{n+1} = 1$, then $V(<\cdot,n+1,G>)$ is simply linear with a known closed form expression (lemma A–7). In this case, we have

$$\frac{\partial}{\partial G} m^J_{n+1}(G) + \frac{\partial}{\partial G} b^J_{n+1}(G) =$$

$$\frac{1-(1-\phi)^{N-K_0+2}}{\phi} + N - K_0 + 1 - (1-\phi)\cdot\frac{1-(1-\phi)^{N-K_0+1}}{\phi}$$

$$> 0.$$

If $\pi_{n+1} < 1$, then $m^J_{n+1}(G)$ and $b^J_{n+1}(G)$ are the parameters of $ECT(<\cdot,n+1,G>)$; that is,

$$\frac{\partial}{\partial G} m^J_{n+1}(G) + \frac{\partial}{\partial G} b^J_{n+1}(G) = \frac{\partial}{\partial G} m^J_{n+1}(G)\cdot 1 + \frac{\partial}{\partial G} b^J_{n+1}(G) \qquad \text{(B–3)}$$

$$= \frac{\partial}{\partial G} ECT(<1,n+1,G>)$$

$$= -(N-n).$$

Regardless of the value of π_{n+1}, we have

$$\frac{\partial}{\partial G} M^K_n(G) + \frac{\partial}{\partial G} B^K_n(G) \geqslant -(N-n-1).$$

The same argument that derives equation B–3 shows that

$$\frac{\partial}{\partial G} m^L_n(G) + \frac{\partial}{\partial G} b^L_n(G) = -(N-n+1).$$

We therefore have

$$\frac{\partial}{\partial G} M^K_n(G) + \frac{\partial}{\partial G} B^K_n(G) > \frac{\partial}{\partial G} m^L_n(G) + \frac{\partial}{\partial G} b^L_n(G),$$

which completes the induction argument, and the proof of the lemma.

□

The next lemma gives general conditions under which

$$\frac{\partial}{\partial G} ECC(<p,n,G>) \geqslant \frac{\partial}{\partial G} ECT(<p,n,G>).$$

Lemma B-3: Suppose $\pi_n \geqslant .6$ for all n, and $\phi\cdot N \leqslant 1$. Then for all $p \geqslant .6$

$$\frac{\partial}{\partial G} ECC(<p,n,G>) \geqslant \frac{\partial}{\partial G} ECT(<p,n,G>).$$

Proof: Let $p_1,\cdots,p_{L-1}$ be $V(<\cdot,n,G>)$'s domain transition points (not 0 or 1). We first prove the following proposition:

For every n, and every $i = 1,2,\cdots,L-1$

$$\frac{\partial}{\partial G}m_n^i(G)\cdot p_i + \frac{\partial}{\partial G}b_n^i(G) \geqslant \frac{\partial}{\partial G}m_n^{i+1}(G)\cdot p_i + \frac{\partial}{\partial G}b_n^{i+1}(G).$$

We induct on n, using $n = N$ as the induction base. The same observations that establish the induction base of lemma B–2 show that this induction base is also satisfied. For the induction hypothesis, we suppose that the conclusion is true for some $n+1$. Let $q_1,\cdots,q_K$ denote $ECC(<\cdot,n,G>)$'s domain transition points; we first show that for every q_j,

$$\frac{\partial}{\partial G}M_n^j(G)\cdot q_j + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^{j+1}(G)\cdot q_j + \frac{\partial}{\partial G}B_n^{j+1}(G).$$

Choose arbitrary (non 0 or 1) q_j. For a function $F(p)$, we denote

$$\frac{\partial^-}{\partial G}F(q) = \frac{\partial}{\partial G}\lim_{\substack{p\to q\\ p<q}} F(q)$$

and

$$\frac{\partial^+}{\partial G}F(q) = \frac{\partial}{\partial G}\lim_{\substack{p\to q\\ p>q}} F(q).$$

Using these definitions, we note that

$$\frac{\partial^-}{\partial G}ECC(<q_j,n,G>) = \frac{\partial}{\partial G}M_n^j(G)\cdot q_j + \frac{\partial}{\partial G}B_n^j(G)$$

and that

$$\frac{\partial^+}{\partial G}ECC(<q_j,n,G>) = \frac{\partial}{\partial G}M_n^{j+1}(G)\cdot q_j + \frac{\partial}{\partial G}B_n^{j+1}(G).$$

Again we let a and z be integers such that $m_{n+1}^a(G)$ and $b_{n+1}^a(G)$ define $V(<\cdot,n+1,G>)$'s linear response at p arbitrarily close to $p^c(q_j)$ on the left; similarly, $m_{n+1}^z(G)$ and $b_{n+1}^z(G)$ define $V(<\cdot,n+1,G>)$'s linear response at p arbitrarily close to $p^{\bar{c}}(q_j)$ on the left. If $p^c(q_j)$ is a domain transition point for $V(<\cdot,n+1,G>)$ and $p^{\bar{c}}(q_j)$ is not, then

$$\frac{\partial^-}{\partial G}ECC(<q_j,n,G>) = q_j + q^c(q_j)\cdot\frac{\partial^-}{\partial G}V(<p^c(q_j),n+1,G>)$$

$$+ q^{\bar{c}}(q_j)\cdot\frac{\partial^-}{\partial G}V(<p^{\bar{c}}(q_j),n+1,G>)$$

$$= q_j + q^c(q_j)\cdot\left[\frac{\partial}{\partial G}m_{n+1}^a(G)\cdot p^c(q_j) + \frac{\partial}{\partial G}b_{n+1}^a(G)\right]$$

$$+ q^{\bar{c}}(q_j)\cdot\left[\frac{\partial}{\partial G}m_{n+1}^z(G)\cdot p^{\bar{c}}(q_j) + \frac{\partial}{\partial G}b_{n+1}^z(G)\right].$$

Furthermore,

$$\frac{\partial^+}{\partial G}ECC(<q_j,n>) = q_j + q^c(q_j)\cdot\frac{\partial^+}{\partial G}V(<p^c(q_j),n+1,G>)$$

$$+ q^{\bar{c}}(q_j)\cdot\frac{\partial^+}{\partial G}V(<p^{\bar{c}}(q_j),n+1,G>)$$

$$= q_j + q^c(q_j)\cdot\left[\frac{\partial}{\partial G}m_{n+1}^{a+1}(G)\cdot p^c(q_j) + \frac{\partial}{\partial G}b_{n+1}^{a+1}(G)\right]$$

$$+ q^{\bar{c}}(q_j)\cdot\left[\frac{\partial}{\partial G}m_{n+1}^z(G)\cdot p^{\bar{c}}(q_j) + \frac{\partial}{\partial G}b_{n+1}^z(G)\right].$$

But

$$\frac{\partial}{\partial G}m_{n+1}^a(G)\cdot p^c(q_j) + \frac{\partial}{\partial G}b_{n+1}^a(G) \geqslant \frac{\partial}{\partial G}m_{n+1}^{a+1}(G)\cdot p^c(q_j) + \frac{\partial}{\partial G}b_{n+1}^{a+1}(G)$$

by the induction hypothesis. It follows that

$$\frac{\partial}{\partial G}M_n^j(G)\cdot q_j + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^{j+1}(G)\cdot q_j + \frac{\partial}{\partial G}B_n^{j+1}(G).$$

Entirely similar arguments establish this same inequality in the cases where (i) $p^{\bar{c}}(q_j)$ is a domain transition point and $p^c(q_j)$ is not; (ii) both $p^c(q_j)$ and $p^{\bar{c}}(q_j)$ are domain transition points for $V(<\cdot,n+1>)$.

For an arbitrary domain transition point q_j of $ECC(<\cdot,n,G>)$, we have established that

$$\frac{\partial}{\partial G}M_n^j(G)\cdot q_j + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^{j+1}(G)\cdot q_j + \frac{\partial}{\partial G}B_n^{j+1}(G). \qquad \text{(B-4)}$$

If $V(<\cdot,n,G>)$ is identically $ECC(<\cdot,n,G>)$ on [0,1], the induction argument is completed. We now suppose that $\pi_{n+1} < 1$; note that we are not considering the special case where $\pi_n < 1$ and $\pi_{n+1} = 1$. This case is treated later.

The proof of lemma B-2 showed that

$$\frac{\partial}{\partial G}M_n^j(G) + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^{j+1}(G) + \frac{\partial}{\partial G}B_n^{j+1}(G). \tag{B-5}$$

Consider

$$\left(\frac{\partial}{\partial G}M_n^j(G) - \frac{\partial}{\partial G}M_n^{j+1}(G)\right)\cdot q$$

as a function of q, it is linear. Furthermore, equations B–4 and B–5 show that when $q = q_j$ or when $q = 1$, this function exceeds the value

$$\frac{\partial}{\partial G}B_n^{j+1}(G) - \frac{\partial}{\partial G}B_n^j(G).$$

It follows from linearity that

$$\left(\frac{\partial}{\partial G}M_n^j(G) - \frac{\partial}{\partial G}M_n^{j+1}(G)\right)\cdot q \geqslant \frac{\partial}{\partial G}B_n^{j+1}(G) - \frac{\partial}{\partial G}B_n^j(G)$$

for all $q \in [q_j, 1]$. By simple algebraic rearrangement, it follows directly that for any $q \in [q_j, 1]$,

$$\frac{\partial}{\partial G}M_n^j(G)\cdot q + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^{j+1}(G)\cdot q + \frac{\partial}{\partial G}B_n^{j+1}(G).$$

For every $j = 1,2,\cdots,K-1$ and $q \in [q_j, 1]$, we then have

$$\frac{\partial}{\partial G}M_n^j(G)\cdot q + \frac{\partial}{\partial G}B_n^j(G) \geqslant \frac{\partial}{\partial G}M_n^K(G)\cdot q + \frac{\partial}{\partial G}B_n^K(G). \tag{B-6}$$

To complete the induction argument, we need to show that

$$\frac{\partial}{\partial G}M_n^{L-1}(G)\cdot \pi_n + \frac{\partial}{\partial G}B_n^{L-1}(G) \geqslant \frac{\partial}{\partial G}m_n^L(G)\cdot \pi_n + \frac{\partial}{\partial G}b_n^L(G).$$

Equation B–6 provides us with a means of bounding

$$\frac{\partial}{\partial G}M_n^{L-1}(G)\cdot \pi_n + \frac{\partial}{\partial G}B_n^{L-1}(G)$$

from below. If we can determine the parameters $M_n^K(G)$, $B_n^K(G)$ of $ECC(<\cdot,n,G>)$'s rightmost segment, then equation B–6 will ensure that

$$\frac{\partial}{\partial G}M_n^{L-1}(G)\cdot \pi_n + \frac{\partial}{\partial G}B_n^{L-1}(G) \geqslant \frac{\partial}{\partial G}M_n^K(G)\cdot \pi_n + \frac{\partial}{\partial G}B_n^K(G). \tag{B-7}$$

The parameter derivatives of $ECC(<\cdot,n,G>)$'s rightmost segment are determined by considering $ECC(<\cdot,n,G>)$ for p so large that $p^{\bar{c}}(p) > \pi_{n+1}$: such p exist since $p^{\bar{c}}(\cdot)$ is continuous and $p^{\bar{c}}(1) = 1$. For such p we have

$$ECC(<p,n,G>) = p\cdot G + q^c(p)\cdot ECT(<p^c(p),n+1,G>) + q^{\bar{c}}(p)\cdot ECT(<p^{\bar{c}}(p),n+1,G>)$$

$$= p\cdot G + D_d + p^*(p)\cdot(D_r - G\cdot(N-n))$$

$$+ (1-p^*(p))\cdot T\cdot G\cdot(N-n-1).$$

From this latter equation it can be determined that

$$\frac{\partial}{\partial G} M_n^K(G) = 1 + (1-\phi)\cdot(-(N-n) - T\cdot(N-n-1))$$

and

$$\frac{\partial}{\partial G} B_n^K(G) = T\cdot(N-n-1) - \phi\cdot(N-n + T\cdot(N-n-1)).$$

We may then restate equation B–7 as

$$\frac{\partial}{\partial G} M_n^{L-1}(G)\cdot\pi_n + \frac{\partial}{\partial G} B_n^{L-1}(G) \geqslant \tag{B–8}$$

$$\Big(1 + (1-\phi)\cdot(-(N-n) - T\cdot(N-n-1)\Big)\cdot\pi_n$$

$$+ T\cdot(N-n-1) - \phi\cdot(N-n-1 + T\cdot(N-n-1).$$

Let $p \geqslant .6$, and consider the difference

$$\left(\frac{\partial}{\partial G} M_n^K(G)\cdot p + \frac{\partial}{\partial G} B_n^K(G)\right) - \left(\frac{\partial}{\partial G} m_n^L(G)\cdot p + \frac{\partial}{\partial G} b_n^L(G)\right) =$$

$$\Big(1 + (1-\phi)\cdot(-(N-n) - T\cdot(N-n-1)\Big)\cdot p - \phi\cdot(N-n+T\cdot(N-n-1))$$

$$+ T\cdot(N-n-1) + \Big((N-n+1) + T\cdot(N-n)\Big)\cdot p - T\cdot(N-n)$$

$$= \Big(1 + (1-\phi)\cdot(-(N-n) - T\cdot(N-n-1)) + ((N-n+1) + T\cdot(N-n)\Big)\cdot p$$

$$- \phi\cdot(N-n+T\cdot(N-n-1)) + (T\cdot(N-n-1) - T\cdot(N-n)).$$

The expression above can be directly coerced into the form

$$2\cdot p - (1-p)\cdot\phi\cdot(N-n + T\cdot(N-n-1)) + (1-p)\cdot(T\cdot(N-n-1) - T\cdot(N-n)). \tag{B–9}$$

Since $0 < T \leqslant 1$, we have

$$T\cdot(N-n-1) - T\cdot(N-n) \geqslant -1; \tag{B–10}$$

likewise

$$T \cdot (N-n-1) < N - n. \tag{B-11}$$

Expression B–9 is thus greater than or equal to

$$2 \cdot p - (1-p) \cdot \phi \cdot 2 \cdot (N-n) - (1-p).$$

The assumption that $\phi \cdot N \leqslant 1$ causes this expression to exceed

$$2 \cdot p - 2 \cdot (1-p) - (1-p),$$

which is non–negative when $p \geqslant .6$. For any $p \geqslant .6$ then,

$$\frac{\partial}{\partial G} M_n^K(G) \cdot p + \frac{\partial}{\partial G} B_n^K(G) \geqslant \frac{\partial}{\partial G} m_n^L(G) \cdot p + \frac{\partial}{\partial G} b_n^L(G).$$

If $\pi_n \geqslant .6$, equation B–7 implies that

$$\frac{\partial}{\partial G} M_n^{L-1}(G) \cdot \pi_n + \frac{\partial}{\partial G} B_n^{L-1}(G) \geqslant \frac{\partial}{\partial G} m_n^L(G) \cdot \pi_n + \frac{\partial}{\partial G} b_n^L(G).$$

Under the assumption that $\pi_{n+1} < 1$, this last observation completes the inductive proof that

$$\frac{\partial}{\partial G} m_n^j(G) \cdot p_j + \frac{\partial}{\partial G} b_n^j(G) \geqslant \frac{\partial}{\partial G} m_n^{j+1}(G) \cdot p_j + \frac{\partial}{\partial G} b_n^{j+1}(G)$$

for each of $V(<\cdot, n, G>)$'s (non 0 or 1) domain transition points p_j.

The final step in this induction proof showed that if $\pi_{n+1} < 1$, then for $p \geqslant .6$

$$\frac{\partial}{\partial G} ECC(<p, n, G>) \geqslant \frac{\partial}{\partial G} ECT(<p, n, G>),$$

the statement of this lemma. We have two loose ends yet to consider. We need still to establish this inequality for n such that $\pi_n = 1$ or $\pi_{n+1} = 1$. The latter situation completes the induction argument that

$$\frac{\partial}{\partial G} m_n^j(G) \cdot p_j + \frac{\partial}{\partial G} b_n^j(G) \geqslant \frac{\partial}{\partial G} m_n^{j+1}(G) \cdot p_j + \frac{\partial}{\partial G} b_n^{j+1}(G)$$

for all domain transition points p_j. If $\pi_n = 1$, this last inequality has already been established, but we have not yet established that

$$\frac{\partial}{\partial G} ECC(<p, n, G>) \geqslant \frac{\partial}{\partial G} ECT(<p, n, G>)$$

under the conditions stated by the lemma.

Suppose then that $\pi_{n+1} = 1$ (note also that if $\pi_n = 1$ we must have $\pi_{n+1} = 1$). Let j be that integer such that $n = N - j$; the proof of lemma A–7 expresses a closed form for $ECC(<\cdot, n, G>)$: We have

$$\frac{\partial}{\partial G} ECC(<p, N-j, G>) = p \cdot \frac{1-(1-\phi)^{N-j+1}}{\phi} + N - j - (1-\phi) \cdot \frac{1-(1-\phi)^{N-j}}{\phi}.$$

Now $(1-\phi) < 1$, and

$$\frac{1-(1-\phi)^{N-j}}{\phi} = \sum_{i=0}^{N-j-1}(1-\phi)^i$$

so that

$$\frac{1-(1-\phi)^{N-j}}{\phi} < N-j-1.$$

and

$$\frac{\partial}{\partial G}ECC(<p,n,G>) > p\cdot\frac{1-(1-\phi)^{N-j+1}}{\phi} + N-j-(1-\phi)\cdot(N-j-1)$$

$$> 0.$$

Now,

$$\frac{\partial}{\partial G}ECT(<p,N-j,G>) = -p\cdot(j+1) + (1-p)\cdot T\cdot j.$$

$$< 0$$

when $p \geqslant .6$ and $0 < T \leqslant 1$. Thus, when $p \geqslant .6$,

$$\frac{\partial}{\partial G}ECC(<p,n,G>) \geqslant ECT(<p,n,G>),$$

as required.

□

$\pi_n(G)$ Is Decreasing

The point at which functions $ECC(<\cdot,n,G>)$ and $ECT(<\cdot,n,G>)$ intersect depends on the value of G. We illustrate this dependence on G by denoting the point of intersection by $\pi_n(G)$. Let $D(<p,n,G>)$ be the function defined by

$$D(<p,n,G>) = ECT(<p,n,G>) - ECC(<p,n,G>).$$

If $\phi\cdot N \leqslant 1$ and $p \geqslant .6$ then lemma B-3 implies that $\frac{\partial}{\partial G}D(<p,n,G>) < 0$. Now consider the graph of the function $y = D(<p,n,G>)$. When $0 < \pi_n(G) < 1$, lemma 9-7 implies that $D(<0,n,G>) > 0$ and $D(<1,n,G>) < 0$. Furthermore, $D(<p,n,G>)$ is continuous and differentiable (almost everywhere) in G, and intersects the line $y = 0$ exactly once, at $p = \pi_n(G)$. $\pi_n(G)$ is itself a continuous and differentiable function of G, a consequence of $D(<p,n,G>)$ continuity and differentiability. We have just seen that for large enough (fixed) p, the value of $D(<p,n,G>)$ at p decreases as we increase G. Consequently, if $D(<\cdot,n,G>)$'s point of intersection with $y = 0$ is greater than .6, the point of intersection decreases as we increase G. If the difference $D(<1,n,G>)$ is positive, then $ECT(<\cdot,n,G>)$ and $ECC(<\cdot,n,G>)$ do not intersect, and $\pi_n = 1$. π_n will not vary as a function of G until $ECT(<1,n,G>) \leqslant ECC(<1,n,G>)$. Thus if $D(<1,n,G>) > 0$, we have $\frac{\partial}{\partial G}\pi_n(G) = 0$. These observations establish

Lemma B-4: If $\phi \cdot N \leqslant 1$, and $\pi_n(G) \geqslant .6$ for all n, then

$$\frac{\partial}{\partial G}\pi_n(G) \leqslant 0.$$

□

We can usefully apply this last observation if a system's true gain G_t is bounded from above and below.

Theorem B-1: Let G_B be the largest G such that for every n and $G \in [0,G_B]$ we have $\pi_n(G) \geqslant .6$. If $0 \leqslant G_l \leqslant G_t \leqslant G_u < G_B$, then

$$\pi_n(G_u) \leqslant \pi_n(G_t) \leqslant \pi_n(G_l).$$

Proof: Lemma B–4 shows that $\pi_n(G)$ is a strictly decreasing function of G on the interval $[G_l, G_u]$. The result follows immediately.

□

The Original Model

The derivation of theorem B–1 rests on a slightly different decision model than was initially proposed. We identify here two inequalities whose establishment will verify theorem B–1's application to the original decision model.

The original decision model identifies the cost of a premature decision to calculate a new partition as $E_v[<0,p,G>]$; the second model uses $T \cdot (N - n)$. Theorem B–1 will hold for the original model if certain inequalities known to be satisfied by the latter cost are also satisfied for the former cost. Equation B–10 remarks that

$$T \cdot (N-n-1) - T \cdot (N-n) \geqslant -1;$$

the corresponding inequality required is

$$\frac{\partial}{\partial G}E_v[<0,n+1,G>] - \frac{\partial}{\partial G}E_v[<0,n,G>] \geqslant -1.$$

Likewise, equation B–11 notes that

$$T \cdot (N-n-1) < N - n;$$

this inequality's counterpart is

$$\frac{\partial}{\partial G}E_v[<0,n+1,G>] < N - n.$$

Both of these inequalities can be established if

$$1 \geqslant \frac{\partial}{\partial G}V(<p,n+1,G>) - \frac{\partial}{\partial G}V(<p,n,G>) \geqslant -1. \qquad \text{(B–12)}$$

This latter inequality seems quite reasonable, especially in light of the fact that we *can* prove that

$$G \geqslant V(<p,n+1,G>) - V(<p,n,G>) \geqslant -G$$

for all p, n, and G. However, so long as inequality B–12 remains unproven, we can only

conjecture that theorem B–1 is applicable to our original decision model.

Significance of Result

Having labored through many detailed arguments leading to theorem B–1, we do well to consider the significance of this result. In the real world, the value of G can only be estimated We want to estimate G at the time that a change is detected: this means that we want to estimate G as quickly as possible. Theorem B–1 shows us that under certain conditions, the optimal cost equations are well–behaved with respect to G. Given rapidly calculated bounds on the true G, we are assured that if we use a point estimate of G between these bounds, the resulting estimation of π_n (for every n) must lie in the smallest interval *known* to contain the true π_n. Theorem B–1's chief significance then is that it tells us that the heuristics we have to use to estimate the π_n are well–founded.

The hypothesis of theorem B–1 assumes that $\phi \cdot N \leqslant 1$ and $\pi_n \geqslant .6$ for all n. We note briefly that we expect these assumptions to be satisfied by most operational scenarios. In general, we anticipate that the costs associated with repartitioning to be significant relative to the gain. If a change has truly occurred, the likelihood of next observing a change is high, and the resultant positive indication of change will substantially increase the probability of change. If $p_n = .6$, and a positive indication of change at time $n+1$ causes $p_{n+1} >> .6$, it seems likely that $\pi_n >> .6$; the additional certainity of change achieved by waiting is relatively cheap. We also assume that $\phi \cdot N \leqslant 1$. In the absence of any detailed knowledge about the possibility of change, it would be reasonable to assume that having a change occur in $[0,N]$ is as likely as not. The failure rate ϕ associated with such an assumption is less than $\frac{1}{N}$, suggesting that $\phi \cdot N \leqslant 1$ is also reasonable.

Bibliography

References

[Bae80] J. Baer, *Computer Architecture*, Computer Science Press, Rockville, MD, 1980.

[Bar76] R. G. Bartle, *The Elements of Real Analysis*, Wiley and Sons, second edition 1976.

[Bok81] S. H. Bokari, A Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System, *IEEE Trans. on Software Eng. 7*, 6 (November 1981), 583–589.

[BoS83] H. Bozedogan and S. Sclove, Multi–Sample Cluster Analysis Using Akaike's Information Criterion, *Annals of the Institute of Statistical Mathematics 36*, 1 (1983), .

[BrF76] M. A. Breuer and A. D. Friedman, in *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press Inc., 1976.

[ChM79] K. M. Chandy and J. Misra, Distributed Simulation: A Case Study in Design and Verification of Distributed Programs , *IEEE Trans. on Software Eng. 5*, 5 (September 1979), 440–452.

[ChA82] T. C. Chou and J. A. Abraham, Load Balancing in Distributed Systems, *IEEE Trans. on Software Eng. 8*, 4 (July 1982), 401–412.

[ChK79] Y. Chow and W. Koehler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, *IEEE Trans. on Computers C-28*, 5 (May 1979), 354–361.

[Cof76] E. G. Coffman, *Computer and Job/Shop Scheduling Theory*, Wiley & Sons, New York, 1976.

[Com82] J. C. Comfort, The Design of a Multi–Microprocessor Based Simulation Computer, *Proceedings of the 15th Annual Simulation Symposium*, , 1982, 45–53 .

[Con85] A. I. Concepcion, *Distributed Simulation on Multi-Processors: Specification, Design, and Architecture*, Ph.D. Dissertation, Wayne State University, January 1985.

[Dav84] D. Davidson, *A Distributed Simulation Implementation*, M.S. Thesis, University of Virginia, January 1984.

[DaR85] M. Davio and C. Ronse, Insertion Networks, *IEEE Trans. on Computers c-34*, 6 (June 1985), 565–569.

[DuB82] M. Dubois and F. A. Briggs, Performance of Synchronized Iterative Processes in Multiprocessor Systems , *IEEE Trans. on Software Eng. 8*, 4 (July 1982), 419–431.

[DKW82] A. Dutta, G. Koehler and A. Whinston, On Optimal Allocation in a Distributed Processing Environment, *Management Science 28*, 8 (August 1982), 839–853.

[ELZ] D. Eager, E. Lazowska and J. Zahorjan, Dynamic Load Sharing in Homogeneous Distributed Systems, 84–10–01, University of Washington.

[El–78] O. I. El–Dessouki, *Program Partitioning and Load Balancing in Network Computers* , Ph.D. Dissertation, Illinois Inst. of Technology, December 1978.

[Elm67] S. E. Elmaghraby, On the Expected Duration of PERT Type Networks, *Management Science 13* , (1967), 299–306.

[FiM82] C. M. Fiduccia and R. M. Mattheyses, A Linear–Time Heuristic for Improving Network Partitions, *Proceedings of the 19th Design Automation Conference*, , 1982, 175–181.

[Fis78] G. S. Fishman, Grouping Observations in Digital Simulation, *Management Science 24*, (1978), 510–521.

[GaT84] A. Gabrielian and D. B. Tyler, Optimal Object Allocation in Distributed Computer Systems, *Proceedings of the Fourth International Conference on Distributed Computing Systems*, , 1984, 88–95.

[Gov75] Z. Govindarajulu, *Sequential Statistical Procedures*, Academic Press, 1975.

[HaW66] H. O. Hartley and A. W. Wortham, A Statistical Theory for PERT Critical Path Analysis, *Management Science 12* , (1966), B469–B481.

[HeI79] P. Heidelberger and D. L. Iglehart, Comparing Stochastic Systems using Regenerative Simulations with Common Random Numbers, *Advanced Applied Probability 11*, (1979), 804–819.

[HoT83] R. Hogg and E. Tanis, *Probability and Statistical Inference*, Macmillan , New York, 1983.

[Hol82] L. J. Holloway, *Task Assignment in a Resource Limited Distributed Processing Environment*, Ph.D. Dissertation, U.C.L.A., 1982.

[HoP84] S. V. Hoover and R. F. Perry, Validation of Simulation Models:The Weak/Missing Link, *Proceedings of the Winter Simulation Conference, Dallas*, , November 1984, 293–295.

[IgS83] D. Iglehart and G. Snedler, Simulation of Non–Markovian Systems, *IBM Journal of Research & Development 27*, 5 (September, 1983), 472–80.

[JeS82] D. R. Jefferson and H. Sowizral, Fast Concurrent Simulation Using the Time Warp Mechanism, *Rand Report to the Air Force FN-1906-AF*, , December 1982.

[Kal79] J. G. Kalbfleish, *Probability and Statistical Inference II*, Springer–Verlag, 1979.

[KeL70] B. W. Kernighan and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs, *Bell System Technical Journal*, , February, 1970, 291–307.

[Kle76] L. Kleinrock, *Queueing Systems, Volume 2:Computer Applications*, Wiley and Sons, New York, 1976.

[Knu73] D. Knuth, *The Art of Computer Programming, vol. 1* , Addison–Wesley, second edition 1973.

[Kri84] B. Krishnamurthy, An Improved Min–Cut Algorithm For Partitioning VLSI Networks, *IEEE Trans. on Computers C-33*, 5 (May 1984), 438–446.

[Law84] A. Law, Steady–State Confidence Interval Methodology, *Proceedings of the Winter Simulation Conference, Dallas*, , November 1984, 243–250 .

[Lo81] V. Lo and J. W. S. Liu, Task assignment in Distributed Multiprocessor Systems, *Proceedings of the 1981 Int'l Conf on Parallel Processing*, , 1981, 358–360.

[Lo84] V. Lo, Heuristic Algorithms for Task Assignment in Distributed Systems, *Proceedings of the Fourth International Conference on Distributed Computing Systems*, , 1984, 30–39.

[Los85] S. Losen, *A Global Algorithm for the Multi-Partitioning of Graphs*, M.S. Thesis, University of Virginia, August, 1985.

[MaO79] A. Marshall and I. Oklin, *Inequalities: Theory of Majorization and Its Applications*, Academic Press, New York, 1979.

[Mar65] J. J. Martin, Distribution of Time Through a Directed, Acyclic Network, *Operations Research 13* , (1965), 46–66.

[MeH82] M. S. Meketon and P. Heidelberger, A Renewal Theoretic Approach to Bias Reduction in Regenerative Simulations, *Management Science 26*, (1982), 173–181.

[Ni81] L. M. Ni and K. Hwang, Optimal Load Balancing Strategies for A Multiple Processor System, *Proceedings of the 1981 Int'l Conf on Parallel Processing*, , 1981, 352–357.

[NiR84] D. M. Nicol and P. F. Reynolds, Problem Oriented Protocol Design, *Proceedings of the Winter Simulation Conference*, Dallas, Texas, November 1984, 471–474.

[Nic84] D. M. Nicol, *Synchronizing Network Performance*, M.S. Thesis, University of Virginia, January 1984.

[NoJ74] M. R. Novick and P. H. Jackson, *Statistical Methods for Educational and Psychological Research*, McGraw–Hill, New York , 1974.

[PMW80] J. K. Peacock, E. Manning and J. W. Wong, Synchronization of Distributed Simulation Using Broadcast Algorithms, in *Computer Networks*, vol. 4, North Holland Publishing Co., 1980, 3–10.

[PhT73] G. M. Phillips and P. J. Taylor, *Theory and Applications of Numerical Analysis*, Academic Press, 1973.

[Pri79] C. C. Price, A Nonlinear Multiprocessor Scheduling Problem, *Ph.D. Dissertation, Texas A&M University*, , May 1979.

[PrK84] C. C. Price and S. Krishnaprasad, Software Allocation Models for Distributed Computing Systems, *Proceedings of the Fourth International Conference on Distributed Computing Systems*, , 1984, 40–48.

[RaH80] C. V. Ramamoorthy and G. S. Ho, Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets, *IEEE Trans. on Software Eng. 6*, 5 (September 1980), 440–449.

[RaS84] K. Ramamritham and J. A. Stankovic, Dynamic Task Scheduling in Distributed Hard Real–Time Systems, *Proceedings of the Fourth International Conference on Distributed Computing Systems*, , 1984, 96–107.

[RSB79] A. Rapoport, W. E. Stein and G. J. Burkheimer, *Response Models for Detection of Change*, D. Reidel Publishing Company, Boston, 1979.

[Rey82] P. F. Reynolds, A Shared Resource Algorithm for Distributed Simulation, *Proceedings of the Ninth Annual International Computer Architecture Conference*, Austin, Texas, April 1982, 259–266.

[Rey83] P. F. Reynolds, Private Communication, , August 1983.

[RoT77] P. Robillard and M. Trahan, The Completion Time of PERT Networks, *Operations Research 25* , 1 (January 1977), 15–29.

[Ros70] S. Ross, *Applied Probability Models with Optimization Applications* , Holden–Day , San Fransico, 1970.

[Ros83] S. Ross, *Stochastic Processes*, Wiley and Sons, New York, 1983.

[SaH76] S. Sahni and E. Horowitz, *Fundamentals of Data Structures* , Computer Science Press, Rockville, MD, 1976.

[Sal83] M. A. Salichs, Task Assignment Across Space and Time in a Distributed System, *Proceedings of the Fifth IFAC Workshop*, , May 1983, 131–141.

[Sch69] S. A. Schmitt, *An Elementary Introduction to Bayesian Statistics*, Addison–Wesley, 1969.

[Sch83] L. Schruben, Simulation Modeling With Event Graphs, *Comm. ACM 25*, 11 (November 1983), 957–963.

[Sip84] H. J. Sips, Task Distribution on Clustered Parallel or Multiprocessor Systems, *Proceedings of the Fourth International Conference on Distributed Computing Systems*, , 1984, 126–130.

[SmL82] C. U. Smith and D. D. Loendorf, Performance Analysis of Software for an MIMD Computer, *1982 ACM SIGMETRICS Conference Proceedings*, , 1982, 151–162.

[Sta84a] J. Stamos, Static Grouping of Small Objects to Enhance Performance of a Paged Virtual Memory, *ACM Transactions on Computer Systems 2*, 2 (May 1984), 155–180.

[Sta84b] J. A. Stankovic, An Adaptive Bidding Algorithm for Processes, Clusters, and Distributed Groups, *Proceedings of the Fourth International Conference on Distributed Computing Systems* , , 1984, 126–130.

[Sta85] J. A. Stankovic, An Application of Bayesian Decision Theory To Decentralized Control of Job Scheuling, *IEEE Trans. on Computers C-34*, 2 (February 1985), 117–130 .

[Sto77] H. S. Stone, Multi–Processor Scheduling With the Aid of Network Flow Algorithms, *IEEE Trans. on Software Eng. 3*, 1 (January 1977), 85–93.

[TaT85] A. N. Tantawi and D. Towsley, Optimal Static Load Balancing, *Journal of the ACM 32*, 2 (April 1985), 445–465.

[TCB78] D. Towsley, K. M. Chandy and J. C. Browne, Models for Parallel Processing Within Programs: Application to CPU:I/O and I/O:I/O Overlap, *Communications of the ACM 21*, 10 (October 1978), 821–830.

[WyS84] D. Wyatt and S. Sheppard, A Language Directed Distributed Discrete Simulation System, *Proceedings of the Winter Simulation Conference*, , November 1984, 463–464.

[YYS81] S. S. Yau, C. C. Yang and S. M. Shatz, An Approach to Distributed Computing System Software Design, *IEEE Trans. on Software Eng. 7*, 4 (July 1981), 427–435.

[Zei84] B. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, London, 1984.