Winter 1982

DECENTRALIZED CONTROL OF DISTRIBUTED PROCESSING SYSTEMS

AHMED KAMAL EZZAT

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again–beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University
Microfilms
International

300 N. Zeeb Road
Ann Arbor, MI 48106

Ezzat, Ahmed Kamal

DECENTRALIZED CONTROL OF DISTRIBUTED PROCESSING SYSTEMS

University of New Hampshire PH.D. 1982

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark __√__.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages __✓__
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Other _____

DECENTRALIZED CONTROL
OF DISTRIBUTED PROCESSING SYSTEMS

BY

AHMED K. EZZAT

B.S. (Electrical Engineering), Cairo University, 1971

M.S. (Computer Engineering), Cairo University, 1976

A DISSERTATION

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

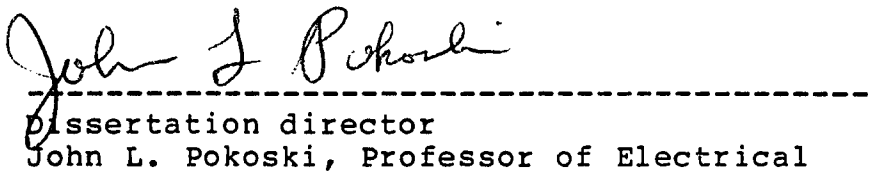
Doctor of Philosophy

In

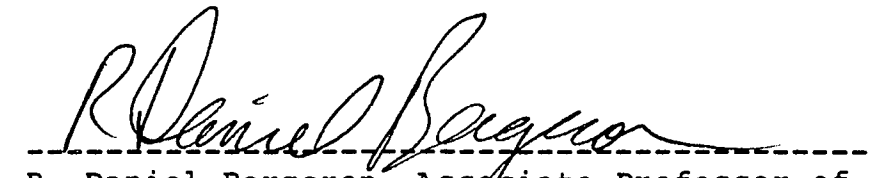
Engineering

December 1982

This dissertation has been examined and approved.

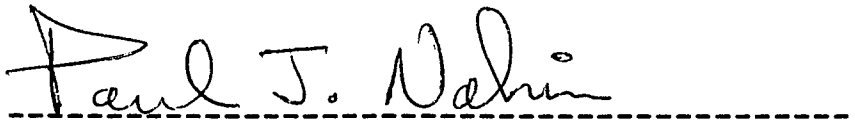


Dissertation director
John L. Pokoski, Professor of Electrical and Computer Engineering

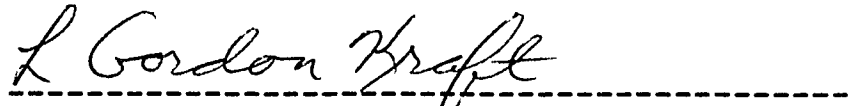


R. Daniel Bergeron, Associate Professor of Computer Science

Loren D. Meeker, Professor of Mathematics



Paul J. Nahin, Associate Professor of Electrical and Computer Engineering



L. Gordon Kraft, Assistant Professor of Electrical and Computer Engineering

October 28, 1982

Date

ACKNOWLEDGEMENTS

The author wishes to thank Dr. J. L. Pokoski, Professor, Electrical and Computer Engineering Department, University of New Hampshire for his effort and contributions to the progress of this work.

The author is deeply indebted to Dr. R. D. Bergeron, Associate Professor and Chairman, Computer Science Department, University of New Hampshire, for his contributions and guidance during this work.

The author would like to express his appreciation to Professor L. D. Meeker, Associate Profeesor P. J. Nahin, and Assistant Professor L. G. Kraft, for their time and effort in behalf of the author.

The author would like to express his gratitude to Dr. R. R. Clark, Professor and Chairman, Electrical and Computer Engineering Department, University of New Hampshire, for his support and advice during the author's assistantship in the Electrical and Computer Engineering Department. Also, the author would like to thank the rest of the staff of the Department of Electrical and Computer

Engineering for their help during his study.

The author would like to express his gratitude to the Computer Science Department, University of New Hampshire, for their help, and the extensive use of their facilities.

Finally, the author would like to thank his parents for their support and encouragement through his career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

ABSTRACT

DECENTRALIZED CONTROL

OF DISTRIBUTED PROCESSING SYSTEMS

by

AHMED KAMAL EZZAT

University of New Hampshire, December, 1982

This thesis presents a methodology for implementing decentralized scheduling for distributed systems. The environment in which the controlling entities make decisions is stochastic and can be described as uncertain since each entity may have a different view of the system state. As a consequence, these entities may make inconsistent decisions.

The methodology is based on defining the system state as a set of distributions and using a queueing model to predict the future behaviour of the system. The predicted state is used to schedule the individual job tasks based on minimum predicted job response time.

A hypothetical real system is simulated. The methodology was tested using different queueing models and under different environments. An evaluation of the proposed

technique using the simulation results indicates a consistent performance improvement over the no network case. Suggestions for extending this research are also presented.

CHAPTER I

INTRODUCTION

Distributed processing systems have now been made possible by the advancement in microelectronics technology and the development of efficient cost-effective interconnection structures. In this research we define distributed processing system characteristics, survey problems unique to distributed processing systems, and propose a control scheme suitable for a distributed processing systems environment. The term "distributed systems" will be used as a synonym for distributed processing systems in the rest of this thesis.

1.1 Characterization of Distributed Systems

The term "distributed system" is frequently used in the literature to imply a collection of processing elements (hosts) that are physically and logically interconnected and share one or more resource(s). This definition includes an extremely wide range of systems including both loosely coupled and tightly coupled architectures. Jensen [JEN78]

has proposed a more precise definition intended to exclude tightly coupled systems on one hand and pure data communication networks on the other hand.

Following Jensen's definition, the term "distributed system" refers to a computing system which has the following physical and logical characteristics which may be interpreted as general rules to be observed in a distributed system.

1. The system includes an arbitrary number of system and user processes.

2. The architecture is modular, consisting of a possibly varying number of processing elements.

3. Communication is achieved via interprocess communication architecture. This excludes shared memory systems.

4. Some system-wide control for all resources is needed to provide dynamic interprocess cooperation and runtime management.

5. Interprocess message transit delays are variable and some non-zero time interval always exists between the production of an event by a process and the materialization of this production at the destination process.

An important consequence of these characteristics is that physical characteristics, such as the distance between components, cannot be used to identify a distributed system.

1.2 Objectives of Distributed Systems

The objectives listed below constitute what is usually expected from distributed systems in general [LAN81]. These objectives may not all be meaningful to a particular system and are probably not equally important. Obtaining these objectives is complicated by the new problems that should be taken into consideration, such as the need for system-wide control to detect and resolve possible conflict without excessively impairing parallelism, and asynchrony in processing.

1.2.1 Increased performance - The processing power of a multiple processor system should certainly be more than the processing power of the system if treated as independent single processors.

1.2.2 Extensibility - This means the ability to change the system performance or the system function without the need to change the system design. This implies that the system should be designed on a modular basis.

1.2.3 Increased availability - Availability is defined as the extent to which a system is able to survive failures. Most reliability mechanisms existing today are based on using redundant hardware, software, and data. The

existence of several processing elements in a system raises the opportunity for utilizing mutual inspection techniques which allow for automatic detection, diagnosis, and recovery. In another way, if processing elements cooperate in a decentralized manner, it becomes possible to take full advantage of redundancy so as to obtain fail-soft computing, i.e., systems which keep on running in spite of faults, errors, or failures.

1.2.4 Resource sharing - the term "resource" should be taken in its widest sense. For example, a resource may be a physical device, a database maintained at one or more processing element(s), or system software such as a specific compiler. Resource sharing involves load sharing and transparency to implemented architectures. Resource sharing should not be restricted to remote access to a variety of resources. The goal is to provide a single computing system with some system-wide control of all activities; in other words, the system would appear to the user as a single virtual machine with optimal and dynamic resource allocation.

1.3 The Problem and Its Importance

In this research we propose and evaluate a control technique which would meet the objectives (1.2.1 - 1.2.4) of distributed systems defined by characteristics (1 - 5) above.

The relationship between objectives (1.2.1 - 1.2.4) and characteristics (1), (2), and (3) are obvious. Characteristic (4) is a vital one because it enables achieving our ultimate goal, the single virtual machine. Characteristic (5) expresses a very important physical constraint which must be taken into account in the design of any control technique for distributed systems. This is because it excludes any system which is based on the premise that all processes in the system share a complete and consistent view of the entire system state at every instant in time. Such systems are referred to as centralized ones. In distributed systems, the existence of such a unique entity is ruled out. Therefore, centralized control techniques are not suitable for distributed systems.

We conclude that such system-wide control for distributed systems should be built in a decentralized manner; consisting of "N" identical physically distributed entities, where "N" is the number of nodes in the system. Each entity makes decisions locally on an equal basis with the other entities based on system-wide objectives.

The environment in which these entities make decisions is stochastic and can be described by the following constraints:

- entities may each have a different view of the system state; consequently, these entities may make inconsistent decisions;

- the difference between what the instantaneous system state really is and its representation communicated to these entities may not be negligible.

1.4 The Approach to the Problem

In designing a decentralized system-wide control algorithm for distributed systems, one should address the following points. First, the objective functions to be achieved by the control algorithm must be defined. Second, best decisions for achieving these objective functions under high uncertainty about the system state must be made. (This is a unique problem in distributed systems). Third, any algorithm implementing decentralized control must run quickly. This is an extremely important aspect of the algorithm for two reasons. One is to minimize the effect of uncertainty in the system state which makes any excessively time-consuming solutions undesirable. The other reason is to minimize the overhead imposed by the control algorithm on the system.

The key idea in our approach is to define the system states by a corresponding number of probability distribution functions, where each distribution at any point in time is completely described by an estimate of its first two moments as functions of time, i.e., the mean(t) and the variance(t). Using queueing theory models with the new defined states, we can predict the future system behaviour

on which basis we can make decisions. A special technique called a moving window of observations is developed to update these estimates for the different entities in the system. This technique is discussed in detail in a later chapter.

Each host sends the parameters corresponding to its own contribution to the system state to every other host in the system. These parameters can be accessed by the corresponding entity at each node. Using a simple queueing model each entity is able to make the decisions needed to achieve the objective functions of the control algorithm.

We show that decisions taken by the different entities using this technique should improve the system performance.

1.5 Related Work

This research is related to work in the area of control and scheduling in distributed processing systems. The Distributed Computer Network (DCN) at the University of Maryland [LAY74,MIL76] is intended to be a research tool for the development and evaluation of resource allocation and management techniques suited for distributed environments. The Distributed Loop Computer Network (DLCN) at Ohio state [REA76] uses the idea of connecting all nodes of the network in a virtual ring and having a resource allocating task floating through the ring to serve resource allocation requests. The ARPANET [KAH72,ROB70,SCH75] high level protocol RSEXEC operates as a subnetwork of the ARPA

network and provides capabilities for remote file and device access. RSEXEC is based on a virtual-storage and virtual processor organization. Processes, devices and files can be accessed from anywhere by a standard communication protocol, but the user must still be aware of which host he is utilizing. Stone [STO77] and Bokhari [BOK77], were concerned with assigning program modules to a processor in a two processor environment. The assignments were handled as a commodity flow problem. An attempt was made to generalize the results to three or more processors with partial success, but an efficient solution has not yet been obtained. They also assume "complete information" is available at a central decision-making point, an approach that we feel is not realistic in any large distributed system. Russell and Bergeron [RUS81] at the University of New Hampshire have implemented a decentralized scheduling mechanism for a heterogeous distributed system based on task migration during execution. The scheduling entities exchange task information and cooperate on the migration decision. A Distributed Real time Operating System (d-RTOS) is a distributed computer simulation developed at the Advanced Research Center in Huntsville, Alabama [MIC80a,MIC80b,MIC80c]. This work is tailored to a ballistic missile defense application to achieve maximum performance at minimum overhead. The resource manager is fully distributed and consists of five tasks which reside identically on each computer in the network. Its main goal

is to dynamically balance the task load on the different computers in the network. The work described above, as most scheduling work in the distributed systems area, is based on using message exchange and cooperation between nodes in the network. Instead of using message exchange, our model uses state prediction to determine task migration. The only other research which handles scheduling under imperfect knowledge of the system state was done by Stankovic [STA81a,STA81b,STA82]. His research is part of the design of an experimental operating system, called Adaptive Decentralized Controlled Operating System (ADCOS). ADCOS and its associated hardware form one computer system that happens to be physically distributed. The decentralized control algorithms are generated by a methodology based on Bayesian Decision Theory and a modified McCulloch-Pitts Neuron. This research is similar to ours in that we both make decisions in a decentralized manner based on local information under conditions of imperfect knowledge of the system state. Stankovic applied his model to a simpler environment than ours. Each job is equivalent to a cpu task whereas our jobs consist of a cpu task, terminal task, and zero or more I/O tasks which can be deterministic or nondeterministic. Also, his simulation utilizes FCFS service discipline whereas ours uses a service discipline which is close to a round-robin.

1.6 Outline of the Thesis

Chapter II discusses different issues in distributed systems, some of which are common to those in nondistributed systems, and others are unique to distributed systems. This chapter also gives a brief overview of an informal model for a distributed system architecture.

Chapter III presents the functional model. Chapter IV discusses how to apply this model to a real system environment. This includes developing some techniques needed by the model such as updating the system state and system state initialization. Chapter V discusses in detail the simulation implemented to test our model. Different test results are shown under different workload environments. Analysis and conclusions for our model from the simulation results are also presented.

Chapter VI summarizes the contributions of this research and presents possible extensions for future research.

Appendix A gives a brief review of stochastic processes as they relate to our research. Appendix B gives a summary of the different resource queueing models used in our decentralized control model. Appendix C gives a brief discussion of the distribution identification problem. One of these techniques is implemented in our simulation.

This thesis assumes that the reader is familiar with queueing theory at the level presented in Appendix B.

CHAPTER II

DISTRIBUTED SYSTEM ARCHITECTURE

The area of distributed systems is new and not well defined. The purpose of this chapter is to survey problems unique to distributed systems and provide an informal framework for a distributed system architecture model, keeping in mind that there are many alternative approaches to the realization of this and similar models.

2.1 Issues Pertaining to Distributed Systems

Distributed systems have many problems and solutions in common with nondistributed systems. New problems, however, are introduced by the physical separation, the potential heterogeneity of the system components, and the need to have multiple entities controlling the system [WAT81]. The following is a discussion of the important problems introduced by distributed systems:

2.1.1 Identification - We need to be able to distinguish between various kinds of identifiers used to refer to

objects at all levels of architecture. Identifiers at different levels referring to the same object should be bound together either statically or dynamically. One way to implement this, is by mapping identifiers into addresses and routes.

Many problems are added to the design of an identification system due to the heterogeneous nature of the system components, each with its own possible local identification system for accessing objects. Another kind of problem is in maintaining distributed context and mapping information in spite of delays and errors in message transmission, and local or network crashes.

The choice of an identification scheme can affect the ease or even the possibility of achieving goals, such as:

- Efficient support of both transaction and stream oriented services. In a transaction oriented system the application has the identifiers of one or more resources on which it desires to perform a given operation, with no implication that these resources or the operation may ever be used again. In a stream oriented system, on the other hand, it is assumed at the start that a stream of operations may be requested over time against an identified set of resources. Efficient support of transaction oriented applications and services implies minimizing the delay and number of messages (overhead) that must be exchanged before

and after the actual message is sent to perform the desired function. On the other hand, supporting efficient stream oriented applications implies that we want to perform identifier mapping at most once before or during the first access.

- Global space identification. A system should be seen as a global space of identified objects rather than one viewed as a space of identified hosts containing local objects.

- Relocation of objects. A system should allow an object to change location. This implies having at least two levels of identifiers, a name and an address, and that the binding between them is dynamic.

- Use of multiple copies of the same object. Multiple copies are required for performance and reliability goals. This means a single identifier at one level can be dynamically bound to more than one address at a lower level according to some criteria.

- Broadcasting. A broadcast capability requires that many different objects be able to share the same identifier.

2.1.2 Resource management - Allocating and scheduling resources at the various distributed system hosts is usually

based on local decisions, because of the need for local autonomy for the different resources at each host. Resource management and state information management are intimately related. In many distributed systems, one would like to use resource and state management philosophies at all levels that could achieve both low delay and high throughput.

Delay is defined as the time interval from the time a process is ready to send a message until the first useful bit of the message reaches the destination. Delay is affected not only by the transmission and queuing properties of the interprocess communication (IPC) mechanisms, but also by the overhead messages at one or more layers that may have to be exchanged to reserve resources, map identifiers, initialize state variables, etc., before the desired request or data can be sent.

Throughput is defined as the number of useful data bits per second that reach the receiver in some interval. Throughput is affected not only by transmission and queuing characteristics of the IPC mechanisms, but also by the amount of identification, control, protection and other overhead information that must accompany the meaningful user data object bits.

The principal difference between transaction and stream oriented communications is the number of identification mappings typically required. Consequently, minimum delay is the better goal for a transaction oriented environment, whereas maximum throughput is the better goal for a stream

oriented environment.

Developing mechanisms that can achieve both low delay and high throughput in a distributed system is a difficult task. Tradeoffs exist between quantity of state information, amount of overhead information carried with messages, and the number of messages needed to initialize state information. Because parameters are a function of the resource management services being supported, one can say that levels of this service also define trade-offs between delay and throughput.

2.1.3 Synchronization - The term synchronization refers to mechanisms used by cooperating entities to coordinate access to shared resources or to order events with some means to keep the system in a consistent state.

In the absence of any particular assumption, the only way to preserve consistency is to guarantee that operations remain atomic [KOH81], i.e., operations are executed one after the other in a strictly sequential fashion. An operation is atomic iff all intermediate states are not visible to any other operation. To explain the principle of atomic operations by example, let us assume two operations A,B; for each resource R_K shared by A and B, let $\{a_i^K\}$ be the set of actions of A that access R_K and $\{b_j^K\}$ the set of actions of B that access R_K. Then A and B are atomic if for every action a in $\{a_i^K\}$ a occurs before b for every b in $\{b_j^K\}$, or all a occurs after all b. Having specific assumptions, it is possible to relax conditions of atomicity

for operations and still preserve consistency. Clearly, if actions activated by a given set of operations manipulate different objects, parallel execution of actions is certainly recommended.

It should be clear now that atomicity implies that it is possible to express or enforce a particular ordering on any given set of events so as to preserve system state consistency, which is the purpose of any synchronization mechanism. For distributed systems where operation executions involve a number of processing elements, it is easy to see that variability in propagation delay(i.e.,the time delay between the production and materialization of an event) may disturb any particular event ordering which was supposed to occur.

2.1.4 Error control - Error control implies error detection and recovery. Two schemes for error control are discussed using the concept of recovery points [LAN81]. A recovery point is a recording of all needed information whereby a consistent state of the system may be reinstalled.

One scheme is the "backward error recovery", where on error detection all activities of any operation are rolled back to the last recovery point. The other one is the "forward error recovery" scheme, which depends on the possibility of detecting the consequences of a fault; it allows activities not affected by the error to proceed to the next recovery point, and those which are affected or will be affected to roll back to the last recovery point.

These two schemes are based on the utilization of those recovery points. Defining the recovery points in a distributed environment is clearly much harder and perhaps infeasible.

2.1.5 Other problems - These include translation issues resulting from heterogeneous data encoding and representation at the different hosts; the need for a message base IPC model; the need to maintain consistency among multiple copies of information; the need to have a uniform resource structure which can support distributed objects.

2.2 A Model Based on Layers

A model which provides a framework for solving these problems should be based on the techniques of layering, message passing, and creation of abstract objects.

The concept of modular and layered design or levels of abstraction [ISO79] has been widely accepted as good software engineering practice. Such concepts when applied to the design of a distributed systems could result in the organization shown in figure 2-1 [WAT81]. Associated with a layer N are two interfaces. Each layer N provides a well-defined set of services at the interface to layers N+1 and higher. In turn, layer N is implemented using the services provided through interfaces with layer N-1 and lower. The layers may be partially ordered since many abstractions at higher levels may involve only some or none

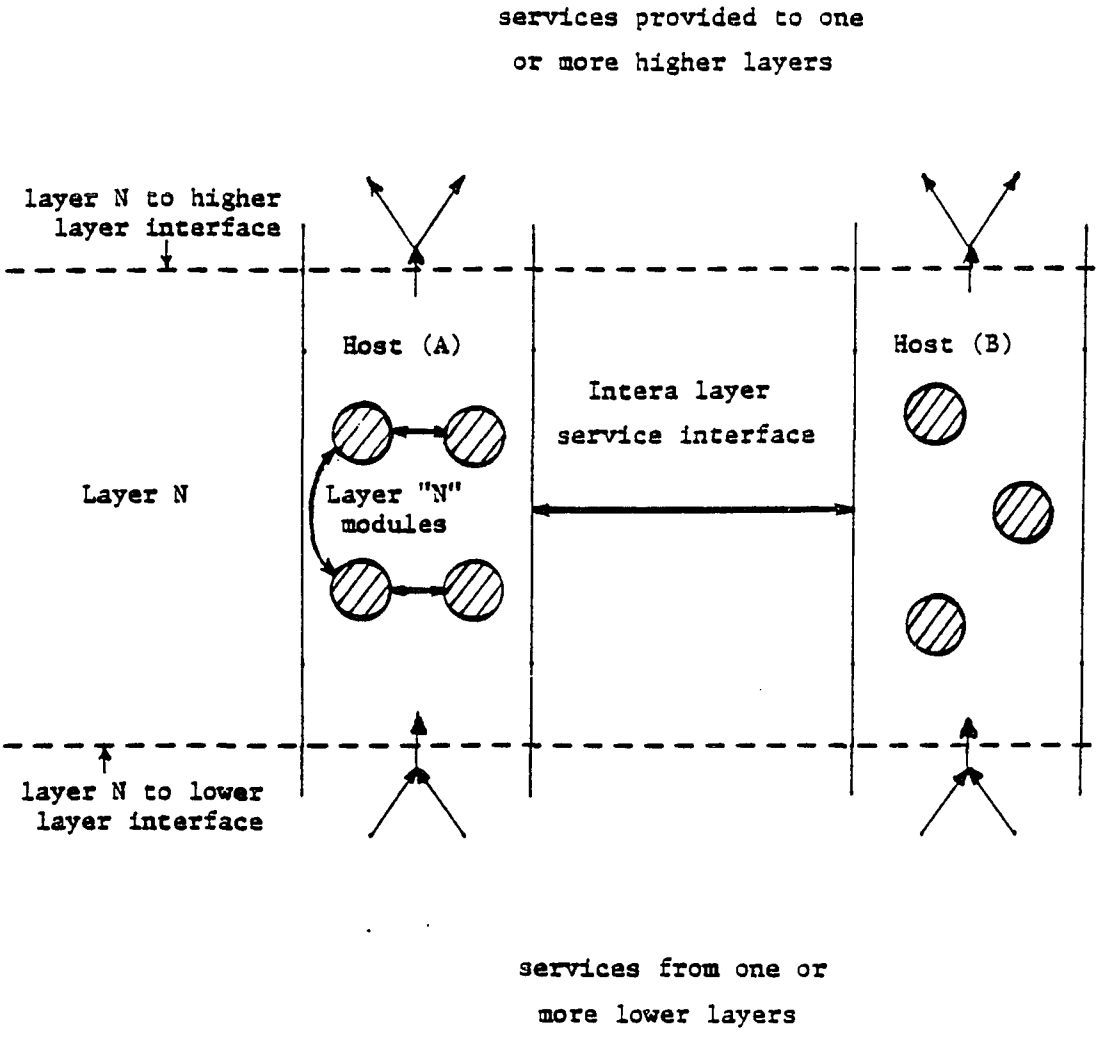


Figure 2-1. Layers and Interfaces Structure

of the abstractions at some particular level below.

Designing a distributed system architecture requires:

- decomposing the system into layers and sublayers according to some set of criteria;
- specifying the services to be offered by layer N to higher layers;
- specifying the services layer N requires of lower layers.

The services of layer N may be further decomposed into modules. The modules implementing a service of a given layer may in turn be distributed. Modules, like layers, provide a well-defined set of services at their interfaces and their internal implementation is not visible on the other side of this interface.

The above description identifies two interfaces, one between adjacent layers and one between cooperating modules within a given layer. Generally, an interface is defined as a set of conventions for the exchange of information between two entities. It consists of a set of abstract objects and for each object a set of allowed operations and associated parameters.

The model is based on the idea of having a distributed operating system (DOS). One of the major design goals of a DOS is to provide users with access to real and abstract objects or resources [JON78] in which the distributed nature

of their implementation is hidden as far as practical. In addition, all objects (system and user defined) are named, communicated with, and shared uniformly. Real objects are entities such as processors, secondary storage, I/O devices. DOS abstract objects or resources are entities such as processes, files, directories, and databases, which are used as a set of basic building blocks for creating higher level objects. Objects at each level interact through, and are created from, lower level objects.

Each type of resource or object is specified by:

- a set of data structures visible at the interface (object representation);
- a set of operations or functions and associated parameters that can be performed on the object representation.

Two resources are said to be of the same type iff they have the same specifications (representation and operation). These specifications are implemented by one or more modules called server(s). Servers can be implemented by hardware/firmware or a set of procedures (processes).

The implementation details of a resource representation are of concern only to a particular server. Two different servers implementing a resource of a given type, such as a file, might internally structure the files they manage quite differently, while presenting the same specification (representation and operations) externally at the

interface. This characteristic is important in dealing with the heterogeneity that results when a DOS is built on top of existing operating systems or even implemented directly on heterogeneous hardware/firmware components.

2.3 The Model Layers

Watson's model for distributed system architecture consists of four basic layers, meeting the layering guidelines and having the concept of objects and their interaction presented above:

- A hardware/firmware component layer - This layer consists of processors, memories, I/O units, terminals, etc.

- A distributed operating system kernel/IPC layer - This layer provides the minimal general purpose services such as creation of abstract objects, interprocess communication (IPC), and interfacing I/O structures to the message passing model.

- A distributed operating system service layer - This layer provides services useful to a wide variety of applications such as resource allocation and process creation, management, and deletion services.

- An application layer - This layer contains processes that provide application dependent services.

An important characteristic of Watson's model is that it explicitly recognizes the need for having a distributed operating system. Most of the existing literature on distributed systems or computer networking has focused on interprocess communication. Developments to date have made valuable services possible such as access to remote interactive programs from terminals on heterogeneous systems, simple file transfer, and electronic mail. However, there have been very few applications of resource sharing or distributed computing as defined in Chapter I. Adding one or more function-oriented protocols, such as file transfer protocols, built on top of the IPC layer is not going to offer resource sharing to support distributed systems as defined in Chapter I.

It is clear that an explicit distributed Kernel/IPC facility should be created as a core of the distributed operating system architecture model. The DOS should provide two basic facilities:

- It should turn a collection of distributed hardware/software resources into a coherent set of real and abstract objects (resources). The Kernel/IPC layer must support naming, sharing, protection, synchronization, and error recovery

- It should multiplex and allocate these resources among distributed processes and possibly in a distributed manner.

The DOS must provide this functionality in the face of the problems of heterogeneity and physical separation of components. In the rest of this chapter we are going to discuss in more detail the model layers starting from the top layer.

2.3.1 Application layer - The service provided by the application layer is clearly dependent on specific applications. Process management, communication, information management, virtual I/O and accounting are the usual services needed in different forms by all applications.

The issue here is how to organize and structure the processing, the data and other resources both physically and logically. For example:

- How should processing be distributed? Should it be distributed on a functional separation basis, a given level of fault tolerance, or other goals?

- Is the control of the application distributed or centralized, and how does the application process maintain synchronization?

- How should data be distributed? Should files be strictly partitioned, fully redundant, or

partially redundant? The decision will depend on cost factors such as money, reliability, responsiveness and also on available mechanisms to support consistency of multiple copies.

- What features should be in the languages used in addition to those desirable in a nondistributed systems? One view of distributed system design uses layers of abstraction to create the illusion of a nondistributed system at some level. How pure should this illusion be? How many of the distributed system naming, error control and resource management facilities should be visible and under user control, and which should be handled automatically by the system?

The application layer should provide answers for these questions.

2.3.2 Distributed operating system service layer - The services required by this layer are very similar to their counterparts for nondistributed systems including the ability to create and destroy resources, interrogate their status, read and write their data structures, account for their usage, and start and stop them. Services in this layer access objects in the lower kernel layer via interrupts and privileged commands, and those in the higher application layer via the message based IPC facility. In this layer, we still have the question about how much

resource management, error control, etc., should be visible at the DOS interface.

There are many approaches to defining a distributed operating system. One is to build a DOS on top of existing operating systems. Another is to define a DOS built as the native system on each host. In any case, common DOS goals are the following:

- Processes, terminal users and programmers should have a uniform coherent view of distributed objects. This means that the user should not have to program differently or use different procedures depending on resource location, i.e., host boundaries are largely or completely hidden.

- The DOS structure should be efficiently implemented. This means that access by local users to local services should be as efficient as a nondistributed operating system in terms of the number and kind of messages exchanged or overhead for the same operations.

- The DOS should be extensible. This implies that users can easily add new services built on existing services without requiring system programmers to add new privileged code. In other words, the focus would be on the definition of general purpose abstract objects and operations that can be used as building blocks for objects of

higher levels of abstraction, rather than elaborate definitions of restricted objects and operations producing restricted functions.

- The DOS should define standard models for logical server, resource structure, protection, error control, resource management, etc.

2.3.3 The distributed system kernel layer - The services to be provided by the kernel could vary depending on assumptions about the underlying hardware support, security, and applications to be supported. However, it is desirable to place the minimum functionality possible in the kernel. Services expected from the kernel layer are:

- Interprocess communication service (IPC). This service should provide the most primitive process synchronization mechanism.

- A message oriented interface to I/O structures. This service converts interrupts from I/O devices into messages to the kernel, and messages from the kernel into low level I/O commands.

- Basis for other services. The kernel should also provide the basis for creating processes; primitive objects; I/O resources; protection and security; and component multiplexing.

Even though it is the responsibility of this layer to create processes, the responsibility of managing these processes would reside in server processes outside the kernel.

Organization of the kernel will vary from one system to another depending on whether it is built from the hardware/firmware up or is built on top of existing operating systems. Of all the layers and sublayers, the IPC is best understood because most of the work in networking in the last decade has been focused on the IPC [CCI80,WAT80a,WAT80b]. Briefly, the design of the IPC service should allow the following goals:

- Each communicating entity should have complete autonomy and control over its own resources and state.

- There should be no a priori restrictions on which processes can communicate with each other. Knowing a process identification should be sufficient to communicate, not including possible access restrictions.

- Efficient support for both transaction and stream-oriented services should be provided.

- The user should have a uniform view of the system, by allowing communication among all processes to use the same mechanism, whether processes are

local or remote or whether they are user or system processes.

2.3.4 Hardware/firmware components - Most existing hardware/firmware components were designed to be used in standalone or tightly coupled systems. In distributed systems, it is important to utilize components that are more appropriate for a message oriented, heterogeneous, communication environment. Some examples are: I/O structures that provide more efficient IPC; components that support system state information and efficient privacy; intelligent components that can be directly connected to the network; and generally enabling much of the architecture to be placed in firmware.

2.4 Summary

The main features of Watson's distributed system architecture model introduced in this chapter are:

- layering and modularity;
- the need to explicitly create a distributed operating system built around a unified view of objects or resources;
- a message based IPC;
- support for both transaction and stream oriented services and applications.

CHAPTER III

FUNCTIONAL MODEL FOR DECENTRALIZED CONTROL IN DISTRIBUTED SYSTEMS

The principle goal of this work is to develop a viable decentralized system-wide control for distributed systems. The main goal of the model is to be able to make decisions locally to schedule job tasks to the available resources in the system. This is done in an uncertain environment for the good of the whole system according to some performance objectives. In order to achieve this goal, we need a definition of system state and a means for predicting a future system state. In such an environment our solution is to make decisions based on predicted values for the actual system state. In this research we assume that each job consists of a group of tasks, one task for each resource required by the job. A task can be deterministic (one which has a predetermined location for the resource) or nondeterministic (one which has no predetermined location for the resource). We also assume that the task's description is known upon the job's arrival to the system,

i.e., the number of operations required by the task from the resource, and a measure of the task length defined as the number of bytes to be used for estimating the communication cost for the task. A typical job consists of a CPU task and possibly one or more I/O task(s).

In the rest of this chapter, we will discuss in details the following points:

1. uniform resource model;
2. system state representation;
3. performance goals for the distributed control model;
4. factors not affecting the model;
5. databases needed for the model;
6. the task allocation algorithm;

3.1 Uniform Resource Model

A resource can be viewed as a data structure*1 (possibly distributed) which represents the resource, plus a set of all operations that can be performed on that resource. In turn, the data structure (resource representation) consists of two major parts:

- The resource state record (heading). This part of the data structure contains resource independent information such as creation time, last access

time, access rights, etc.

. The resource proper (body). This part of the data structure contains resource dependent information. For example, the body of a file resource may be the information itself and how to access it, i.e., the format to store or retrieve the information.

The functions required to perform operations on the data structures consequently can be classified into two main groups. One group performs operations on the resource state record data structure and involves accessing or modifying the resource heading. The other group performs operations on the resource body and are clearly resource dependent. A third group of functions is needed to create and destroy a resource.

3.2 System State Representation

In order to implement a decentralized control model we must:

1. define a representation for the system state;

2. be able to estimate the current system state;

---

1. For more details on abstract data types and data structures, refer to [GUT77,SHA77].

3. be able to predict the future system state.

3.2.1 Definition of system state - The system state is basically defined in terms of the resources in the system. A resource state is defined by: The resource workload arrival time distribution; the resource workload service time distribution; and the resource capacity. A node state is defined as the collection of the individual resource states at the node. The system state is defined as the collection of the node states in the system.

3.2.2 Estimation of the current system state - Since each node in the system has the ability to make scheduling decisions, each node must have an approximate value (estimation) of the current system state. Because of communication delays, a node's knowledge of other node's resource states will always be more or less inaccurate. To get other node's states, there are two main approaches:

- Centralize the system state information and make it available on demand to any remote process that needs it. This approach does not require saving the system state at each node and is relatively more accurate. However this approach requires high overhead and is less reliable since failure of the central node affects all nodes.

- Replicate the system state information at every node and update it periodically. This approach

requires saving the system state at each node. It is less accurate because of delays in updating. However this approach allows more concurrent decisions to be taken simultaneously and is more reliable as functioning of the system is independent from any node in the system.

We chose the second approach and use a "moving window technique" to update the system state at each node. This technique is discussed in detail in Chapter IV.

3.2.3 Prediction of the future system state - The representation of each state by a distribution allows the local controllers at each node to use the past history of the state to predict the actual present and future value of the state. We hope that this approach will minimize the effect of the communication time delay and its variability. Estimation of the resource state distribution parameters is discussed in detail as part of the distribution identification problem in Appendix C.

3.3 Performance Goals of the MOdel

The proposed model makes decisions based on predicted response time defined as the difference in time between job completion and job arrival to the system.

3.4 Factors not Affecting the Model

The model is not affected by the physical implementation of the following factors even though the model needs information related to them, such as the actual communication cost.

- Network switching policy. The model does not depend upon whether the network is implemented using circuit switching (a physical path is actually established between the source and destination nodes), or packet switching (messages are divided into packets and routed through the interconnection network without establishing a physical connection path).

- Network topology. The model is not affected by the network topology or whether the topology is static (the links between nodes are passive and dedicated), or dynamic (links can be reconfigured).

- Routing algorithm. This refers to a mechanism establishing which path the message is going to take between the source and destination nodes. Although the model is not affected by the physical implementation of the routing algorithm, it uses an estimation for the communication cost which must be calculated by the routing algorithm.

3.5 Data-Bases Needed for the Model

Two databases are needed as a part of the resource state record. One has static information such as resource name, how to reach the resource (address), and resource capacity. This database is updated only on resource addition/failure. The other database has dynamic information representing the workload arrival and service time distribution parameters for each resource in the system. This information is updated dynamically whenever the resource state changes beyond a predetermined limit.

3.6 Task allocation algorithm

A decentralized task allocation algorithm is one algorithm composed of "N" physically distributed entities, $\{e_1, e_2, \ldots, e_N\}$. Each of the entities is considered a local controller running asynchronously (potential performance improvement) and concurrently with the others, continually making decisions over time. Each entity "e_i" makes decisions based on system-wide objectives, rather than on local ones. Each e_i makes decisions on an equal basis with the other entities (no master entity). It is intended that the task allocation algorithm adapts to the changes in the state of the various nodes in the system.

It is hypothesized that these controlling entities acting together can produce greater benefits than cost. These include the cost of running the algorithm itself, the cost of transmitting the update information, and

the cost of moving the jobs between hosts.

Upon job arrival at any node, the local controller will assign the individual job tasks (CPU and I/O tasks) to server queues for the resources available in the system in a manner which achieves the model objectives (see Figure 3-1). For example, a job can arrive at node "j" and be migrated to node "i" where the CPU task will be executed, and from node "i" perform I/O tasks possibly at other nodes. Decisions taken by the local controllers in the general form are based on minimizing the total time cost (T) given by the estimated job response time. Defining the response time depends on the way the job is executed in the system. There are two main approaches to defining the job response time:

1. We can assume that all the tasks belonging to one job start execution together and independently, i.e., parallel execution. The response time will be defined as the summation of the cpu task migration time from the arrival node to the assigned cpu task execution node plus the maximum time for executing any of the job tasks including necessary communication cost for I/O tasks.

2. We can assume that the various tasks of the job alternate execution in time, i.e., serial execution. The response time will be defined

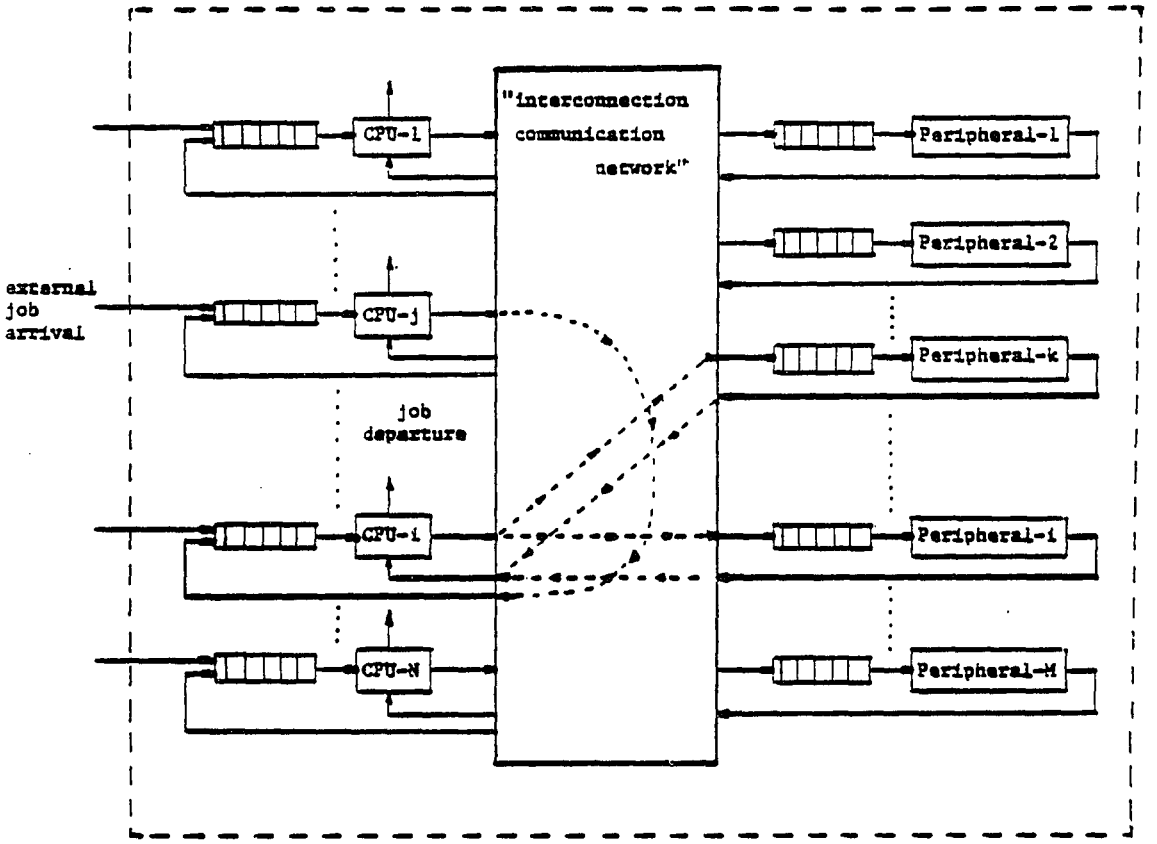


Figure 3-1. Decentralized Control Model for Distributed Systems

as the sum of the cpu task migration time from the arrival node to the assigned cpu task execution node plus the summation of the execution time costs of all the job's tasks individually. I/O tasks which are not executed at the cpu task node include their communication time cost as part of their execution time cost.

We feel that the second approach is closer to what happens in a real system.

On arrival of a new job to the system (consisting of N nodes) at node "j", the local controller at the job arrival node will calculate the estimated total time cost (T_j^i), for assigning the job to each of the nodes in the system, i= 1,2,..,j,..,N. Estimation of T_j^i will depend on the resource queueing model we assume. In the simulation We use three models M/M/1, M/G/1, and G/G/1. A brief description of each model and their steady state analysis results are presented in Appendix B. Of particular importanats are the equations for computing the estimated time a process wait in resource queue before starting execution. We can thus compare the simplicity of the M/M/1 model, the more accurate but relatively more complicated M/G/1 model, and the G/G/1 model which is the closest to the real environment but substantially more complex. The local controller will assign the job tasks to the system resources corresponding to minimum T_j^i .

3.6.1 Computing cost for one allocation - For a system having "N" nodes, let us assume an external job which arrives at node "j", consists of a cpu task and only one nondeterministic I/O task. Given a possible allocation for the CPU task to node "i" and the I/O task to node "k", apply the task allocation algorithm. The predicted job response time (T_j^i) for that specific allocation would be:

response time=C_{ji} + (time cost for CPU task)+
(time cost for I/O task)

$=C_{ji} + (T_{i1} + T_{i2}) + (T_{i3} + T_{i4} + T_{i5})$

Where:

C_{ji} ..Estimated time cost for migrating the CPU task from the arrival node "j" to the CPU task execution node "i", i.e., the size of the CPU task (S_{cpu}) multiplied by the cost of communication from node "j" to node "i" per unit size (c_{ji}).

$C_{ji} = S_{cpu} * c_{ji}$

T_{i1} ..Estimated "waiting time" at node "i" before the CPU task starts execution. This cost depends on the model used to represent the CPU resource, i.e., M/M/1, M/G/1 or G/G/1 model. Formulas for these models are given in Appendix B. In the case of the M/M/1 model:

$$T = \left[\frac{\rho/\mu C}{1-\rho}\right]_{CPU}$$

T_{i2} ..Estimated execution time for the CPU task at node "i".

$$T_{i2} = \left(\frac{1}{\mu C}\right)_{CPU}$$

T_{i3} ..Estimated time cost to access the required I/O resource from node "i", i.e., the size of the I/O task ($S_{i/o}$) multiplied by the cost of communication from node "i" to node "k" per unit size (C_{iK}).

$$T_{i3} = S_{i/o} * C_{iK}$$

T_{i4} ..Estimated "waiting time" for the I/O task before it starts execution. This cost depends on the model used to represent the I/O resource. In the case of the M/M/1 model:

$$T_{i4} = \left[\frac{\rho/\mu C}{1-\rho}\right]_{i/o}$$

T_{i5} ..Estimated execution time for the I/O task at node "k".

$$T_{i5} = \left(\frac{1}{\mu C}\right)_{i/o}$$

We can see that T_{i1} and T_{i4} are the only components which depend on the resource model representation used, i.e., M/M/1, M/G/1 or G/G/1 model.

In a real system environment, the job arrives at node "j" and we want to find node "i" and node "k" to allocate the job tasks achieving the system objectives. Applying the above algorithm for all possible allowed combinations, the job tasks will be scheduled according to the allocation combination which gives minimum cost. Assume $T_j^{i,K}$ is the total time cost for one specific allocation (CPU task to node "i" and I/O task to node "k") and apply the following algorithm:

```
for ( i=1 TO N ){
     assign CPU task to node "i";
     for ( k=1 TO N ){
          assign I/O task to node "k";
          compute T_j^{i,K} ;
     }
}
```

Schedule the CPU task to node "i" and the I/O task to node "k" having minimum $T_j^{i,K}$ of all possible combinations.

3.6.2 General job task allocation cost - In general, a job may consist of a CPU task, "m" deterministic I/O tasks, and "n" nondeterministic I/O tasks. The general equation for computing the predicted job response time for any one specific allocation is the summation of the estimated time cost for migrating the cpu task from the arrival node "j" to the cpu task execution node "i" and the sum of the time cost of the cpu task plus all I/O tasks of the job individually

including their communication cost.

As discussed in Chapter IV, it is not always necessary to apply the full general algorithm in a real world system. Most probably there will be only a limited number of nodes accessible from a particular node. This means the algorithm will require less time to perform the task allocation calculations.

3.7 Conclusion

The decentralized control model presented in this chapter offers a promising approach to the uncertainty problem in the system state of distributed systems. It offers a compact representation of the system resource states, and requires simple calculations which implies low overhead.

CHAPTER IV

APPLICATION OF THE MODEL

4.1 Introduction

Chapter III presents relatively simple model for making scheduling decisions based on queueing theory. If all the assumptions embedded in the model are correct for a given environment, we could utilize the model for scheduling without requiring further validation. However, there are several important assumptions that do not generally hold in real systems. First, the resource queueing models used in the prediction (M/M/1, M/G/1 or G/G/1) assume First Come First Service (FCFS) service scheduling, whereas real systems usually use a form of scheduling that is more like round robin. Second, in calculating the estimated time required to finish a job, we assume that the job consists of a set of independent tasks. In this chapter we discuss how the scheduling model of Chapter III can be applied to a real system. In this case, however, the analytical results are not sufficient - we must verify the validity of this

application by simulation. We hope that the simulation results will show a consistent performance improvement using our model.

4.2 A Feasible Real Environment

In order to simulate the performance of our model, we must create an environment that approximates a real environment. This environment has the following characteristics:

1. Job description. A job is assumed to be one cpu task, one terminal task, and zero or more I/O tasks. The terminal task is always assumed to be executed at the arrival host independent of the algorithm used for scheduling. I/O tasks can be deterministic (predetermined location for execution) or nondeterministic (location is determined by the scheduling).

2. Resource scheduling. Scheduling of all resources at every host uses a round-robin discipline for as many tasks up to the resource capacity. Additional tasks must wait in another queue until capacity is available before they enter the round-robin queue. Each task utilizes a percentage of the resource capacity until it is done.

3. Task description. The task descriptions are known, i.e., number of operations required per task and a measure of the task size which is used to estimate communication cost.

In large distributed systems, it is likely that the network would be partitioned into a set of overlapping subnetworks, i.e., each node is only allowed to communicate directly with part of the network. This issue of partitioning is not considered in our research, i.e., we assume that a job at any node in the network can access any resource at any other node in the network. This is not a major restriction since partitioning can be reflected by assigning appropriate communication costs between nodes.

4.3 Task Allocation Scheduling Algorithm

This section describes the three allocation algorithms that are implemented and compared via the simulation.

Upon external job arrival, the local controlling entity using any of the three schedulers will allocate the job tasks according to minimum predicted response time as discussed in Chapter III. In the mean time, each local controlling entity updates the resource's state to the network using a moving window technique. This technique is discussed in detail later in the chapter. The main difference between the three algorithms is the degree of freedom in scheduling the job tasks to the different resources in the network. We assume that each job consists

of a CPU task, a terminal task, and zero or more I/O tasks. For all three algorithms, the terminal task is always assigned to the arrival node resource, and deterministic I/O tasks are assigned to the corresponding predetermined node resources.

1. The Joint Migration Algorithm

The local controlling entity may schedule the cpu task and nondeterministic I/O tasks to any node in the network but they must be assigned to the same host. The time cost for executing this algorithm is linearly proportional to the number of nodes in the network.

2. The I/O Migration Algorithm

The local controlling entity schedules the CPU task to the arrival node, but each nondeterministic I/O task can be scheduled independently to any node in the network. The time cost of this algorithm is only slightly better than the full migration algorithm discussed below.

3. The Full Migration Algorithm

Both CPU task and nondeterministic I/O tasks can be scheduled independently to any host in the network. This approach should achieve the optimal minimum response time. However, the time cost for executing the algorithm can be high - it is proportional to N^r, where N is the number of nodes in the network and r is the number of tasks

to be allocated for the job.

In general, each scheduler allocates tasks based on calculating a predicted cost for executing the job assuming a specific allocation pattern. In doing this, each resource can be treated as an M/M/1, M/G/1 or G/G/1 queueing model. All models were tested in the simulation under different workloads to see the tradeoffs between them.

The three schedulers were simulated using the three queueing models and the performance of each were compared to the base-line case where no migration is allowed, i.e., both CPU task and nondeterministic I/O tasks must be allocated to the arriving host.

The simulation results provide important insights into the feasibility of the scheduling algorithms, the different resource models, and the effects on performance improvement of migration and communication costs.

4.4 Moving Window Technique

The resource state is represented by the workload arrival distribution, the workload service time distribution, and the resource capacity. It is clear that both arrival and service time distribution parameters will vary with time as the workload on the resource varies. Updating these two distribution parameter values involves two issues.

1. How much time is needed to estimate the distribution parameter values? We define a window of time within which we retain sufficient data to be able to compute these distribution parameters. The window moves forward in time with system time. Window size is defined as the most recent period of time used to estimate the state distribution parameters.

2. How often is the resource state updated? In this regard we have a tradeoff between a high update rate which means imposing high overhead on the communication capacity of the network, and a low update rate which means a greater chance that the current state value does not represent the actual value.

4.4.1 Window size - It is possible to define either a fixed-sized window or a variable-sized window.

1. Fixed window size - With a fixed size window of size "T", the tasks that have arrived at a resource during the most recent "T" time units are used in estimating the arrival and service time distribution parameters. This approach can not handle both high arrival rate and low arrival rate efficiently. In the case of high arrival rate, a small window will contain enough tasks to be able to calculate the distribution parameters

accurately. In the case of low arrival rate, we will need a larger window in order to have enough tasks to be able to calculate the distribution parameters. Now, if we have a fixed window size which is enough to estimate the distribution parameters in case of low arrival rate (large window size) this means an unnecessary overhead when the workload changes to high arrival rate. On the other hand, if we choose the window size small enough to handle high arrival rate efficiently, the window may be too small to handle a low arrival rate.

2. Variable window size - With a variable size window the controlling entity maintains enough past history to allow us to calculate the arrival and service time distribution parameters.

We have chosen the second approach because it is more accurate and more efficient. If we assume that at least "M" tasks are required to be able to calculate the distribution parameters. Then at any point in time, the window size would be "M" multiplied by the current mean interarrival time between two tasks at that resource. This means also that at any point in time, every resource in the network will have a different window size, and for any specific resource the window size is variable with time according to the workload variations on this resource. On the other

hand, the storage overhead for maintaing the data is constant, and the window size is linearly proportional to the total average arrival rate at a resource for all resources independently.

4.4.2 Resource state updating rate - There are two approaches to selecting the state update rate:

1. Fixed rate state update - Having a fixed rate of state update does not allow the system to respond to system state variations. If we have high fixed rate of state update, there will be high overhead when the system state varies slowly. On the other hand, having a low fixed rate of state update may lead to making decisions based on the wrong system state.

2. Variable rate state update - This approach basically tends to update the system state only when the system state changes significantly.

We chose the second approach. In our model recalculations are made following each time period equal to the mean interarrival time for that resource. Using the current window size ("M" multiplied by the current mean interarrival time for this resource), we calculate the new arrival time and service time distribution parameters. If the percentage difference between the new state and the old state is higher than a predetermined value, update the

system state. Having new parameter values will affect the next time to check the state of the resource (mean interarrival time for the resource) and the window size.

4.5 Resource State Initialization

Estimating the resource state (arrival time and the service time distribution parameters) is based on using the past history of the workload on the resource. At system start time, the resource does not have a history about its state. In a real system, these values would be determined based on experience. In the simulation, we initialize the resource arrival time and service time distribution parameters with the corresponding values of the arrival and service time distributions for jobs arriving to the node which has this resource. Knowing no history for the resource, this seems to be a reasonable guess for the resource's initial state. Each local controlling entity will adapt its resource state with time. In other words, the controlling entities will adjust its resources' states as it learns more about its history. This approach of handling the initial resource states has a special case which must be handled separately. Let us assume that one node in the network has no external job arrivals (mean job interarrival time is infinity and mean service time is zero). This means that the next time the controlling entity at this node updates the resource states at the node will be at time infinity, i.e., the resource states at this node

will never be adapted to the arrival of the internal task arrivals in the network. Consequently, all nodes in the network will think that this node's resources are free all the time and migrate their tasks to that node even though eventually the resources at this node will become heavily loaded. Our solution to this specific case is to force a maximum time to recheck the resource state. This value is defined as four times the initial average time to update the states at other nodes in the network.

4.6 The Model Operation

The controlling entity at each host has access to a data base representing the system state (i.e., distribution parameters for each resource in the system). On system start-up, the initialization phase takes place where each local controlling entity initializes its resource states (initial resource arrival time and service time distribution parameters) as discussed in the previous section. Upon mean interarrival time elapsing for any resource, the local controlling entity recalculates the new resource state (new arrival time and service time distribution) and compares them to the old values and updates the resource state in the network if the difference exceeds a fixed predetermined percentage. As implemented, the local task allocation controlling entity has two options.

1. The resource can be treated as an M/M/1, M/G/1 or G/G/1 queueing model.

2. The algorithm doing the task allocation can be any of the three algorithms - joint migration, I/O migration or full migration.

On external job arrival to any node, the local controlling entity according to the type of scheduling (algorithm) will allocate the different job tasks to the available resources in the system using the current local values for the resource state distribution parameters. Arrival of a migrated task to its assigned resource will affect this resource's state the next time the local controlling entity updates the resource state.

CHAPTER V

SIMULATION OF THE DECENTRALIZED CONTROL MODEL

5.1 Introduction

This chapter is intended to describe the simulation model used in testing and evaluating our decentralized control model under different workload environments. The entire simulation and auxiliary programs were implemented under the UNIX operating system using the C programming language [RIT78]. As implemented, the simulation can run many different variations of the basic scheduling model applied to any workload for an arbitrary network. In order to interpret the results of the simulation in a reasonable fashion, we present here comparative results of running nine variations on nine different workloads for a fixed network. We then present results from isolated experiments using other workloads and networks. Comparison between the various tests are based on job mean response time (the mean time a job spends in the system), load balancing of system

resources, and the number of migrations. It is important to note that all algorithms are relatively simple and require minimal run time cost.

5.2 Workload Description

5.2.1 Workload characteristics. The workloads used in the simulation are divided into three sets, light, moderate, and heavy. A light workload is defined as a workload which if equally balanced in the network will give around twenty percent cpu utilization in the network. A moderate workload is defined as a workload which if equally balanced in the network will give around forty percent cpu utilization in the network. A heavy workload is defined as a workload which if equally balanced in the network will give over sixty percent cpu utilization in the network. In addition, the workloads used can be categorized by their arrival and service time distributions as follows:

1. M/M Distribution Workload. The arrival is generated from a Poisson distribution, and the service time is generated from exponential distribution. The purpose of generating this kind of workload is to be able to compare the performance of applying the M/M/1 resource queueing model and the M/G/1 resource queueing model, i.e., comparing the use of both resource models under a workload which is typically biased to the M/M/1 resource model.

2. M/G Distribution Workload. The arrival distribution is generated from a poisson distribution, and the total service time is generated from a general distribution. The general distribution can be generated from any arbitrary distribution defined by its mean and variance. In all our workloads, the general distribution is generated from a normal distribution. The purpose of generating this kind of workload is to be able to compare the performance of the M/M/1 resource model under a general service time distribution with the more accurate M/G/1 resource model for this workload.

3. G/G Distribution Workload. The arrival distribution is generated from a normal distribution and the total service time is generated from a normal distribution. The purpose of generating this workload is to compare the M/M/1 and M/G/1 resource queueing models under a workload that does not have Poisson arrival and Exponential service distributions.

5.2.2 Workload Generation. The workload inputs to the simulation are generated separately. Input data to the workload generator program includes the following information: Number of hosts in the system, maximum number of deterministic I/O tasks allowed in a job, maximum number

of nondeterministic I/O tasks allowed in a job, total simulation time, and parameters for each host including the following:

1. Arrival distribution type and its parameter values. The arrival distribution can be Poisson, defined by its mean, or normal, defined by its mean and variance.

2. Total service time distribution. The service time is defined as the sum of the service times required by all tasks of a job, i.e., cpu, terminal, and I/O. The total service time can be generated from an exponential distribution, defined by its mean, or a normal distribution, defined by its mean and variance. We actually generate the total number of operations required by a job instead of the total service time so that the job is independent of specific resource capacities. The total number of operations generated is then divided among the job's individual tasks on a random basis using a uniformally distributed random number generator.

3. For the terminal and I/O tasks, we assume the number of operations required by the individual tasks also represents the size of the task in bytes to be used as a measure for the communication cost. In effect, each operation

represents the transfer of one byte. Since there is no direct correlation between the size of a program and the number of operations it executes, another distribution was used to determine the cpu task size.

The network was selected to consist of five hosts with each host having three resources: a cpu, a terminal, and an I/O resource. The capacities of the resources can be assigned for each host independently. In most of the tests reported here we assumed identical hosts with the resource capacities shown in table 5-1. It is important to mention that M/G and G/G workloads are generated separately but from distributions having the same mean and variance to ensure equivalent total resource demand for both workloads over a sufficiently long period of time. Similarly, programs were developed to generate equivalent M/M workloads from the M/G workloads

The parameters of the different M/G and G/G workloads are shown in tables 5-2a, 5-2b, 5-2c. There are no explicit parameters for the M/M workloads since they were regenerated from the actual M/G workloads.

5.3 The Simulation Model

The simulation model, consists of a network of an arbitrary number of hosts with any required topology. The topology of the network is represented by the communication costs in a routing table. Each host is represented by a cpu

resource type	capacity
CPU	300 M oper./hr
terminal	90 K oper./hr
I/O	300 K oper./hr

TABLE 5-1. Host Resources' Capacities

Host number	G/G (General arrival/General service distribution					
	interarrival mean	interarrival variance	mean service time	service variance	CPU length mean	CPU length variance
1	0.25	0.0025	0.3	0.003	40	10
2	0.35	0.0035	0.3	0.003	40	10
3	0.50	0.0050	0.3	0.003	40	10
4	0.50	0.0050	0.3	0.003	40	10
5	1.00	0.0100	0.3	0.003	40	10

Host number	M/G (Poisson arrival/General service distribution)				
	arrival rate	mean service time	service variance	CPU length mean	CPU length variance
1	4.0	0.3	0.003	40	10
2	3.0	0.3	0.003	40	10
3	3.0	0.3	0.003	40	10
4	2.0	0.3	0.003	40	10
5	1.0	0.3	0.003	40	10

TABLE 5-2a. Light Workload Parameters (time units in hours, task length units in Kbytes)

Host number	G/G (General arrival/General service distribution					
	interarrival mean	interarrival variance	mean service time	service variance	CPU length mean	CPU length variance
1	0.1666	0.001666	0.3	0.003	40	10
2	0.1666	0.001666	0.3	0.003	40	10
3	0.2500	0.00250	0.3	0.003	40	10
4	0.3333	0.00333	0.3	0.003	40	10
5	0.5000	0.0050	0.3	0.003	40	10

Host number	M/G (Poisson arrival/General service distribution)				
	arrival rate	mean service time	service variance	CPU length mean	CPU length variance
1	6.0	0.3	0.003	40	10
2	6.0	0.3	0.003	40	10
3	4.0	0.3	0.003	40	10
4	3.0	0.3	0.003	40	10
5	2.0	0.3	0.003	40	10

TABLE 5-2b. Moderate Workload Parameters (time units in hours, task length units in Kbytes)

Host number	G/G (General arrival/General service distribution					
	interarrival mean	interarrival variance	mean service time	service variance	CPU length mean	CPU length variance
1	0.1	0.001	0.3	0.03	40	10
2	0.1	0.001	0.3	0.03	40	10
3	0.2	0.002	0.3	0.03	40	10
4	0.2	0.002	0.3	0.03	40	10
5	0.25	0.0025	0.3	0.03	40	10

Host number	M/G (Poisson arrival/General service distribution)				
	arrival rate	mean service time	service variance	CPU length mean	CPU length variance
1	10.0	0.3	0.03	40	10
2	10.0	0.3	0.03	40	10
3	5.0	0.3	0.03	40	10
4	5.0	0.3	0.03	40	10
5	4.0	0.3	0.03	40	10

TABLE 5-2c. Heavy Workload Parameters (time units in hours, task length units in Kbytes)

resource, terminal resource, and one I/O resource. The units of time in the simulation is hours. Delay due to communication in the network is modeled as a simple function, i.e., the size of the information in Kbytes to be sent multiplied by a corresponding communication cost per Kbyte supplied to the system in the form of the routing table. The cost of running the algorithm is considered fixed.

Three algorithms are implemented as discussed in the previous chapter. Any of the three algorithms can use different models representing the resources in the prediction process used in scheduling jobs upon arrival. This includes the M/M/1, M/G/1, and G/G/1 queueing models.

5.3.1 Statistics. The following statistics are gathered:

1. The utilization of each resource in the network;
2. The mean time a task spends at a specific resource;
3. The mean time a task spends in the queue of a resource;
4. The mean time a job spends in the system;
5. The total number of jobs which arrived to the network during the simulation period;

6. The total number of jobs completed;

7. The total number of cpu tasks migrated;

8. The total communication cost in the network.

5.3.2 Resource State Update. The arrival and service time distribution parameters are updated upon task arrival to the resource or elapsed mean interarrival time for this resource. Resource state update uses the moving widow technique discussed in the previous chapter.

5.3.3 Job Execution. A job starts execution only after the cpu task arrives at the executing host and all its tasks can have their requirements from the different resources' capacities. Each task requires a certain percentage of the resource capacity based upon how the task relates to the other tasks comprising the job. In a real system the various parts of a job alternate execution - some cpu, then I/O, more cpu, more I/O etc., with considerable cooperation. To approximate that behaviour without simulating it exactly, we have spread out the execution of each task over the total time required to complete the job. We have accomplished this as follows:

1. Compute total job execution time as the sum of the cpu task execution time, the terminal execution time, I/O execution time, and total communication cost.

2. Calculate each task percentage requirements from its corresponding resource as the task time divided by the total job time.

3. Assign only that percentage of the resource capacity to this task.

In other words, if a job's cpu task is 10% of its total time, then its cpu task will be assigned 10% of the capacity of the host executing it. This is similar to the effect of any multitasking scheduling such as round-robin.

5.3.4 The Bias. The bias is a variable included in the simulation to compensate for the inaccuracies inherent in the model since the state information is necessarily uncertain as described in Chapter III.

Some inhibitation to migration is desirable to ensure stability. This bias against migration is defined as a percentage of the job execution time at the arrival host. The bias is closely related to the window size used for calculating the resource state and the workload.

5.4 Distribution Identification

We have discussed different types of workloads and different resource models used by the scheduling algorithms. It is important to note that even though the external arrival at a specific resource may be Poisson, there is no reason to believe that the total effective arrival at the resource (external arrivals plus internal

arrivals from other hosts) will be Poisson. It is important to be able to get conclusions from comparing the above workloads under different resource models to know the goodness-of-fit of our hypothesized distribution*2. We implemented a goodness- of-fit test to test on-line the hypothesis that the arrival process is drawn from a Poisson distribution and the service process is drawn from an exponential distribution according to a predetermined level of significance. This test is activated for every resource in the system every time the resource state is updated, i.e., every time a task arrives at a resource and every mean interarrival time for the resource.

At the end of the simulation, as part of the statistics gathered in the simulation, there is a summary for each resource in the network indicating how often the arrival distribution was Poisson, and how often the service distribution was exponential to a predetermined level of significance. These statistics help to provide better understanding of the results of the application of different resource models to the same workload. Table 5-3a, 5-3b show the workload distribution statistics for the heavy G/G workload when no migrations are allowed and when using the full-migration algorithm respecively. Table 5-4a, 5-4b show

---

2. A survey of the distribution identification problem and the goodness-of-fit tests are discussed in Appendix C.

resource type	arrival Poisson	service expon.
CPU	12%	15.6%
term.	0.7%	5.7%
I/O	2.7%	21.2%

a. No Migration

resource type	arrival Poisson	service expon.
CPU	5.5%	6.8%
term.	0.72%	5.4%
I/O	5%	15.5%

b. Full Migration

TABLE 5-3. Distribution Test Results for Heavy G/G Workload

resource type	arrival Poisson	service expon.
CPU	73.8%	12.2%
term.	78.4%	7.1%
I/O	71.1%	7.8%

a. No Migration

resource type	arrival Poisson	service expon.
CPU	46.3%	9%
term.	78.3%	7.1%
I/O	50%	6.3%

b. Full Migration

TABLE 5-4. Distribution Test Results for Heavy M/G Workload

the same statistics for the heavy M/G workload. Both results were tested under a fully connected network with communication capacity of 55.5 Kbyte/sec and a bias of 35%.These tables show that the implemented distribution identification scheme is working correctly, and also that the internal migrations reduce how often the arrival distribution is Poisson as shown in table 5-4a, 5-4b.

5.5 The Task Allocation Algorithm

In Chapter IV we discussed the three algorithms which are implemented in the simulation:

1. Joint migration. The cpu task and the nondeterministic I/O's are scheduled anywhere in the network but together as one unit.

2. I/O migration. The cpu task is scheduled to the arrival host and the nondeterministic I/O's are scheduled anywhere in the network and independently.

3. Full migration. The cpu task and all nondeterministic I/O's are scheduled anywhere in the network and independently.

As a base line of comparison, a fourth scheduler was implemented where the cpu task and all nondeterministic I/O's are forced to be executed locally, i.e., no task migrationsare allowed. For any of the schedulers, a

resource in the network can be represented as an M/M/1, M/G/1 or G/G/1 queueing model. However, we present only a few results using the G/G/1 model because this model has no accurate simple equation for the estimated waiting time for a task at a resource before it starts execution. There are only relatively simple lower and upper bounds as a function of the resource utilization. In the G/G/1 model we assume a linear interpolation between the lower bound and the upper bound for waiting time as a function of the resource utilization.

5.6 The Simulation Parameters

The following is a list of the simulation parameters:

1. The workload category, i.e., light, moderate or heavy;
2. The workload distribution type, i.e., M/G, M/M or G/G;
3. The resource model, i.e., M/M/1, M/G/1 or G/G/1;
4. The scheduling algorithm, i.e., full migration, joint migration or I/O migration;
5. Network topology and communication cost;
6. The bias;
7. The window size;

8. The level of significance used for distribution identification.

It is clear that it is impossible to try every possible combination of parameters. Some variables, however, are not crucial for the purpose of our tests. Since our goal is to understand and validate the model, we have chosen one homogeneous fully connected network for most tests. A simple structure helps in understanding the effects of varying other parameters. Most of our experiments used 55.5 Kbyte/sec communication speed. A set of early tests showed a window of size twenty tasks and a 20% level of significance for distribution identification were reasonable and they were used in all tests described below.

5.7 Simulation test goals

The main goals of the simulations tests presented are:

1. Evaluate the performance of different resource models under different workloads.

2. Evaluate the effect of communication cost on the different models.

3. Validate the workload distribution at any resource with respect to the resource model used.

4. Determine the value of the bias.

5.8 Simulation Results And Analysis

This section presents the simulation results, compares the different algorithms and discusses the applicability of the different resource models.

Three main workload sets were used in the simulation as discussed above (table 5-2). Before the network simulation was run for every workload, it was decided to obtain an upper bound on response time for the system. Therefore, the five hosts of our model were run for each different workload with no task migrations, i.e., cpu task and nondeterministic I/O tasks must be executed at the arrival host. The performance of the system with no migration is then used as a baseline for comparing the results of the other simulations.

5.8.1 Light Workload Set

1. Poisson arrival/General service distribution (M/G). For the no migration case table 5-5a shows the resource utilization in the network. Applying the full migration algorithm for the same workload with different biases, a homogeneous fully connected network with communication cost of 55.5 Kbyte/sec using both the M/M/1 and M/G/1 resource models, table 5-5b shows the response time and corresponding number of migrations which tends to be flat for bias values over 20%. Table 5-5c shows the resource utilization in the network for

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.38	0.108	0.348
2	0.31	0.092	0.314
3	0.20	0.064	0.193
4	0.19	0.058	0.218
5	0.11	0.042	0.141
response time = 0.231776 # of jobs completed = 233 Total # of jobs = 239			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.34	0.107	0.303
2	0.29	0.089	0.289
3	0.21	0.064	0.193
4	0.19	0.058	0.219
5	0.16	0.043	0.202
response time = 0.227544 # of jobs completed = 233			

c. Full Migration - M/G/1 Model Resource Utilization

Bias		0%	10%	15%	20%	25%	30%	40%	50%
M/M/1	response time	0.3799	0.2723	0.2324	0.2387	0.2370	0.2309	0.2302	0.2317
	# of migrations	172	78	41	28	24	14	4	1
	# of jobs completed	232	234	232	233	233	233	233	233
M/G/1	response time	0.4116	0.2456	0.2336	0.2275	0.2296	0.2302	0.2317	0.2318
	# of migrations	171	37	25	17	10	6	0	0
	# of jobs completed	232	232	233	233	233	233	233	233

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-5. Light M/G Workload

the M/G/1 resource model. This algorithm shows little improvement (1.8%) in response time and not much improvement in load balancing.

2. Poisson arrival/Exponential service workload (M/M). This workload is generated from the light M/G workload, table 5-6a shows the no migration case, table 5-6b shows results of the full migration algorithm using M/M/1 with improvement 8.5% and the M/G/1 model with improvement 6.6% under a bias of 20%. The M/M/1 model gives better results because it matches the M/M workload better than the M/G/1 model does. Table 5-6c shows the resource utilization for the M/M/1 model.

3. General arrival/General service workload (G/G). For the no migration case, table 5-7a shows the resource utilization in the network. Table 5-7b shows the results of applying the full migration algorithm under a 20% bias and communication cost of 55.5 Kbyte/sec using the M/M/1, M/G/1, and G/G/1. Table 5-7c shows the resources utilization in the network for the M/G/1 model.

Generally for the light workload, there is little or no improvement since there is sufficient capacity at each host to handle the external arrivals to that host efficiently, i.e., there is little or no incentive for migration. An

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.41	0.116	0.407
2	0.39	0.065	0.316
3	0.18	0.056	0.202
4	0.24	0.049	0.185
5	0.09	0.031	0.174
response time = 0.304049 # of jobs completed = 232 Total # of jobs = 239			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.29	0.114	0.290
2	0.33	0.068	0.242
3	0.24	0.054	0.250
4	0.23	0.049	0.256
5	0.23	0.031	0.252
response time = 0.2780 # of jobs completed = 232			

C. Full Migration - M/M/1 Model Resource Utilization

Algorithm	response time	# of migrations	# of jobs completed
no migrat.	0.304	0	232
M/M/1	0.278	76	232
M/G/1	0.284	61	230

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-6. Light M/M Workload

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.43	0.13	0.372
2	0.29	0.107	0.297
3	0.21	0.065	0.196
4	0.18	0.071	0.197
5	0.09	0.035	0.101
response time = 0.190337 # of jobs completed = 236 Total # of jobs = 239			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.373	0.13	0.335
2	0.294	0.107	0.297
3	0.209	0.065	0.196
4	0.197	0.071	0.197
5	0.154	0.035	0.138
response time = 0.19005 # of jobs completed = 236			

c. Full Migration - M/G/1 Model Resource Utilization

Algorithm	response time	# of migrations	# of jobs completed
no migrat.	0.1903	0	236
M/M/1	0.1912	25	236
M/G/1	0.1900	13	236
G/G/1	0.2431	91	235

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-7. Light G/G Workload

interesting result was the output of the G/G/1 workload. We believe that the 27% degradation of performance is caused by the fact that our linear interpolation approximation is not accurate, at least at low resource utilization.

5.8.2 Moderate Workload Set

Taking the same procedure as in the light workload set:

1. Poisson arrival/General service workload (M/G). For the no migration case, table 5-8a shows the resource utilization in the network. Table 5-8b shows the results of the full migration algorithm using M/M/1 and M/G/1 resource models under different biases. This table shows also that the response time tends to be flat after a 20% bias. Table 5-8c shows the resource utilization in the network for the M/G/1 resource model with bias 20%. The improvement over the base line is 14%.

2. Poisson arrival/Exponential service workload (M/M). Table 5-9a shows the no migration resource utilization in the network. From table 5-9b we can see that the full migration algorithm using M/M/1 gives 20% improvement, and using the M/G/1 model gives 11.6% improvement. Table 5-9c shows the resource utilization for the M/M/1 model.

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.61	0.169	0.538
2	0.51	0.165	0.49
3	0.38	0.108	0.348
4	0.31	0.092	0.314
5	0.21	0.064	0.193
response time = 0.315488 # of jobs completed = 390 Total # of jobs = 399			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.45	0.163	0.386
2	0.44	0.161	0.388
3	0.41	0.11	0.39
4	0.36	0.094	0.359
5	0.34	0.064	0.336
response time = 0.2713 # of jobs completed = 390			

c. Full Migration - M/G/1 Model Resource Utilization

Bias		0%	10%	20%	25%	30%	40%	50%
M/M/1	response time	0.5946	0.3962	0.2950	0.2833	0.2826	0.2841	0.2864
	# of migrations	295	187	100	75	67	47	26
	# of jobs completed	385	386	390	390	390	390	390
M/G/1	response time	0.5778	0.3002	0.2713	0.2801	0.2895	0.2897	0.2874
	# of migrations	303	118	65	58	48	27	21
	# of jobs completed	385	390	390	390	390	390	390

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-8. Moderate M/G Workload

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.64	0.172	0.605
2	0.55	0.164	0.619
3	0.41	0.116	0.407
4	0.39	0.065	0.316
5	0.18	0.056	0.202
response time = 0.525601 # of jobs completed = 387 Total # of jobs = 399			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.43	0.172	0.449
2	0.36	0.167	0.341
3	0.43	0.109	0.422
4	0.44	0.064	0.456
5	0.48	0.053	0.424
response time = 0.4184 # of jobs completed = 382			

c. Full Migration - M/M/1 Model Resource Utilization

Algorithm	response time	# of migrations	# of jobs completed
no migrat.	0.5256	0	387
M/M/1	0.4184	180	382
M/G/1	0.4645	161	382

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-9. Moderate M/M Workload

3. General arrival/General service workload (G/G). Table 5-10a shows the no migration resource utilization. For the moderate workload, a new experiment was run to determine the effect of communication cost on the response time and number of migrations. Table 5-10b compares the performance of the various models at 55.5 Kbyte/sec communication speed. The improvement for the M/M/1 and M/G/1 model are 7% and 15.7% respectively. The G/G/1 model is 31% worse than the baseline. Table 5-10b shows also a variety of communication speeds for the M/G/1 model. Variations from different communication speeds are uniform. This result occurs due to the nature of the jobs in the workload which place a relatively small demand on communication relative to the cpu and I/O demands. Table 5-10c shows the resource in the network utilization using the M/G/1 model.

Generally, under the moderate workloads, the M/M/1 and M/G/1 models show performance improvements over the baseline of no migrations, but the G/G/1 model is still worse. This further supports the belief that our approximation is not accurate. It is also worth noting that the relative performance of the M/M/1 and the M/G/1 are fairly close in all workloads.

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.599	0.1975	0.582
2	0.66	0.199	0.505
3	0.44	0.129	0.372
4	0.293	0.105	0.293
5	0.209	0.065	0.196
response time = 0.29384 # of jobs completed = 409 Total # of jobs = 420			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.47	0.2	0.48
2	0.52	0.2	0.43
3	0.46	0.13	0.37
4	0.41	0.11	0.35
5	0.35	0.06	0.34
response time = 0.2478 # of jobs completed = 412			

c. Full Migration - M/G/1 Model Resource Utilization

Comm. Cost		2.22 Kbyte/sec	5.55 Kbyte/sec	55.5 Kbyte/sec	0.55 Mbyte/sec	10 Mbyte/sec
M/M/1	response time			0.2731		
	# of migrations			79		
	# of jobs completed			412		
M/G/1	response time	0.2423	0.2507	0.2478	0.2405	0.2404
	# of migrations	55	58	59	57	57
	# of jobs completed	411	412	412	412	412
G/G/1	response time			0.3853		
	# of migrations			107		
	# of jobs completed			408		

b. Full Migration - Response Time Using Different Communication Capacities

TABLE 5-10. Moderate G/G Workload

5.8.3 Heavy Workload Set

1. Poisson arrival/General service distribution (M/G). Table 5-11a shows the no migration case resource utilization in the network. Table 5-11b shows the results of the full migration algorithm using the M/M/1 and M/G/1 resource models under different biases. This table shows that unlike the light and moderate workloads, there is a clear optimal bias for both resource models under heavy workloads. Because both models had very close performance in the range 30-40%, we have chosen 35% as a compromise optimal bias for both. The M/G/1 model shows an improvement of 48.1% over the baseline and the M/M/1 model shows 45.4% improvement. Table 5-11c shows the resource utilization in the network for the M/G/1 model under a 35% bias.

2. Poisson arrival/Exponential service distribution (M/M). Table 5-12a shows the no migration case resource utilization in the network. Table 5-12b shows the results of the full migration algorithm using the M/M/1 model which yields a 30% improvement and the M/G/1 model which yields 42% improvement. Table 5-12c shows the resource utilization for the M/G/1 model in the network.

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.87	0.246	0.838
2	0.85	0.274	0.848
3	0.50	0.138	0.434
4	0.45	0.15	0.452
5	0.38	0.107	0.344
response time = 1.045712 # of jobs completed = 586 Total # of jobs = 652			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.674	0.289	0.701
2	0.665	0.302	0.670
3	0.655	0.143	0.611
4	0.620	0.149	0.569
5	0.613	0.115	0.593
response time = 0.5423 # of jobs completed = 622			

c. Full Migration - M/G/1 Model Resource Utilization

Bias		0%	10%	20%	30%	35%	40%	50%
M/M/1	response time	1.0085	0.9193	0.7415	0.5956	0.5707	0.5543	0.5789
	# of migrations	491	460	377	297	246	217	155
	# of jobs completed	593	590	613	619	619	625	620
M/G/1	response time	0.9605	1.057	0.7309	0.5369	0.5423	0.5462	0.5585
	# of migrations	499	476	355	225	195	164	138
	# of jobs completed	596	588	606	619	622	620	628

c. Full Migration - Response Time Using Different Resource Models

TABLE 5-11. Heavy M/G Workload

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.816	0.228	0.846
2	0.77	0.269	0.862
3	0.523	0.154	0.506
4	0.508	0.148	0.575
5	0.418	0.114	0.404
response time = 1.392 # of jobs completed = 569 Total # of jobs = 652			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.639	0.242	0.668
2	0.577	0.326	0.631
3	0.629	0.146	0.631
4	0.649	0.135	0.674
5	0.642	0.109	0.683
response time = 0.8036 # of jobs completed = 599			

c. Full Migration - M/G/1 Model Resource Utilization

Algorithm	response time	# of migrations	# of jobs completed
no migrac.	1.392	0	569
M/M/1	0.8534	314	603
M/G/1	0.8036	317	599

b. Full Migration - Response Time Using Different Resource Models

TABLE 5-12. Heavy M/M Workload

3. General arrival/General service distribution (G/G). Table 5-13a shows the no migration resource utilization in the network. As shown in table 5-13b the full migration algorithm with a fixed bias 35% and 55.5 Kbyte/sec communication speed gives substantial improvement: 54.4%, 55.4%, and 49.7% for the M/M/1, M/G/1, and G/G/1 models respectively. Table 5-13c shows the resource utilization in the network for the M/M/1 model.

Generally the M/M/1 resource model seems to be stable and performs very close to the M/G/1 model under different environments. The G/G/1 model performs very poorly. Even though it does better than the baseline for the heavy workloads, it is still worse than either of the other models. As a result of these conclusions, we decided to use only the M/M/1 model on the heavy G/G workload for additional experiments described below.

5.8.4 Various Communication Speeds

To test the effect of the communication speed on any of the algorithms discussed before, we selected the heavy M/G workload and changed the resource capacities in the network to make the communication delay a more effective factor in the scheduling process. Applying the full migration algorithm and using the M/M/1 model under different communication speeds, table 5-14 shows that the algorithm is stable as the communication speed changes from 2.22

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.86	0.291	0.816
2	0.885	0.275	0.767
3	0.535	0.162	0.477
4	0.549	0.174	0.434
5	0.442	0.131	0.376
response time = 1.235791 # of jobs completed = 600 Total # of jobs = 678			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.75	0.33	0.67
2	0.72	0.31	0.65
3	0.71	0.16	0.60
4	0.72	0.17	0.61
5	0.68	0.13	0.60
response time = 0.56304 # of jobs completed = 651			

c. Full Migration - M/M/1 Model Resource Utilization

Comm. Cost		2.22 Kbyte/sec	5.55 Kbyte/sec	55.5 Kbyte/sec	0.55 Mbyte/sec	10 Mbyte/sec
M/M/1	response time			0.5630		
	# of migrations			160		
	# of jobs completed			651		
M/G/1	response time	0.5718	0.5621	0.5508	0.5389	0.5389
	# of migrations	142	138	137	139	139
	# of jobs completed	655	659	656	654	654
G/G/1	response time			0.6207		
	# of migrations			165		
	# of jobs completed			651		

b. Full Migration - Response Time Using Different Communication Capacities

TABLE 5-13. Heavy G/G Workload

Comm. Cost		2.22 Kbyte/sec	5.55 Kbyte/sec	55.5 Kbyte/sec	0.55 Mbyte/sec	10 Mbyte/sec
M/M/1	response time	0.037196	0.03704	0.03680	0.036775	0.036774
	# of migrations	6	13	22	23	24
	# of jobs completed	650	650	650	650	650
No - migration response time = 0.037308 # of jobs completed - 650 Total # of jobs = 652						

TABLE 5-14. Full Migration - Response Time Using Different Communication Speeds for the Heavy M/G Workload

Kbyte/sec to 10 Mbyte/sec. The response time consistently decreases and the number of migrations slightly increases.

5.8.5 G/G Workload With Deterministic I/O

This test utilized a G/G workload in which each job had one deterministic I/O task plus one or two nondeterministic I/O tasks. Table 5-15a shows the no migrations resource utilization. Applying the full migration algorithm and using the M/M/1 model under various biases, table 5-15b shows that the bias for the minimum response time is 40% and yields an improvement of 35% in response time. Table 5-15c shows the resource utilization in the network with bias 40%.

Table 5-15b shows also the response time for the full migration, I/O migration, and joint migration algorithms, using the M/M/1 model with a fixed bias of 40%, and communication speed of 55.5 Kbyte/sec. Even though the results here show close performance for the three algorithms, it is important to mention that other experiments show that normally the joint migration algorithm gives closer performance to the full migration than the I/O migration algorithm. Occasionally the joint migration algorithm even performed better than the full migration. We believe that the joint migration is a feasible alternative in most real environments. We would expect significantly worse performance only in a heavily loaded network with nodes that have widely disparate relative capacities of their CPU and I/O resources.

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.63	0.212	0.888
2	0.62	0.19	0.905
3	0.38	0.126	0.784
4	0.42	0.121	0.788
5	0.31	0.103	0.405
response time = 2.00266 # of jobs completed = 545 Total # of jobs = 680			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.54	0.26	0.849
2	0.50	0.22	0.835
3	0.47	0.12	0.795
4	0.48	0.11	0.839
5	0.50	0.10	0.796
response time = 1.304 # of jobs completed = 593			

c. Full Migration - M/M/1 Model Resource Utilization

Bias		20%	30%	35%	40%	50%
Joint migrat. M/M/1	response time				1.3682	
	# of migrations				290	
	# of jobs completed				585	
I/O migrat. M/M/1	response time				1.3739	
	# of migrations				0	
	# of jobs completed				585	
Full migrat. M/M/1	response time	1.3569	1.327	1.334	1.304	1.334
	# of migrations	328	275	257	229	207
	# of jobs completed	588	590	587	593	584

b. Response Time - Bias for the Three Algorithms

TABLE 5-15. Moderate G/G Workload with Deterministic I/O

5.8.6 Not Fully Connected Network

The above workloads with nondeterministic I/O tasks were tested under the network configuration shown in Figure 5-1 under different biases. Table 5-16 shows that a bias of 40% gives the minimum response time yielding an improvement of 36%.

5.8.7 System initialization. In order to test the sensitivity of the algorithm performance to initialization, we reran the heavy G/G workload with the M/M/1 model with initialization factors that were far from their true values. The results were almost identical to the results with correct initialization. We believe that any initialization which discourages migrations during the startup period will lead to a reasonable performance result.

5.9 Conclusions

We can summarize the conclusions presented in the previous section as follows:

For light workloads, there is not much performance improvement, but the algorithm should be tuned to guarantee stability.

For moderate workloads, there is a performance improvement of around 20% over the no migration case and more improvement in load balancing in the network. Also the M/M/1 model gives stable results that are very close to the M/G/1 model in a variety of workload environments.

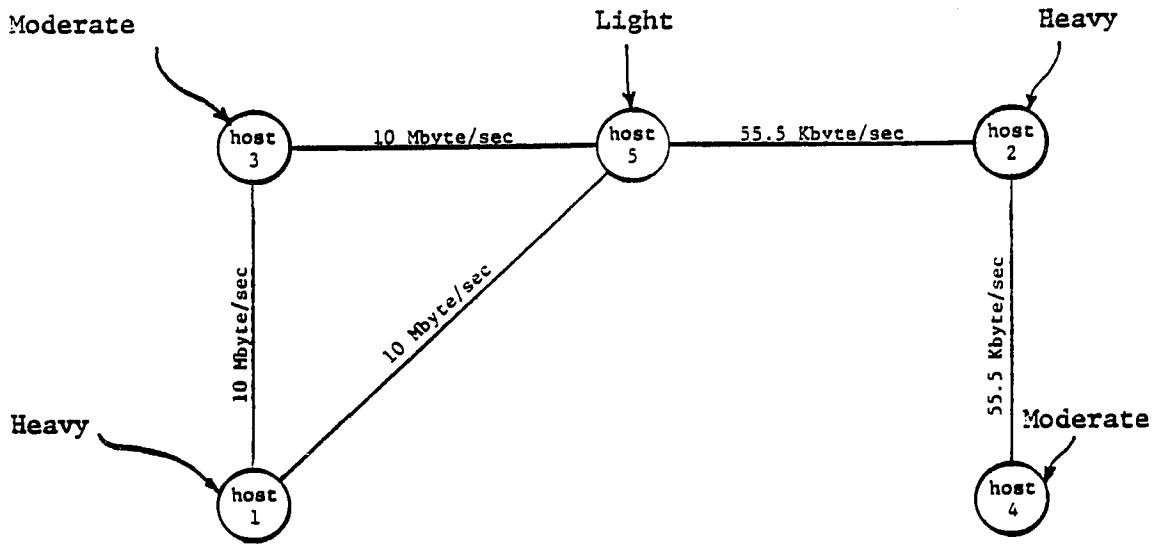


Figure 5-1. Not Fully Connected Network

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.64	0.217	0.902
2	0.63	0.179	0.886
3	0.38	0.125	0.791
4	0.41	0.111	0.749
5	0.31	0.103	0.405
response time = 1.9814 # of jobs completed = 543 Total # of jobs = 680			

a. No Migration Resource Utilization

Host number	CPU utiliz.	term utiliz.	I/O utiliz.
1	0.54	0.256	0.818
2	0.48	0.218	0.815
3	0.51	0.123	0.832
4	0.50	0.115	0.832
5	0.47	0.092	0.814
response time = 1.2539 # of jobs completed = 592			

c. Full Migration - M/M/1 Model Resource Utilization

Bias		20%	30%	40%	50%
M/M/1	response time	1.3558	1.2906	1.2539	1.3707
	# of migrations	323	276	237	203
	# of jobs completed	591	592	592	586

b. Full Migration - Response Time for Different Biases

TABLE 5-16. Moderate G/G Workload with Deterministic I/O for Not Fully Connected Network

For heavy workloads, there is a clear performance improvement of around forty percent over the no migration case and resource utilization statistics show very good load balancing in the network. The bias for minimum response time is almost the same for both M/M/1 and M/G/1 models. The G/G/1 model is not suitable for implementation since its performance is unstable and worse than both the M/M/1 and M/G/1 models.

CHAPTER VI

CONTRIBUTIONS AND FUTURE RESEARCH

6.1 Contributions

The major contribution of this thesis is the establishment of a methodology for implementing a decentralized scheduling mechanism for distributed systems. Our approach is to handle the problem in a similar way to the prediction problem in modern control theory, i.e., using the past history to predict the actual and future system state with a form of feedback to guarantee system performance improvement. This approach required solving two problems:

1. Defining the system state and making it independent of a specific system implementation.

2. Building a model which can use these states in predicting the actual and future state of the system.

The system state is defined as the collection of resource arrival and service distributions. By using queueing theory models, we can predict on-line the future system state, in an attempt to minimize the uncertainty in the system state.

The following are the important conclusions of our model demonstrated by the simulation.

1. The simulation indicates that the uncertainty problem in the system state can be approached using the past history of the resource to predict the actual system state. The simulation indicates also that representing the past history of a resource by a distribution is reasonable and valid.

2. The moving window technique discussed in Chapter IV for updating the system state proves to be efficient, i.e., it requires low overhead cost, and is a stable technique for updating the system state. By stability we mean that, even if the system is initialized in an incorrect initial state, it was shown that the system will converge to the correct system state after a transition period. This transition period is a period in which the system learns more about its history.

3. The results of using the M/M/1 model show very comparable results to the M/G/1 model even under a

general distribution (G/G) workload. This result is important in the applicability of the algorithm proposed because the M/M/1 model requires simpler calculations.

4. The simulation shows that the algorithm gives significant improvement if tuned properly. It seems that the performance of the algorithm is more sensitive to the bias under heavy workloads than under light or moderate workloads.

5. We believe that, depending upon the topology of a network and the nature of the workload, either the full migration algorithm or the joint migration algorithm can be used.

6. The results of using the G/G/1 model with the linear interpolation approximation proved to be not useful as it often gives worse performance than the no migration case, and worse performance than the M/M/1 and M/G/1 models consistently.

6.2 Future Extensions

The research presented in this thesis can be extended in several areas.

1. Improving the way the simulation executes the jobs, i.e., executing the jobs in the simulation closer to a real round-robin.

2. Addressing more carefully the interdependence between the different tasks belonging to one job.

3. Incorporating in our model more performance variables such as throughput.

4. Incorporating the effect of task migration on the communication capacity.

5. Evaluating the M/M/1 resource model under different general distributions other than the normal distributions.

6. Investigating the relation between the bias and the window size.

7. Implementing the M/M/1 resource model in our simulation using round-robin scheduling rather than first come-first service scheduling.

8. Investigating the possibility of allowing remigration after scheduling.

9. Investigating the possibility of adapting the model to handle the unknown job description. We can see two approaches to handle this problem:

    1. Assume the job description as the average job description at the arrival node.

2. Let the job start execution at the arrival node and gather statistics to identify the nature of the job and then evaluate whether to reschedule the job to another node.

10. Implementing other models in our simulation such as the Bayesian model (or others) for performance comparison purposes.

BIBLIOGRAPHY

[BOK77] Bokhari, S. H., Dual Processor Scheduling with Dynamic Reassignment, The Second Distributed Processing Workshop, Brown University, Providence, Rhode Island, (Augustl977).

[BRI80] Britton, D. E., and Stickel, M. E., An Interprocess Communication Facility for Distributed Applications, Proc. Distributed Computing, (Septemberl980), 590-598.

[CCI80] CCITT Rapporteur Report, X.25, Characteristics of Concern to Transport Services Working Paper, ANSI Document Number X3537-80-32R, (Aprill980).

[CLA80] Clark, D. D., Sovobodova, L., Design of Distributed Systems Supporting Local Anatonomy, COMPCON 80, (Springl980), 438-444.

[GRO73] Gross, D., Sensitivity of Queueing Models to the Assumption of Exponentiality: I-Single-Channel Queues, Technical Memorandum Serial 64121, Institute for Management Science and Engineering, The George Washington University, (1973).

[GUT78a] Guttag, Horowitz, and Musser, Abstract data types and software validation, CAMC, (Decemberl978).

[GUT78b] Guttag, The Design of Data Type Specification, In Current Trends in Programming Methodology, R. T. Yeh, Ed. Prentice-Hall, (1978).

[HOF80] Hoffman, R. H., and Smith, R. W., Applied Techniques for Testing Distributed Data Processing Software, Distributed Data Acquistion, Computing, and Control Symposium, (Decemberl980), 202-212.

[ISO79] Reference Model of Open Systems Interconnection, ISO/TC97/SC16N227, (Augustl979).

[JAF78] Jafari, H., Spragins, J., and Lewis, T., A New Modular Loop Architecture for Distributed Computer Systems, Trends and Applications: 1978 Distributed Processing, (1978), 72-83.

[JEN78] Jensen, D. E., The Honeywell Experimental Distributed Processor An Overview, Computer 11, 1, (January1978), 28-39.

[JON77] Jones, A. K., Software Management of Cm*-A Distributed Multiprocessor, Proc. AFIPS Conf. 46, (1977).

[JON78] Jones, A. K., The Object Model: A Conceptual Tool for Structuring Software, Operating Systems: An Advanced Course, Springer Verlag, (1978).

[KAH72] Kahn, Robert E., Resource-Sharing Computer Communications Networks, Proceedings of the IEEE 60, 1, (January1977), 85-93.

[KLE75] Kleinrock, L., Queueing Systems Theory: Volume 1, John Wiely & Sons, Inc., (1975).

[KLE76] Kleinrock, L., Queueing Systems Theory: Volume 2, John Wiely & Sons, Inc., (1976).

[KOH81] Kohler, W. H., A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems, ACM Computing Surveys 13, 2, (June81), 149-184.

[LAN77] Le Lann, G., Distributed Systems-Towards a Formal Approach, Proc. IFIP Congress, Toronto, North Holland Publishing Company, (August1977), 155-160.

[LAN78] Le Lann, G., Algorithms for Distributed Data Sharing Systems Which Use Tickets, Proc. 3rd Berkeley Workshop, (August1978), 259-272.

[LAN79] Le Lann, G., An Analysis of Different Approaches to Distributed Computing, Proc. 1st ICDPS, (Ocober1979), 222-232.

[LAN81] Le Lann, G., Motivations, Objectives and Characterization of Distributed Systems, Lecture Notes in Computer Science 105, 1, (1981), 1-9.

[LAY74] Lay, W. M., Mills, D. L., and Zelkowitz, M.V., Operating System Architecture for a Distributed Computer Network, ACM Conference on Trends and Applications of Minicomputer Networks, (April1974).

[LIN79] Lin, W. K., Concurrency Control in a Multiple Copy Distributed System, Proc. 4th Berkeley Workshop, (1979).

[LIU77] Liu, M. T., Pardo, R., Babic, G., A Performance Study of Distributed Control Loop Network, Proc. of the 1977 International Conference on Parallel Processing, (1977), 137-138.

[MAR74] Marchal, W. G., Some Simple Bounds and Approximations in Queueing, Technical Memorandum Serial T-294, Institute for Management Science and Engineering, The George Washington University, (January1974).

[MIC80a] Michael, L. G., Edward, Y. S., Douglas, C. S., A Distributed Real Time Operating System, Distributed Data Acquisition, Computing, and Control Symposium, (December1980), 175-184.

[MIC80b] Michael, L. G., Samprakash, M., A Distributed Real Time Resource Manager, Distributed Data Acquisition, Computing, and Control Symposium, (December1980), 185-193.

[MIC80c] Michael, L. G., Edward, Y. S., Douglas, C. S., A Distributed Data Base Manager, Distributed Data Acquisition, Computing, and Control Symposium, (December1980), 194-201.

[MIL76] Mills, David L., An Overview of the Distributed Computer Network, AFIPS Conference Proceedings NCC 45, 1, (1976), 523-531.

[MIN79] Minoura, T., A New Concurrency Control Algorithm for Distributed Database Systems, Proc. 4th Berkeley Workshop, (1979).

[RAN78] Randolph, T. Y., and Chandy, K. M., On the Design of Elementary Distributed Systems, Proc. of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, (August1978), 289-321.

[REA76] Reames, C. C., and Liu, M. T., Design and Simulation of the Distributed Loop Computer Network (DLCN), Proceedings of the Third Annual Symposium on Computer Architecture, ACM Computer Architecture News 4, 4, (January1976), 124-129.

[RIT78] Ritchie, D. M., Brian W. K., The C Programming Language, Prentice-Hall, Inc. (1978).

[ROB70] Roberts, L. G., and Wessler, B. D., Computer Network Development to Achieve Resource Sharing, AFIPS Conference Proceedings SJCC 36, 1, (May1970), 543-549.

[RUS81] Russell, R. D., and Bergeron, R. D., Machine Independent Migration and Scheduling in a Heterogeneous Distributed Computer System, University of New Hampshire, Department of Computer Science Technical Report, (December1981).

[SAP80] Saponas, T. G., and Crews, P. L., A Model for Decentralized Control in a Fully Distributed Processing System, Proc. Distributed Computing, (September1980), 307-312.

[SCH75] Schelonka, E. P., Resource Sharing with ARPANET, Computer Networks: A Tutorial, IEEE Computer Society, (1975), 5.19-5.22.

[SHA76] Shaw, and Wulf, An Introduction to the Construction and Verification of Alphard Programs, IEEE Transaction on Software Engineering, (December1976).

[SIN80] Sincoskie, W. D., Farber, D. J., The Series/1 Distributed Operating System: Description and Comments, Proc. Distributed Computing, (September1980), 579-584.

[STA80] Stankovic, J. A., A Comprehensive Framework For Evaluating Decentralized Control, Proc. at the 1980 International Conference on Parallel Processing, (1980), 181-187.

[STA81a] Stankovic, J. A., ADCOS-An Adaptive, System-Wide, Decentralized Controlled Operating System, University of Massachusetts, Amherst, Department of Electrical & Computer Engineering Technical Report ECE-CS-81-2, (November1981).

[STA81b] Stankovic, J. A., Bayesian Decision Theory and its Application to Decentralized Control Algorithms, University of Massachusetts, Amherst, Department of Electrical & Computer Engineering Technical Report, (November1981).

[STA82] Stankovic, J. A., Simulation of Adaptive, Decentralized Controlled, Job Scheduling Algorithms, University of Massachuestts, Amherst, Department of Electrical & Computer Engineering Technical Report, (January1982).

[STE81] Stephen, S. Y., Chen-Chau, and Sol, M. S., An Approach to Distributed Computing System Software Design, IEEE Transaction on Software Engineering se-7, 4, (July1981).

[STO77] Stone, H. S., Multiprocessor Scheduling with the Aid of Network Flow Algorithms, IEEE Transaction on Software Engineering 13, 1, (January1977), 85-93.

[TSA80] Tsay, D. P., Liu, M. T., Design of A Robust Network Front-End for the Distributed Double-Loop Computer Network (DDLCN), Distributed Data Acquisition, Computing, and Control Symposium, (December1980), 141-155.

[WAT79] Watson, R. W., Fletcher, J. G., An Architecture for Support of Network Operating System Services, Proc. 4th Berkeley Workshop, (August1979), 18-50.

[WAT80a] Watson, R. W., Comments on the EMCA Transport Protocol, Contributors to ISO, SC-16, (August1980).

[WAT80b] Watson, R. W., Network Architecture Design Issues: With Applications to Backend Storage Networks, Computer 13, 2, (February1980), 32-49.

[WAT81] Watson, R. W., Distributed System Architecture Model, Lecture Notes in Computer Science 105, 1, (1981), 10-43.

[WHI75] White, J. A., Schmidt, J. W., Bennett, G. K., Analysis of Queueing Systems, Academic Press, (1975).

APPENDIX A

Stochastic Process

A.1 Introduction

The environment in which the decentralized controllers make decisions is stochastic. This is because the information that these controllers have about the state of the system is uncertain; that is, they have only an estimate of the system state. In addition, a decision made using the estimated state of the system may prove less than optimal due to future random variations that are independent of the control decisions that can occur.

In this section, we will define a stochastic process*3, and discuss in more detail specific stochastic processes to be used later.

---

3. This is a brief definition and classification of a stochastic process, for more details, refer to [KLE75].

A.2 Definition and Classification of a Stochastic Process

Associate with each point of time "t" in the range of $\{-\infty<t<\infty\}$, a random variable X which has a sample space $\{-\infty<X<\infty\}$ and a corresponding probability density function f(x). A set of such random variables $\{X_t\}$ is called a stochastic process and is completely described by defining the probability density function of each random variable in the set and the joint probability density function of these random variables f(X1,X2,...,Xn). The state of a stochastic process at any point "t" in time is the value of the random variable "X ". If the permitted points in time when the process can change its state are finite, it is called a "discrete-time stochastic process", otherwise, it is called a "continous-time stochastic process".

In the following we will study some stochastic processes which are characterized by different kinds of dependency relations among their random variables $\{x_t\}$:

- Stationary process. In general, the properties of a stochastic process are time dependent. The process is called stationary if the probability density functions {f(x)} and the joint probability density function of the process f(x1,x2,.....,xn) are independent of absolute time. Clearly, consequences of that condition are that the mean E[x] and the variance var[x] are constants.

. Markov process. A stochastic process is called a Markovian process if the next state of the process depends only upon the current state and not on the previous states. In other words, the way in which the entire past history affects the future of the process state is completely summarized in the current state of the process (memoryless process).

In the case of a discrete-time Markov process, transitions from the current state to the next state are allowed only at a specific set of times; however, it should be clear that there is no restriction on the new state (within the state space). In the case of a continous-time Markov process, the transition between states may take place any time. Thus, we have to consider another random variable describing how long the process remains in its current state before making a transition. It has been shown that the exponential distribution is the only distribution which satisfies the continous-time Markov process definition, and the geometric distribution is the only one in the case of a discrete-time Markov process [KLE75].

. Birth-Death process. A special case of the Markov process is called a Birth-Death process. It allows state transitions only to take place between neighboring states. If the process is in

state X_n, then the next state can be only one of three; remain in its current state; move to state X_{n+1}; or move to state X_{n-1}. Birth-Death process can be either discrete or continous in time.

. Poisson process. A Poisson process is a special case of the birth-death process. In state transitions, the only allowed transitions are pure birth; i.e., a transition from state X would be to state X . Also, the distribution of time between transitions is memoryless but independent of the current state, i.e., the distribution of time between transitions in case of continous-time is exponential with a constant birth rate instead of being exponential with birth-rate a function of the current state.

Figure (A-1) shows the relationship between the above processes in the form of a Venn diagram. The symbols P_{ij} denote the probability of making a transition to state "j" given that the process is currently in state "i". Also, "f_τ" denotes the distribution of time between transitions; to say that "f_τ is memoryless" implies that if the process is discrete , then "f_τ" is a geometric distribution, whereas if it is a continous-time process, then "f_τ" is an exponential distribution. Furthermore, it is implied that f_τ may be a function of the current state for the process.

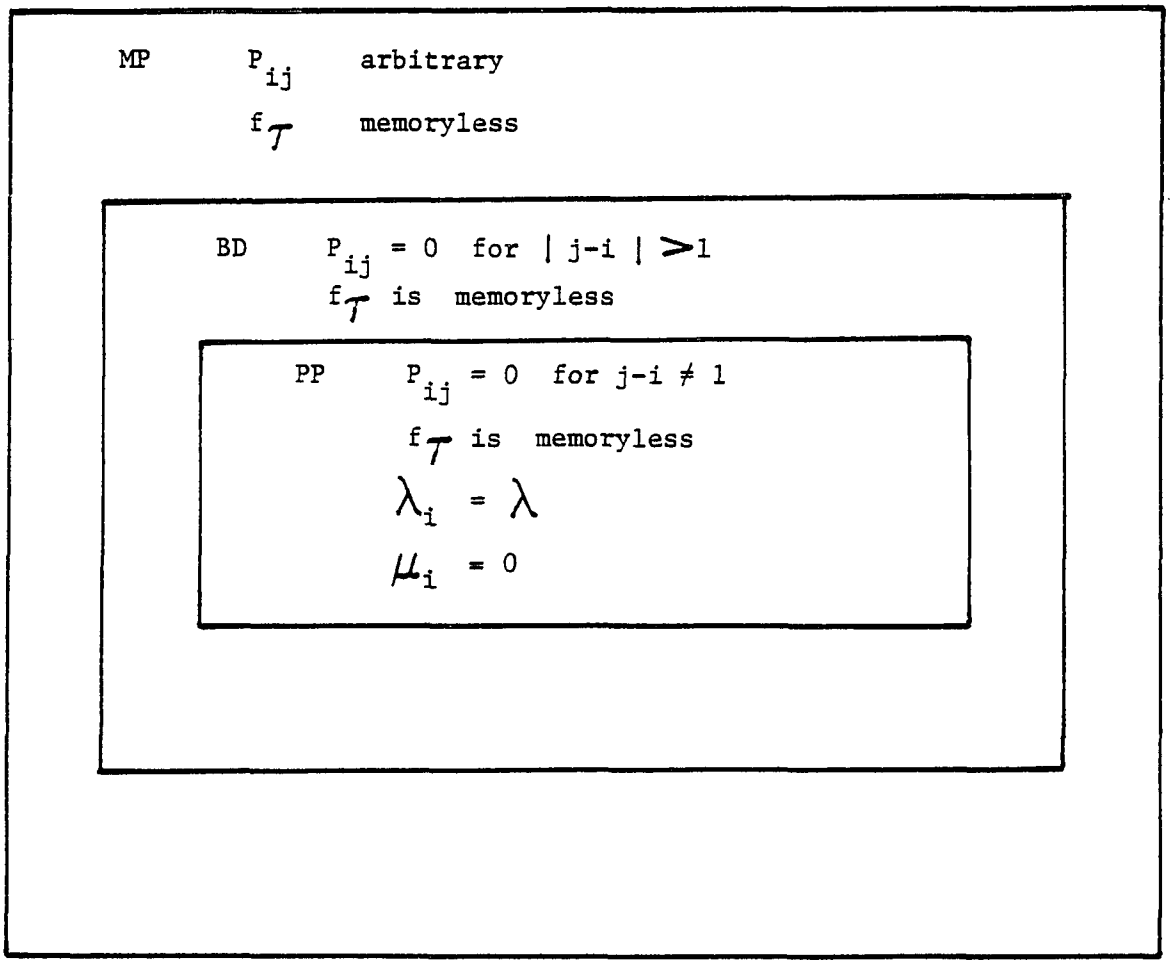


Figure A-1. Venn Diagram for Some Stochastic Processes
MP: Markovian Process; BD: Birth-Death Process; PP: Poisson Process

APPENDIX B

QUEUEING MODEL FOR A SINGLE RESOURCE

B.1 Introduction

Queueing theory offers a simple mathematical approach for modeling any resource using a probabilistic distribution. Queueing models are used in many applications including performance evaluation of computer systems and computer networks. They play a key role in understanding computer communication networks, where one of the primary factors for message delay in traversing a network is the queuing delay. A single resource can be represented by a "server-queue model" which requires defining the following:

1. The stochastic process describing the arrival time distribution of customers. This process is normally described in terms of the probability distribution of the interarrival time of customers.

   $A(t) = P[$ time between arrival $< t]$

2. The stochastic process describing the service time distribution required by the different

customers. This process is normally described in terms of length of time that customers need to spend in the service facility.

$B(x) = P[\text{service time} < x]$

3. The queue length. This represents the storage capacity available to hold customers waiting to receive service.

4. Server specification. This includes number of servers and their capacities. For example, for one CPU as a resource, capacity is defined as the maximum number of operations per unit time that can be performed by the CPU.

5. The queueing discipline. The order in which customers are taken from the queue and allowed into service.

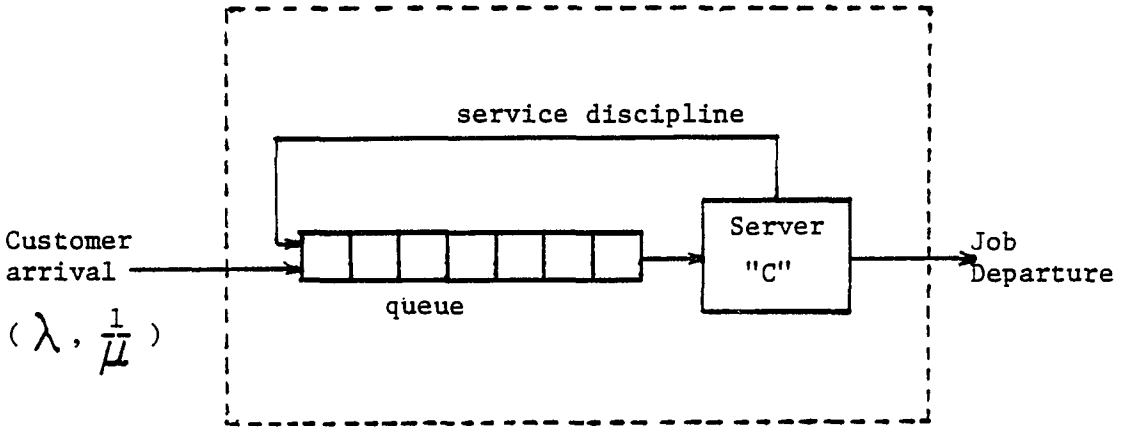
Figure (B-1) shows a single resource model. The following notation is usually used in the literature to define a queueing model:

A/S/N/L/C

Where,

A...Customers arrival time distribution;

S...Customers service time distribution;



λ : average job arrival rate;

$1/\mu$: average number of operations/job;

C : server capacity;

μC : average service rate;

ρ : resource utilization

= average arrival rate / average service rate

= $\lambda / \mu C$, $0 \leq \rho < 1$;

T : average response time

= average (completion time — arrival time);

W : average waiting time

= (response time — service time)

= $(T - \frac{1}{\mu C})$.

Figure B-1. Single Resource Queue Model

N...Number of servers in the resource;

L...Resource queue length;

C...Number of customers.

The last two terms are omitted if the queue length and number of customers are infinite which is the case in our model. Arrival and service time probability density functions are abbreviated using the following symbols:

M...Markovian (exponential) distribution

$$b(x) = \mu e^{-\mu X}$$

Er..r-stage Erlangian distribution

$$b(x) = \frac{r\mu(r\mu x)^{r-1} e^{-r\mu X}}{(r-1)!}$$

D...Deterministic distribution

$$b(x) = \delta\left[X - \frac{1}{\mu}\right]$$

G...General (arbitrary) distribution.

Two models are of interest in this research, both assume a First come-First service (FCFS) queueing discipline. Results of the steady-state analysis are listed below [KLE75].

B.2 M/M/1 Server Model

This model assumes a Poisson arrival distribution, i.e., exponential (Markovian) interarrival distribution and Exponential (Markovian) service time distribution; this

implies that the remaining service time for a job is independent of how long that job has been in service. The steady state solution for this model provides the following results:

T average response time

$$= \frac{1}{\mu C - \lambda}$$

W average waiting time in the queue/job

$$= \frac{\rho / \mu C}{1 - \rho}$$

also,

W = (average response time - execution time)

$$= T - (\frac{1}{\mu C})$$

B.3 M/G/1 Server Model

This model assumes a Poisson arrival distribution, and General service time distribution, this implies that the remaining service time for a job may depend on how long that job has been in service. The steady state solution for this model provides the following results:

1. Average waiting time

$$W = \frac{\sigma_b^2 + \bar{X}^2}{2\bar{t}(1 - \rho)}$$

Where,

$\bar{t}$...average interarrival-time

$\bar{X}$...average service-time

σ_b^2...Variance of the service-time distribution

2. Average response time

$$T = W + \left(\frac{1}{\mu C}\right)$$

B.4 G/G/1 Server Model

This model assumes general arrival and service distributions. General distribution is defined as a process where the future state depends on the current state and how the process reached that state. Applying this for general service time distribution, means that the remaining service time for a job may depend on how long that job has been in service. The results of the steady state solution for the average waiting time (W) and the average response time (T) are given below [KLE75].

1. Average waiting time

$$W = \frac{\sigma_a^2 + \sigma_b^2 + \bar{t}^2 (1-\rho)^2}{2\bar{t}(1-\rho)} - \frac{\overline{I^2}}{2\bar{I}}$$

where,

$\bar{t}$...average interarrival-time

σ_a^2...variance of interarrival-time distribution

σ_b^2...variance of service-time distribution

I...idle period distribution

The second term $(\frac{\overline{I^2}}{2\overline{I}})$ is hard to evaluate. However it is possible to find upper and lower bounds for the waiting time [KLE75]. The upper bound exceeds the known exact mean wait time for the M/G/1 model. Marchal [MAR74] has proposed that the upper bound be scaled down so that it is exact for the M/G/1 model; thus his approximation gives

$$W = \frac{1 + C_b^2}{(1/\rho)^2 + C_b^2}\left[\frac{\sigma_a^2 + \sigma_b^2}{2\bar{t}(1-\rho)}\right]$$

Where C_b, the service time coefficent of variation, is defined as $C_b = \frac{\sigma_b}{\bar{X}}$, $\bar{X}$ is the average service time. Both Marchal and Gross [GRO73] considered the effectiveness of this approximation to W. Their numerical studies show that the fit to G/G/1 is fair, improving as ρ increases, and degrading with an increase in C_a or C_b.

2. Average response time

$$T = W + (\frac{1}{\mu C})$$

APPENDIX C

DATA ANALYSIS AND DISTRIBUTION IDENTIFICATION

C.1 Introduction

In modeling a queueing system, certain assunptions must be made about the system in order to represent it by mathematical equations. For example, we frequently assume that interarrival and service times are exponentially distributed random variables. However, since our attempt is to model physical systems, we would wish to identify these distributions properly rather than simply make assumptions. The area of particular concern in this Appendix is the identification of the probability distribution of a random variable [WHI75]. This is a threefold problem. First, the analyst must collect data that characterize the random variable of concern. Using these data he can develop and plot a relative-frequency distribution for the random variable. By visually comparing the relative-frequency distribution with known probability density functions (pdf), one is often able to hypothesize that certain families of distribution describe the random variable under study. The selection of these candidate distributions is the first of the three problems. Second,

the problem is to determine the numerical values of the parameters of each candidate distribution based on the data collected. Third, the hypothesis that each distribution is, in fact, the true distribution of the random variable under study must be tested. These three problems can be statistically termed data collection, parameter estimation, and goodness-of-fit testing, respectively. In the next section, we will treat each of these topics in some detail.

C.2 Data collection

One common way for summarizing the collected data is the frequency distribution. In the case of discrete variables, we simply record the number of times (frequency) each value was observed. For continous random variables, we break the range of observed values into intervals and record the frequency that occurs within each interval.

Once a frequency distribution has been tabularized, a plot of entries is generally helpful in determining the distribution of the random variable under study. One useful method of displaying these data is to plot the relative frequency distribution. The relative frequency for each interval is simply the observed frequency count divided by the total frequency count. With this information at hand we can now proceed to select an appropriate probability distribution. This suggests also that more than one pdf may represent the collected data.

C.2 Distribution Parameters Estimation

The theory of estimation can be divided into two parts, point estimation and interval estimation. In point estimation, we concern ourself with the problem of producing a value that, in some sense, represents our best estimate as to the actual value of the parameter of interest. In interval estimation, we are interested in establishing an interval that would contain the true value of the parameter with some given level of probability. Such an interval is called a confidence interval. These intervals can also be viewed as possible measures of the precision of a point estimator. This is the view that we will adopt in the goodness-of-fit testing problem.

The general problem of point estimation can be stated as follows: There exists a random variable 'X' whose distribution function is characterized by some parameter 'Θ' that we would like to estimate. A random sample 'X1, X2,.....,Xn' is to be drawn and a function $\hat{\Theta} = \hat{\Theta}(X1, X2, ..., Xn)$ of this sample is formed. The value of $\hat{\Theta}$ is then used to estimate the parameter Θ. The function $\hat{\Theta}$ is referred to as an estimator of Θ and the value that $\hat{\Theta}$ takes on is called the estimate of Θ. Note that $\hat{\Theta}$ is itself a random variable, since it is a function of the random observations X1, X2,....., Xn. It should be apparent that there exist many estimators for a parameter depending on our measure which describes the function $\hat{\Theta}$.

We will talk now about various properties of an estimator that are considered desirable, and then discuss in detail one procedure to estimate a distribution parameter satisfying these desired properties.

An estimator for a parameter is itself a random variable and must therefore have a distribution of its own. Obviously we cannot specify this distribution until we have completely described the estimator itself. Let us define the estimator distribution <u>mean</u> $E[\hat{\Theta}]$, and its <u>variance</u> by

$$\mathrm{Var}(\hat{\Theta}) = E[\hat{\Theta} - E(\hat{\Theta})]^2 = E(\hat{\Theta}^2) - [E(\hat{\Theta})]^2$$

The standard deviation of $\hat{\Theta}$, is defined by $[\mathrm{Var}(\Theta)]^{1/2}$. Also of importance are the concepts sampling error, bias, and mean square error. <u>The sampling error</u>, defined by $\hat{\Theta} - \Theta$, is simply the difference between the value of the estimator and the true value of the parameter. <u>The bias</u>, defined by $E(\hat{\Theta}) - \Theta$, is the difference between the expected value of the estimator and the true value of the parameter. Whereas the sampling error may vary from sample to sample, the bias is fixed and may or may not be zero. <u>The mean-squared error</u>, defined by $E(\hat{\Theta} - \Theta)^2$, measures the dispersion of an estimator and is therefore similar to the concept of variance. The difference is that while the variance measures the dispersion of $\hat{\Theta}$ around its mean $E(\hat{\Theta})$, the mean squared error measures the dispersion around the true value of the parameter. If it turns out that $E(\hat{\Theta})$ and the true value of the parameter coincide, then the

mean-squared error and the variance are equivalent.

It should be recognized that by itself unbiasedness is not enough, since it implies nothing about the dispersion of the distribution of the estimator. Thus, an estimator can be unbiased but yet lead to estimates that lie far from the true value of the parameter. On the other hand, a biased estimator even with small variance often can be less useful. An efficient estimator is frequently called a minimum-variance unbiased estimator, which is an unbiased estimator with minimum variance.

Now we come to the problem of devising procedures to obtain estimators that have all or at least most of the above properties *4. We will discuss in detail the method of moments to estimate the parameters of a distribution.

.Method of moments estimator. This estimator is based on the principle that one should estimate a moment of a population by the corresponding moment of the sample. For example, consider a population whose density f(x) is characterized by K parameters $\Theta_1, \Theta_2, \ldots\ldots, \Theta_K$, which are to be estimated. Let X1,X2,.....,Xn be a random sample of size "n" from the population. The i_{th} sample moment is

---

4. There are other methods such as the maximum-likelihood method and least-square regression method [WHI75].

defined as

$$Mi = \frac{1}{n} \sum_{j=1}^{n} (X_j)^i$$

The i_{th} moment of the f(x) can be calculated mathematically and assumed to be E(x). Equating the first K population and sample moments, we have

$$M1=E(x),\ M2=E(x^2),.....,Mk=E(x^K)$$

The solution values $\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_K$ obtained by solving these simultaneous equations are referred to as the method of moment estimators for the K parameters $\theta_1, \theta_2, ..., \theta_K$.

C.4 Goodness-of-Fit Tests The third and final step in identifying a distribution is to test the hypothesis that there is no detectable difference between the hypothesized distribution and the sample distribution. We will discuss later in this section one test for a Poisson distribution and the Kolmogorov-Smirnov (K-S) test as a general purpose test for any distribution*5. The following are important terms which are commonly used in most of these tests.

1. Universe and sample. Any set of individuals (objects) having some common observable

---

5. There are also other common goodnes-of-fit tests such as Chi-Square test [WHI75].

characteristic constitutes a Universe. Any subset of the universe is a sample from the universe. There is then a "distribution of the measurements of a sample" which actually we observe and study and a "distribution of the measurements of the universe" which needed to be estimated. The problem is to decide what information about the distribution of the universe can be inferred from a study of the sample. For the universe distribution we will denote the mean of the universe by μ and the variance of the universe by σ^2. The sample size represented by the letter N, is the number of individuals in the sample. A sample may be any size from N=1 to the number of items in the universe.

The measurements on the individuals in the sample will form a distribution which will have a mean $\bar{X}$, and a variance S^2. Presumably $\bar{X}$ and S^2, which we can actually measure, should give us some information about μ and σ^2, whose values are usually not known. $\bar{X}$ and S^2 are different from sample to sample, while μ and σ^2 are constant, i.e., have particular values for a particular universe.

2. Central limit theorem. If X has any distribution with finite mean μ and finite variance σ^2, then the distribution of $\bar{X}$ approaches the normal

distribution with mean μ and variance $\frac{\sigma^2}{N}$ as the sample size increases. If, however, the distribution of the universe, is exactly normal, then the sampling distribution of the mean of any size sample, even N equal to one, will be exactly normal.

This does not say that we are likely to be close to the correct value of the parameter, but we shall estimate the parameter correctly on the average. Most of the goodness-of-fit tests are based on this theorem.

3. Confidence interval. From the central limit theorem, we have seen that the sampling distribution of $\bar{X}$ has a mean μ, standard deviation $\frac{\sigma}{\sqrt{N}}$, and is normal in shape. Therefore, we can use the area under the normal distribution $(\mu, \frac{\sigma}{\sqrt{N}})$ tables to find the proportion of the time we can expect to obtain a sample mean within a certain distance of μ. Then it is clear that a new random variable $Z = \frac{\bar{X} - \mu}{(\sigma/\sqrt{N})}$ which is a normalized normal distribution having mean equal to zero and standard deviation equal to one could be used.

4. Level of significance (α). The hypothetic distribution will be rejected if the sample mean value lies in the region where the proportion of

the time we can expect to find the sample mean is equal to the level of significance; assuming the hypothetic distribution is true. In another way, the hypothetic distribution is rejected if the sample mean value lies outside a confidence interval equal to $(1-\alpha)$.

C.4.1 Poisson-Process Test. This test is particularly useful when sample data are sparse, since it does not require any knowledge of the universe distribution parameters.

Let $t_1, t_2, \ldots, t_n$ denote the times at which n units enter a queueing system during a time interval of length T. If these arrivals are from a Poisson process then the times are independent and uniformally distributed over the interval (0-T) with the mean of the universe distribution $=T/2$, and the variance of the universe $=T^2/12n$. The mean of the sample $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} t_i$

Thus, to test the hypothesis that the arriving units are from a Poisson process, we simply compute the normal test variable Z

$$Z = \frac{S_n - T/2}{\sqrt{T^2/12n}}$$

Choose a level of significance α, and locate the critical values $Z_{1-\frac{\alpha}{2}}$ and $Z_{\frac{\alpha}{2}}$, in Table C-1. If $Z_{1-\frac{\alpha}{2}} < Z < Z_{\frac{\alpha}{2}}$ we reject the hypothesis that our arrival is from a Poisson process with level of significance α.

To summarize, the Poisson test proceeds as follows:

1. Compute $\overline{X}$;

2. Compute the normal test variable Z;

3. Locate the critical values $Z_{\frac{\alpha}{2}}$ and $Z_{1-\frac{\alpha}{2}}$ in Table C-1;

4. If $Z_{1-\frac{\alpha}{2}}<Z<Z_{\frac{\alpha}{2}}$, then we reject the hypothesis that the data could have come from a poisson process.

Since the test is based on the central limit theorem, as a rule of thumb we can say that the test can be applied tsafely whenever $N \geq 20$.

C.4.2 Kolmogorov-Smirnov test. This test compares the PDF for the hypothesized distribution F(x) with the sample cumulative distribution $S_n(x)$. The sample cumulative distribution is defined by $S_n(x)=i/n$, where i is the number of observations less than or equal to X, and n is the sample size. The comparison between $S_n(x)$ and F(x) is based on the absolute value of their difference:

$$D_{max} = \max_x \left| F(x) - S_n(x) \right|$$

The value D_{max} is called the maximum deviation. The distribution of D_{max} is known and can be shown to be independent of F(x). Several values from the distribution have been tabulated as a function of n, the number of observations, and α, the level of significance. Table C-2 lists some of these values for different n and combinations.

The hypothesis that the data come from the hypothesized distribution is rejected at the level of significance whenever $D_{max} \geq D_n^{\alpha}$, where D_n^{α} are the critical values listed in Table C-2.

Figure C-1 illustrates a typical situation in which F(x) is continous. It is important to note that two differences must be computed at each step point X_i, $\left|F(x_i) - S_n(x_i)\right|$ and $\left|F(x_i) - S_n(x_{i-1})\right|$, since one is permitted to choose either the bottom or top of each step.

To summarize, the (K-S) test proceeds as follows:

1. Determine $S_n(x)$ from the sample data;

2. Compute $\left|F(x_i) - S_n(x_i)\right|$ and $\left|F(x_i) - S_n(x_{i-1})\right|$ at each step point X_i, if F(x) is continous. If F(x) is discrete, only $\left|F(x_i) - S_n(x_i)\right|$ need to be computed;

3. Determine the maximum value D_{max} from step 2;

4. Locate the critical value D_n^{α} in Table C-2;

5. If $D_{max} \geq D_n^{\alpha}$, reject the hypothesis that the data have come from the population described by F(x).

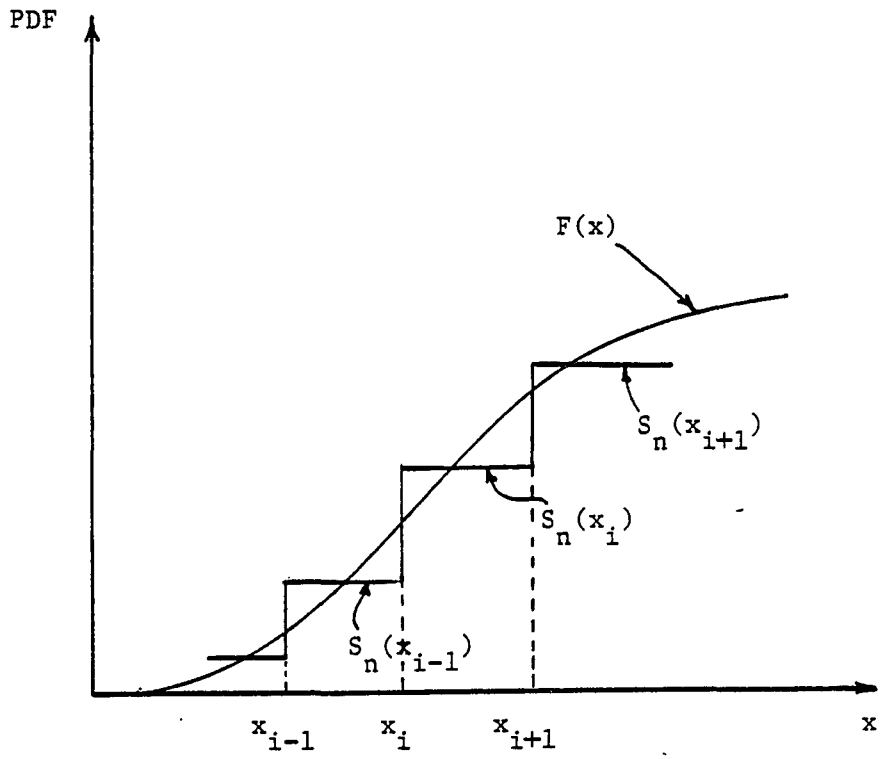


Figure C-1. Step Points for F(x) Continuous

Z	F(z)	Z	F(z)	Z	F(z)	Z	F(z)
−4.000	0.0000	−3.750	0.0001	−3.500	0.0002	−3.250	0.0006
−3.990	0.0000	−3.740	0.0001	−3.490	0.0002	−3.240	0.0006
−3.980	0.0000	−3.730	0.0001	−3.480	0.0003	−3.230	0.0006
−3.970	0.0000	−3.720	0.0001	−3.470	0.0003	−3.220	0.0006
−3.960	0.0000	−3.710	0.0001	−3.460	0.0003	−3.210	0.0007
−3.950	0.0000	−3.700	0.0001	−3.450	0.0003	−3.200	0.0007
−3.940	0.0000	−3.690	0.0001	−3.440	0.0003	−3.190	0.0007
−3.930	0.0000	−3.680	0.0001	−3.430	0.0003	−3.180	0.0007
−3.920	0.0000	−3.670	0.0001	−3.420	0.0003	−3.170	0.0008
−3.910	0.0001	−3.660	0.0001	−3.410	0.0003	−3.160	0.0008
−3.900	0.0001	−3.650	0.0001	−3.400	0.0003	−3.150	0.0008
−3.890	0.0001	−3.640	0.0001	−3.390	0.0004	−3.140	0.0009
−3.880	0.0001	−3.630	0.0001	−3.380	0.0004	−3.130	0.0009
−3.870	0.0001	−3.620	0.0002	−3.370	0.0004	−3.120	0.0009
−3.860	0.0001	−3.610	0.0002	−3.360	0.0004	−3.110	0.0009
−3.850	0.0001	−3.600	0.0002	−3.350	0.0004	−3.100	0.0010
−3.840	0.0001	−3.590	0.0002	−3.340	0.0004	−3.090	0.0010
−3.830	0.0001	−3.580	0.0002	−3.330	0.0004	−3.080	0.0010
−3.820	0.0001	−3.570	0.0002	−3.320	0.0005	−3.070	0.0011
−3.810	0.0001	−3.560	0.0002	−3.310	0.0005	−3.060	0.0011
−3.800	0.0001	−3.550	0.0002	−3.300	0.0005	−3.050	0.0012
−3.790	0.0001	−3.540	0.0002	−3.290	0.0005	−3.040	0.0012
−3.780	0.0001	−3.530	0.0002	−3.280	0.0005	−3.030	0.0012
−3.770	0.0001	−3.520	0.0002	−3.270	0.0005	−3.020	0.0013
−3.760	0.0001	−3.510	0.0002	−3.260	0.0006	−3.010	0.0013

TABLE C-1. Cumulative Distribution Function F(z)
of the Standard Normal Random Variable Z

Z	F(z)	Z	F(z)	Z	F(z)	Z	F(z)
−3.000	0.0014	−2.500	0.0062	−2.000	0.0227	−1.500	0.0668
−2.990	0.0014	−2.490	0.0064	−1.990	0.0233	−1.490	0.0681
−2.980	0.0014	−2.480	0.0066	−1.980	0.0239	−1.480	0.0695
−2.970	0.0015	−2.470	0.0067	−1.970	0.0244	−1.470	0.0708
−2.960	0.0015	−2.460	0.0069	−1.960	0.0250	−1.460	0.0722
−2.950	0.0016	−2.450	0.0071	−1.950	0.0256	−1.450	0.0735
−2.940	0.0016	−2.440	0.0073	−1.940	0.0262	−1.440	0.0750
−2.930	0.0017	−2.430	0.0075	−1.930	0.0268	−1.430	0.0764
−2.920	0.0018	−2.420	0.0078	−1.920	0.0274	−1.420	0.0778
−2.910	0.0018	−2.410	0.0080	−1.910	0.0281	−1.410	0.0793
−2.900	0.0019	−2.400	0.0082	−1.900	0.0287	−1.400	0.0808
−2.890	0.0019	−2.390	0.0084	−1.890	0.0294	−1.390	0.0823
−2.880	0.0020	−2.380	0.0086	−1.880	0.0301	−1.380	0.0838
−2.870	0.0021	−2.370	0.0089	−1.870	0.0307	−1.370	0.0854
−2.860	0.0021	−2.360	0.0091	−1.860	0.0314	−1.360	0.0869
−2.850	0.0022	−2.350	0.0094	−1.850	0.0322	−1.350	0.0885
−2.840	0.0023	−2.340	0.0096	−1.840	0.0329	−1.340	0.0901
−2.830	0.0023	−2.330	0.0099	−1.830	0.0336	−1.330	0.0918
−2.820	0.0024	−2.320	0.0102	−1.820	0.0344	−1.320	0.0934
−2.810	0.0025	−2.310	0.0104	−1.810	0.0352	−1.310	0.0951
−2.800	0.0026	−2.300	0.0107	−1.800	0.0359	−1.300	0.0968
−2.790	0.0026	−2.290	0.0110	−1.790	0.0367	−1.290	0.0985
−2.780	0.0027	−2.280	0.0113	−1.780	0.0375	−1.280	0.1003
−2.770	0.0028	−2.270	0.0116	−1.770	0.0384	−1.270	0.1021
−2.760	0.0029	−2.260	0.0119	−1.760	0.0392	−1.260	0.1038
−2.750	0.0030	−2.250	0.0122	−1.750	0.0401	−1.250	0.1057
−2.740	0.0031	−2.240	0.0125	−1.740	0.0409	−1.240	0.1075
−2.730	0.0032	−2.230	0.0129	−1.730	0.0418	−1.230	0.1094
−2.720	0.0033	−2.220	0.0132	−1.720	0.0427	−1.220	0.1112
−2.710	0.0034	−2.210	0.0135	−1.710	0.0436	−1.210	0.1132
−2.700	0.0035	−2.200	0.0139	−1.700	0.0446	−1.200	0.1151
−2.690	0.0036	−2.190	0.0143	−1.690	0.0455	−1.190	0.1170
−2.680	0.0037	−2.180	0.0146	−1.680	0.0465	−1.180	0.1190
−2.670	0.0038	−2.170	0.0150	−1.670	0.0475	−1.170	0.1210
−2.660	0.0039	−2.160	0.0154	−1.660	0.0485	−1.160	0.1230
−2.650	0.0040	−2.150	0.0158	−1.650	0.0495	−1.150	0.1251
−2.640	0.0041	−2.140	0.0162	−1.640	0.0505	−1.140	0.1272
−2.630	0.0043	−2.130	0.0166	−1.630	0.0516	−1.130	0.1293
−2.620	0.0044	−2.120	0.0170	−1.620	0.0526	−1.120	0.1314
−2.610	0.0045	−2.110	0.0174	−1.610	0.0537	−1.110	0.1335
−2.600	0.0047	−2.100	0.0179	−1.600	0.0548	−1.100	0.1357
−2.590	0.0048	−2.090	0.0183	−1.590	0.0559	−1.090	0.1379
−2.580	0.0049	−2.080	0.0188	−1.580	0.0571	−1.080	0.1401
−2.570	0.0051	−2.070	0.0192	−1.570	0.0582	−1.070	0.1423
−2.560	0.0052	−2.060	0.0197	−1.560	0.0594	−1.060	0.1446
−2.550	0.0054	−2.050	0.0202	−1.550	0.0606	−1.050	0.1469
−2.540	0.0055	−2.040	0.0207	−1.540	0.0618	−1.040	0.1492
−2.530	0.0057	−2.030	0.0212	−1.530	0.0630	−1.030	0.1515
−2.520	0.0059	−2.020	0.0217	−1.520	0.0643	−1.020	0.1539
−2.510	0.0060	−2.010	0.0222	−1.510	0.0655	−1.010	0.1563

TABLE C-1. Continued

Z	F(z)	Z	F(z)	Z	F(z)	Z	F(z)
−1.000	0.1587	−0.500	0.3086	0.000	0.5000	0.500	0.6915
−0.990	0.1611	−0.490	0.3121	0.010	0.5040	0.510	0.6950
−0.980	0.1636	−0.480	0.3157	0.020	0.5080	0.520	0.6985
−0.970	0.1660	−0.470	0.3192	0.030	0.5120	0.530	0.7020
−0.960	0.1685	−0.460	0.3228	0.040	0.5160	0.540	0.7054
−0.950	0.1711	−0.450	0.3264	0.050	0.5200	0.550	0.7089
−0.940	0.1736	−0.440	0.3300	0.060	0.5240	0.560	0.7123
−0.930	0.1762	−0.430	0.3336	0.070	0.5279	0.570	0.7157
−0.920	0.1788	−0.420	0.3373	0.080	0.5319	0.580	0.7191
−0.910	0.1814	−0.410	0.3409	0.090	0.5359	0.590	0.7224
−0.900	0.1841	−0.400	0.3446	0.100	0.5399	0.600	0.7258
−0.890	0.1867	−0.390	0.3483	0.110	0.5438	0.610	0.7291
−0.880	0.1894	−0.380	0.3520	0.120	0.5478	0.620	0.7324
−0.870	0.1922	−0.370	0.3557	0.130	0.5517	0.630	0.7357
−0.860	0.1949	−0.360	0.3595	0.140	0.5557	0.640	0.7389
−0.850	0.1977	−0.350	0.3632	0.150	0.5596	0.650	0.7422
−0.840	0.2005	−0.340	0.3670	0.160	0.5636	0.660	0.7454
−0.830	0.2033	−0.330	0.3707	0.170	0.5675	0.670	0.7486
−0.820	0.2061	−0.320	0.3745	0.180	0.5714	0.680	0.7518
−0.810	0.2090	−0.310	0.3783	0.190	0.5754	0.690	0.7549
−0.800	0.2119	−0.300	0.3821	0.200	0.5793	0.700	0.7581
−0.790	0.2148	−0.290	0.3859	0.210	0.5832	0.710	0.7612
−0.780	0.2177	−0.280	0.3898	0.220	0.5871	0.720	0.7643
−0.770	0.2207	−0.270	0.3936	0.230	0.5910	0.730	0.7673
−0.760	0.2236	−0.260	0.3975	0.240	0.5949	0.740	0.7704
−0.750	0.2266	−0.250	0.4013	0.250	0.5987	0.750	0.7734
−0.740	0.2297	−0.240	0.4052	0.260	0.6026	0.760	0.7764
−0.730	0.2327	−0.230	0.4091	0.270	0.6064	0.770	0.7794
−0.720	0.2358	−0.220	0.4130	0.280	0.6103	0.780	0.7823
−0.710	0.2389	−0.210	0.4169	0.290	0.6141	0.790	0.7853
−0.700	0.2420	−0.200	0.4208	0.300	0.6179	0.800	0.7882
−0.690	0.2451	−0.190	0.4247	0.310	0.6217	0.810	0.7911
−0.680	0.2483	−0.180	0.4286	0.320	0.6255	0.820	0.7939
−0.670	0.2515	−0.170	0.4325	0.330	0.6293	0.830	0.7968
−0.660	0.2547	−0.160	0.4365	0.340	0.6331	0.840	0.7996
−0.650	0.2579	−0.150	0.4404	0.350	0.6368	0.850	0.8024
−0.640	0.2611	−0.140	0.4444	0.360	0.6406	0.860	0.8051
−0.630	0.2644	−0.130	0.4483	0.370	0.6443	0.870	0.8079
−0.620	0.2677	−0.120	0.4523	0.380	0.6480	0.880	0.8106
−0.610	0.2710	−0.110	0.4562	0.390	0.6517	0.890	0.8133
−0.600	0.2743	−0.100	0.4602	0.400	0.6554	0.900	0.8160
−0.590	0.2776	−0.090	0.4642	0.410	0.6591	0.910	0.8186
−0.580	0.2810	−0.080	0.4681	0.420	0.6628	0.920	0.8212
−0.570	0.2844	−0.070	0.4721	0.430	0.6664	0.930	0.8238
−0.560	0.2878	−0.060	0.4761	0.440	0.6700	0.940	0.8264
−0.550	0.2912	−0.050	0.4801	0.450	0.6737	0.950	0.8290
−0.540	0.2946	−0.040	0.4841	0.460	0.6773	0.960	0.8315
−0.530	0.2981	−0.030	0.4881	0.470	0.6808	0.970	0.8340
−0.520	0.3016	−0.020	0.4920	0.480	0.6844	0.980	0.8365
−0.510	0.3051	−0.010	0.4960	0.490	0.6879	0.990	0.8389

TABLE C-1. Continued

Z	F(z)	Z	F(z)	Z	F(z)	Z	F(z)
1.000	0.8414	1.500	0.9332	2.000	0.9773	2.500	0.9938
1.010	0.8438	1.510	0.9345	2.010	0.9778	2.510	0.9940
1.020	0.8462	1.520	0.9357	2.020	0.9783	2.520	0.9941
1.030	0.8485	1.530	0.9370	2.030	0.9788	2.530	0.9943
1.040	0.8509	1.540	0.9382	2.040	0.9793	2.540	0.9945
1.050	0.8532	1.550	0.9394	2.050	0.9798	2.550	0.9946
1.060	0.8554	1.560	0.9406	2.060	0.9803	2.560	0.9948
1.070	0.8577	1.570	0.9418	2.070	0.9808	2.570	0.9949
1.080	0.8599	1.580	0.9429	2.080	0.9812	2.580	0.9951
1.090	0.8622	1.590	0.9441	2.090	0.9817	2.590	0.9952
1.100	0.8643	1.600	0.9452	2.100	0.9821	2.600	0.9953
1.110	0.8665	1.610	0.9463	2.110	0.9826	2.610	0.9955
1.120	0.8687	1.620	0.9474	2.120	0.9830	2.620	0.9956
1.130	0.8708	1.630	0.9484	2.130	0.9834	2.630	0.9957
1.140	0.8729	1.640	0.9495	2.140	0.9838	2.640	0.9959
1.150	0.8749	1.650	0.9505	2.150	0.9842	2.650	0.9960
1.160	0.8770	1.660	0.9515	2.160	0.9846	2.660	0.9961
1.170	0.8790	1.670	0.9525	2.170	0.9850	2.670	0.9962
1.180	0.8810	1.680	0.9535	2.180	0.9854	2.680	0.9963
1.190	0.8830	1.690	0.9545	2.190	0.9857	2.690	0.9964
1.200	0.8849	1.700	0.9554	2.200	0.9861	2.700	0.9965
1.210	0.8869	1.710	0.9564	2.210	0.9865	2.710	0.9966
1.220	0.8888	1.720	0.9573	2.220	0.9868	2.720	0.9967
1.230	0.8907	1.730	0.9582	2.230	0.9871	2.730	0.9968
1.240	0.8925	1.740	0.9591	2.240	0.9875	2.740	0.9969
1.250	0.8944	1.750	0.9599	2.250	0.9878	2.750	0.9970
1.260	0.8962	1.760	0.9608	2.260	0.9881	2.760	0.9971
1.270	0.8980	1.770	0.9616	2.270	0.9884	2.770	0.9972
1.280	0.8997	1.780	0.9625	2.280	0.9887	2.780	0.9973
1.290	0.9015	1.790	0.9633	2.290	0.9890	2.790	0.9974
1.300	0.9032	1.800	0.9641	2.300	0.9893	2.800	0.9974
1.310	0.9049	1.810	0.9648	2.310	0.9896	2.810	0.9975
1.320	0.9066	1.820	0.9656	2.320	0.9898	2.820	0.9976
1.330	0.9082	1.830	0.9664	2.330	0.9901	2.830	0.9977
1.340	0.9099	1.840	0.9671	2.340	0.9904	2.840	0.9977
1.350	0.9115	1.850	0.9678	2.350	0.9906	2.850	0.9978
1.360	0.9131	1.860	0.9686	2.360	0.9909	2.860	0.9979
1.370	0.9147	1.870	0.9693	2.370	0.9911	2.870	0.9979
1.380	0.9162	1.880	0.9699	2.380	0.9914	2.880	0.9980
1.390	0.9177	1.890	0.9706	2.390	0.9916	2.890	0.9981
1.400	0.9192	1.900	0.9713	2.400	0.9918	2.900	0.9981
1.410	0.9207	1.910	0.9719	2.410	0.9920	2.910	0.9982
1.420	0.9222	1.920	0.9726	2.420	0.9922	2.920	0.9982
1.430	0.9236	1.930	0.9732	2.430	0.9925	2.930	0.9983
1.440	0.9251	1.940	0.9738	2.440	0.9927	2.940	0.9984
1.450	0.9265	1.950	0.9744	2.450	0.9929	2.950	0.9984
1.460	0.9279	1.960	0.9750	2.460	0.9931	2.960	0.9985
1.470	0.9292	1.970	0.9756	2.470	0.9932	2.970	0.9985
1.480	0.9306	1.980	0.9762	2.480	0.9934	2.980	0.9986
1.490	0.9319	1.990	0.9767	2.490	0.9936	2.990	0.9986

TABLE C-1. Continued

Z	F(z)	Z	F(z)	Z	F(z)	Z	F(z)
3.000	0.9986	3.250	0.9994	3.500	0.9998	3.750	0.9999
3.010	0.9987	3.260	0.9994	3.510	0.9998	3.760	0.9999
3.020	0.9987	3.270	0.9995	3.520	0.9998	3.770	0.9999
3.030	0.9988	3.280	0.9995	3.530	0.9998	3.780	0.9999
3.040	0.9988	3.290	0.9995	3.540	0.9998	3.790	0.9999
3.050	0.9988	3.300	0.9995	3.550	0.9998	3.800	0.9999
3.060	0.9989	3.310	0.9995	3.560	0.9998	3.810	0.9999
3.070	0.9989	3.320	0.9995	3.570	0.9998	3.820	0.9999
3.080	0.9990	3.330	0.9996	3.580	0.9998	3.830	0.9999
3.090	0.9990	3.340	0.9996	3.590	0.9998	3.840	0.9999
3.100	0.9990	3.350	0.9996	3.600	0.9998	3.850	0.9999
3.110	0.9991	3.360	0.9996	3.610	0.9998	3.860	0.9999
3.120	0.9991	3.370	0.9996	3.620	0.9998	3.870	0.9999
3.130	0.9991	3.380	0.9996	3.630	0.9999	3.880	0.9999
3.140	0.9991	3.390	0.9996	3.640	0.9999	3.890	0.9999
3.150	0.9992	3.400	0.9997	3.650	0.9999	3.900	0.9999
3.160	0.9992	3.410	0.9997	3.660	0.9999	3.910	0.9999
3.170	0.9992	3.420	0.9997	3.670	0.9999	3.920	1.0000
3.180	0.9993	3.430	0.9997	3.680	0.9999	3.930	1.0000
3.190	0.9993	3.440	0.9997	3.690	0.9999	3.940	1.0000
3.200	0.9993	3.450	0.9997	3.700	0.9999	3.950	1.0000
3.210	0.9993	3.460	0.9997	3.710	0.9999	3.960	1.0000
3.220	0.9994	3.470	0.9997	3.720	0.9999-	3.970	1.0000
3.230	0.9994	3.480	0.9997	3.730	0.9999	3.980	1.0000
3.240	0.9994	3.490	0.9998	3.740	0.9999	3.990	1.0000
						4.000	1.0000

TABLE C-1. Continued

Sample size N	Kolmogorov-Smirnov level of significance α				
	0.20	0.15	0.10	0.05	0.01
3	0.451	0.479	0.511	0.551	0.600
4	0.396	0.422	0.449	0.487	0.548
5	0.359	0.382	0.406	0.442	0.504
6	0.331	0.351	0.375	0.408	0.470
7	0.309	0.327	0.350	0.382	0.442
8	0.291	0.308	0.329	0.360	0.419
9	0.277	0.291	0.311	0.341	0.399
10	0.263	0.277	0.295	0.325	0.380
11	0.251	0.264	0.283	0.311	0.365
12	0.241	0.254	0.271	0.298	0.351
13	0.232	0.245	0.261	0.287	0.338
14	0.224	0.237	0.252	0.277	0.326
15	0.217	0.229	0.224	0.269	0.315
16	0.211	0.222	0.236	0.261	0.306
17	0.204	0.215	0.229	0.253	0.297
18	0.199	0.210	0.223	0.246	0.289
19	0.193	0.204	0.218	0.239	0.283
20	0.188	0.199	0.212	0.234	0.278
25	0.170	0.180	0.191	0.210	0.247
30	0.155	0.164	0.174	0.192	0.226
> 30	$\frac{0.86}{\sqrt{N}}$	$\frac{0.91}{\sqrt{N}}$	$\frac{0.96}{\sqrt{N}}$	$\frac{1.06}{\sqrt{N}}$	$\frac{1.25}{\sqrt{N}}$

TABLE C-2. Critical Values for Kolmogorov-Smirnov Test for the Exponential Distribution