1983

Distributed computation in computer networks

Ching-Liang You

Rocheste: Institute of Technology
School of Computer Science and Technology

DISTRIBUTED COMPUTATION IN COMPUTER NETWORKS

by

CHING-LIANG YOU

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

May 4, 1983

Approved by: John L. Ellis May 11, 1983

Thesis Advisor John L. Ellis Date

Lawrence Coon May 11, 1983

Committee Member Lawrence Coon Date

Michael J. Lutz May 11, 1983

Committee Member Michael J. Lutz Date

I grant permission to the Wallace Memorial Library, of R.I.T., to reproduce my thesis in whole or in part. Any reproduction will not be for commerical use or profit.

Ching-Liang You

May 11, 1983

Date

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

IIST CF TABIES

ACKNOWIEDGEMENTS

I would like to thank my thesis advisor, Dr. John Ellis, for the endless hours he spent directing my efforts on this thesis. I would also like tc thank Dr. Iawrence Ccon for the ideas for this thesis. I wish to thank Mr. Michael Iutz for the time he has spent as a member of my committee.

CHAPTER 1 INTRODUCTION

1.1 Objective

The purpose of this thesis is to investigate distributed computation by using the following different models: the hierarchical model, the CPU cache model, the user-server model, the pool processor model and the data flow model. This research also presents functions, characteristics and applications of distributed computation by describing various current implementation of these models.

1.2 Definition of distributed system

Enslow (1978) defines a distributed system as a system-wide operating system with services requested by name, rather than by location. In other words, the user of a distributed system should not be aware that there are multiple processors; it should look like a virtual uniprocessor. Allocation of jobs to processors, processors scheduling, allocation of files to disks, movement of files between where they are stored and where there are needed, and all other system function must be automatic.[1]

1.3 Definition of distributed computation

---

[1] Enslow, P.H., Jr., "What is a distributed data processing system?", Computer, vol 11, pp.13-21, Jan. 1978.

A distributed computation is a computation partitioned into pieces and implemented on different physically autonomous machines in the computer communication network. The computation may be user problems, tasks, procedures, programs, etc.

1.4 Origin of distributed computation

The traditional method for solving large, complex problems on a computer has been to partition the problem into many smaller components, each of which is capable of being processed by an individual programmer. Once the components of the solution have been developed, we typically merge all elements of the solution into one large program on a single machine and collect all data to be processed in data bases on the same machine. As the problems to be solved grow increasingly large and complex, our traditional approaches seem to be less and less effective. The reason is that we use a single computer to solve the entire problem and do not allow the variations in hardware and software architectures that make it easier to process various elements of a problem in different machine environments that are available in a typical large system. We require that all data be physically located on one system, even though for many decentralized organizations, much data could be collected and stored locally without any adverse affect on the operation of central facilities.

Distributed computing systems provide an attractive alternative for the implementation of complex systems. In a distributed network, many autonomous machines cooperate to solve a large problem with each machine manipulating only a small manageable piece of the whole. The general properties of such distributed networks are:

. Programs in different machines can exchange data in real time, on a cooperative basis.

. Resources such as peripheral devices, files, and data bases can be shared across system boundaries.

. A program on one system can activate programs on other systems so that multi-tasking can be effected across system boundaries.

In order to understand distributed computation, we now introduce five different models to describe it. These models differ in the following ways:

(1) Different network structures - We cover from loosely coupled long-haul networks, such as the hierarchical model and CPU cache model, to tightly coupled data flow machines, such as the data flow model.

(2) Different methods of distributed computation - According to different network structures, we have different methods of distributed computation. We deal with the problem of how to achieve distributed computation for each network structure. More detailed discussion will be given in Chapter 2.

1.5 Introduction of five models

1.5.1. The hierarchical model

The hierarchical model is commonly used in a hierarchical organized distributed computer system. It has a tree structure shown in Figure 1-1, with increasingly powerful computers as one progresses from the leaves of the tree to the root. This model has one unique characteristic, that is, the machine of the root always coordinates all computation. Each level performs its function by using a computer sufficiently powerful to handle the various computations required at that level. For example, microcomputers in a supermarket can control equipment and collect data, sending them to minicomputer in a factory for analyzing and storing data for local queries. Finally, each minicomputer sends local information to a large mainframe at the headquarters.

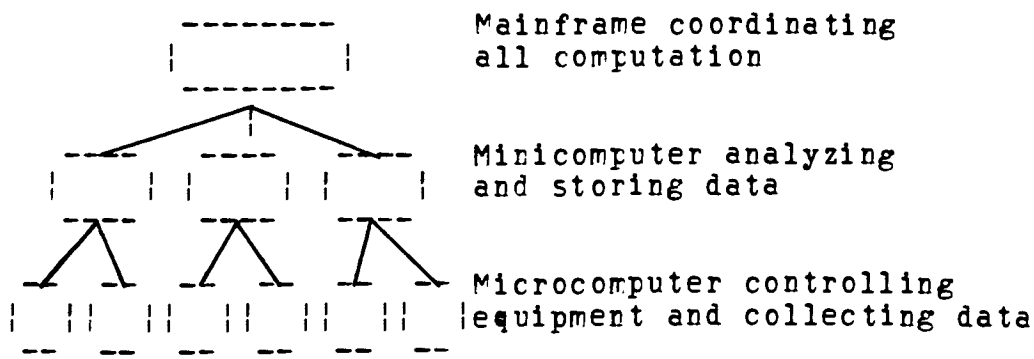


Fig. 1-1 A hierarchical model

1.5.2. The CPU cache model

The CPU cache model shown in Figure 1-2 has two levels of computers, each user has a minicomputer connected between his terminal and the large central computer. Part of the computation is done on the central computer and part or the minicomputer depending on the current workload of either computer. How to allocate the tasks to each processor to find the minimum total running time and interprocessor communication cost is the most important question with this model.

The CPU cache model, also called the task allocation model, allocates application tasks among processors in

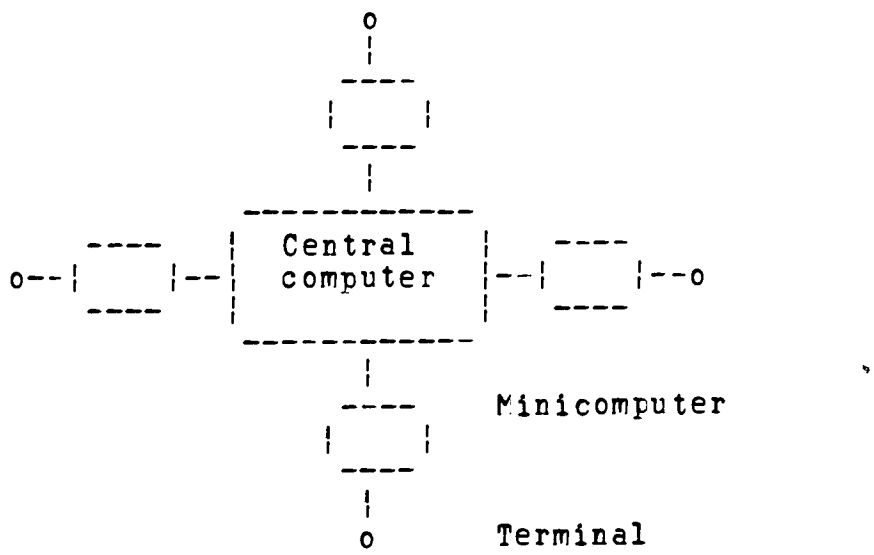


Fig. 1-2 The CPU cache model

distributed computing systems satisfying : (1) minimum interprocessor communication cost, (2) balanced utilization of each processors, and (3) all engineering application requirements. Several approaches to the task allocation problem have been identified. They are basically graph theoretical[2] , integer programming[3] [4], and heuristic methods[5].

1.5.3 The user-server model

As the power of the individual user's personal computers increases and their price decreases, the user-server model which does not have a central computer becomes feasible. The personal computers of all the users are connected by a high speed local network to allow users to share data, send mail to each other and share expensive peripherals. In this model, shown in Figure 1-3, most of the computation will be done on the individual user's personal computer. Specialized computations will be carried out by various server machines. Possible services to be

---

[2]Stone, H.S., "Multiprocessor scheduling with the aid of network flow algorithm", IEEE Trans. Software Eng., vol. SE-3, pp.85-93, Jan. 1977.

[3]El-Dessouki, O.I. and Huan, W.H., "Distributed enumeration on network computers", IEEE Trans.Comput., vol.C-29, pp818-825, Sept, 1980.

[4]Chu, W.W., "Optimal file allocation in a multiple computing system", IEEE Trans.Comput.,vol.C-18, pp885-889, Oct. 1969.

[5]Gylys, V.B. and Edwards, J.A., "Optimal partitioning of workload for distributed systems", in Dig. COMPCON Fall 1976, pp.353-357.

provided include : time of day provided by time server; allocating, return, reading and writing a disk block provided by disk servers; file storage and retrieval provided by file servers and data base managements provided by data base servers.[6], [7]

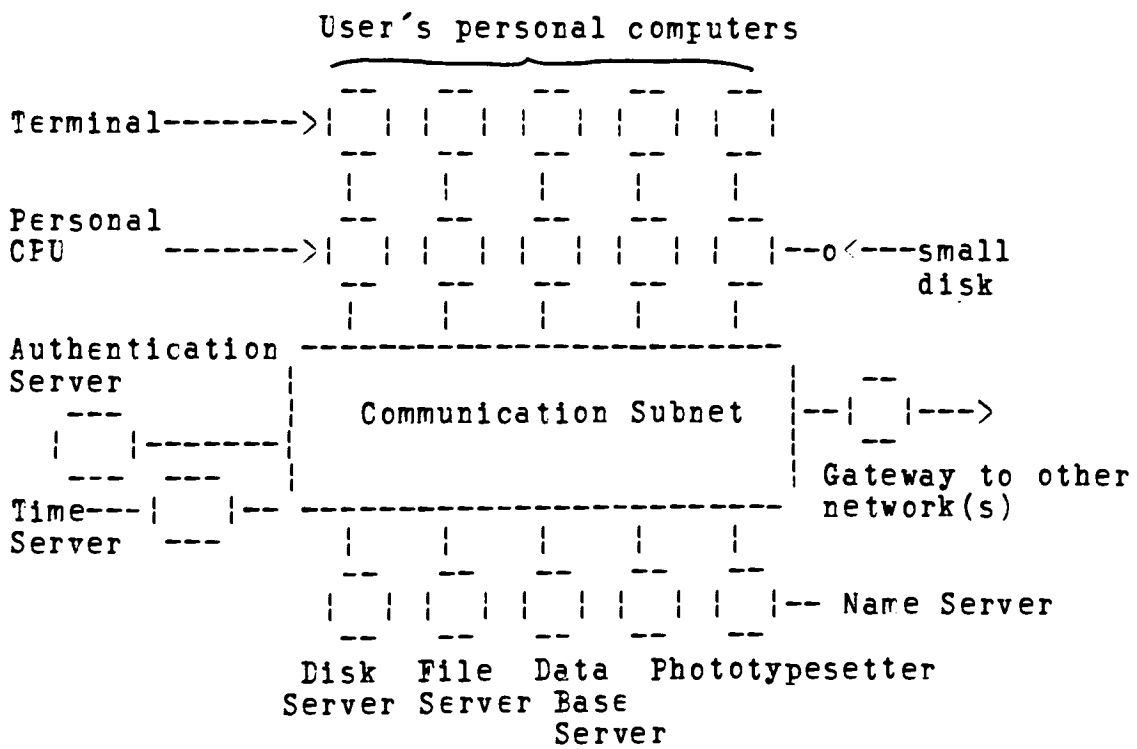


Fig. 1-3 The user-server model

---

[6] Maryanski, F.J., "Backend database systems", Comput. Surv., vol.12, pp.3-25, March 1980.

[7] Su, S.Y.W., Chang, H., Copeland, G., Fisher, P., Lowenthal, E., and Schuster, S., "Database machines and some issues on DBMS standards", Proc. NCC, pp.191-208, 1980.

1.5.4. The pool processor model

Sometimes a user may need more computing power than his machine can provide. In the pool processor model shown in Figure 1-4, users do not have powerful personal computers. Work is carried out by a pool of processors, some of which may have fixed functions, such as file server, and some of which are dynamically allocatable on demand. Wittie[8] has envisioned a system of this type containing 10,000 processors.

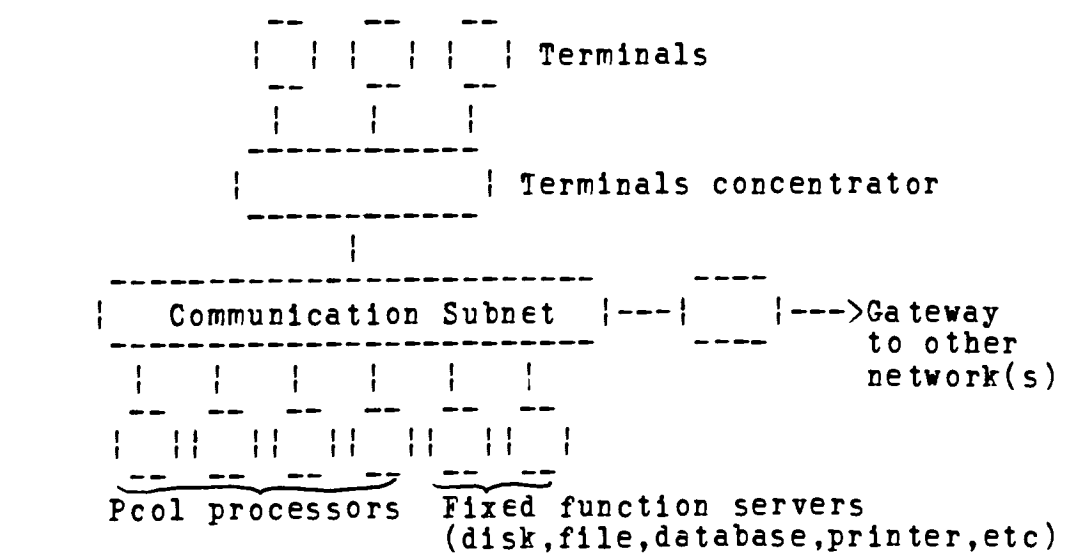


Fig. 1-4 The pool processor model

---

[8] Wittie, L.D., "A distributed operating system for a reconfigurable network computer", Proc. First Int.Conf. on Distrib.Comput.Syst., IEEE, pp.669-677, 1979.

1.5.5. The data flow model

The data flow model is another alternative model of computation which is particularly promising. The basic principles of data flow are asynchrony and functionality, and thus are in distinct contrast to the von Neumann model.[9] In a data flow computer, an instruction is ready for execution when its operands have arrived - there is no concept of 'control flow', and data flow computers do not have program location counters. Many instructions of data flow program may be available for execution at once. Thus highly concurrent computation is a natural characteristic of the data flow model.

1.6 Contrast between five models

Each of those five models differ in their structure and the methods used to carry out the computations to be performed on them. Let us give an overview contrasting and comparing these five different models by looking at their structures and method of computation.

First let us look at the structure of the different models.

(1) The hierarchical model is a tree structured model where computers of the top level are more powerful than

---

[9] Gostelow, K.P. and Thomas, R.E., "A View of Data Flow", AFIPS Conf. Proc. NCC, pp629-636, 1979.

those at the lower level.

(2) The CPU cache model has two kinds of computers: fast access time and slow access time computers. Whenever the fast computer has light workload, modules will be run there because of the better response time.

(3) The user-server model does not have a large central computer. Instead there are many user's personal computers with local disk storage as well as several server machines that can carry out specialized functions whenever requested by users.

(4) In the pool processor model, users do not have powerful personal computers. Each terminal is connected to the network, either directly or through a terminal concentrator. Several processors are connected together in a so called pool of processors to handle the computation.

(5) Figure 1-5 shows a data flow system. The templates are stored in the program memory. (Note: Template contains enough information associated with each processor. Information include opcode, operand(s) and destination(s).) As soon as a template is ready to fire, the fetch unit can extract the opcode, operands, and destination addresses, make a packet from them, and send the packet to one of the processors. The output packets generated by the processors are stored in the appropriate templates by the store unit.

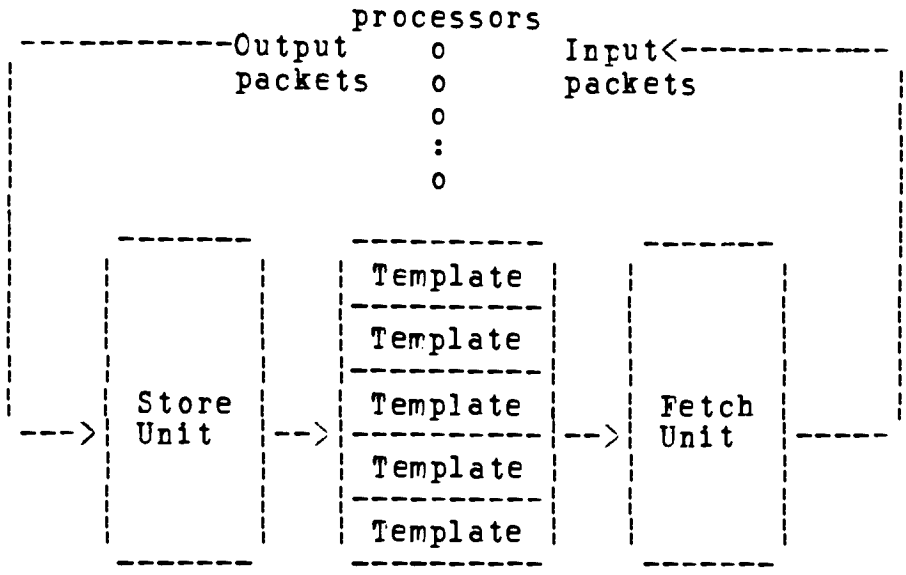


Fig. 1-5 A data flow machine

Finally, we will examine the way computations are carried out by each of these five models.

(1) In the hierarchical model, each level of computers carrys out different computations.

(2) In the CPU cache model, we have to consider how to assign modules to computers in order to decrease the total running time and intermodule communication cost and to balance the total workload.

(3) In the user-server model, most of the computations are done in user's personal computer, although part of the computations are done in various specialized server machines which can be requested by users.

(4) In the pool processor model, the computations are

done by a pool of processors, some of which may have fixed functions, such as file servers, and some of which are dynamically allocatable on demand. The difference between the pool processor model and the user-server model is that resources are allocated dynamically rather than statically resulting in faster response time.

(5) The data flow concept is a different way of looking at instruction execution in machine level programs. In a data flow computer, an instruction is ready for execution when its operands have arrived. Hence many instructions of a data flow program may be executed at once. Highly concurrent and parallel computation are important characteristics of data flow model.

1.7 Outline of the rest of the thesis

In Chapter 2 we give the detailed descriptions of the five different models of distributed computation. In particular, the origins, structures, characteristics, functions, and distributed computing methods for each model.

In Chapter 3 we present several examples which have been implemented or in the progress to determine why they are using distributed computations and which model they fit into. Such examples are

(1) Task allocation system
    - distributed computing system

(2) Central file system

- distributed file system

(3) SWALLOW

- distributed data storage system

(4) WORM

- distributed computation

(5) LOCUS

- distributed operating system

(6) Backend storage networks

(7) File servers for network-based distributed systems

CHAPTER 2 DIFFERENT MODELS OF DISTRIBUTED COMPUTATION

2.1 Intention

In this chapter, we will investigate five possible models for describing distributed computations. In each of the next five sections one model will be described, its functions and characteristics outlined, and one or more applications for that model introduced.

2.2 The Hierarchical Model

2.2.1 Introduction

In some networks the computers are programmed to cooperate with one another to solve a common set of problems. This is often the case in a hierarchical system. The lower-level machines are programmed to pass work to the higher-level machines.

In hierarchical model, functions are distributed. The lower-level machines may be intelligent terminals or intelligent controllers for such operations as message editing, screen formatting, data collection dialogue with terminal operators, security,and message compaction or concentration. They do not complete the processing of an entire set of transactions.

The transaction must enter and leave the computer system at the lowest level. The lowest level may be able to prccess the transaction or may execute certain

functions and pass it up to the next level. Some, or all, transactions may eventually reach the highest level, which will probably have access to on-line files or some data base.

The machine at the top of a hierarchy might be a computer system in its own right, performing its own type of processing on its own transactions and the data passed from lower-level system. The machine at the top might be a head-office system which receives data from factory, branch, warehouse, and other systems.

2.2.2 Applications

One example of an application where a hierarchical model would apply is a distributed computing system for an insurance company.

The branchs of an insurance company each have their own processor with a printer and terminals. This processor handles most of the computing requirements of the branch. Details of the insurance contracts made are sent to a head office computer for risk analysis and actuarial calculations. The head-office management has up-to-the-minute information on the company's financial position and exposure, and can adjust the quotations given by the salesman accordingly.

Similar example is the computer system for a chain store.

Each store in a chain has a minicomputer which records sales and handles inventory control and accounts receivable. It prints sales receipts for customers at the time of sale. Salesmen and office personnel can use the terminals to display pricing, inventory and accounts receivable information, and customer statements. The store management can display salesman performance information and goods aging and other analysis reports (Figure 2-1). The store systems transmit inventory and sales information to the head office system and receive inventory change information each night. The fast receipt of inventory and sales information enables the head office system to keep the inventory of the entire organization to a minimum. The store systems run unattended with any program changes transmitted to the system from the head office computer.

A more complex example is the overall computer system for a large corporation. As shown in Figure 2-2, such a system must have multiple levels, with a diversity of computers at each level.

The lowest level consists of intelligent terminals for data entry, or microprocessors in a factory that scan instruments. The next level is a computer in a sales region assembling and storing data that relates to that

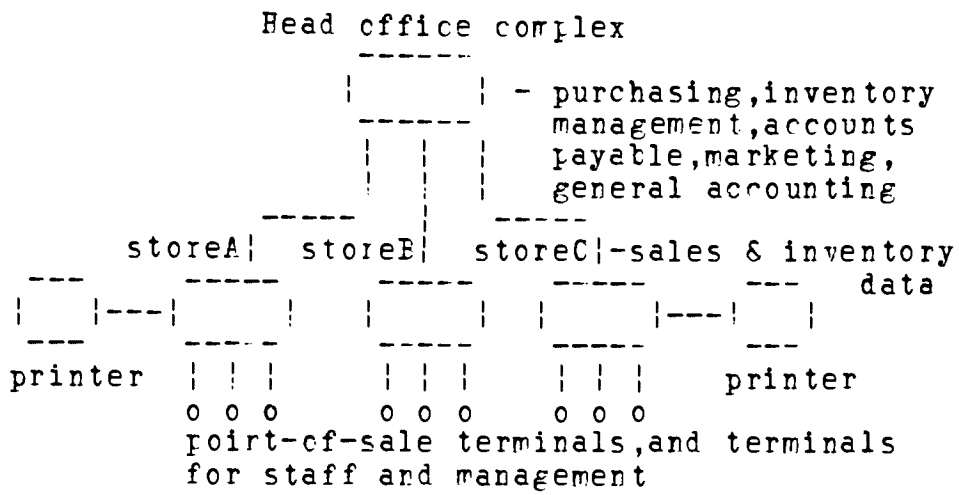
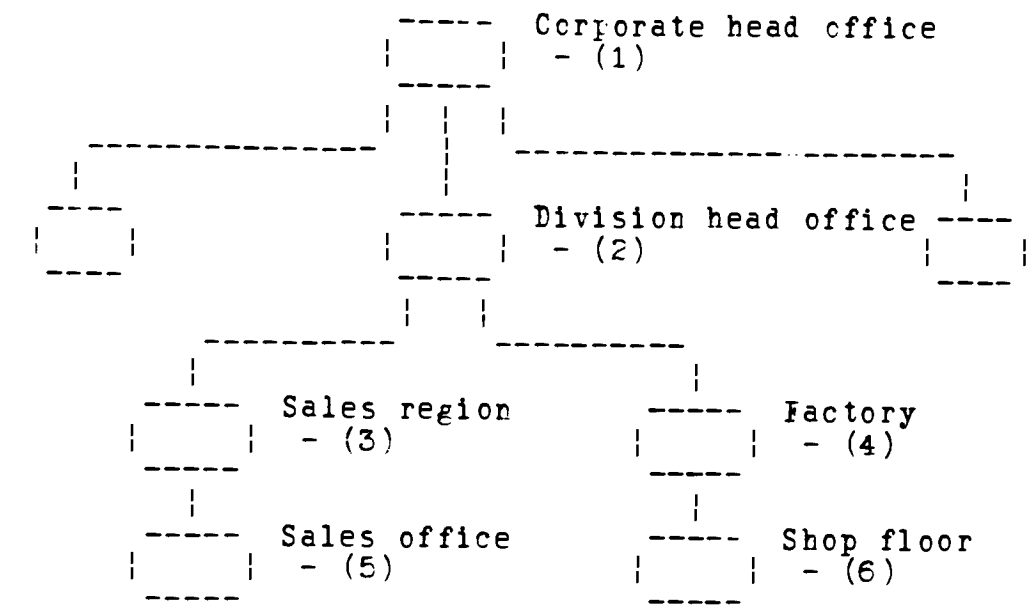


Fig. 2-1 A hierarchical configuration in a chain store



(1) Management information system
Financial modeling
(2) Sales summary and analysis
Production planning
General accounting
Routine data processing
(3) Data base of orders and customers
Accounts receivable
Salesman records
(4) Process control planning
Process data gathering and analysis
Investigation of problems
(5) Intelligent terminals for sales order entry and administrative operations
(6) Microprocessors for scanning instruments process monitoring and control

Fig. 2-2 Multiple levels

region, or a computer in a factory assembling the data from the microprocessors and being used for production planning. The third level is a conventional large computer system in the divisional head office, performing many types of data processing and maintaining large data

bases for routine operations. This computer center receives data from the lower systems and sends instructions to them. The highest level is a corporate management information system, with data structured differently from that in the systems used for routine operations. This system may be designed to assist various types of high-management decision making. It may run complex corporate financial models or elaborate programs to assist in optimizing certain corporate operations. It receives summary data from lower systems.

2.2.3 Characteristics

The following characteristics are generally found when comparing this model with a system comprised of a centralized computer.

(a) Cost - Total system cost may be lower than that of a system with a single large computer because of less data transmission is needed and many functions are moved to inexpensive lower level machines.

(b) Capacity - The central host may not be able to handle the workload without distribution. Many functions can be performed in parallel in distributed systems.

(c) Response Time - Local responses to critical function can be fast; no telecommunications delay; no scheduling problem; instruments are scanned and controlled by a local device.

(d) User Interface - A better terminal dialogue can be used when the user interacts with a local machine; also better graphics or screen design, and faster response time.

(e) Simplicity - Separation of the peripheral functions can give a simpler, more modular system design.

2.3 The CPU Cache Model

2.3.1 Introduction

Many problems are not structured hierarchically. Besides, if the workload on one computer exceeds the CPU capacity of that computer, a cache of CPUs can be easily accessed to distribute the load. One example of the CPU cache model is shown in Figure 1-2. Each user has a minicomputer between his terminal and the large central computer, part of the computation is done on the central computer and part on the mini.

The decision to run a particular part of the computation on one machine or the other is based on the (a) suitability of the two machines, (b) the relative costs of the two machines, (c) the bandwidth between the machines, (d) the current workload. Let us examine a system in which all the modules are available to both the central computer and one or more of the minicomputers. When the workload on the central computer is light, most modules will run faster there. When the central computer is

heavily loaded, the mini may actually have a better response time. A major question in this model is to determine which parts of the program should be run where.

2.3.2 Using Graph Theory to Assign Modules

Stone and Bokhari (1978) used graph theory to assign modules to machines.[10] There are two costs that must be considered in finding a good assignment:

(1) Each module has a running cost that depends on the processor to which it is assigned. Let RUNx (i) be the cost for running module i in assignment x.

(2) Each pair of modules have a communication cost if they have to communicate over a link because they are assigned to different computers. Let the communication cost be COMMx(i,j) for the module pair i,j in assignment x, and this cost is zero if modules i and j are placed on the same processor by assignment x. Hence we want to find an assignment x such that x minimizes $\sum_i$ RUNx(i) $+\sum_i \sum_{j \neq i}$ COMMx(i,j).

2.3.3 The Result for Two-Processor Systems

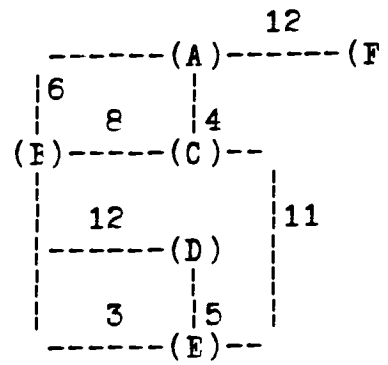
A network flow algorithm[11] may be used to find the optimal assignment of modules to processors in a two-

---

[10]Stone, H.S., and Bokhari, S.H., "Control of Distributed Processes", Computer, vol 11, pp97-106, July, 1978.
[11]Tanenbaum, A.S., "Computer Networks", Prentice-Hall, pp.40-49, 1981.

processor system. A program graph and a table of execute times for program modules in a two-processor distributed system is shown in Figure 2-3&4. The nodes represent modules, and the links represent intermodule communication patterns. The numbers on the edges, called edge weights, represent the cost of communicatcn between modules when the modules are not coresident on the same computer. Intermodule communication costs are assumed to be zero when the pair of module are coresident. Both costs are given in units of time or dollars.

The table in Figure 2-3&4 shows the execution costs of the program module when run on computer P1 or computer P2. An infinite cost indicates that a module cannot be executed on that computer. For any given module assignment, the cost of the assignment is the sum of the execu-



MODULE	P1 TIME	P2 TIME
A	5	10
B	2	INFINITY
C	4	4
D	6	3
E	5	2
F	INFINITY	4

Fig. 2-3&4 Execution and communication costs for a distributed program

tion costs plus the sum of the intermodule communication costs for those modules that are not coresident. The latter can be determined by summing the weights of the edges which connect non-coresident modules.

Tc find the optimal allocation, we have to create a modified graph as shown in Figure 2-5. This graph has two additional nodes, one for computer P1 and one for computer P2. Each original module has an edge to each of the computer nodes. The weights of the new edges are such that the edge to the P1 node carries the P2 execution cost and the edge to the P2 node carries the P1 execution cost. A minimal cut in this new graph, shown by the dark line is a collection of edges such that when removed from the graph, cause node P1 to be disconnected from node P2.

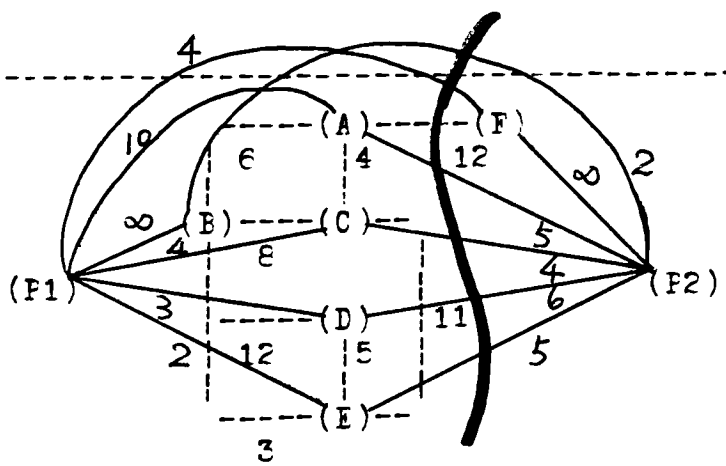


Fig. 2-5 Modified graph with execution and communication costs in one graph

Each minimal cut corresponds to a module assignment, and every assignment corresponds to a minimal cut. The weight of a minimal cut is the sum of the weights of the branches in a minimal cut. In Figure 2-5, the weight of the minimal cut is equal to the cost of the corresponding module assignment since the weight of a minimal cut sums up the execution and communication costs for that assignment. Note that if module A is assigned to P1 then the edge to P2 is cut, but this edge carries the cost of executing on P1. Similarly, other edges cut between module A and other nodes of the graph represent actual communication costs. The optimal assignment corresponds to a minimum weight cut, that is a minimum cut.

2.3.4 The Dynamic Assignment Problem

A module assigned to a particular processor is assumed to stay on that processor for the lifetime of that assignment. This is called the optimal static assignment. The objective of the dynamic assignment is to relocate modules during the execution of the program to take advantage of changes in the locality of the program. [12]

The basis for our mathematical model of the dynamic assignment problem is the concept of the phase of a modu-

---

[12] Bokhari, S.H., "Dual-Processor Scheduling with Dynamic Reassignment", IEEE Trans. on Software Engineering, vol. SE-5, No 4, pp.338-349, July 1979.

lar program. The phase of a modular program is a contiguous period of time during which only one module executes. During this period the executing module may communicate with any number of the remaining modules. A module may not be moved from one processor to another during a phase, it may be moved only between phases.

For each phase we need the following information:

1. Which module executes during this phase.

2. Run costs of this module for either of the two processors.

3. Costs of residence of the remaining modules for each of the two processors.

4. Intermodule communication costs between the executing module and all other modules assigned to different processors.

5. Relocation cost for each module - the cost of relocating each module from one processor to the other if relocation were to be carried out at the end of this phase.

In Table 1, a program consists of three modules (A,B,C) and five phases. During phase 1 module A executes. The cost of running this module on processor 1 is 4, on processor 2 is 10. The residence costs of the nonexecuting modules (B,C) are also in Table 1. The cost of communication between module A and B is 3, A and C is 2. Costs for

relocating modules A, B, C at the end of this phase are 2, 1, 3 respectively.

This information can be represented by the graph in Figure 2-6. The number of program modules (in this case, 3) multiplied by the number of phases (in this case, 5) equals the number of nodes in this graph. The vertical

PHASE	MODULE	EXECUTION COST		RESIDENT COST	
		P1	P2	P1	P2
1	A	4	10	-	-
	B	-	-	1	3
	C	-	-	3	1
2	A	-	-	1	2
	B	INFINITY	12	-	-
	C	-	-	3	1
3	A	4	17	-	-
	B	-	-	1	2
	C	-	-	4	2
4	A	-	-	1	1
	B	-	-	4	2
	C	6	4	-	-
5	A	-	-	3	1
	B	-	-	3	2
	C	9	3	-	-

PHASE	COST OF COMMUNICATION BETWEEN EXECUTING MODULE			RELOCATION COST OF		
	A	B	C	A	B	C
1	-	3	2	2	1	3
2	4	-	3	2	3	2
3	-	7	0	3	2	3
4	0	4	-	2	1	4
5	4	3	-	-	-	-

Table 1. Tabulation of costs for a dynamic assignment problem with three modules

lines of nodes represent the modules and the horizontal ones represent the phases. Each node represents the residence of a module during a specific phase. The asterisk stands for the single module that executes during each phase. The weight of the edge which connects successive residences of the same module represents the cost of relocating that module. The weight of the edge which connects the executing module with other modules during the same phase represents intermodule communication costs between the executing module and the other modules.

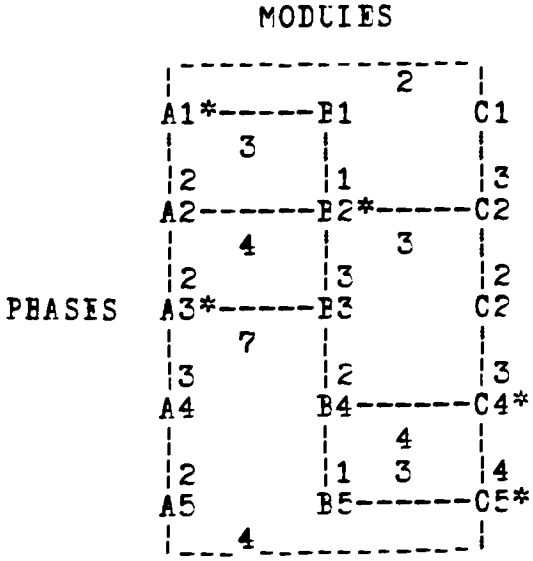


Fig. 2-6 Incomplete dynamic assignment graph, for 3 modules and 5 phases. The horizontal edges represent communication costs, and the vertical edges represent relocation cost. Asterisk denotes executing module.

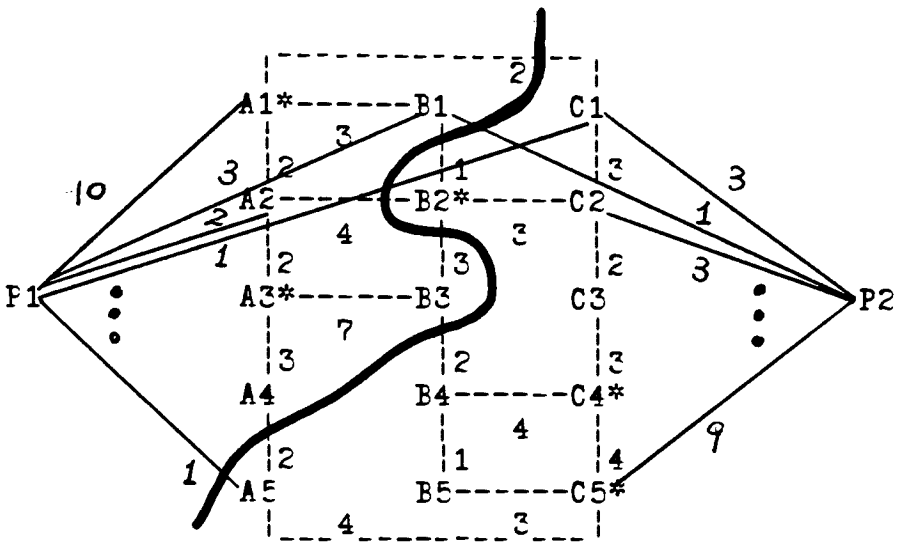


Fig. 2-7 A minimum weight cut
in the dynamic assignment graph

In Figure 2-7, edges representing the run costs are drawn from P1 and P2 to each of the nodes representing executing modules, and edges representing residence costs are drawn from P1 and P2 to the remaining nodes. This graph (Figure 2-7) is called a dynamic assignment graph. A minimum cut gives a dynamic assignment of modules. It specifies which modules are to reside on which processor during each phase. The weight of a minimum cut is the total cost of the optimal dynamic assignment. This minimum cut may be found by running a network flow algorithm between nodes P1 and P2 in the dynamic assignment graph.

Stone and Bokhari(1978) also presented how the module assignments to three processors by using an extension of

the network flow approach described in Sec. 2.3.3. They believed the assignment problem for four or more processors should be very hard problems for which an efficient solution may not even exist. It is still an open question for future research.

2.4. The User-Server Model

2.4.1 Introduction

As the power of small computers increases and their price decreases, the value of the central computer in the above model also decreases. At some point the most cost effective thing to do is to discard the central computer entirely. Each user has a personal minicomputer with local disk storage. The personal computer may range from a 16-bit machine with 64 kilobytes of memory and a floppy disk to a 32-bit machine with several megabytes of memory and a 100 megabyte hard disk.

The personal computers are connected by a high speed local network in order to allow users to share data and send mail, and to share expensive peripherals. An example of the resulting user-server model is shown in Figure 1-3. In this model the personal computer can execute most of the user's actual computing, but any user can request various existing centrally located server machines to carry out specific functions.

In a computer network, hosts have resources that can be accessed remotely by other hosts. Some resources are processes, files, mailboxes, terminals, and unique peripheral devices. We call these resources, the generic service. A generic service performs some set of actions for its customers. Some example of services are : time of day, pascal compilation, file storage and retrieval, and data base management. Associated with each service is a process called a server, whose task is to provide the service to any authorized process that requests it.

2.4.2 Disk Server and File Server

The function of a disk server is to read and write raw disk blocks, without regard to their contents. Typical messages that could be sent to a disk server are requests to allocate, return, read and write a disk block.

A file server provides long-term reliable storage of data for the individual workstations and supports data sharing among workstations. The file servers are responsible for maintaining directories and connection status information to turn the primitive disk service into usable file service. Typical messages that can be sent to a file server are requests to open a file, close a file and read, write, or seek on a file.

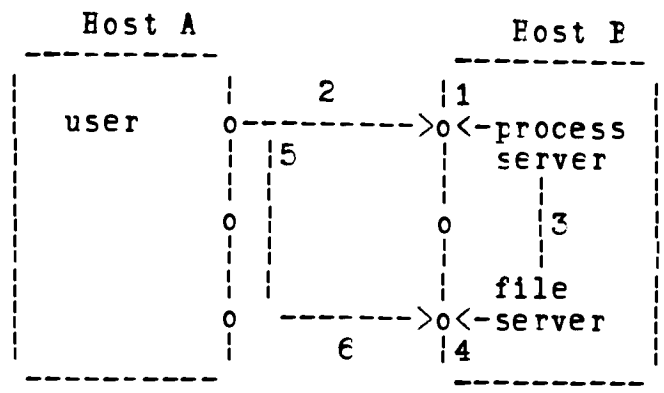
2.4.3 Name Server

In order to implement services each server process must listen to some transport address. To use a service, a user must connect to a remote process by specifying the remote transport address. We use a "name" that indicates which service the user wants and an "address" that specifies which transport address the server is listening to. The mapping which must be done between the name and the address can be accomplished by a name server, like telephone directory server. In some cases servers may listen to well-known address, which are widely known and rarely changed.

Users normally access generic services by name, not by address. The local host must use the name to find the transport address of a process willing to do the work. Once the address of such a process is known, contact can be established. However, a difficulty arises when a host potentially offers a very large variety of services. For example, if a time-sharing system has a thousand programs available, it is not feasible to have each one perpetually listening on a well-known address on the off chance that some remote user wants to use it. Furthermore, multiple instances of many programs would be needed, in case several remote users wanted to run the same program at the same time.

In the early days of the ARPANET this problem arose and was solved by the ARPANET initial connection protocol. We now introduce an equivalent, but simpler mechanism. Instead of every conceivable server listening at a well-known address, each host that wishes to offer service to remote users has special process servers through which all services must be requested. Whenever the process server is idle, it listens to a well-known address. Potential users of any service must specify the address of the process server. Once the connection has been set up, the user sends the processor server a message telling which program it wants to run. The process server then chooses an idle address and spawns a new process, telling the new process to listen to the chosen address. Finally, the process server sends the remote user the chosen address, terminates the connection, and goes back to listening to its well-known address.

At this point the new process is listening on an address that the user now knowns, so it is possible for the user to close the connection to the process server and connect to the new process. Once this connection has been set up, the new process executes the desired program, the name of which was passed to it by the process server, together with the address to listen to. The entire protocol is shown in Figure 2-8.



1. Process server listens to a well known address.
2. User connects to the well known address.
3. Process server creates a file server and tells it where to listen.
4. File server listens.
5. Process server tells user where to connect, and closes the connection.
6. User connects to the file server.

Fig. 2-8 How a user in host A connects to a file server in host B

2.4.4. SPICE

Recent advances in hardware technology are opening up new possibilities for scientific computing. Today the time-shared machine and a CRT provides the user with his own powerful machine, far more powerful than microprocessors and equipped with such features as high-resolution color graphics and audio I/O devices. This development will avoid compromises and limitations inherent in time-sharing which user has encountered. New high-speed network technologies makes it possible to move to this personal computing environment without foregoing the attrac-

tive features of time-sharing such as shared information, good inter-user communication and expensive peripherals.

An example of such a system which fits into the user-server model is SPICE (Scientific Personal Integrated Computing Environment), which has been proposed by the Computer Science Department of Carnegie-Mellon University. SPICE would provide facilities for computer science research in a diversity of areas and a number of office automation services : report preparation, personal communication, time and record management, etc. Some essential components of SPICE as follows:

1. A set of personal computers with the following features:

speed:	10**6 macro-instruction/second.
control store:	at least 16k micro-instruction, reasonably structured and user writable.
virtual address:	2**30 to 2**32 bytes address space smoothly addressable.
primary memory:	1 Mbyte.
secondary storage:	100 Mbytes, suitable for paging.
display:	color, bit-mapped, high-resolution (1kx1k pixels); with keyboard and pointing device.
other:	audio input and output.
cost:	around $10,000 by the mid-1980's.

2. A 10 megabit/second high-bandwidth network

The personal SPICE computers must be linked together with high-performance communication. The major communications are likely to be :

(a) Personal communication among researchers using electronic multi-media mail.

(b) Access to services centralized for economy - file storage, printing and magnetic tape.

(c) Interprocess communication - multi-processor applications.

There are three reasons for supporting these communication needs with a high-bandwidth design as indicated above.

(a) High total capacity is required so that the share available to each pair of communicating processes is adequate.

(b) Transparent access to network services requires high speeds; if a file is to be accessed on a remote file system as quickly as on local storage, the network transmission bandwidth must be comparable with disk transfer bandwidths.

(c) The media-intensive design of SPICE (voice, video) requires larger communication capacity than for most computer communication networks.

3. A shared file system - file server

A shared file system is required on the network so that all user can share information. This file system may run on one central file computer or it may be distributed around the network, but it will appear to users as a single unified facility. A large library of reference material will be available on-line using video disk or other comparable technology.

4. Shared expensive peripherals

To share hard-copy printers, phototypesetter, etc.

5. SPICE software environment

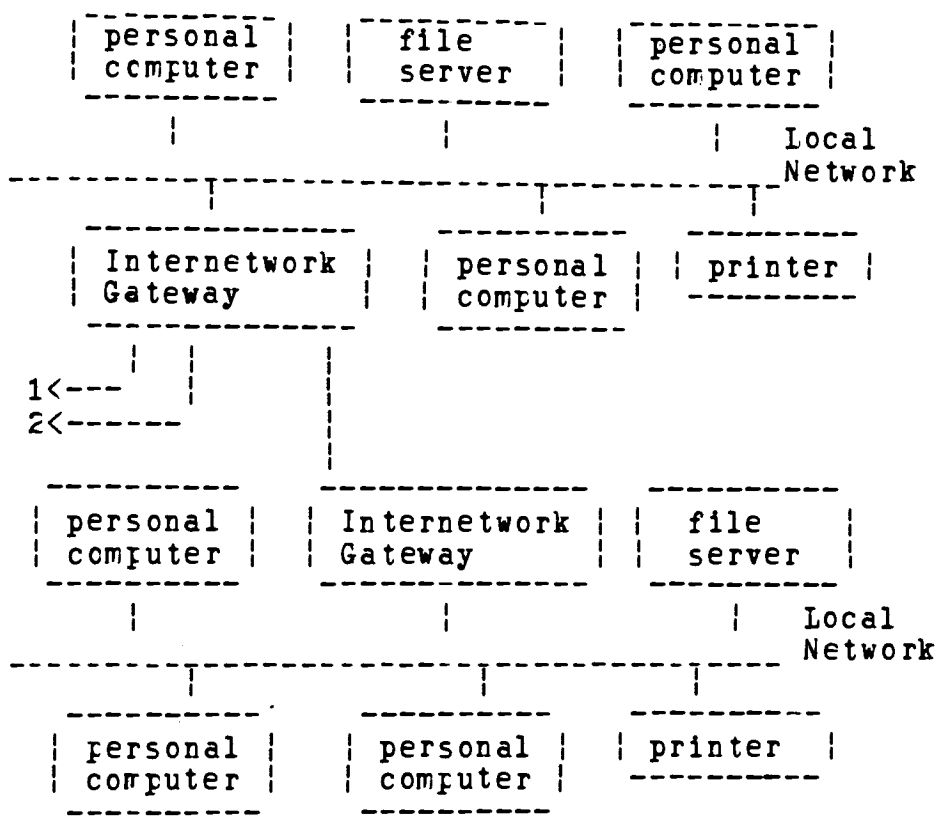
This environment should provide a comprehensive editing and text-processing system; mail, message-handling, and other forms of inter-user communication; a system for sending reminders and scheduling the user's appointments; extensive program development facilities in several high level language (creation, debugging, analysis, management of program, and library maintenance); and an integrated graphics-oriented hardware design facility.

2.4.5. Internet

An advantage of distributed systems is that the system structure can be made to compliment organizational structure. We will consider distributed systems with the

following general characteristics. The overall system is an internet, comprised of interconnected networks with varying communications media. The communications system provides store-and-forward packet switching communications. The processing power of a node on the internet can range from that of an electronic typewriter to that of a large, time-shared computer. Each node provides some collection of physical and logical resources such as a printer or file storage. Some server nodes provide services that permit clients on other nodes to access their resources. Other workstation nodes serve as a user's personal computer and act as clients of remote services on their user's behalf when necessary. It is possible that a node supports both multiple users and shared services. The nodes of such a system communicate for various reasons: to retrieve a file, send a file for remote printing, deliver electronic mail, exchange routing information, etc. Nodes that provide the service of communication between two or more networks in the internet are called Internetwork Gateways or just Gateways. Such an internet is shown in Figure 2-9.

In order to understand the functions of these gateways and how they provide the communication services, we must introduce the concept of binding and look at various binding strategies.



1. To other local Networks
2. To other Long Haul Networks

Fig. 2-9 A typical internet

2.4.5.1. Binding Strategies

A main advantage of decentralized network-based systems is that they can be reconfigured to meet current needs easily. Binding strategies are approaches to distributed systems which can automatically cope with dynamic reconfiguration of networks, resources, and services without requiring an excessive amount of on-site technical

support or human intervention. For example, we might not have to bring an entire internet down and back up simply to add a new node. If we lack binding strategies, system configurations must remain relatively static so that resources cannot be added or deleted.

In a distributed system, system components must cooperate in order to satisfy client requests, and clients must know something about their distributed environment, such as the location of services or resources. Knowledge about the environment is obtained when system components are bound to system resources.

Definitions:

(a) Binding - resolves a reference to an object in the distributed system by replacing the reference with the object's address.

(b) Reference - may be a human-sensible string name, a machine-sensible identification, etc.

(c) Object - may be a physical resource such as a machine, a logical resource such as a file directory, an abstract entity such as a route used for delivering data, etc.

(d) Address - may be the network address of an object or of some system component that will access the object on the client's behalf.

(e) <name, address> - the binding of an object identi-

fied by "name" to "address".

A particular binding strategy is characterized by when the binding occurs. For example, binding might be static and carried out at the time of system generation. It might occur early when the system is initialized. It might occur late when a resource is accessed. We need to evaluate performance versus flexibility when selecting a particular binding strategy. Reconfiguration generally implies the need to correct some established bindings, such as updating, deleting, or adding to a binding. The early binding affected system elements must often be bought down and reinitialized. On the other hand, since each binding operation requires the resolution of one or more indirect references, later binding increases flexibility and decreases performance.

We can use both the performance benefits of early binding and the flexibility of late binding by using early binding wherever possible, but performing runtime rebinding whenever the binding needs correction. With such a scheme, the client must verify that an object with the recorded name is actually located at the address of high speed memory each time an access is attempted. If the object is not at the expected location, the client must rebind. In order to carry out the name verification correctly, the name in <name, address> binding must

uniquely identify the object.

Binding agents in a distributed system are system components that maintain <name, address> associations in a database. For example, a directory assistance telephone operator is a binding agent, and the phone directory is the database.

Late binding with a binding agent can be used to establish or correct bindings at runtime. A binding agent's database could be a dynamically constructed in-core table or a set of complex data structures permanently maintained on secondary storage. There may be many different binding agents in a distributed system performing different binding functions. For example, a binding agent could keep a database of <name, address> pairs for hosts on the internet and for network-based services that provide access to resources.

Whenever a client needs to find the address of an object, it presents the object name to the suitable binding agent. The binding agent returns the address which the client can save in order to bypass lookups the next time the object must be accessed. If the address is saved, the client must be prepared to go back to the binding agent to rebind if name verification shows the object to be absent from the expected location. For example, to print a file on a remote printing server, a user might present the name

of the desired printing services to a client module on his workstation. The client module would then present the name to a binding agent which would subsequently return the server's network address.

2.4.5.2. Internetwork Gateways

Store-and-forward packet switching internets consist of individual networks interconnected by internetwork gateways that store and forward the packets to their destinations. They may use adaptive routing algorithms that dynamically determine the path a packet should take at every internetwork gateway. This is an application of late binding, where the object is a host identified by host and network numbers, address is a route by which the packets should travel. In this case, the binding agent is the internetwork gateway and its database is the routing table.

The routing algorithm in an internetwork gateway needs to be told the numbers of the networks to which they are connected so that they can participate in the routing of packets to other internetwork gateways and to machine on the directly connected network. The internetwork gateway must generate routing table information for exchange with other internetwork gateways. This is how routing tables are dynamically updated. The routing machinery of a packet switching internet cannot work unless:

(1) Each internetwork gateway knows the network address of other internetwork gateways on the same network.

(2) Each workstation or host machine knows how to create routing tables so that they may direct a packet to the suitable internetwork gateway.

2.4.5.3. Xerox Pup Internet

The Xerox Pup Internet supports a large Alto-based distributed system consisting of over 1000 machines on 25 networks of 5 different types using 20 internetwork gateways. The predominant type of network is the Ethernet system. Such local networks permit broadcast delivering of data. The other networks in the internet are used as transmission media to transport packets from one internetwork gateway to another.

This system offers some services such as file transfer, interactive sessions with foreign hosts, exchange of electronic mail, remote printing, etc. The Xerox Pup Internet mainly supports the Alto personal minicomputer. It uses a hierarchical addressing scheme for identifying a machine with the internet. It assigns a fixed socket number for the active listener that provides a particular type of service. Socket numbers are statically assigned according to service type.

A service is identified by the host's internet name or number, followed by the service's socket number. Because of the static association between a service and a socket number, there is no mapping between service ID and network address. Clients must bind the internet host name to a network address before using a service.

Xerox Pup Internet has an unique string name for hosts, ie. there are no local contexts for host names. Machines on every local broadcast network have access to a local clearinghouse (Clearinghouse is a binding agent that maintain the associations between local string service names, global service IDs, and service addresses.) called the name-lookup server that translates a host's string name into its network address. This mapping is performed dynamically every time a service or host is accessed. Workstations need not know the network address of the local name-lookup server because they can broadcast their binding request. There are several name-lookup servers with the same database of the internet. Once a new database is inserted at one name-lookup server, it propagates through the entire system until each name-server has been updated.

Network access software is divided into two classes: (1) for workstation and server machines, this class is identical for every machine. (2) for internetwork gateway

and name-lookup server machines, in this class, network access software is parameterized for every internetwork gateway. Software in the first class executes in an Alto connected to the Ethernet system. The software determines the number of its network by broadcasting a packet in order to locate an internetwork gateway, and then examining the reply. The routing tables are initialized and updated via broadcast packets from internetwork gateways. Abraham and Dalal (1980) have found that this way of managing software configurations is acceptable, as most machines are workstations, and the internet has only 20 internetwork gateways.

2.4.6 Data Base Server - Backend Machine

2.4.6.1. Introduction

For data base users, even record oriented files may be far too primitive to use directly in application programs. Consequently, the network may provide one or more data base servers, also called backend machines. A data base server might accept queries about the data base, analyze them itself, and return the answers.

2.4.6.2. Objectives and Characteristics

A database machine (DBM) can be defined as any hardware, software, and firmware complex dedicated to perform some or all of the functions of the database manage-

ment portion of a computing system. The DBM may range from a small, personal query machine to a large, public-utility information machine.

Backend computers in database systems are dedicated computers for carrying out database processing functions such as the retrieval and manipulation of databases, the verification of data access, the formulation of responses, the enforcement of integrity and security rules and constraints, etc. Figure 2-10 shows one possible backend system. The operating system, application programs, and DBMS interface run on the host computer and actual DBMS runs on the backend computer.

The main concept of backends is to off-load the database management function from the host computer to dedi-

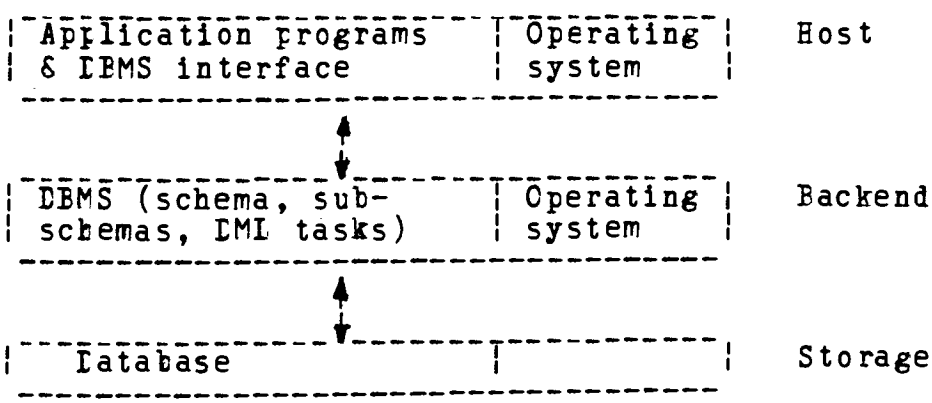


Fig. 2-10 A configuration of a backend computer system

cated processor(s) in order to (1) release the host from tedious and time-consuming operations involved in database manipulation, maintenance and control. (2) increase system performance through functional specialization of and through parallel processing among the host and the backend(s). (3) reduce the cost of managing data. The backend approach can be viewed as a cost-effective alternative to upgrading to the host or to achieving the level of functionality and performance that no conventional system can provide.

The isolation of the DBMS, the mass storage devices and the database from the host can bring a number of additional advantages. (1) several hosts, possibly dissimilar, can share on-line data as shown in Figure 2-11. A single backend may handle the processing of the database and

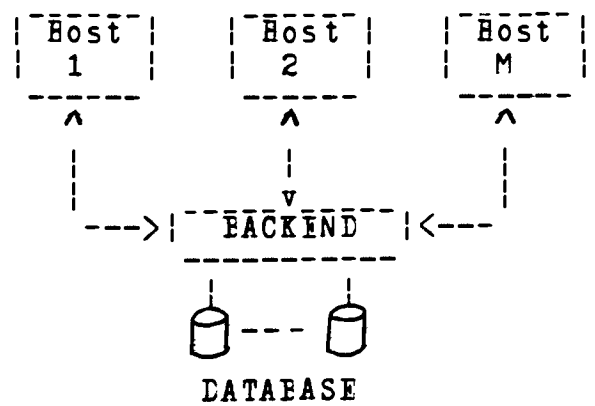


Fig. 2-11 Multiple host configuration

present data in forms suitable to the dissimilar hosts. (2) databases and the DBMS itself can be transported from an old mainframe to a new one with relatively little conversion effort. Similarly, changes to the databases, the mass storage devices, and the DBMS can be made without entailing changes to the host. (3) multiple numbers of backends (Figure 2-12) can be used to process large databases, which can be stored either in a distributed manner across secondary memory devices to facilitate parallel processing or in a manner such that each database can be processed by one backend. (4) The enforcement of database integrity and security can be separated from that of operating system integrity and security, thus the failure of one will not endanger the other.

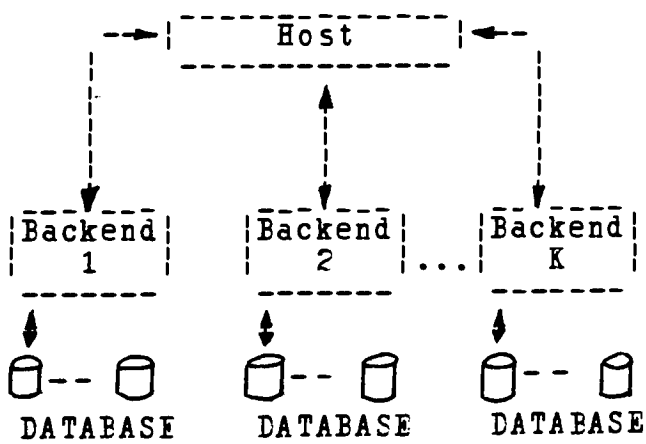


Fig. 2-12 Multiple backend configuration

2.5 The Pool Processor Model

2.5.1. Introduction

As long as each user is content with a single machine for running user programs, the user-server model is reasonable. However, dedicating a large minicomputer to each user is an inefficient way to do business, because computing requirements are bursty. In this model, work is carried out by a pool of processors, some of which may have fixed functions, such as file servers, and some of which are dynamically allocatable on demand. Wittie (1979)[13] has envisioned a system of this kind which we will discuss in the following section.

2.5.2 MICROS - Distributed operating system

MICROS is the distributed operating system for the MICRONET network computer. MICRONET is a reconfigurable and extensible network of sixteen loosely-coupled LSI-11 microcomputer nodes connected by packet-switching interfaces to pairs of high-speed shared communication buses. MICROS simultaneously supports many users, each running multicomputer parallel programs. MICROS is intended for control of network computers of up to 10,000 nodes.

---

[13]Wittie, L.D.: "A Distributed Operating System for A Reconfigurable Network Computer," Proc. First Inc. Conf. on Distrib. Comput. Syst., IEEE, pp. 669-677, 1979.

MICRONET is an extensible, reconfigurable network of autonomous computer modules, each sharing two of the communication buses over which packet-switched messages are transmitted. Figure 2-13 shows the architecture of one node of MICRONET. Each node consists of a task computer and a communications frontend computer. The task and frontend memories are linked by a 0.3 megabyte per second (MB/S) direct memory access (DMA) channel controlled by the frontend. The task computer is a DEC LSI-11 with a full complement of 28 kilowords of 16-bit RAM.

Figure 2-14 shows one possible configuration of a sixteen node version of MICRONET. There are eight shared buses: four represented by horizontal lines and four by vertical lines. Each node shares exactly two buses, those that pass nearest it horizontally and vertically. Peripherals attached to the LSI-11's in MICRONET include five interactive CRT and hard copy TTY terminals, two minicomputers linked by 1920 byte per second serial interfaces, and two floppy disk drives on 16-bit parallel interfaces. Each node has a direct bus connection to only a few (six in Figure 2-14) of the other nodes. A message from node 1 to node 15 must be relayed through node 3 (via buses 1 and 7) or node 13 (via buses 5 and 4) for a minimum delay connection.

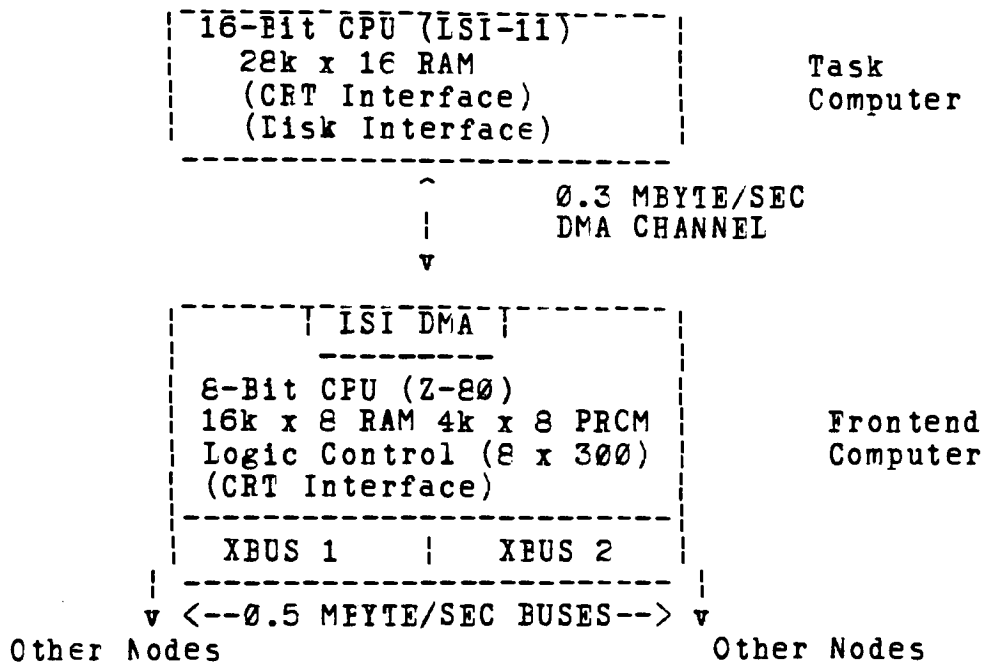


Fig. 2-13 Architecture of a MICRONET node. Each node consists of separate task and frontend computer linked by a DMA channel. Each node has two ports to share external buses. Some nodes have terminal or disk peripheral interfaces.

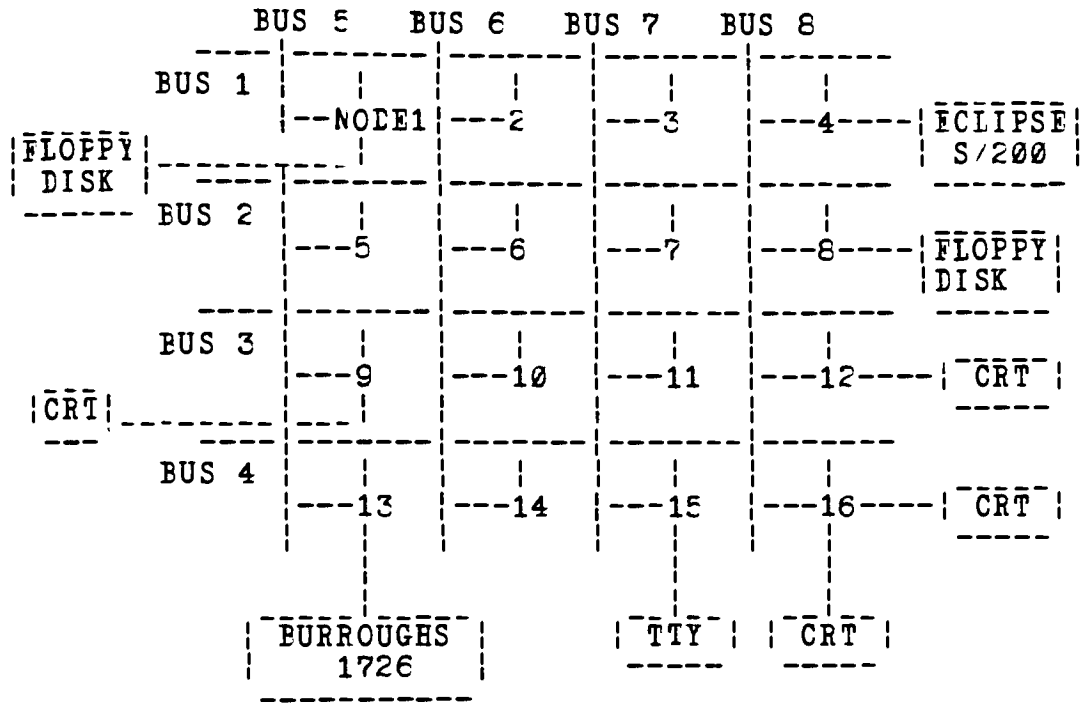


Fig. 2-14 One possible configuration of eight buses connecting sixteen microcomputer nodes in MICRONET. Each node shares two buses. Each bus is shared by four nodes. Terminals and minicomputers are attached by serial interfaces.

In summary, the MICRONET hardware consists of nodes of paired task and communications computers. Each node is linked to two shared buses. Nodes cannot directly access data in the memory of other nodes but rather communication via packet-switching. Packet latencies are 1 to 3 milliseconds. In the following section, we will explore methods by which MICROS controls the distributed MICRONET hardware.

The main research problem in the design of operating systems for massively distributed computers is global resource management, the control of shared resources that may be requested by nodes throughout the network. Since MICRONET is a loosely-coupled network whose nodes cannot directly access each other's memories, MICROS cannot utilize centralized resource tables accessed by several management nodes.

The global management structure proposed for MICROS to control networks of thousands of nodes is shown in Figure 2-15. An example will be given in next section. The circles represent nodes in the network. The arcs represent message paths between them. The lowest nodes in the tree perform user tasks and handle I/O devices connected to the network. The upper nodes manage resources and provide regional control for the nodes directly below them.

Although globally accessible, network resources are managed in nested pools. Each management node controls all resources within its subtree of the network hierarchy. Monitoring, scheduling, and allocation of each resource associated with a low-level node are performed by one or more of the management nodes in the chain between the resources node and the top of the hierarchy. For example, idle nodes which can accept new user tasks are a network

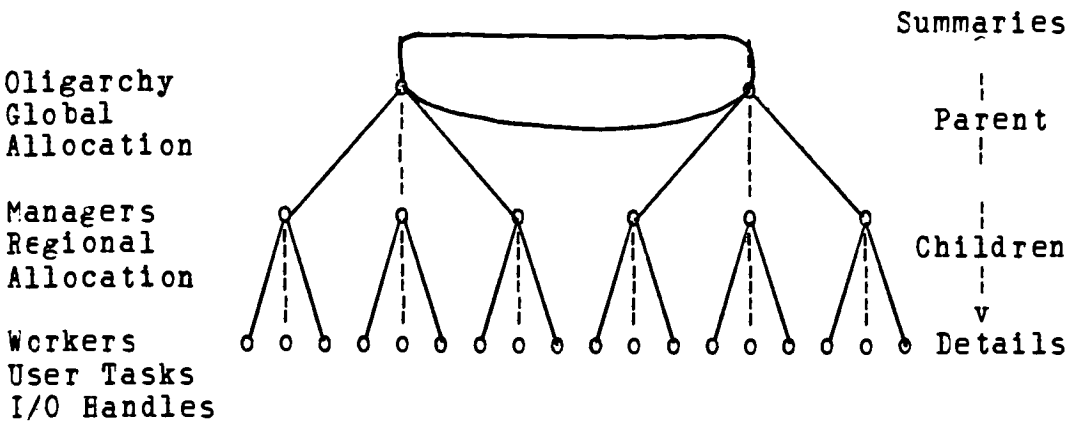


Fig. 2-15 A logical hierarchy of resource management nodes for a distributed operating systems. Path between nodes may involve multiple hardware links. Nodes and paths are chosen for efficient message passing within each resource pool.

resource. Each manager keeps track of the approximate number of idle nodes available in the subtree below it. High-level nodes request groups of nearby nodes for scheduled groups of intercommunicating tasks. Middle-level managers subdivide the requests to match groups of available nodes. Low-level managers allocate individual tasks to nodes, keep exact amount of idle nodes, and lock busy nodes against further loading.

In order to avoid communication and processing overloads, each node can exchange control messages only one level upward with its parent and one level downward with its children. However, any node can have physical links to many others at various levels. User task and I/O data

message can be relayed along any links. Also to avcid overloads, nodes in each higher level of management keep only summaries of the resource information known to the nodes in the next lower level. The higher in the tree, the more global is the scope of management information but the less is known in detail. On the other hand, low level managers will know which user task nodes are idle but their parent will know only how many idle nodes each manager controlled.

In summary, for large networks MICROS has resource management tasks and system tables distributed over a logical hierarchy of nodes. There is a hierarchy of nested resource pools controlled by managers whose load is kept rcughly constant regardless of level.

2.5.3. Wave Scheduling for MICROS

To solve user problems in parallel, programming language compilers for network computers must generate separate simultaneous executable task modules. Such collections of related tasks are known as task forces. Wave Scheduling[14] is used by the MICROS operating system to schedule task forces in MICRONET. Assume that a task force of size S, which needs S nodes to execute, enters a queue

---

[14]Van Tilborg, Andre M., Wittie, Larry D., "Distributed task force scheduling in multi-microcomputer networks", National Computer Conference, 1981, pp. 283-289.

of ready task force at an arbitrary network node. Task forces may enter at any level of the control hierarchy, at managers and worker nodes. If a task force enters the network at a level which cannot schedule a task force of size as large as S, the task force request is passed up the tree until a suitable manager is reached. In this case, a manager at the appropriate level is the Task Force Master (TFM) for the task force. A task force too large to be processed will be rejected when it reach the top level.

Each TFM keeps track of the number of non-busy workers in the subtree below it. TFM are responsible for reserving enough nodes for the task forces which they control, that is, to reserve R>=S workers to get S. Tilborg and Wittie[15] have shown that a useful value of R is R=(S*k)/(1-u), where u is the instantaneous fraction of busy worker nodes in the subtree of a TFM and k is a value from 1.1 to 1.3. The request for R workers is divided among the submanagers of the TFM and proceeds down the tree as a wave of subrequest. Each subrequest is repeatedly divided until it reaches the lowest managers. Managers at that level store accurate information regarding the status of worker nodes.

---

[15] Van Tilborg, A.M., Wittie, I.D., "Wave Scheduling: Distributed Allocation of Task Forces in Network Computers", Proceedings of 2nd Int. Conf. on Dist. Comp. Sys., Paris, 1981.

An example of wave scheduling is shown in Fig. 2-16. A task force consisting of S=8 tasks enters the network at the rightmost level-1 manager. The task force is too large to be scheduled by that manager, so it is passed one level upward. From there a request for R=11 worker nodes travels as a wave of subrequests downwards through the subtree of the TFM. When a subrequest reaches a level-1 manager, that manager reserves as many as possible of the requested number of workers. Finally, a total of ten worker nodes are reserved by the wave scheduling. Only two workers were

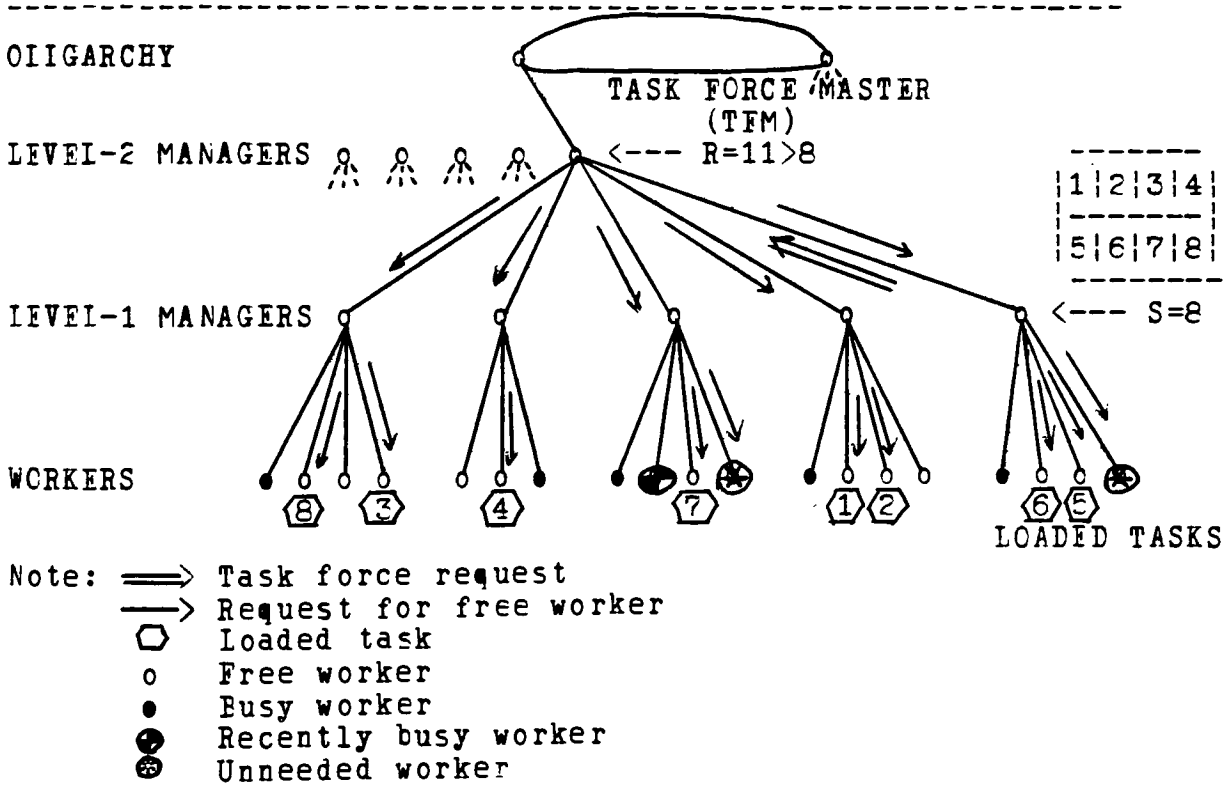


Fig. 2-16 The wave of subrequests to reserver R nodes for a size S task force.

actually idle but the TFM thought there were three idle workers. The worker marked as 'recently busy' was believed to be available by the TFM because the status summaries had not yet been updated. Eight of these workers are loaded with task modules (marked by hexagons) while the other two (marked by *) are released.

2.5.3. Characteristics of pool processors model

The reason we present the pool processor model is that it has faster response time than the user-server model. We can allocate resources dynamically in this model. In order to fully utilize idle machines, each user may have ten pool processors 10% of the time to carry out computation for him, but in the user-server model each user owns his personal CPU all the time.

In order to schedule the pool processors, we need complicated scheduling algorithms. In the next section, we will discuss three possible algorithms.

2.5.4. Scheduling algorithms

2.5.4.1. Parent-Children algorithms

This algorithm consists of dedicating one processor to schedule the others. The dedicated processor keeps track of the status of all processors and tells the others what to do next.

2.5.4.2. Wave scheduling algorithm

MICROS in the previous section is a good example of this algorithm. The pool processors are statically partitioned into groups, with each group having a group manager. The group managers are divided into divisions, with each division having a division manager. Each manager schedules the processors under it.

2.5.4.3. Bidding algorithm

When a new process is to be created, the parent broadcasts a request for bids which might include special properties of the computation. If any of the pool processors wants to bid for that work, it will send messages to the requesting processor. The parent will review the characteristics of all bidding processors, and then choose the most qualified.

In order to understand how the bidding algorithm works, we will discuss the Distributed Computing System (DCS) in the following section.

2.5.5. Distributed Computing System (DCS)

Farber and Larson (1972)[16] presented the Distri-

---

[16]Farber, D.J., and Larson, K.C.: "The System Architecture of the Distributed Computer System - the Communication System," symp. on Comput. Networks, Polytechnic Institute of Brooklyn, April 1972.

buted Computing System (DCS) which is an experimental computer network under study at the University of California at Irvine. The network has been designed with the following goals: reliability, low cost facilities, easy addition of new processing services and low incremental expansion cost. The computers used are small to medium scale and are interfaced to the ring (Note: Ring is one kind of local networks. In a ring, a special bit pattern circulates around the ring.) using a sophisticated piece of hardware called a Ring Interface (RI).

DCS (Figure 2-17) consists of a set of different processors interconnected by a common transmission system - the ring. All messages flowing through the ring, are addressed to some processes, not a processor. Further, each address is the name of that process. Thus the only addresses are the names of sending and receiving processes. An advantage of this proposed method of addressing is the easier and more dynamic entry and exit protocols available to processors on the ring.

There are two features which make the communications protocols unique. First, messages are addressed to processes, not processors. This is accomplished by placing an associative memory in each RI. The memory contains the names of all processes active on the attached processor. When a message arrives over the ring, the destination pro-

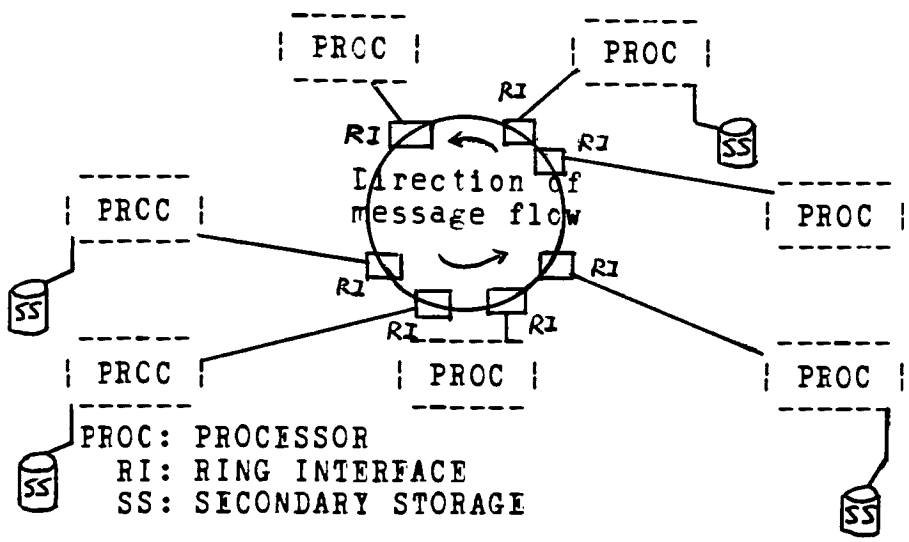


Fig. 2-17 An example DCS

cess name is matched against the associative memory. If a match occurs the message is copied and passed over the ring to the next RI. Second, messages are only removed at the RI from which they originate. The ring may be thought of as a series of message slots. To transmit a message the RI waits for an empty slot and places the message on the ring. The message is copied when necessary as it proceeds around the ring and checked against the original when it returns to the originating RI.

Farber and Larson considered that system performance could be improved by connecting a set of small machines. For example, linking the small machines permits the sharing of the pool of peripheral facilities. Interconnecting

the small machines will also improve the reliability of the system.

In DCS, all processors and terminals are attached to the communication system via a RI. The RIs are connected together via a single unidirectional data path. We shall use a fixed block length message format, that is, the ring is analogous to a lazy Suzan with trays on it that are rotating past a set of people (the nodes) gathered around it. Suppose a process p1 desired to send a message to process p2. The principal that we will use is as follows:

(1) p1 waits until an empty tray appears.

(2) When p1 finds an empty one, p1 places two items in the tray, the name of the process and the message p1 wishes to send it to.

(3) p1 then waits until it sees the message passing p1 again. At that point p1 removes the message from the tray and checks to see if the process p2 received the message. If not it repeats the sequence for a fixed number of times.

(4) From p2's point of view, it looks for messages with its name on them as the trays pass it. If it sees one, p2 makes a copy of the message and drops in the tray a note telling the sending process that it has received the message.

As has been mentioned previously, messages are addressed by the name of a process. Thus each message is broadcast to all nodes, there to be accepted or disregarded depending on whether or not the desired destination process is active on the associated host. A complete description of the software is given in [17] and a definition of a distributed file system for the DCS is given in [18].

2.5.6. CNET - Contract Net Protocol

In bidding systems, the question arises whether a processor that is currently occupied may continue to bid for worked maintain an internal work queue. If processors do queue work, they may confer from time to time, to allow processors with long queues to offload work onto processors with short queues. Smith (1979)[19] presented the contract net protocol which had been developed to specify problem-solving communication and control for the nodes in a distributed problem solver. Task distribution is effected by contract negotiation. It is based on a

---

[17]D.J. Farber and K.C. Larson, "The Structure of a Distributed Computing System - Software," Proc.Symp. Computer-Communications Networks and Teletraffic, Polytechnic Press, 1972.

[18]F. Heinrich, "The Systems Architecture of the Distributed Computer System - The Distributed File System," Technical Report, University of California, Irvine.

[19]Smith, R.: "The Contract Net Protocol : High Level Communication and Control in a Distributed Problem Solver," proc. First Int. Conf. Distrib. Comput. Syst., IEEE, pp. 185-192, 1979.

discussion carried on between a node with a task to be executed and a group of nodes that may be able to execute the task.

The key issue to be resolved in task-sharing is how tasks are to be distributed among the processor nodes. There must be a means whereby nodes with tasks to be executed can find the most appropriate idle nodes to execute those tasks. This is called the connection problem. The connection problem has two parts: Nodes with tasks to be executed seek other nodes capable of their execution, and nodes not engaged at any given time seek tasks that can execute. Both sets of nodes can proceed simultaneously, engaging each other in a process of negotiation to solve the connection problem.

The collection of nodes is referred to as a contract net and the execution of a task is dealt with as a contract between two nodes. Each node in the net takes on one of two roles related to the execution of an individual task : manager or contractor. A manager is responsible for monitoring the execution of a task and processing the results of its execution. A contractor is responsible for the actual execution of the task.

The normal method of initiating contract negotiation is for a node that generates a task to advertise existence of that task to the other nodes with a task-announcement

message. It then acts as the manager of the task. In the absence of any information about the specific capabilities of the other nodes in the net, a manager is forced to issue a general broadcast to all nodes.

The eligibility specification is a list of criteria that a node must meet to be eligible to submit a bid. It enables a node receiving the message to decide whether or not it is able to execute the task. The task abstraction is a brief description of the task to be executed. It enables a node to rank the announced task relative to other announced tasks.

Each node checks the eligibility specifications of all task announcements that it receives. If it is eligible to bid on a task, then it must rank that task, relative to others that have been announced. In CNET, all tasks are typed and a node maintains a list containing the best task announcements of each type that it has received. Each time a task announcement for which a node is eligible is received, the node compares the task abstraction of the new announcement with the task abstraction of the best task announcement of the same type received thus far. It then decides whether to replace the current best task announcement of the given type with the new task announcement or to discard the new announcement.

This announcement-ranking activity proceeds concurrently with task processing until the node goes idle. At this point, the node checks its current list of task announcements and selects a task on which to submit a bid. If there are a number of task types available, the node must select one of them. The current version of CNET selects the oldest task on the queue.

Contracts are queued locally by the node that generates them until they can be awarded. When the manager for a contract receives a bid, it compares the new bid with the list of bids thus far received for the task. If the new bid is determined to be satisfactory, then the contract is awarded immediately to the node that made that bid. Otherwise, the new bid is sorted into the list of bids.

The selection is communicated to the successful bidder through an announced-award message. This message contains a complete specification of the task to be executed. Bids in a contract net are binding. This avoids a confirmation message for each transaction[20]. If a node receives multiple awards as a result of bidding on several contracts, then it queues them for processing in order of receipt.

---

[20]D.J. Farber, K.C. Larson,. "The Distributed Computing System." Conference Digest, IEEE COMPCON Spring, 1973, pp. 31-34.

For example:

```
< Managers make announcements of this form >

To : * ( * indicates a broadcast message )
From : 25
Type : TASK ANNOUNCEMENT
Contract : 22-3-1
Message :
          Task Abstraction :
               TASK TYPE `signal
               NODE NAME `25  POSITION `p

          Eligibility Specification :
               MUST HAVE DEVICE TYPE `Sensor
               MUST HAVE NODE NAME `SELF POSITION

          Bid Specification :
               NODE NAME `SELF POSITION
               EVERY DEVICE TYPE `Sensor TYPE NUMBER

< Sensor nodes respond to the nearest manager >

To : 25
From : 42
Type : BID
Contract : 22-3-1
Message :
          Node Abstraction :
               NODE NAME `42  POSITION `q
               Sensor TYPE `s  NUMBER `3
               Sensor TYPE `t  NUMBER `1

< Several similar awards are transmitted >

To : 42
From : 25
Type : AWARD
Contract : 22-3-1
Message :
          Task Specification :
               Sensor NAME `s1
               Sensor NAME `s2
```

Finally, we discuss dynamic distributing of information. The contract net protocol enables dynamic distribution of information via three methods. First, a node can

transmit a request directly to another node for the transfer of the required information. Second, a node can broadcast a task announcement in which the task is a transfer of information. Third, a node can note in its bid on a task that it requires particular information in order to execute the task. The manager can then send the required knowledge in the award message if the bid is accepted.

2.6 The Data Flow Model

2.6.1. Introduction

The important components of computer networks are nodes and data links. The nodes are processing elements, each with its own memory and processor unit. The data links are communication channels that permit messages to be sent among the nodes. Data links might have a queue at the receiver to store messages until the receiver is ready to process them. A network in which the nodes are activated by the arrival of input packet is called a data flow network.

Data flow is an alternative model of computation which is particularly promising. The basic principles of data flow are asynchrony and functionality, and thus are in distinct contrast to the von Neumann model.[21]

---

[21]Taub, A.H., "Collected Works of John von Neumann",

The key idea of a data flow system is that there are no variables in the usual sense, that is, no memory locations that can be stored into. Instead, values are represented by packets that are transmitted between processing units. Each processing unit has the task of computing some function of its inputs and producing an output containing the result.

Because there are no variables or side effects in a data flow computer, there is no concept of `control flow` - an instruction is ready for execution when its input packet has arrived. Data flow computers do not have a program counter and there is no explicit sequencing of computations, other than that implicit in one calculation depending on the result of another one. For example, if y=f(x) and z=g(y), then f must run before g. A consequence of data-activated instruction execution is that many instructions of a data flow program may be available for execution at once. Thus highly concurrent computation is a natural accompaniment of the data flow idea.[22]

2.6.2 Data flow programs

2.6.2.1. Type 1

---

vol.5, The Macmillan Company, New York, 1963, pp.34-79.

[22]Dennis, J.B.: "The Varieties of Data Flow Computers", First Int. Conf. Data Flow Comput., IEEE, pp. 430-431, 1979.

A data flow program is a directed graph which made up of actors connected by directed arcs. One kind of actor is the operator shown in Figure 2-18 which is drawn as a circle with a function symbol written inside - in this case '+' - indicating addition. An operator also has input arcs and output arcs which carry tokens bearing values. The arcs define paths over which values from one actor are conveyed by tokens to other actors.

Tokens are placed on and removed from the arcs of a program graph according to firing rules, which are shown in Figure 2-19. To be enabled, tokens must be presented on each input arc, and there must be no token on any output arc of the actor. Any enabled actor may be fired. This means removing one token from each input arc, applying the specified function to the values associated with those tokens, and placing tokens labelled with the result value on the output arcs.

Operators may be connected as shown in Figure 2-20 to form program graphs. Here, presenting tokens bearing

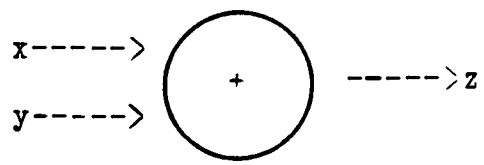


Fig. 2-18 Data flow actor

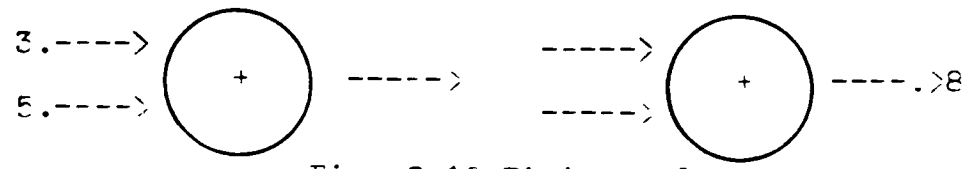


Fig. 2-19 Firing rule

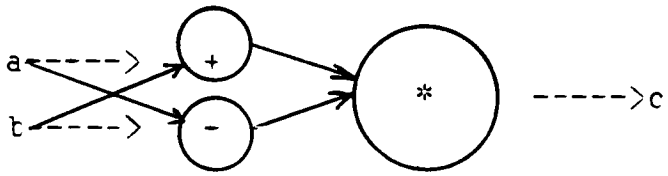


Fig. 2-20 Interconnection of operators

values for a and b at the two inputs will enable computation of the value c=(a+b)*(a-b) by the program graph, and cause the placement of a token carrying the result value on output arc c.

2.6.2.2. Type 2

There is another representation for data flow programs - one that is much closer to the machine language used in prototype data flow computers. In this scheme, a data flow program is a collection of activity templates, each corresponding to one or more actors of a data flow program graph. Figure 2-21 shows an activity template corresponding to the plus operator (Figure 2-18). It consists of four fields : (1) an operation code specifying

the operations to be performed; (2) receivers (in this case, 2), which are places waiting to be filled in with operand values; and (3) destination fields which specify what is to be done with the result of performing the operation on the operands.

Figure 2-22 shows how activity templates are joined to represent the program graph in Figure 2-20. An instruction in a data flow program is the fixed portion of an activity template and consists of the operation code and the destinations

instruction: <opcode, destination>

Each destination field specifies a target receiver by giving the address of some activity template and an input integer specifying which of the receivers in the template is the target.

destination: <address, input>

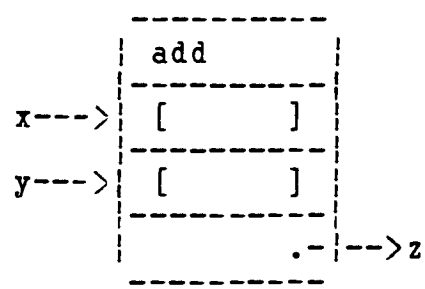


Fig. 2-21 An activity template

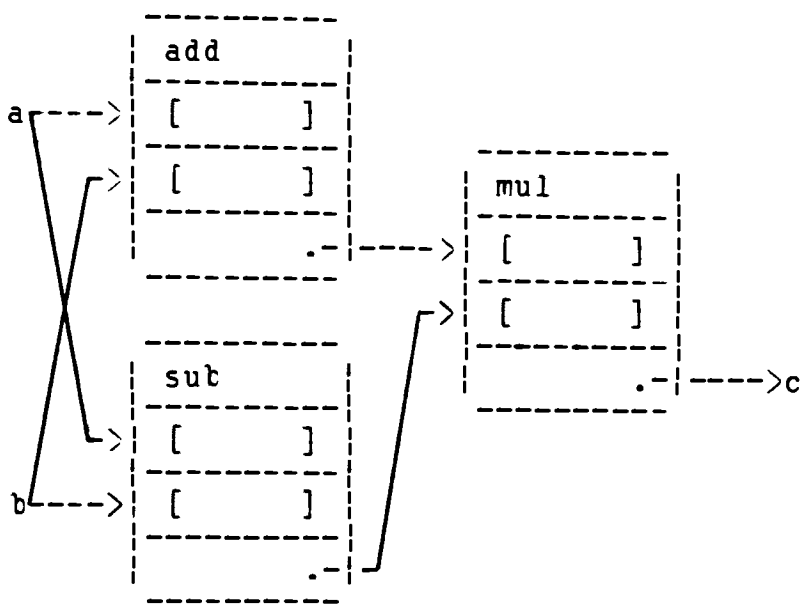


Fig. 2-22 Configuration of activity templates for the program graph of Fig. 2-20

There are two new data flow actors, switch and merge, which control the routing of data values. The switch actor which is for conditional structure sends a data input to its T or F output according to whether a boolean control is true or false. The merge actor which is for iterative structure forwards a data value from its T or F input according to its boolean input value. The conditional program graph and implementation in Figure 2-23 represent computation of

b := if (a>5 then a+3 else a-2) *6

and the program graph and implementation in Figure 2-24

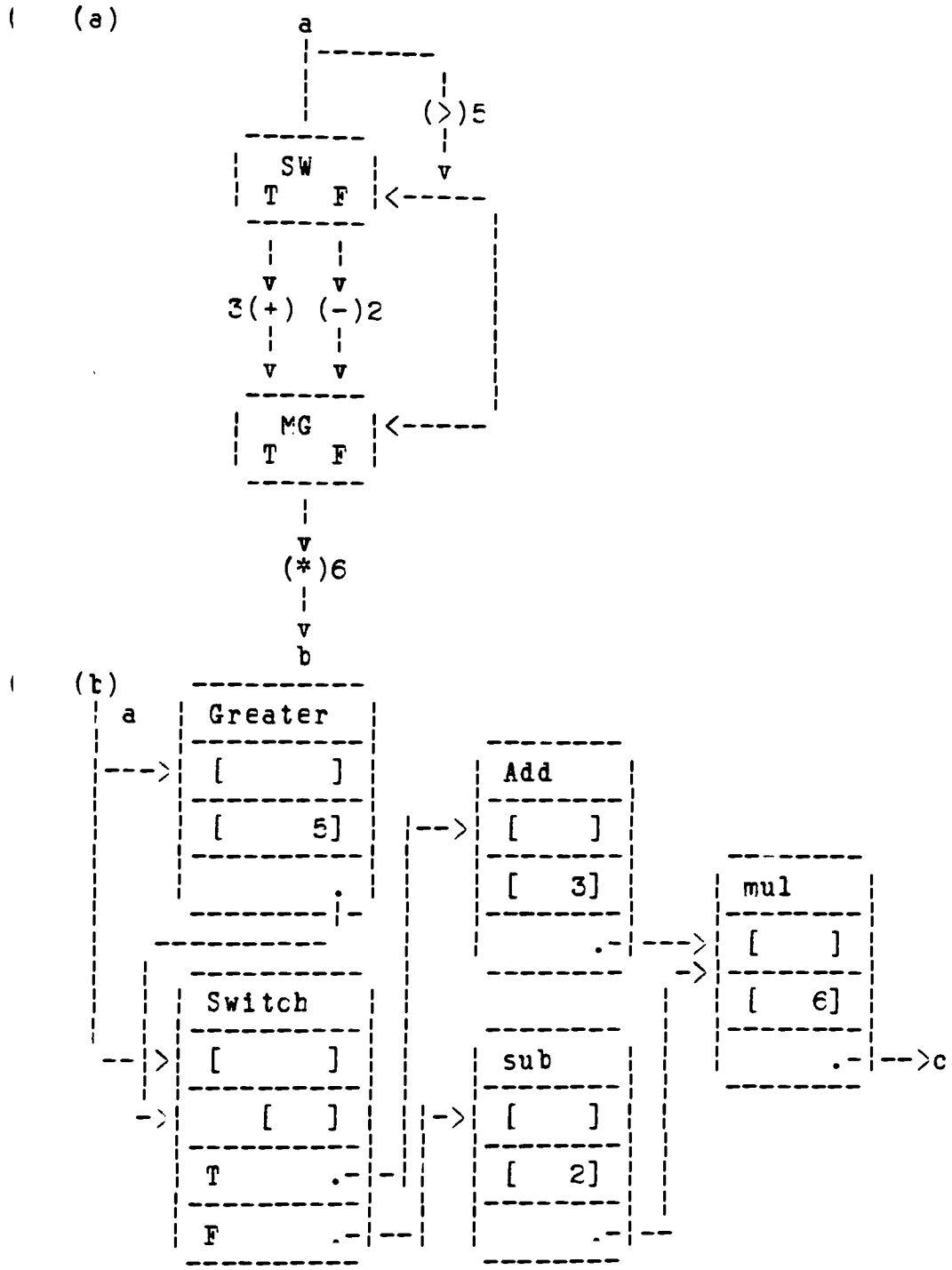


Fig. 2-23 (a) A conditional schema and (b) its implementation for b:= if(a>5 then a+3 else a-2)*6

represents the iterative computation while a<Ø do a := a+4.

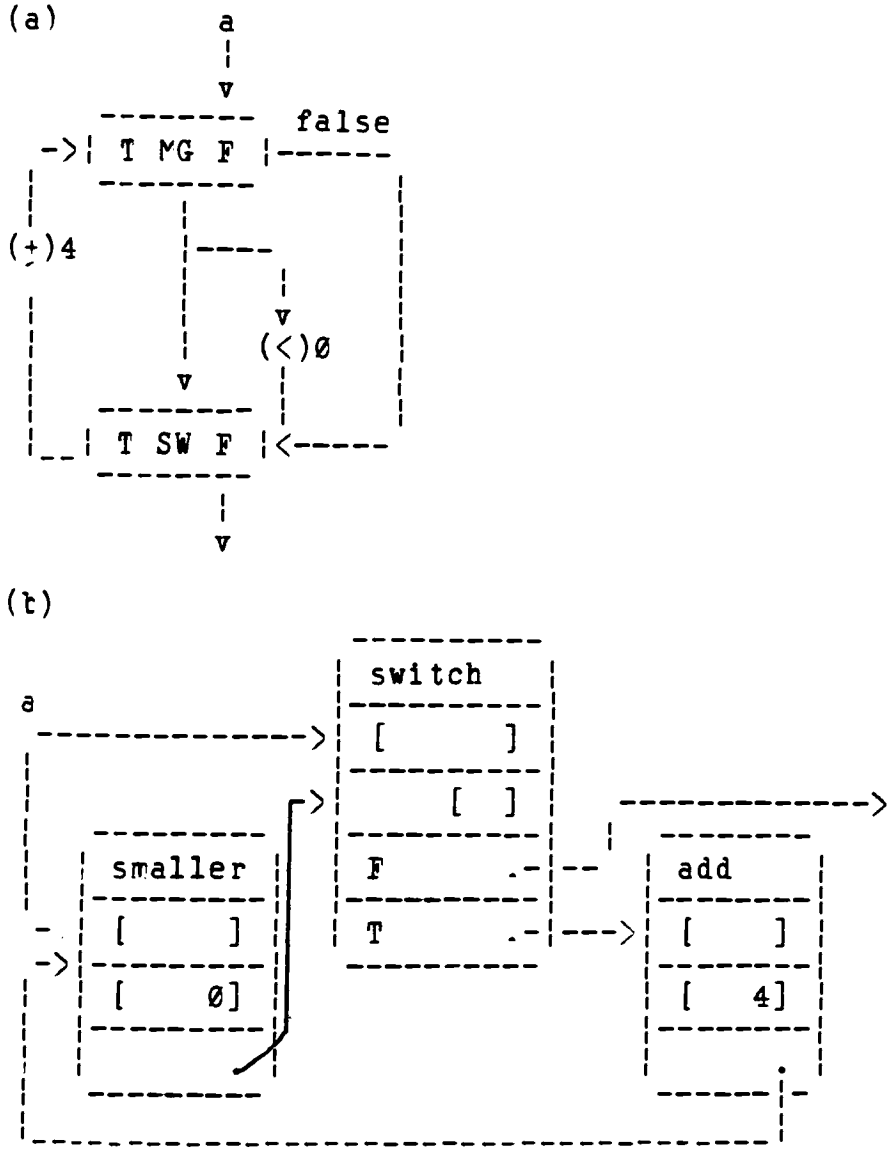


Fig. 2-24 (a) An iterative schema and (b) its implementation for while (a<Ø) do a:= a+4

Execution of a machine program consisting of activity templates is viewed as follows: When a template is activated by the presence of an operand value in each receiver, the contents of the template from an operation packet of the form

operation packet : <opcode, operands, destinations>

such a packet specifies one result packet having the form

result packet : <value, destination>

for each destination field of the template. Generation of a result packet, in turn, causes the value to be placed in the receiver designated by its destination field.

Another example is Butterfly of a Fast Fourier Transform shown in Figure 2-25. The expressions being evaluated are :

$A' = A+C\cos\alpha+D\sin\alpha$

$B' = B-C\sin\alpha+D\cos\alpha$

$C' = A-C\cos\alpha-D\sin\alpha$

$D' = B-D\cos\alpha+C\sin\alpha$

These can be built up from six simple nodes performing the functions of addition, subtraction, multiplication, sin, cosine, and duplication of a value. A node becomes executable when its input values(tokens) are available. Following this principle it can be seen that, assuming at least four execution units, the computation can be performed in seven steps as indicated by the number adjacent

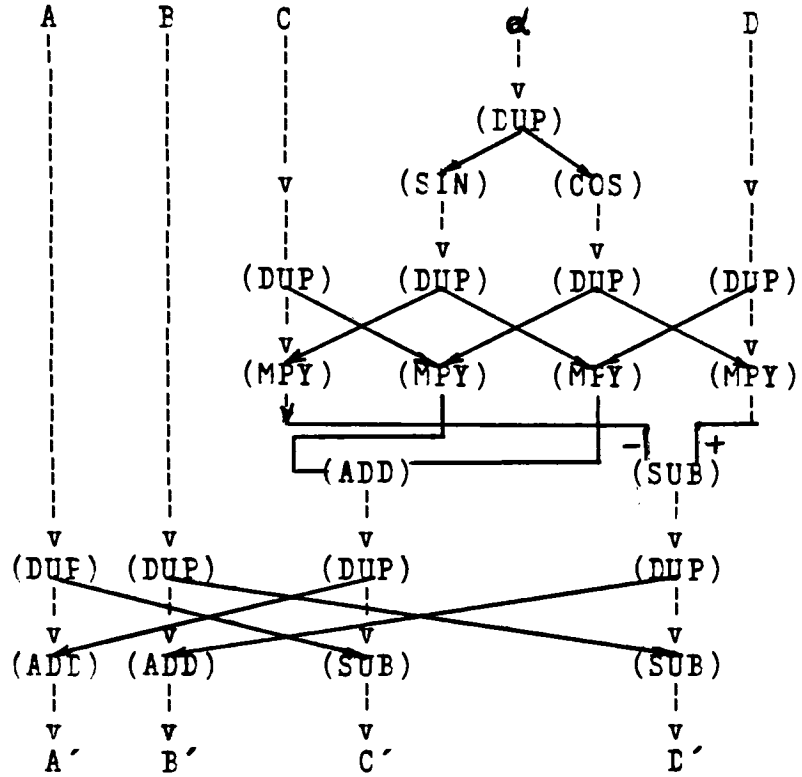


Fig. 2-25 Graph of an F.F.T. butterfly

to the nodes. The total number of nodes in the graph is 21 and therefore the computation has an average parallelism of three.

2.6.3. Data flow principles

Treleaven (1979)[23] presented four basic principles of all data flow schemes :

(1) An operation is executed as soon as all its input

[23]Treleaven, P.C.:"Exploiting Program Concurrency in Computing System, Computer, vol.12, pp.42-49, Jan. 1979.

values are available.

(2) When an operation is executed, the input data values are no longer available as inputs to any operation. They can produce more than one output value.

(3) There is no concept of data storage and operations are not allowed to retain internal state information.

(4) No sequencing constraints are given beyond those implied by the data flow specification - that is, there are no control flow specifications.

2.6.4. Instruction Execution

The basic instruction execution mechanism used in a number of past data flow projects is shown in Figure 2-26. The data flow program describing the computation to be performed is held as a collection of activity templates in the Activity Store. Each activity template has a unique address which is entered in the Instruction Queue unit (A FIFO buffer store) when the instruction is ready for execution.

The Fetch unit takes an instruction address from the Instruction Queue and reads the activity template from the activity store, forms it into an operation packet, and passes it on to the Operation Unit. The Operation Unit performs the operation specified by the operation code on the operand values, generating one result packet for each destination field of the operation packet. The Update

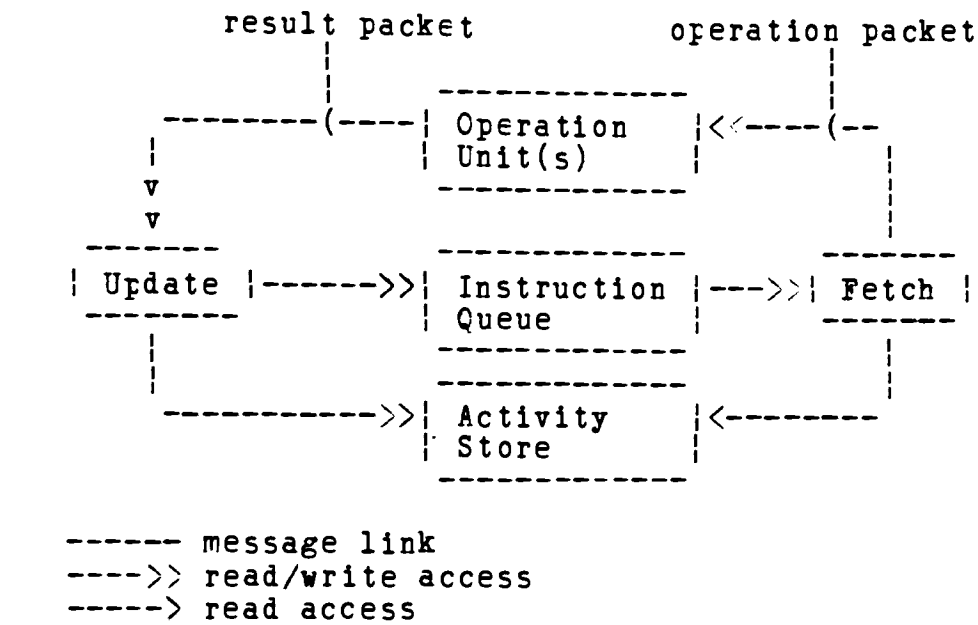


Fig. 2-26 Basic instruction execution mechanism

unit receives result packets and enters the values they carry into operand fields of activity templates as specified by their destination fields. The Update unit also tests whether all operand and acknowledge packets required to activate the destination instruction have been received, and, if so, enters the instruction address in the Instruction Queue.

During program execution, the number of entries in the Instruction Queue measures the degree of concurrency present in the program. Once the Fetch unit has sent an operation packet off to the Operation Unit, it may immediately read another entry from the Instruction Queue

without waiting for the instruction previously fetched to be completely processed. Thus a continuous stream of operation packets may flow from the Fetch unit to the Operation Unit so long as the Instruction Queue is not empty.

2.6.5 Manchester data flow architecture

Watson and Gurd (1979)[24] presented the Manchester Data Flow Prototype at the University of Manchester. They considered the basic elements of a data flow machine must contain parallel processing units which perform the nodal operations, a stored description of the directed graph and a mechanism for collecting and matching tokens. The first two requirements can be realized using standard processing and storage techniques; the matching operation is more complex.

Figure 2-27 shows the basic architecture of the Manchester Data Flow Prototype. The Instruction Store is a RAM which holds the directed graph description. Each entry is in the form of a nodal operation and the addresses of the subsequent nodes to which the output tokens will be directed. The processing units are microprogrammed microprocessors with a distribution system. The distribu-

---

[24]Watson, I., Gurd, J. "A Prototype Data Flow Computer with Token Labelling", AFIPS, Conference Proceedings, NCC, pp.623-628, 1979.

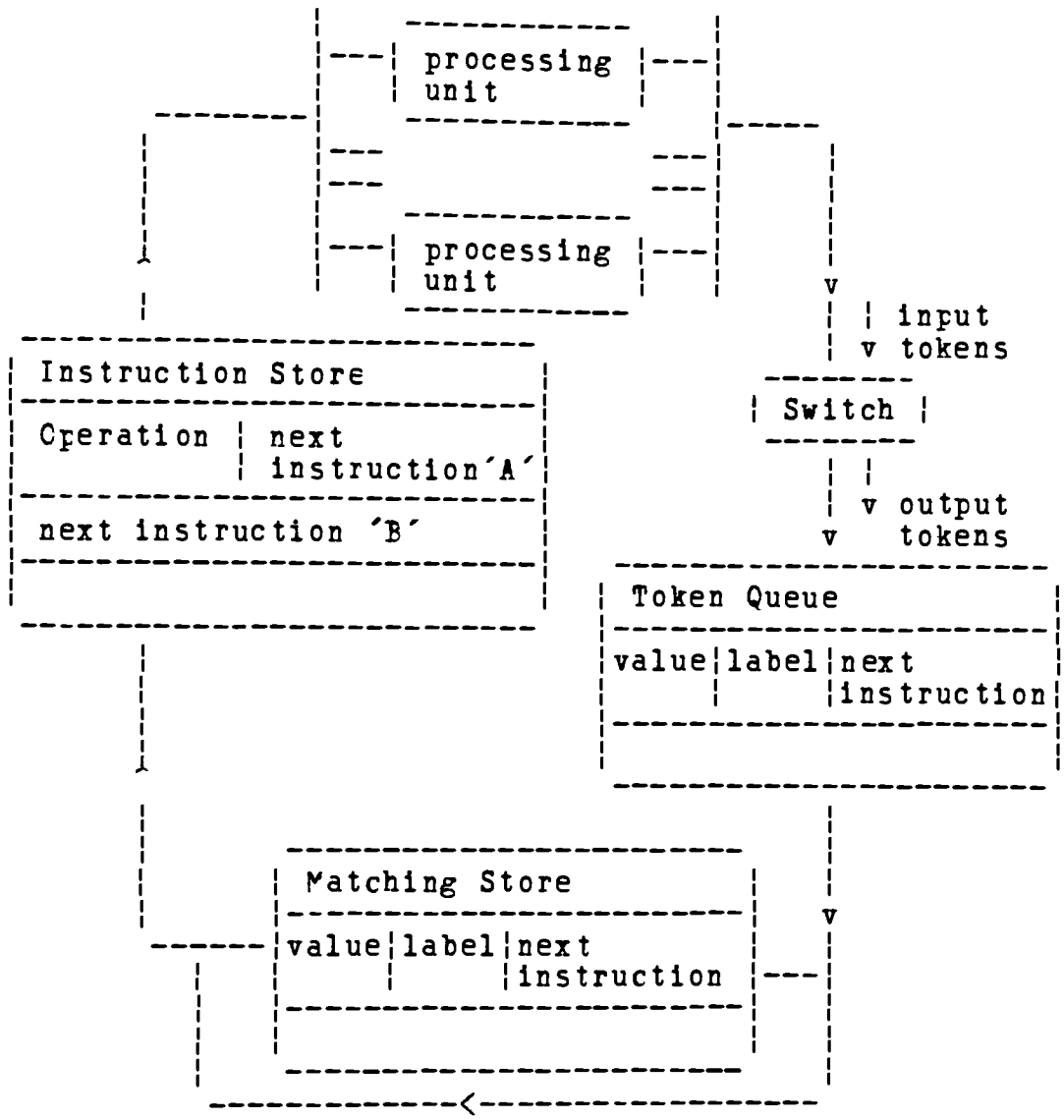


Fig. 2-27 Basic architecture

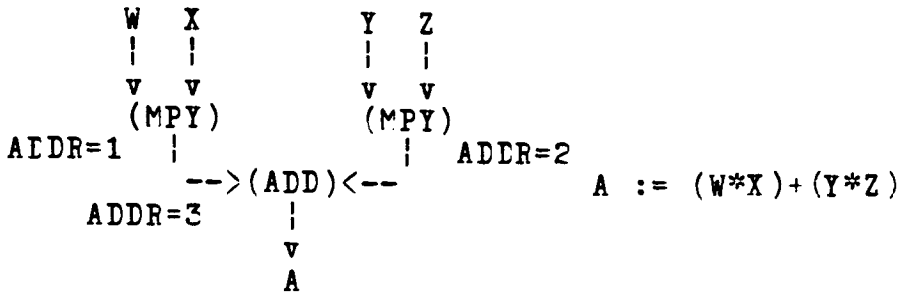
tion system, on receipt of an executable package, will select any processor which is free and allocate the nodal operation. The Switch provides input and output for the system. Initial tokens are directed to the starting nodes

of the computation. A special destination address in the final nodes of the graph allows tokens to be output.

The Token Queue is a FIFO buffer which equalizes data rates around the system. The Matching Store is associative in nature, although it is implemented using conventional random access store with hardware hashing techniques. The associative address is formed from a concatenation of the label and next instruction fields; the value field is the token value.

In order to execute a program, the graph description is entered into the instruction store. The initial data tokens are input into the Token Queue via the Switch. For example, Figure 2-28 shows a very simple graph to form the sum of the product of two pairs of numbers, together with an indication of the Instruction Store entries and initial token formats. The label is not used in this case and is thus set to zero.

The tokens, on reaching the front of the Token Queue, can access the Matching Store dependent on the Next Instruction Information. An access to the Matching Store will cause an associative search of the store. If a token is found with the same label and Instruction Address it is removed to form a token pair. If no match is found, the incoming token is written to the store.



Instruction store entries

Address	Operation	Next instruction 'A' Addr	Left/Right	Next instruction 'B' Addr	Left/Right
1	MPY	3	L	NONE	
2	MPY	3	R	NONE	
3	ADD	OUTPUT		NONE	

Initial tokens

Value	Label	Next Addr	instruction Ieft/Right
W	Ø	1	L
X	Ø	1	R
Y	Ø	2	L
Z	Ø	2	R

Fig. 2-28 A simple graph and its machine representation

Token pairs from the Matching Store now access the Instruction Store and form an executable package. This is distributed to any free processor for execution. Tokens produced by the processors are entered on the back of the Token Queue via the Switch.

This operation proceeds in a pipelined manner around the ring structure until the computation is complete and

all cutput tokens have been produced. Each unit communicates with its successor and predecessor by a two-way handshake interface. This ensures that if, for example, all processing units are busy, the ring operation is suspended until the necessary resources become available.

2.6.6. The MIT Architecture

Another example of the data flow model is the MIT data flow architecture which was presented by Dennis (1979)[25]. The structure of this machine is shown in Figure 2-29. Each instruction is equally accessible to result packets generated by any other instruction, independent of where they reside in the machine. The heart of this architecture is a large set of Instruction Cells, each cf which holds one activity template of a data flow program. Result packets arrive at Instruction Cells from the Distribution Network. Each Instruction Cell sends an operation packet to the Arbitration Network when all operands and signals have been received. The function of the Operation Section is to execute instructions and to forward result packets to target instruction by way of the Distribution Network.

---

[25]Dennis, J.B., Leung, C.K.C., and Misunas, D.P., "A Highly Parallel Processor Using a Data Flow Machine Language," Laboratory for Computer Science, M.I.T., CSG Memo 134-1,p.31, June 1979.

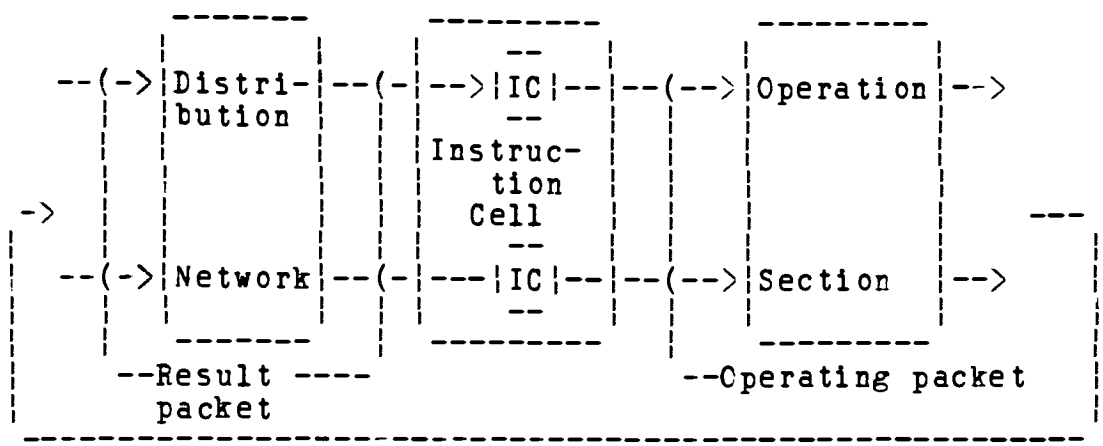


Fig. 2-29 MIT data flow processor

As shown in Figure 2-29, the Instruction Cells are grouped into blocks and each block realized as a single unit. Such an Instruction Cell Block has a single input port for each result packet and a single output port for each operation packet. Thus one Cell Block unit replaces many Instruction Cells together with the associated portion of the Distribution Network. In order to reduce the number of interconnections between Cell Blocks and other units, a byte-serial format for result and operation packets is chosen.

The resulting structure is shown in Figure 2-30. Several Cell Blocks are served by a shared group of functional units P1,..., Pk. The Arbitration Network in each section of the machine passes each operation packet to the appropriate functional unit according to its opcode.

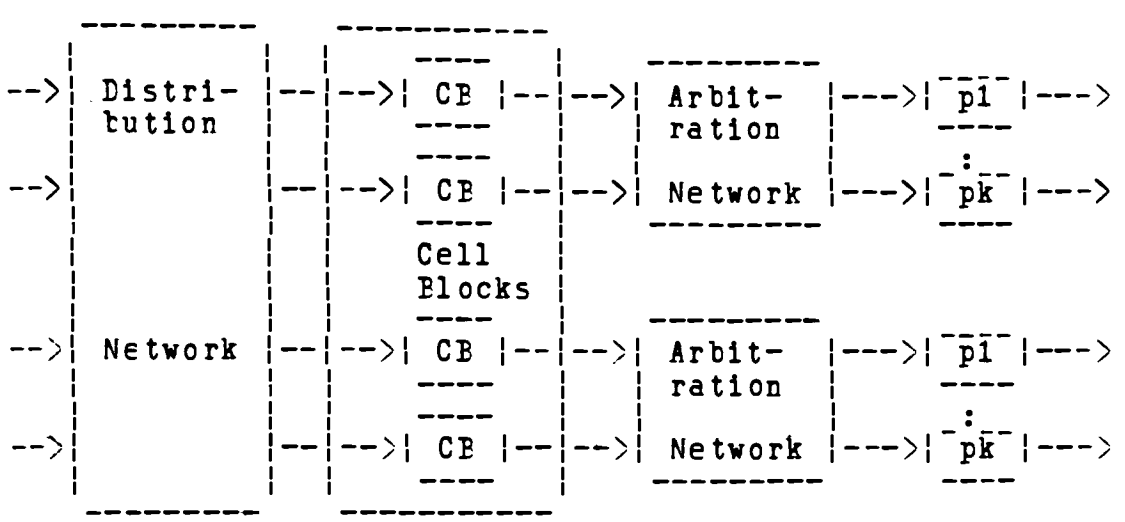


Fig. 2-30 Practical form of the MIT architecture

The number of functional unit types in such a machine is likely to be small or just one universal functional unit type might be provided in which case the Arbitration Network becomes trivial.

The relationship between the MIT architecture and the basic mechanism described earlier becomes clear when one considers how a Cell Block unit would be constructed. As shown in Figure 2-31 a Cell Block would include storage for activity templates, a buffer store for addresses of enabled instructions, and control units to receive result packets and transmit operation packets that are functionally equivalent to the Fetch and Update units of the basic mechanism. The Cell Block differs from the basic data flow processing element in that Cell Block contains no

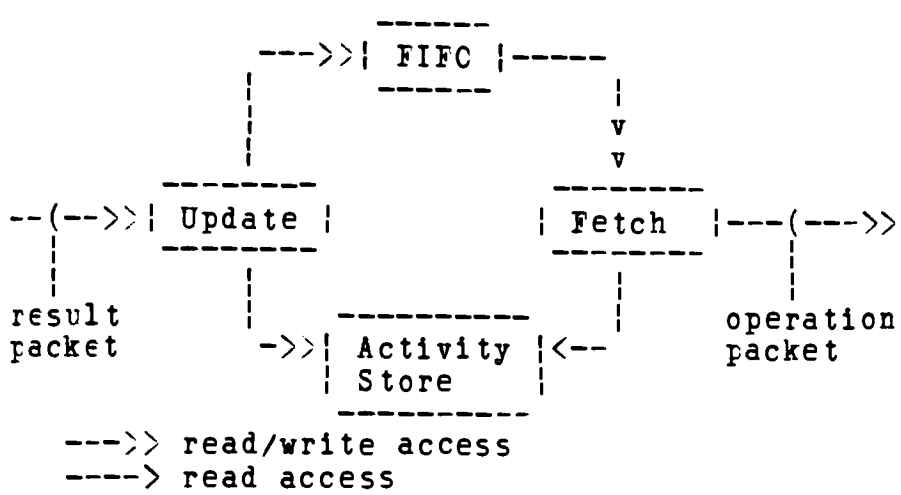


Fig. 2-31 Cell Block implementation

functional units, and there is no shortcut for result packets destined for successor instructions held in the same Cell Block.

2.7. Summary

In this chapter we investigated five possible models for describing distributed computations. The hierarchical model is the simplest one which can apply to the hierarchical systems. The main topics of the CPU cache model are how to balance the work load between the user's minicomputer and central machine and how to minimize interprocessor communication.

To eliminate the central machine and connect all the personal computers by a high speed local network is another approach of distributed computation. The user-

server model allows a user to request support specific functions from various centrally located server machines.

The pool processor model provides a way to fully utilize idle machines by having computations be carried out by a pool of processors.

Finally, the data flow model is a paradigm for highly concurrent and parallel distributed computation.

CHAPTER 3 EXAMPLES OF DISTRIBUTED COMPUTATION IMPLEMENTATIONS

In this Chapter, we will present seven examples of distributed computation that have been implemented or are in progress. We will describe the design and functions of each system, and then classify them into the five models presented in Chapter 2.

3.1 Task Allocation System - Distributed Computing System

3.1.1. Introduction

Ma and Lee (1982)[26] presented a task allocation system that allocates application tasks among processors in a distributed computing system. The system must allocate tasks among distributed processors to achieve the following goals:

(1) allow specification of a large number of constraints to facilitate a variety of engineering application requirements.

(2) balance the utilization of individual processors in the distributed computing system.

(3) minimize interprocessor communication cost.

The task allocation system should have a structure with supporting functions as shown in Figure 3-1 in order

---

[26]Ma, P.R., Lee, E.Y. and Tsushiya, M., "A Task Allocation Model for Distributed Computing Systems," IEEE Trans. on Comp., vol c-31, NO.1, Jan.1982. pp41-47.

to effectively satisfy these goals. We define items as follows:

Task preprocessor - The task preprocessor analyzes the application task to acquire relevant information such as coupling factors among tasks and task attribute sizes.

Coupling factor - The coupling factor between tasks is

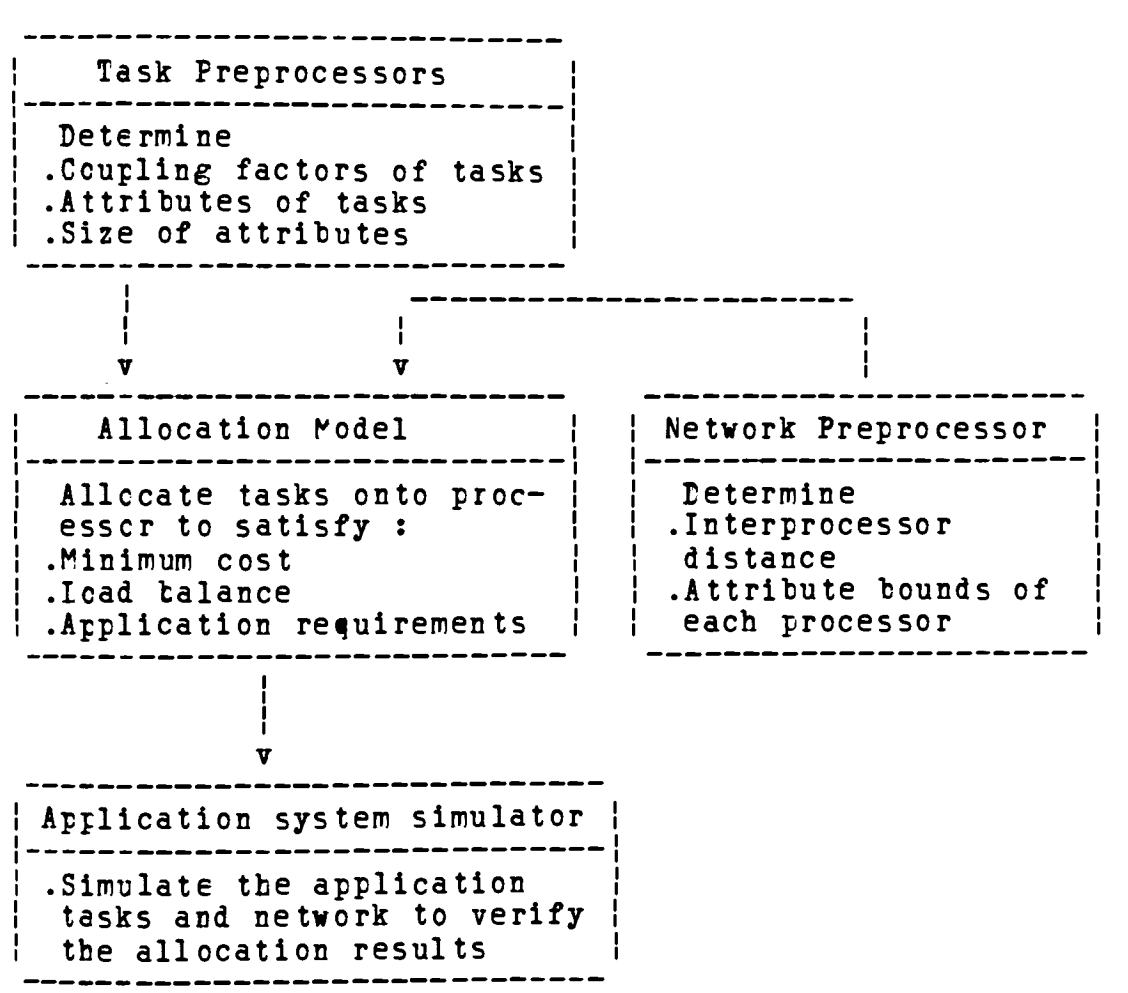


Fig. 3-1 General structure of the allocation system and its supporting functions

measured by the number of data units transferred from one task to another. The unit of data may be word, byte, or bit.

Attributes - The attributes are the inherent characteristics of tasks, for example, the number of executable statements, or the maximum allowable execution time.

Network preprocessor - The network preprocessor examines the distributed network and provides information on the network architecture which includes the interprocessor distance and processor constraints.

Interprocessor distance - The interprocessor distance is conceptually the physical distance between two processors. However, it may represent any type of cost that can be measured in time or in dollars.

Processor attributes - Processor attributes are the hardware constraints of the processor. For example, the attributes may be the MIPS (million instructions per second) rate, storage space, etc.

3.1.2 Functional design

In order to design a mathematical model for a task allocation system, we need three steps:

(1) formulate the cost function to measure the interprocessor communication (IPC) cost and processing cost.

(2) formulate a set of constraints to meet the diverse requirements.

(3) derive an iterative algorithm to obtain a minimum total cost solution.

A. Cost function

Cost function - The cost function is formulated as the sum of the IPC cost and the processing cost.

IPC cost - IPC cost is a function of both task coupling factors and interprocessor distance.

Coupling factor Cij - is the number of data units transferred from task i to task j.

Interprocessor distance Dkl - is certain distance-related communication costs associated with one unit of data transferred from processor k to processor l.

Interprocessor cost - if task i and j are assigned to processors k and l, respectively, the interprocessor cost is (Cij*Dkl).

Processing cost Qik - the cost to process task i on processor k. It can be used to control the processor assignment. For example, if task i must not be executed on processor k, a very large value can be assigned to Qik to inhibit the assignment.

Assignment variable - Xik = 1, if task i is assigned to processor k, Xik = 0, otherwise.

Total cost for processing the tasks is $\sum_i \sum_k (WQ_{ik}X_{ik} + \sum_i \sum_j$ (Cij * Dkl) XikXjl). The constant W is used to scale processing cost and IPC cost to account for any difference in

measuring units.

B. Constraints

In order to achieve load balance and meet application requirements, the task allocation system has to consider several constraints as shown in Table 2. The memory attribute is represented by $\sum_i MiXik <= Sk$, where Mi is the amount of memory required by task i, Sk is the memory capacity at processor k. This attributes states that the amount of memory required for all tasks assigned to a processor must not exceed the processor memory capacity.

The task preference matrix is represented by an m*n matrix P. m is the number of tasks and n is the number of processors, where Pik = Ø implies that task i cannot be

Bounded Attributes	Representation of the constraints associated with the tasks and/or processor, e.g., task size is bounded by available memory storage
Task Preference	A task is preferred to be allocated to a certain processor
Task Exclusion	Certain pairs of tasks must not be assigned to the same processor
Task Redundancy	Certain tasks must be assigned to two or more processors

Table 2 Allocation constraints

assigned to processor k, and Pik = 1 otherwise. The task exclusion matrix is represented by an m*m matrix E, where Eij = 1 implies that tasks i and j cannot be assigned to the same processor, Eij = Ø otherwise.

C. Algorithm derivation

The algorithm was derived from a branch and bound (BB) method[27]. To use the BB technique, the allocation problem is represented by a search tree. The allocation decision represents a branching at the node corresponding to the given task. Consider a problem of allocating m tasks among n processors. Staring with task 1, each task is allocated to one of the n processors subject to the constraints imposed on the relations on tasks and processors. The number of tree level m corresponds to m tasks. A path from the root node to the last node is a complete allocation; otherwise, it is a partial allocation. The cost of a path is computed according to the cost equation.

This has been modified for the task allocation algorithm. The following rules determine whether the selected branch i for a given node k should be eliminated.

(1) Rule F checks the preference matrix P for task k and prccessor i. If Pik = Ø, the branch is eliminated.

(2) Rule E checks the exclusive matrix for task k. If

---

[27]F.G.Coffman, Jr.,Ed., "Computer and Job-Shop Scheduling Theory," N.Y.:Wiley,1976.

task l which is mutually exclusive to task k has already been allocated to processor i, branch i for node k is eliminated.

(3) Rule RP checks the cumulative requirement of tasks against the processor memory capacity. As task k is assigned to processor i, its size is added to all tasks which are already in processor i. If the cumulative size of all tasks exceeds the processor's memory capacity, branch i for node k is eliminated.

(4) Rule D compares the partial cost L with the complete cost U. (Note: Lower bound function L assigns a cost lower bound to each partial solution. Cost upper bound U of a complete solution is known at the beginning of the algorithm.) If L is greater than U, the solution cannot be improved. Hence, branch i for node k is eliminated.

If branch i of node k satifies all these four rules, the allocation is made.

3.1.3. Classification - CPU cache model

The task allocation system seems to best fit the CPU cache model for the following reasons:

(1) The goals of allocating tasks among distributed processors are the same, that is,

(a) balance the workload of each processor.

(b) minimize the interprocessor communication cost.

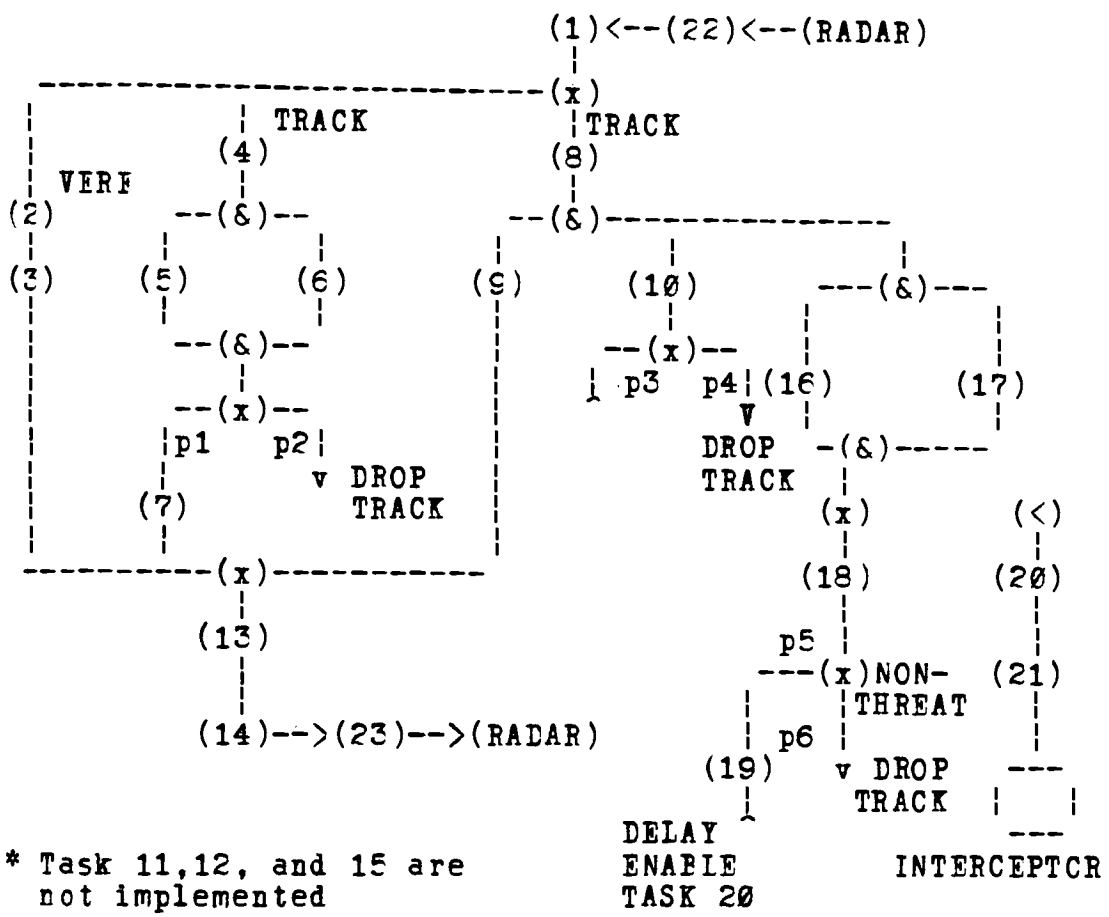
(2) In Chapter 2.3, we described graph theory to assign

modules to processors. In this example, we have a different approach to allocate 23 real-time software tasks into three processors.

3.1.4 Application

One example where the task allocation model could be applied is in an Air Defense (AD) system (see Figure 3-2). This described the data processing system interfacing with the radar and the interceptor. Sensor data from the radar are input to the system which tracks and keeps under constant surveillance all possible attacking reentry vehicles (RV) and commands the radar and the interceptor to prevent any RV from penetrating the defended area. 23 tasks are identified in Figure 3-2 (only 20 are shown, the other 3 are not implemented yet). These tasks are arranged into seven major processing threads, each consisting of a number of tasks as shown in Figure 3-3. Because of the real-time critical nature of this application, they must meet the port-to-port (PTP) processing times which confines the total execution time of the specified thread within a certain prescribed limit. Therefore, any allocation of the tasks to the processors must first satisfy these PTP time requirement in order to be qualified as a valid allocation.

A simulator[28] was designed to simulate the AD data



* Task 11,12, and 15 are not implemented

Fig. 3-2 Task flow diagram for a model AD application

THREADS	PTP DELAY LIMIT (MSEC)
(22,1,2,3,13)	100
(22,1,4,13)	50
(22,1,4,5,6,7,13)	40
(22,1,8,9,13)	40
(22,1,8,10)	40
(22,1,8,16,17,18)	40
(22,1,8,17,18,19)	50

Fig. 3-3 Major processing threads and port-to-port time requirements

processing system. The three processors were simulated as homogeneous, fully connected computers with 1.5 million instruction per second (MIPS) rate each. They also built the task sizes (in machine language instructions) and task coupling factors (in words) used as inputs into the simulator. Some examples are shown in Table 3. In the following, the task allocation model is applied to the AD data processing system and its results are compared with those obtained from engineering experience.

Allocation A (Figure 3-4a), based on the engineering experience, was designed to satisfy the PTP time by evently loading the tasks on the processors with an equal MIPS rate. Allocation B (Figure 3-4b) was based on the

---

[28]M.L.Green et al., "The DDP underlay simulation experiment: Tractical applications and d-RTOS model," TRW Defense Space Syst. Group, May 1980.

(A)

TASK	TASK SIZE(MIL)
1	200
13	1200
14	350
22	150
23	350

(B)

	13	14	20	21	23
13		1000			
14	1000				100
20				200	
21			200		
23		100			

Table 3 (A) Size of tasks(Machine Language Instructions)
(B) Couping factor among tasks(Words)

	A		
	p1	p2	p3
TASKS	1	2	3
	4	5	6
	7	8	9
	10	13	16
	17	14	19
	20	18	23
	22	21	

(a)

	B		
	p1	p2	p3
TASKS	13	2	1
	14	3	4
	17-19	6	5
		8	7
		9	10
		23	16
			20-22

(b)

Fig. 3-4 (a) Engineering experience allocation to achieve load balance
(b) Allocation from the allocation system

	A	B
PTP	PASS	PASS
CPU1	39%	40%
CPU2	55%	52%
CPU3	42%	40%
IPC	222	184

Table 4 Performance of allocation A and B as measured by the simulator

allocation algorithm derived previously. These two allocations A and B were tested on the simulator. The result is shown in Table 4. A satisfies the PTP time requirement and achieves good load balancing at the expense of high IPC cost. B shows similar load balancing effects, but at a lower IPC cost.

3.2 Central File System (CFS) - Distributed file system

3.2.1 Introduction

The Central File System (CFS) has been designed as a back-end file system for the computing environment in the Department of Computer Science at Carnegie-Mellon University. This environment currently consists of a variety of machine such as PDP-10s, VAX-11/780s, PDP-11s and Altos[29] connected together by a local network. The Spice[30] machines currently under development will be added to this environment when they become available.

The CFS will provide a central facility on which local file systems may store files and from which they may retrieve them. Files on the CFS are accessible to all machines on the local network. Besides permitting sharing of files, the CFS will also provide a means of indefinitely archiving the current and past versions of all such files.

The design of the CFS draws upon ideas from a variety of existing file systems. The directory structure is based on that of the UNIX file system. The stable storage and

---

[29] Tacker, C.P., McCreight, E.M., Lampson, B.W., Sproull, R.F. and Boggs, D.R., "Alto: A personal computer," Technical Report CSL-79-11, Xerox Palo Alto Research Center, August 1979.

[30] Department of Computer Science, Carnegie-Mellon U., "Proposal for a Joint Effort in Personal Scientific Computing," August 1980.

transactions are based on work done at Xerox and IBM. The Multics[31] and Hydra[32] file systems have also had an influence on the design of the CFS. Besides, the promise of extremely high storage density write-once media such as videc disks[33] has influenced the archival and migration features of the CFS.

The prototype system will be implemented on a VAX under the UNIX operating system but completely distinct from the UNIX file system. The first two file systems to be interfaced to the CFS will be the VAX UNIX file system and the Spice file system[34]. UNIX will treat the CFS as a central storage area for common system files and as the primary backup and archiving area for general files. The Spice file system is being designed with the intent of providing the Spice user with a local file system which resembles the CFS as closely as possible.

3.2.2. Functions of the CFS

The CFS provides users with a wide variety of services. CFS services will be provided over the network by

---

[31]Organick,E.L., "The Multics System:an Examination of its Structure," MIT Press, Cambridge, Mass.,1972.

[32]Almes,G.and Robertson,G., "An Extensible File System for Hydra," Proc.of 3rd Intern. Conf.on Software Engin. IEEE,May 1978.

[33]White,R.M.,"Disk Storage Technology," Scientific American 243(2), August 1980.

[34]Thompson,M.,Robertson,G.,Satyanarayanan,M. and Accetta,M. "Spice File System," Technical Report, Department of Computer Science, CMU 1980.

four server processes:

(1) Authentication Server - which handles the initial connection protocol between users and the CFS, and establishes the authenticity of the former.

(2) File Server - which provides access to the physical data in files.

(3) Name Server - which supports a directory structure.

(4) Archive Server - which interacts with the File Server to handle automatic archiving and migration function. It is invisible to user.

Communication - Interprocessor Communication Facility (IPC)[35] will be the sole means of communication between users and the CFS, and between the components of the CFS.

At the lowest level, the CFS will provide the common storage mechanism for files from all users. It will support files with differing data characteristics to simplify the sharing of compatible data between a variety of local file systems. At the higher level, the CFS may be viewed as an extension to each local file system. Basic functions of the CFS are:

(1) Creating and deleting files.

(2) Reading and writing file data.

(3) Updating file statistics and changing access

---

[35] Rashid, R.F., "An Inter-Process Communication Facility for UNIX," Technical Report CMU-CS-80-124, Department of Computer Science, CMU, March 1980.

rights.

3.2.3 Server machines

Authentication Server

In order to use the services of the CFS, a user must first establish a secure connection with it. This connection is set up by sending a login message, consisting of a CFS user name and password, to a public port of the Authentication Server. As a result of a successful login, three authenticated ports are built and returned: one to the File Server, another to the Name Server, and the third to the Authentication Server. These ports are used for all further communications with the CFS.

The connection to the CFS is terminated by sending a logout message to the Authentication Server port. This deallocates the File, Name, and Authentication Server ports and closes the connection in an orderly manner.

File Server

A file is stored on the CFS as an ordered collection of data units divided into pages containing a fixed number of those data units. A file may be stored in either standard storage or stable storage. The data pages of a file stored in standard storage are recorded in only one place on the CFS while the data pages of a file stored in stable

storage are recorded in two different places on the CFS. Files stored in stable storage are less likely to be damaged during a system crash than are files stored in standard storage. Because it is expensive to store and change data in stable storages, most files will be stored in standard storage. Only extremely important files will be stored in stable storage. The storage locations of stable pages are chosen so as to reduce the chance that both copies can be damaged at the same time (on different platters of a disk pack). The two physical copies of any stable logical page are always written by the File Server in the same order so that they can be brought into agreement after a crash when they are not identical.

Every file is uniquely identified by its file ID (FID). FID's are the only means by which files are identified to the File Server. The file has four properties:

(1) Mutability: {Variant, Invariant}

The mutability of a file determines whether the data in that file can be altered after file creation. An invariant file can never have its data altered, whereas variant files can be so altered.

(2) Storage: {Standard, Stable}

Standard storage files are archived soon after their creation on the CFS. Stable storage files are stored in two places on the CFS.

(3) Data: {Universal text, binary of various sizes}

The CFS will be connected to a various of machines with different word sizes and different software conventions regarding text files.

(4) Advisory File Type: {User-defined}

To create a new type, a user could create a file and enter a description of the type in that file. He could then try to enter the FID of this file as version one of the type name in the special directory.

Name Server

The Name Server provides three primary functions:

(1) It maps names chosen by users to unique identifiers.

(2) It provides a directory structure that aids users in organizing their files in a logical manner.

(3) It aids sharing of files by supporting shared directories and provides control over the extent of this sharing.

The Name Server maps name to values, and so allows user-defined entries as well as files. The directories are stored in stable storage as variant files. There is a name archive associated with each directory so that names would not be damaged during a system crash.

Archive Server

The primary function of the Archive Server is to maintain on tertiary storage copies of all files ever created by the File Server on secondary storage. The design of the Archive Server is based on the premise that extremely large (of the order of 10**10 bits per platter) randcm access write-once memories such as video disks will be ccmmercially available in the near future.

The only component of the CFS that is aware of the presence of the Archive Server is the File Server. The relationship between secondary storage on the File Server and tertiary storage on the Archive Server is analogous to that between cache and main memory when a write-through caching strategy is used. Alterations to secondary storage cause teritary storage to be updated. However, unlike a write-through cache, interactions between secondary and tertiary storage are asynchronous. There can be indeterminate delays between the queuing of a request by the File Server and its execution by the Archive Server. During this period the File Server may continue to attend to user processes and queue further requests to the Archive Server. On completion of a request, the Archive Server notifies the File Server of this fact.

We will classify CFS example and the next SWALLOW example together, because they both fit the user-server model.

3.3. SWALLOW - Distributed data storage system

3.3.1 Introduction

SWALLOW is a distributed data storage system that supports highly reliable long term storage of arbitrary sized data objects with special mechanisms for implementing multi-site atomic action. SWALLOW[36], was developed at MIT, is an integrated system of servers that provides reliable, secure and efficient storage for clients through a network. The components of SWALLOW are repositories, authentication servers and brokers. Each item is described as follows:

(1) Repository - A repository is a server that provides very reliable storage for client data in SWALLOW. It is a processor that is connected to a configuration of storage devices.

(2) Authentication server - An authentication server acts as intermediary to ensure that all communications within SWALLOW are secure.

(3) Broker - A broker is a module in the client node that acts as an interpreter for client requests. It mediates interactions between the clients and servers in SWALLOW.

---

[36] Reed, D.P., Svobodova.L., "SWALLOW:A Distributed Data Storage System for a Local Network," Intern. Workshop on Local Networks by IBM Zurich Lab, August 1980.

Figure 3-5 shows the general configuration of SWALLOW in relationship to its clients. SWALLOW has five basic features:

(1) Reliability - It provides extremely reliable storage. Thus the probability that any client objects will ever be lost is near zero.

(2) Generality - SWALLOW enables the clients to perform any number of accesses (read and write) on an arbitrary set of objects as a single, indivisible operation.

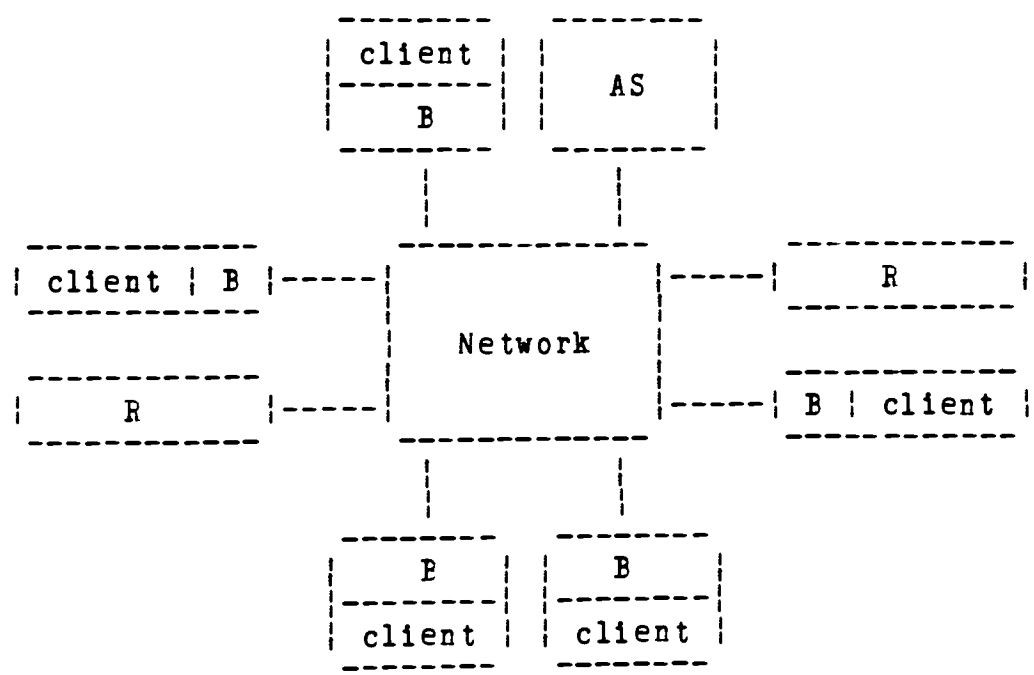


Fig. 3-5 Configuration of SWALLOW

(3) Security - SWALLOW protects all objects from unauthorized access, using encryption-based mechanisms.

(4) Distribution - SWALLOW provides a uniform interface for accessing the objects which may be distributed over a local node and/or several remote repositories. In effect, the clients can specify where they would like each object to be stored, but need not remember the location in order to access the object.

(5) Flexibility - SWALLOW supports objects of any size, and in particular, very small objects. Thus, SWALLOW gives the client flexibility in structuring and managing its data, since each object is treated as a separate entity with respect to protection and synchronization as well as with respect to storage and retrieval.

3.3.2. SWALLOW mechanisms

SWALLOW is intended to be a very reliable storage system. Basically, it is a set of protocols that allow for proper management of data that may distributed over the local node and several remote repositories. There are various underlying mechanisms that are used in order to implement these protocols. These mechanisms are based on those described by Reed[37].

---

[37] Reed, D.P., "Naming and Synchronization in a Decentralized Computer System," PhD Thesis, MIT, September 1978.

In SWALLOW, the functional unit of client data is called an object. The fundamental requests that a client can submit to SWALLOW (through a broker) to be performed on an object are:

Create Object : write a new object into storage.

Delete Object : eliminate an object from storage.

Read Object : return the current value of an object.

Modify Object : assign a new value to an object and write it into storage.

3.3.3. Comparison of CFS with SWALLOW

Similarity

CFS is a collection of various types of servers that cooperate in order to provide a single, coherent system. Also, CFS makes the location of the data distributed over the local and remote nodes transparent to the clients, as does SWALLOW.

Difference

The most fundamental difference between CFS and SWALLOW lies in the amount of flexibility the client is given for structuring his data.

SWALLOW supports arbitrarily small objects and allows the client to access these objects in whatever faction suits the particular application. Thus, SWALLOW provides separate protection for every object. Besides, SWALLOW provides synchronization for accesses to any arbitrary set

of objects.

CFS forces the client to structure and access his object within the confines a file system. Thus, CFS only provides protection for files as a whole. Furthermore, CFS provides synchronization for access to arbitrary sets of objects within a single file.

3.3.4. Classification - User-Server model

CFS and SWALLOW both fit the user-server model for the following reasons:

(1) CFS consists of Alto personal computers and four server processes: Authentication Server, File Server, Name Server, and Archive Server. That fits user-server model in Figure 1-3. Note: CFS has the Archive Server which we did not present in user-server model.

(2) SWAILOW consists only two servers: Authentication Server and Repository Server. The function of Repository Server is the same as that of file server in user-server model. Note: SWALLOW contains a new component - Broker which mediates interactions between the clients and servers.

3.4. WORM - Distributed computation

3.4.1 Introduction

The WORM programs were an experiment in the develcpment of distributed computations - programs that would

span machine boundaries, and also replicate themselves in idle machines. A WORM is composed of multiple segments each running on a different machine. The WORM maintenance mechanisms were responsible for maintaining the WORM - finding free machines when needed, and replicating the program for each additional segment.

A WORM is simply a computation which lives on one or more machines. The programs on individual computers are described as the segments of a WORM. The segments in a WORM remain in communication with each other; should one of those segments fail, the remaining pieces have the task of finding another free machine, initializing it, and adding it to the WORM. As segments (machines) join and then leave the computation, the WORM itself seems to move through the network. The WORM mechanism is used to gather and maintain the segments of the WORM, while actual user programs are then built on top of this mechanism.

The WORM programs were constructed at the Xerox Palo Alto Research Center. This includes over 100 Alto computers each connected to an Ethernet local network[38]. In addition, there is a diverse set of specialized network servers, including file system, printers, boot-servers, name-lookup servers, and other utilities. The whole system

---

[38]Metcalfe,R.M.,Boggs,D.R.,"Ethernet: Distributed Packet Switching for Local Computer Networks," Communications of the ACM, July 1976.

is connected together with the PUP architecture of internetwork protocols[39]. This system can be viewed as a 100-element multiprocessor to run a program.

3.4.2 Functions of the WORM

The first task of a WORM is to fill out its full complement of systems. In order to do that, a WORM must find some number of idle machines. We define a protocol to aid in this process, that is, a special packet format is used to inquire if a host is free. These inquiries could be broadcast to all hosts, or transmitted to specific destinations. Since multiple WORMs might be competing for the same idle machines, we could use a series of specific probes, addressed to individual machines. In order to determine which possible host to probe next when looking for an additional segment, we have the following procedure : a segment begins with its own local host number, and simply works its way up through the address space. A segment sends packets to successive hosts until it finds one that is idle; at that point the program is copied to the new segment and this host begins probing for next segment.

After finding an idle machine, a WORM segment then asks it to go through the standard network boot procedure.

---

[39] Boggs, D.R., Shoch, J.F., Taft, E.A., and Metcalfe, R.M., "PUP: An Internetwork Architecture," IEEE Trans. on Communications, April 1980.

In this case, however, the specified source for the new program is the WCRM segment itself. Thus, we have following sequences:

1. Existing segment asks if a host is idle.

2. The host answers that it is.

3. The existing segment asks the new host to boot through the network, from the segment.

4. The newcomer uses the standard PUP procedures for requesting a boot file.

5. The Easy File Transfer Protocol (EFTP) is used to transfer the WORM program to the newcomer.

In general, the program sent to a new segment is just a copy of the program currently running in the WORM; this makes it easy to transfer any dynamic state information into new segments.

3.4.3. Classification - Pool processor model

WCRM programs fit the pool processor model for the following reasons:

(1) WORM includes 100 Alto computers as pool processors connected to an Ethernet local network. Besides, there are several fixed function server machines including file servers, printers, boot-servers, and name-lookup servers. Note: WCRM has boot-server which we did not present in pool processor model.

(2) WORM uses bidding algorithm to broadcast inquires

to all hosts. This fits the scheduling algorithm described in Sec 2.5.4.

3.5 LOCUS - Distributed operating system

3.5.1. Introduction

LOCUS is a distributed operating system that provides a very high degree of transparency and supports high performance as well as automatic replication of storage.[40] LOCUS was designed at UCLA and its goals are as follows:

(1) making the development of distributed applications easy,

(2) designing for high reliability, availability and good performance of distributed system.

It is application code compatible with UNIX and runs on DEC-11s, model 44,45 and 70, connected by a high bandwidth, low delay 1 megabit local ring network.

Network transparency is one of the characteristics of LOCUS. Network transparency could be viewed as requiring all resources to be accessed in the same manner independent of their location. If open (file-name) is used to access local files, it also is used to access remote files. That is, the network becomes invisible, in a similar manner to the way that virtual memory hides secondary

---

[40]Popek,G.,Walker,B.,Chow,J.,Edwards,D.,Kline,C., Rudisin,G. Thiel,G., LOCUS: A Network Transparent, High Reliability Distributed System, UCLA, February 1981.

storage.

3.5.2. System architecture

There are two significant levels of abstraction in the LCCUS file system, which serves as the heart of the naming mechanism. Low level names in LCCUS are globally unique. The name space is organized around the concept of file groups. As in UNIX, each file group is composed of a small set of file descriptors which serve as a simple low level directory and a large number of standard sized data blocks. A file descriptor contains pointers to the data blocks which compose that file. Therefore, a low level file name is a <file group number, file descriptor number>. The application visible naming hierarchy is implemented by having certain low level files serve as high level directories. An entry in such a directory contains an element of a path name and the index of the file descriptor corresponding to the next directory or data file in the path.

The collection of file groups therefore represent a set of naming trees. These are patched together to form the single, network wide naming tree by system code and a network wide mount table.

A centralized synchronization protocol with distributed recovery was chosen for ICCUS. To open a file, a

request is made to the Current Synchronization Site (CSS) for a given file group. This site is responsible for coordinating access to the files contained in the associated file group. However, it is not necessary for the CSS to be the site from which data access is obtained. Any site which has a copy of the file can support this open request; that is, a particular Storage Site (SS) is designated to support an open request at the time of the open.

The two most significant specific conclusions which were drawn from the LOCUS experience are:

(1) High performance network transparency in a local network is feasible.

(2) Network transparency in a local network possesses so many advantages that a choice not to adopt it ought to be very carefully justified.

3.5.3. Classification - Integrated model

LOCUS does not fit easily into any of the five models we described in Chapter 2. An alternative to the user-server model is to design each machine's software as a complete facility, with a general file system, name interpretation mechanism, etc. Each machine in the local network would run the same software, so that there would be only one implementation. The system would be highly configurable, so that adaptation to the nature of the supporting hardware as well as the characteristics of use

could be made. IOCUS fits into this model, which we will call an integrated model.

3.6. Backend Storage Networks (BSN)

3.6.1. Introduction

A backend storage network (BSN) is a logical subnetwork within a general purpose high-performance local computer network. According to Watson (1980)[41], a BSN is the collection of interconnected equipment that provides shared storage services to a set of general purpose host computers.

The Lawrence Livermore Laboratory (LLL) Octopus network was designed as a BSN (see Figure 3-6) with a set of host computers connected to several functional hardware subnetworks. At least one of these subnetwork is the 'frontend' terminal network, while the others are high-performance 'backend' shared peripheral networks. The Octopus network has several deficiencies:

(1) Lack of hardware modularity - The addition of a new kind of host or new functional subnetwork requires several special interfaces.

(2) Lack of software or protocol modularity - Each functional subnetwork tends to have its own special proto-

---

[41]Watson,R.W.,"Network architecture design issues for backend storage networks," Compcon, pp.224-234,Spring 1980.

col.

(3) Difficulties of host-host or subnetwork-subnetwork intercommunication.

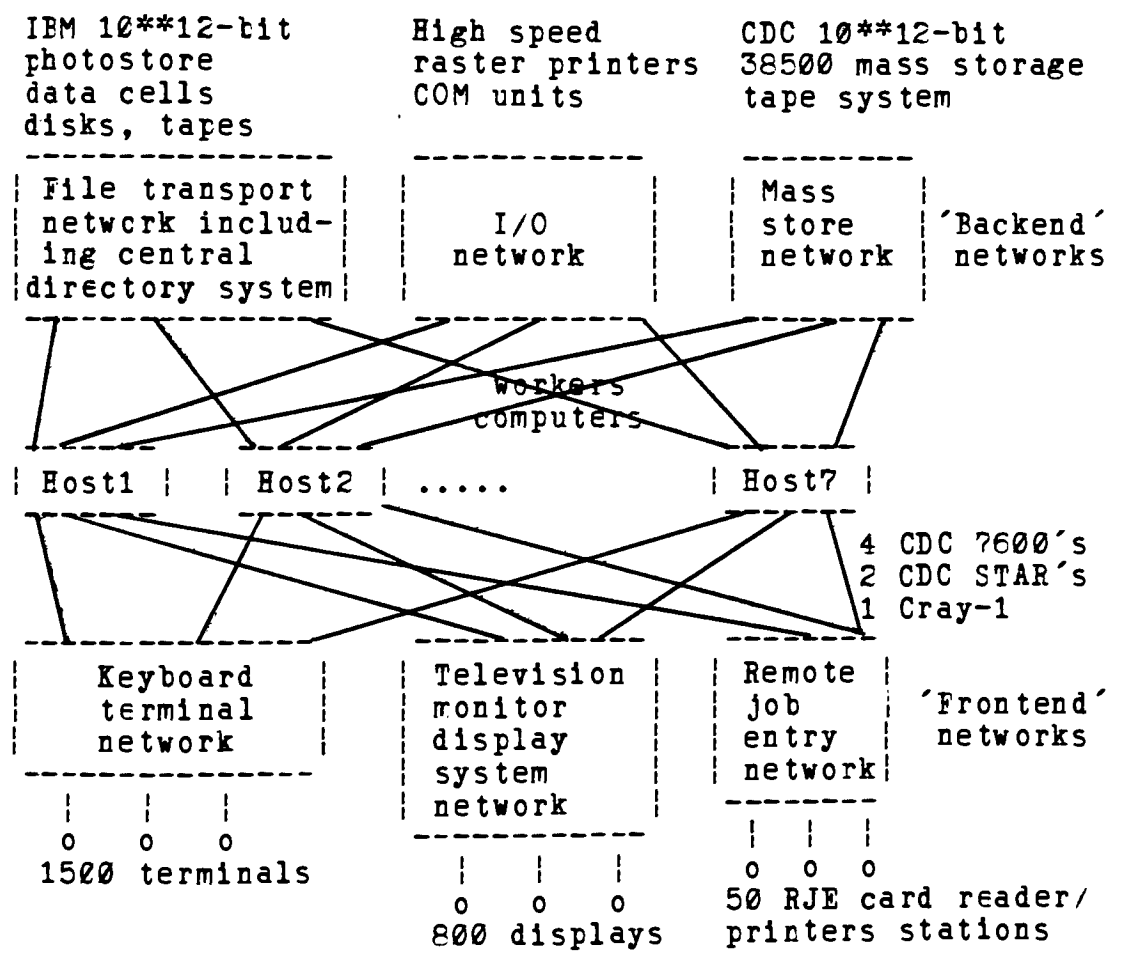


Fig. 3-6 Recent octopus network structure

Tc solve these problems, Watson is evolving the BSN system toward the topological structure shown in Figure 3-7. The important elements of this future network environment are:

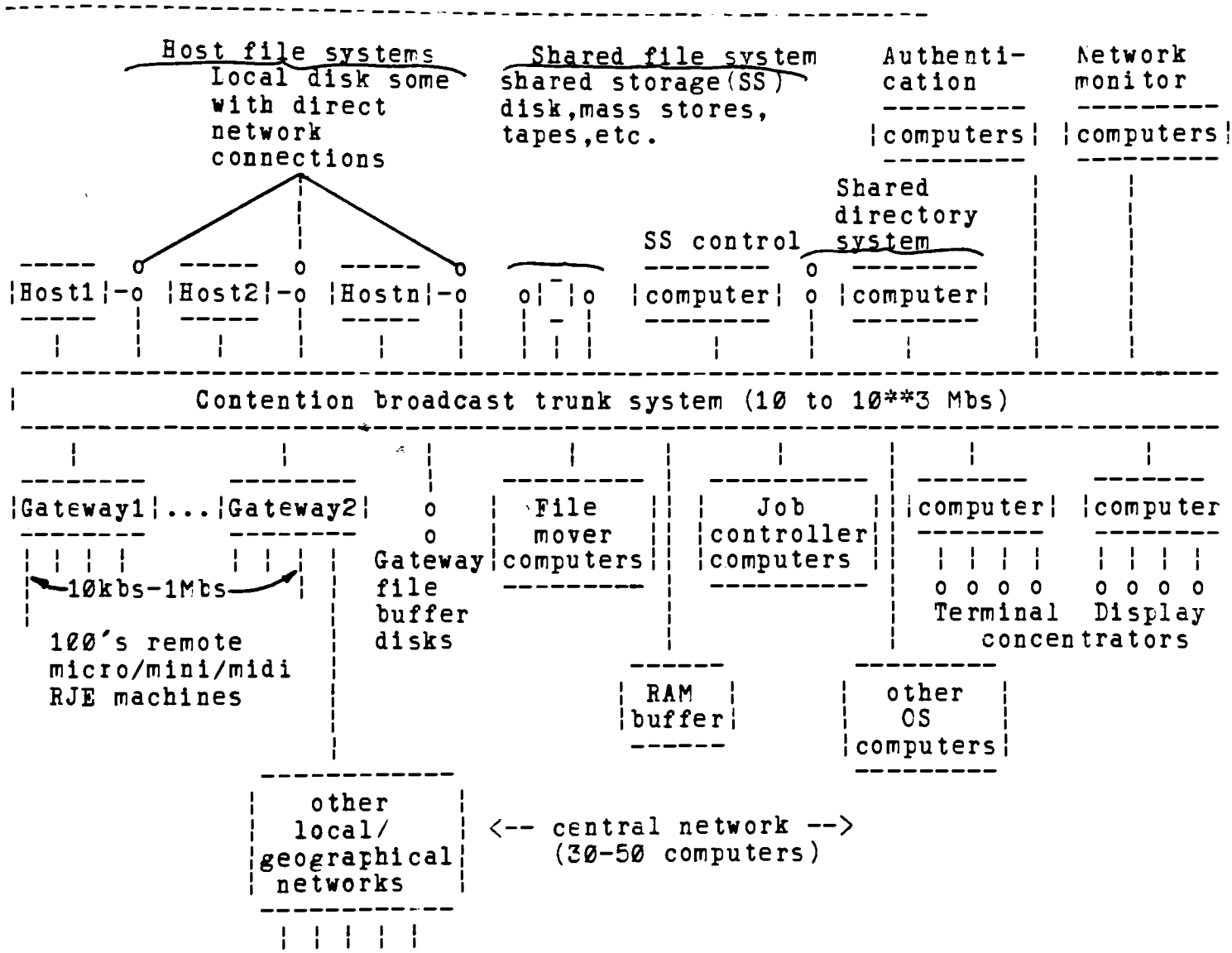


Fig. 3-7 Future structure toward which the octopus network is evolving

(1) direct network interconnectivity among all systems.

(2) a central high-performance local network providing large-capacity processing and storage services.

(3) a remote or satellite environment consisting of other local networks and remote computers.

3.6.2. Classification -Combined User-Server Model and Pool Processor Model

BSN seems to best fit a combined user-server model and pool processor model for the following reasons: it allocates some resources (processors, disks) permanently to each user and remains other resources in a pool to be allocated on demand.

3.7. File servers for network-based distributed systems

3.7.1. Introduction

Several file servers and data storage systems designed and developed in recent years will be briefly overviewed in this section. The first two systems - CFS and SWALLOW, we have already described in section 3.2 and 3.3. The purpose of this example is not to give a complete description of each system, but to present the overview. A reader interested in a specific system is advised to read the original papers listing below.

(1) CFS (Carnegie-Mellon Central File System)

{Accetta (1980)[42], Ball (1982)[43]}

(2) SWALLOW (MIT SWALLOW)

{Reed (1980)[44], Svobodova (1980)[45],
Svobodova (1981)[46], Arens (1981)[47]}

(3) WFS (Woodstock File Server)

and its client-based extension

{Swinehart (1979)[48], Paxton (1979)[49]}

(4) XDFS (Xerox Distributed File System)

---

[42]Accetta,M.,Robertson,G.,Satyanarayanan,M.,Thompson, M., "The design of a network-based central file system," Tech.Rep. CMU-CS-80-134,Dep. of Computer Science, Carnegie-Mellon U. Pittsburgh,Penn.,Aug. 1980.

[43]Ball,E.J.,Barbacci,M.R.,Fahlman,S.E.,Harbison,S.P., Hibbard,P.G., Rashid,R.F.,Robertson,G.G., and Steele,G.L.Jr"The SPICE project," Computer Science Research Review 1980-1981, Dep. of Computer Science, CMU, Pittsburg, Penn., 1982, 5-36.

[44]Reed,D.,Svobodova,L.,"SWALLOW: A distributed data storage system for a local network,"in Proc.Int.Workshop Local Networks,Zurich, Switzerland,Aug. 1980,pp.355-373.

[45]Svobodova,L.,"Management of the object histories in the SWALLOW repository,"Tech.Rep. MIT/ICS/TR-243, Lab. for Computer Science,MIT,Cambridge,Mass.,July 1980.

[46]Svobodova,L.,"A reliable object-oriented data repository for a distributed computer system,"in Proc. 8th ACM Symp. Operating systems principles,ACM SIGOPS Operating Syst. Rev. 15,Dec. 1981,pp.47-58.

[47]Arens,G.C.,"Recovery of the SWALLOW repository," Tech.Rep. MIT/LCS/TR-252,Lab. for Computer Science,MIT, Cambridge,Mass.,Jan. 1981.

[48]Swinehart,D.,McDaniel,G.,Boggs,D.,"WFS: A simple shared file system for a distributed environment," in Proc. 7th ACM Symp. Operating Systems Principles,ACM SIGOPS Operating Syst. Rev. 13, 4,Dec 1979,9-17.

[49]Paxton,W.H.,"A client-based transaction system to maintain data integrity,"in Proc. 7th ACM Symp. Operating Systems Principles,ACM SIGOPS Operating Syst. Rev. 13, 4,Dec. 1979,18-23.

[50]Lampson,B.W.,Sturgis,H.E.,"Crash recovery in a distributed data storage system,"Tech.Rep.Xerox Palo Alto Research Center, Palo Alto,Calif.,Apr. 1979.

{Lampson (1979)[50], Strugis (1980)[51],
Mitchell (1982)[52], (1982)[53]}

(5) UCFS (University of Cambridge File Server)

{Dion (1980)[54], Dion (1981)[55],
Garnett (1980)[56], Dellar (1980)[57],

(6) FELIX (Bell-Northern Research)

{Fridrich(1981)[58]}

(7) RSS (Research Storage System)

{Gray (1978)[59], Gray (1981)[60],
Williams (1981)[61]}

---

[51]Sturgis,H.E.,Mitchell,J.G.,Israel,J.E.,"Issues in the design and use of a distributed file system," ACM SIGOPS Operating Syst. Rev. 14, July 1980,pp.55-69.

[52]Mitchell,J.G.,Dion,J.,"A comparison of two network-based file servers," Commun. ACM 25, April 1982,pp.233-245.

[53]Mitchell,J.G.,"File servers for local area networks," Lecture Notes,Course on Local Area Network, U. of Kent, Canterbury,England,March 1982,pp.83-114.

[54]Dion,J.,"The Cambridge file server," ACM SIGOPS Operating Syst. Rev. 14,Oct. 1980,pp.26-35.

[55]Dion,J.,"Reliable storage in a local network," Tech.Rep. No. 16,U. of Cambridge,Computer Laboratory, Cambridge,England,Feb. 1981.

[56]Garnett,N.H.,Needham,R.M.,"An asynchronous garbage collection for the Cambridge file server," ACM SIGOPS Operating Syst. 14, Oct. 1980,pp.36-40.

[57]Dellar,C.,"Removing backing store administration from the CAP operating system," ACM SIGOPS Operating Syst. Rev. 14, Oct. 1980,pp.41-49.

[58]Fridrich,M.,Older,W.,"The FELIX file server," in Proc. 8th ACM Symp. Operating Systems Principles, ACM SIGOPS Operating Syst. Rev. 15,Dec. 1981,pp.37-44.

[59]Gray,J.N.,"Notes on database operating systems," Lecture Notes in Computer Science 60, Springer-Verlag, 1978,pp.393-481.

[60]Gray,J.N.,McJones,P.,Blasgen,M.W.,Lorie,R.A.,Price, T.G. Putzulu,G.F.,Traiger,I.L., The recovery manager of a data management system," ACM Comput. Surv. 13, June 1981,pp.223-242.

[61]Williams,R.,Daniels,D.,Haas,I.,Lapis,G.,Lindsay,B.,

3.7.2. Overview

(1) CFS from the Carnegie-Mellon University puts strong emphasis on the filing and access control aspects. On the data-manipulation side, CFS offers two different kinds of update: one is an update of an entire file which creates a new version of the file; the other is a transaction update that allows several transactions to use and update the same version of a file. Transactions are limited to a single file. CFS includes various simple mechanisms which the clients can use to build customized filing systems and multiple file transactions. This system also includes automatic archiving.

(2) SWALLOW is a data storage system designed and currently being implemented at the Laboratory for Computer Science of MIT. SWALLOW provides atomic transactions on multiple files and multiple servers. It can also handle even small objects efficiently so that they can be stored as separately nameable and retrievable entities.

(3) WFS, developed at Xerox Palo Alto Center, was one of the first dedicated network-based file servers. WFS provides a page-level access to remote shared files. A client can lock a file, but the original WFS did not

---

Ng,P., Obermark,R.,Selinger,P.,Walker,A.,Wilms,P., and Yost,R., "R*: An overview of the architecture," Tech.Rep. RJ3325, IBM San Jose Res. Lab., San Jose, Calif., Dec. 1981.

include any mechanisms to ensure or to help implement atomicity with respect to failures. Later in 1979, the extended WFS system (EWFS) was designed to support atomic transactions on multiple files and multiple servers using mechanisms similar to the mechanisms used in XDFS.

(4) XDFS, also developed at Xerox Palo Alto Research Center, was a research project in designing very robust multiple-server systems, using a two-phase commit protocol. It provides a base for general database-oriented applications. Clients can read and write small sequential sets of bytes within a file. Atomic transactions may involve multiple files on the same or different servers. XDFS uses novel time-limited breakable locks for concurrency control.

(5) UCFS, developed at the University of Cambridge, is an attempt to implement a universal file server for clients that are general-purpose operating systems. Special objects called indices that can be used to build file directories are implemented. It is used by two different operating systems, each of which implements its own filing system using the UCFS indices. It also supports the virtual memory of one of these operating systems[62] but provides atomic transaction on single files only.

---

[62]Wilkes,M.V.,Needham,R.M.,"The Cambridge CAP computer and its operating system," Operating and Programming System Series, Elsevier/North Holland, 1979.

(6) FELIX, developed at Bell-Northern Research, is a general-purpose file server that provides atomic transactions on multiple files, but transactions are currently limited to a single server. Every time a file is updated, FELIX creates a new version of the file. It supports the notion of version copies: clients can get their own read-only or writable copy without having to copy the actual data into another file. At the beginning of the transaction, clients can declare all of the files to be included in a transaction. In such a case, it is guarantted that the transaction will not be aborted because of a conflict with another transaction.

(7) RSS was developed at the IBM San Jose Research laboratory as part of the system R, a relational database management system. It is neither a server nor a system that contains a server; it runs on the same machine as its clients. RSS is the part of the system R which is responsible for transaction management and database recovery. RSS provides atomic transactions on multiple files. A distributed system R, called R*, which extends atomic transactions to multiple machines, is under development. Because of its thorough design of transaction management and recovery functions, RSS was a pioneer project that demonstrated usefulness and feasibility of the transaction concept.

3.8. Summary

In this chapter, we present seven examples of distributed computations. Some of which fit into five models we described in Chapter 2, they are:

(1) Task allocation system - CPU cache model

(2) CFS and SWALLOW - User-Server model

(3) WORM - Pool processor model

(4) BSN - Combined user-server and pool processor model

LCCUS in example 5 does not fit easily into any of the model. It is an alternate to the user-server model, which we call an integrated model. As several file servers were designed and developed in universities and industrial fields, it is advisable that file server is the high light of user-server model. That is why we presented it in example 7.

BIBLIOGRAPHY

1. Abraham, S.M., and Dalal, Y.K.: "Techniques for Decentralized Management of Distributed Systems," Compcon, pp. 430-437, Spring 1980.

2. Accetta, M., Robertson, G., Satyanarayanan, M., Thompson, M., "The design of a network-based central file system," Tech.Rep. CMU-CS-80-134,Dep. of Computer Science, Carnegie-Mellon U. Pittsburgh,Penn.,Aug. 1980.

3. Almes,G.and Robertson,G., "An Extensible File System for Hydra," Proc.of 3rd Intern. Conf.on Software Engin. IEEE,May 1978.

4. Ardrew S. Tanenbaum. "Computer Networks," Prentice Hall, pp. 460-476, 1981.

5. Arens, G.C., "Recovery of the SWALLOW repository," Tech.Rep. MIT/LCS/TR-252, Lab. for Computer Science, MIT, Cambridge, Mass., Jan. 1981.

6. Arvind, and Gostelow, K.: "A Computer Capable of Exchanging Processing Elements for Time," proc. IFIP congr.77, pp. 849-854, 1977.

7. Ball, E.J., Barbacci, M.R., Fahlman, S.E., Harbison, S.P., Hibbard, P.G., Rashid, R.F., Robertson, G.G., and Steele, G.L.Jr "The SPICE project," Computer Science Research Review 1980-1981, Dep. of Computer Science, CMU, Pittsburg, Penn.,

1982, 5-36.

8. Boggs,D.R.,Shoch,J.F., Taft,E.A., and Metcalfe, R.M., "PUP:An Internetwork Architecture," IEEE Trans. on Communications, April 1980.

9. Bokhari, S.H., "Dual-Processor Scheduling with Dynamic Reassignment", IEEE Trans. on Software Engineering, vol. SE-5, No 4, pp.338-349, July 1979.

10. Chu, W.W., "Optimal file allocation in a multiple computing system", IEEE Trans.Comput.,vol.C-18, pp885-889, Oct. 1969.

11. Clark, D.D., and Svobodova, L.: "Design of Distributed System Supporting Local Autonomy," Compcon, pp. 438-444, Spring 1980.

12. Coffman, E.G., Jr., Ed., "Computer and Job-Shop Scheduling Theory," N.Y.:Wiley,1976.

13. Dellar,C.,"Removing backing store administration from the CAP operating system," ACM SIGOPS Operating Syst. Rev. 14, Oct. 1980,pp.41-49.

14. Dennis, J.B.: "The Varieties of Data Flow Computers," First Int. Conf. Data Flow Comput., IEEE, pp. 430-431, 1979.

15. Dennis, J.B., Leung, C.K.C., and Misunas, D.P., "A Highly Parallel Processor Using a Data Flow Machine Language," Laboratory for Computer Science, M.I.T., CSG Memo 134-1,p.31, June 1979.

16. Department of Computer Science, Carnegie-Mellon U.,

"Proposal for a Joint Effort in Personal Scientific Computing," August 1980.

17. Lion, J., "The Cambridge file server," ACM SIGOPS Operating Syst. Rev. 14,Oct. 1980,pp.26-35.

18. Dion, J.,"Reliable storage in a local network," Tech.Rep. No. 16,U. of Cambridge,Computer Laboratory, Cambridge,England,Feb. 1981.

19. El-Dessouki, O.I. and Huan, W.H., "Distributed enumeration on network computers", IEEE Trans.Comput., vol.C-29, pp818-825, Sept, 1980.

20. Enslow, P.H., Jr.,"What is a distributed data processing system?", Computer, vol 11, pp.13-21, Jan. 1978.

21. Farber, D.J., and Larson, K.C.: "The System Architecture of the Distributed Computer System- the Communication System," symp. on Comput. Networks, Polytechnic Institute of Brooklyn, April 1972.

22. Farber, D.J. and Larson, K.C., "The Structure of a Distributed Computing System - Software," Proc.Symp. Computer-Communications Networks and Teletraffic, Polytechnic Press, 1972.

23. Farber, D.J., Larson,K.C., "The Distributed Computing System." Conference Digest, IEEE COMPCON Spring, 1973, pp. 31-34.

24. Fridrich,M.,Clder,W.,"The FELIX file server," in Proc.

8th ACM Symp. Operating Systems Principles, ACM SIGCPS Cperating Syst. Rev. 15,Dec. 1981,pp.37-44.

25. Garnett, N.H., Needham, R.M.,"An asynchronous garbage collection for the Cambridge file server," ACM SIGCPS Cperating Syst. 14, Oct. 1980,pp.36-40.

26. Gostelow, K.P. and Thomas, R.E., "A View of Data Flow", AFIPS Conf. Proc. NCC, pp629-636, 1979.

27. Gray,J.N.,"Notes on database operating systems," Lecture Notes in Computer Science 60, Springer-Verlag, 1978,pp.393-481.

28. Gray,J.N., McJones, P., Blasgen, M.W., Lorie, R.A., Price, T.G. Putzulu,G.F.,Traiger,I.L.,"The recovery manager of a data management system," ACM Comput. Surv. 13, June 1981,pp.223-242.

29. Green, M.I. et al., "The DDP underlay simulation experiment: Tractical applications and d-RTOS model," TRW Defense Space Syst. Group, May 1980.

30. Gylys, V.B. and Edwards, J.A., 'Optimal partitioning of workload for distributed systems", in Dig. COMPCON Fall 1976, pp.353-357.

31. Hansen, P.B., "The programming Language Concurrent Pascal", IEEE Trans. on Software Engineering, vol. SE-1, No. 2, June 1975, pp. 197-207.

32. Heinrich, F., "The Systems Architecture of the Distributed Computer System - The Distributed File

System," Technical Report, University of California, Irvine.

33. Lampson,B.W.,Sturgis,H.E.,"Crash recovery in a distributed data storage system,"Tech.Rep.Xerox Palo Alto Research Center, Palo Alto,Calif.,Apr. 1979.

34. Ma, P.R., Lee, E.Y. and Tsushiya, M., "A Task Allocation Model for Distributed Computing Systems," IEEE Trans. on Comp., vol c-31, NO.1, Jan.1982, pp41-47.

35. Maryanski, F.J.: "Backend Database Systems," Comput. Surv., vol. 12, pp.3-25, March 1980.

36. Metcalfe,R.M.,Boggs,D.R.,"Ethernet: Distributed Packet Switching for Local Computer Netwcrks," Communications of the ACM, July 1976.

37. Mitchell,J.G.,Dion,J.,'A comparison of two network-based file servers,' Commun. ACM 25, April 1982,pp.233-245.

38. Mitchell,J.G.,"File servers for local area networks," Lecture Notes,Course on Local Area Network, U. of Kent, Canterbury,England,March 1982,pp.83-114.

39. Newell, A., Fahlman, S.E., Sproull, R.F., and Wactlar, H.D.:"CMU Proposal for Personal Scientific Computing," Compcon, pp. 480-483, Spring 1980.

40. Organick,E.I., "The Multics System:an Examination of

its Structure," MIT Press, Cambridge, Mass.,1972.

41. Paxton,W.H.,"A client-based transaction system to maintain data integrity,"in Proc. 7th ACM Symp. Operating Systems Principles,ACM SIGOPS Operating Syst. Rev. 13, 4,Dec. 1979,18-23.

42. Popek,G., Walker, B., Chow, J., Edwards, D., Kline, C., Rudisin,G. Thiel,G.,"LOCUS: A Network Transparent, High Reliability Distributed System," UCLA, February 1981.

43. Rashid,R.F.,"An Inter-Process Communication Facility for UNIX," Technical Report CMU-CS-80-124, Department of Computer Science, CMU, March 1980.

44. Reed, D.P., Svobodova.I., "SWALLOW: A Distributed Data Storage System for a Local Network," Intern. Workshop on Local Networks by IBM Zurich Lab, August 1980.

45. Reed,D.P.,"Naming and Synchronization in a Decentralized Computer System," PhD Thesis, MIT, September 1978.

46. Rosen, B.:"PERQ: A Commercially Available Personal Scientific Computer," Compcon, pp. 484-485, Spring 1980.

47. Smith, R.:"The Contract Net Protocol : High Level Communication and Control in a Distributed Problem Solver," proc. First Int. Conf. Distrib. Com-

put. Syst., IEEE, pp. 185-192, 1979.

48. Stone, H.S., and Bokhari, S.H.:' Control of Distributed Processes. " Computer, vol. 11, pp. 97-106, July 1978.

49. Stone, H.S., "Multiprocessor scheduling with the aid of network flow algorithm", IEEE Trans. Software Eng., vol. SE-3, pp.85-93, Jan. 1977.

50. Sturgis,H.E.,Mitchell,J.G.,Israel,J.E.,"Issues in the design and use of a distributed file system," ACM SIGOPS Operating Syst. Rev. 14, July 1980,pp.55-69.

51. Svobodova,L.,"Management of the object histories in the SWALLOW repository,"Tech.Rep. MIT/LCS/TR-243, Lab. for Computer Science, MIT, Cambridge, Mass., July 1980.

52. Svobodova,L.,"A reliable object-oriented data repository for a distributed computer system,"in Proc. 8th ACM Symp. Operating systems principles,ACM SIGOPS Operating Syst. Rev. 15,Dec. 1981,pp.47-58.

53. Swinehart,D.,McDaniel,G.,Boggs,D.,"WFS: A simple shared file system for a distributed environment," in Proc. 7th ACM Symp. Operating Systems Principles,ACM SIGOPS Operating Syst. Rev. 13, 4,Dec 1979,9-17.

54. S.Y.W., CHANG, H., Copuand, G., Fisher, P., Lowenthal,

E., and Schuster,s.: "Database Machines and some Issues on DBMS Standards," proc. NCC, pp. 191-208, 1980.

55. Tacker,C.P., McCreight, E.M., Lampson, B.W., Sproull, R.F. and Boggs,D.R.,"Alto: A personal computer," Technical Report CSL-79-11,Xerox Palo Alto Research Center, August 1979.

56. Taub, A.H., "Collected Works of John von Neumann", vol. 5, The Macmillian Company, New York, 1963, pp.34-79.

57. Thompson, M., Robertson, G., Satyanarayanan, M. and Accetta, M. "Spice File System," Technical Report, Department of Computer Science, CMU 1980.

58. Thorton, J.E.:"Backend Network Approaches," Compcon, pp. 217-223, Spring 1980.

59. Treleaven, P.C.:"Exploiting Program Concurrency in Computing Systems, Computer, vol.12, pp. 42-49, Jan. 1979.

60. Van Dam, a., Stabler, G., and Harrington, r.:"Intelligent Satellite for Interactive Graphics," proc. IEEE, vol.62. pp. 83-92, April 1974.

61. Ward, S.A., and Terman, C.J.: "An Approach to Personal Computing," Compcon, pp. 460-465, Spring 1980.

62. Watson, R.W.: "Network Architecture Design Issues for Backend Storage Networks," Compcon, pp. 224-234,

Spring 1980.

63. Watson, I., Gurd, J. "A Prototype Data Flow Computer with Token Labelling", AFIPS, Conference Proceedings, NCC, pp.623-628, 1979.

64. White,R.M.,"Disk Storage Technology," Scientific American 243(2), August 1980.

65. Williams, R., Daniels, D., Haas, L., Lapis, G., Lindsay, B., Ng, P., Obermark, R., Selinger, P., Walker, A., Wilms,P., and Yost,R.,"R*: An overview of the architecture," Tech.Rep. RJ3325, IBM San Jose Res. Lab., San Jose, Calif., Dec. 1981.

66. Wilkes,M.V.,Needham,R.M.,"The Cambridge CAP computer and its operating system," Operating and Programming System Series, Elsevier/North Holland, 1979.

67. Wittie, L.D.: "A Distributed Operating System for A Reconfigurable Network Computer," Proc. First Int. Conf. on Distrib. Comput. Syst., IEEE, pp. 669-677, 1979.