

A Functional Processor System Model

on

Loosely-Coupled Machines

by

Shell Ying Huang, B. Sc. (Eng), A. C. G. I.

January, 1990

**A thesis submitted for the degree of Doctor of Philosophy
of the University of London and for
the Diploma of Membership of the Imperial College.**

**Department of Computing
Imperial College of Science Technology and Medicine
London SW7.**

Abstract

This is a study of the feasibility and the performance of the Functional Processor System(FPS), running on a group of logically isolated virtual machines. The FPS is a confederation of cooperating service machines, each is dedicated to provide a subset of the services of the whole system like compilation or editing. In other words, the application services are partitioned among the service machines.

A model of the FPS, to be run on the virtual machines supported by the VM/SP system, was built. It consists of a real communication subsystem, utilising the inter-virtual machine communication facility supported by VM/SP, and a submodel of the service machines(SM). The communication subsystem is realised by a Central Communication Controller, which coordinates the activities of the SMs and runs on a virtual machine. The SM submodel runs in two modes, the independent mode and the functional mode, to facilitate the performance comparison between the CP/CMS system and the FPS.

Experiments involving different resource requirements and workloads were carried out using the model. The inter-SVM communication costs were assessed and the performance of the FPS and the CP/CMS system were analysed and compared.

The model was examined in the context of a network of loosely-coupled real machines. It was shown to provide a suitable tool for studying the FPS for some networks of homogeneous or heterogeneous machines with certain types of interconnection schemes, the only modifications needed were the system calls to send/receive messages. With other types of interconnection schemes some structural changes in the communication subsystem of the model would be required as well. All major design decisions for the model need not be changed.

Acknowledgement

I would like to thank Prof. D. J. Howarth, my supervisor, for his guidance and encouragement throughout this project, and also for reading the scripts of this thesis and the valuable comments he made. His knowledge and insight in this field has made this work a more rewarding and exciting experience.

I am deeply indebted to my parents. Without their sacrifice and hard work, I would not have received the education I had. Words are not enough for expressing my gratitude to them.

My best friend, Ong Ghim Hwee, also deserves a special acknowledgement here. Without his unfailing support, I probably would not have enough perseverance to finish this project while working full time.

To my father

Contents

Abstract	2
Acknowledgement	3
Chapter 1 Introduction	8
Chapter 2 The VM Systems on VM/370	16
2.1 The CP/CMS System	16
2.1.1 The System Overview	16
2.1.2 The Control Program	17
2.1.3 The CMS	24
2.1.4 Virtual Machines as a Vehicle for System Services	28
2.2 The Functional VM System(FVMS)	30
2.2.1 The System Structure	30
2.2.2 The Potential Advantages of FVMS and its Disadvantage	35
2.2.3 Previous Developments and Research	36
2.3 Experimental Approach or Analytical Approach?	38
Chapter 3 The Communication Subsystem	40
3.1 Introduction	40
3.2 The Functional Requirements	42
3.2.1 Sender Point of View	43
3.2.2 Receiver Point of View with Potential Waiting Involved	44
3.2.3 Receiver Point of View with No Potential Waiting	44
3.2.4 Concurrent Communications	45
3.3 Facilities Available	46
3.3.1 Shared Segments	46
3.3.2 The Spooling Facility	48
3.3.3 Message Passing	49
3.3.4 A Preliminary Assessment	53

3.4	The Inter Program Communication Facility(IPCF)	55
3.5	The Assessment of IPCF	60
3.5.1	Functions Provided and Those Not Provided	60
3.5.2	Errors	62
3.5.3	Some Performance Considerations	67
3.5.4	Conclusion	73
3.6	The Design and Implementation of the Communication Subsystem for FVMS	74
3.6.1	Round Robin	74
3.6.2	Virtual Machines as Service Controllers	77
3.6.3	Virtual Machines as Communication Controllers	79
3.6.4	One Central Communication Controller(CCC)	80
3.6.5	The Implementation of CCC	86
Chapter 4	The Service Virtual Machine Submodel	91
4.1	Why a Model ?	91
4.2	Some Design Issues	94
4.2.1	Simulation Controlled by Resource Consumption Definitions	94
4.2.2	Simulation Flexibility	96
4.2.3	The Scope of the Model and its Limitations	98
4.2.4	Error Recovery in the SVM	103
4.2.5	A Guideline in the Design of the Model	103
4.3	The Model Components	104
4.3.1	The Overview	104
4.3.2	The Data Structures	106
4.3.3	The Initialisation of the SVM Model	114
4.3.4	The Communication Interface	118
4.3.5	The Simulation of the Services	125
4.4	The Assessment of the Model	140
4.4.1	The Model Characteristics	140
4.4.2	More on the Scope of the Model	141
4.5	The Validation of the Model	142
4.5.1	Validation Method	142
4.5.2	Examples of Validation Results	143
Chapter 5	Experiments , Results and Analysis	147

5.1	The Experiments	147
5.1.1	The Environment of the Experiments	147
5.1.2	Experimental Data	148
5.1.3	Experiment Plan	151
5.1.4	The Collection of Data	157
5.2	The Performance of the Communication Subsystem	160
5.2.1	The Communication Costs to Users	162
5.2.2	The Communication Costs to Non-Multiplexed SVMs	172
5.2.3	Is the CCC a Bottleneck?	176
5.3	The Performance of the FVMS and that of the IVMS	178
5.3.1	Comparison of the Average Response Times	178
5.3.2	The Fairness of the FVMS and the IVMS to users	187
5.4	The Performance Comparison between the Different Configurations of the FVMS	190
5.4.1	The Performance of the FVMS with Different Number of SVMs	190
5.4.2	The Performance of the Multiplexed SVMs	192
Chapter 6	How the Model Relates to a Network of Real Machines	198
6.1	Introduction	198
6.1.1	The Functional Processor System (FPS)	198
6.1.2	The FVMS Model for the FPS?	199
6.2	The SVM Submodel to Simulate a Service Machine(SM)	200
6.3	The Communication Subsystem	202
6.3.1	A CCC for the FPS	203
6.3.2	Multiprocessor Interconnection Schemes	205
6.3.3	The CCC in These Interconnection Topologies	207
Chapter 7	Conclusions	213
Appendix A	The Machine	217
Appendix B	The Analysis of Variance on the Communication times	218
Appendix C	The Regression Analysis on the Response Times	228
References		251

Chapter One

Introduction

The VM/System Product (VM/SP), which came about in 1980, was an extension to VM/370. A history of the development of the VM/370 and of its predecessor CP-67/CMS are presented in [1] and [2]. VM/SP is widely installed on IBM's 43xx, 303x and 308x series machines. The basic system architecture consists of a Control Program(CP) and some virtual machines. The CP supports the virtual machines by time-sharing them on the real machine. The virtual machines are architecturally identical to the real machine and each of them runs its own operating system. The virtual machine concepts have been discussed at length in [3], [4] and the virtual machine environment in [5].

This system architecture has the following advantages and they are unique among the mainframe systems:

-- concurrent running of dissimilar operating systems in different virtual machines.

This provides a very convenient facility for developing and testing new operating systems or new releases of a current operating system without interrupting users' production work.

-- the programs in each virtual machine are restricted to use only the (virtual) resources of the virtual machine. As a result, the CP is protected from the virtual machines. Also each virtual machine is totally isolated from the effects of software failures occurring in other virtual machines and is protected from each other against malicious or inadvertent destruction.

In the VM/SP system, as its predecessor, the sharing of real machine resources between concurrent virtual machines are managed by the CP while each

user's interactive environment is provided by the operating system called CMS in the virtual machine. That is, CP simulates and multiprograms the virtual machines and CMS manages one user's (the user's) personal computing in each virtual machine. The system gained great popularity in institutions all over the world in the early half of the eighties.

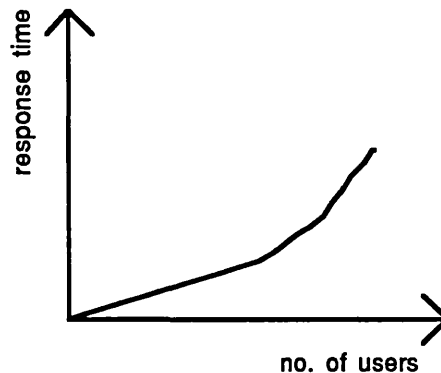


Fig 1.1 Relationship between response time and user load.

The system works satisfactorily when the number of users is lower than a certain value, depending on the system configuration and the individual workload. However when the total workload put on the system by the users approaches the system capacity, the system performance as viewed by the user degrades drastically. It is said that the system is saturated. One measurement of the system performance is the response time to user work. Fig 1.1 shows the general shape of the relationship between the response time and the number of (active) users on the system [6]. At first when the system has not reached the saturation point, the response time rises more or less linearly with the increase of the number of users. When the system is saturated the response time is inflated.

Fig 1.2 shows the conceptual views of the sharing of real resources among different number of concurrent virtual machines. The black area is the resources consumed by the CP to support the concurrent operations of the virtual machines.

They are referred to as the system overhead. The grey area represents the resources shared fairly by the virtual machines. Sharing the resources *fairly* does not necessarily mean sharing them *equally*. One example is the amount of storage that each virtual machine is allocated depends on the size of its working set. Nevertheless the CPU time is expected to be shared equally over an extended period of time. The white area indicates the spare resources left. The system overhead includes all the work done by the CP to maintain the illusion of a single machine for each user for all the users. For example, to maintain the virtual I/O devices for the virtual machines, their status has to be stored and maintained. [7] has a summary of the additional work that must be accomplished in a virtual machine environment by the CP. The more virtual machines there are on the system, the more work there is for the control program.

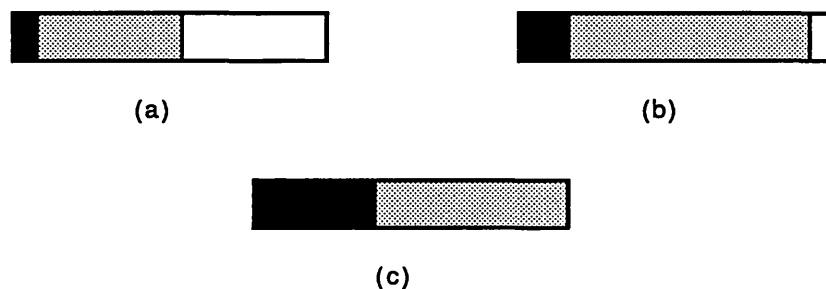


Fig 1.2 A conceptual view of the sharing of system resources.

Fig 1.2(a) shows the state when the machine is only lightly loaded with plenty of spare capacity. Fig 1.2(b) is the situation when the machine is not saturated but not far from it. As the number of virtual machines increases, the system overhead increases. However, as shown in Fig 1.2(b), there is still spare capacity in the machine(no component of the machine is saturated), therefore there is no difficulty supporting the concurrent virtual machines. The result is the system overhead increases, but not abnormally. As soon as the system crosses the saturation point, it becomes increasingly hard to support the virtual machines. As illustrated in Fig

1.2(c), the system overhead increases out of the proportion and virtual machines have to share the resources remained. An example is when the CPU is spending a lot of time organising excessive paging activities. [6] presents a detailed analysis of the saturation conditions.

When saturation is detected, there are a number of ways to improve the system performance. If only one part of the system is delaying the system's production work, that is, a bottleneck exists in the system, upgrading that component's capacity, or if possible, improving that component's efficiency will remove the bottleneck. For example, if the CPU is saturated with overhead due to paging, expanding the main memory may release the CPU and the paging subsystem for more production work. If the workload is such that the whole system's capacity is not up to the demand, upgrading the whole system will achieve better performance. However, consider the system itself, given the capacity, the machine performance in terms of response time and throughput may be improved by improving the efficiency of running it. In other words the system performance may be enhanced by reducing the system overhead. Various Virtual Machine Assists [7] increased the efficiency of a number of CP functions. Another way is to keep the number of virtual machines down. It is found that when the system performance is good (this happens when the number of users, that is in fact, the number of virtual machines, is low) more work gets done [8] within the same amount of time.

In this study, the feasibility and the efficiency of organising the virtual machines in a different way than the CP/CMS structure is investigated. In the CP/CMS structure a virtual machine is the environment where all the computing facilities a user needs are provided, and where all the user application jobs, trivial and non-trivial, are carried out. It is a service station for one user where comprehensive services are offered. In the structure studied by this research, a virtual machine only provides a subset of the computing facilities, such as compiling

or editing. There may be more than one virtual machine offering a particular type of facilities. A user has to be processed in different virtual machines at different times in a session depending on the type of facility required. The management of the real resources and the scheduling of the virtual machine operations are still the responsibility of the Control Program, but the virtual machines cooperate to process the total workload of all the users. These virtual machines are called Service Virtual Machines (SVM) and the system is called Functional Virtual Machine System (FVMS).

Obviously in a FVMS the number of SVMs does not have to be equal to the number of users. Thus by having less number of SVMs than the number of users the system overhead to support them would be lower than that in the CP/CMS system and the work can be processed in a shorter time. It is noted that in the CP/CMS system the scheduler in the CP has the job of trying to keep the system overhead down at a reasonable level by keeping the multiprogramming level to a value the scheduler thinks best at the time. This limits the number of virtual machines which are using the real resources concurrently for production work at each moment. But the scheduler may underestimate the working sets of the virtual machines. Besides, making the decision of keeping a virtual machine out of the multiprogramming list or allowing it in also consumes resources, though maybe an insignificant amount. However the accumulative effect of organising many virtual machines might be significant. This study tries to answer these questions: (1) will the FVMS use the real resources more efficiently and therefore be able to give faster response times and higher throughput when the machine is heavily loaded? (2) on the other end will the FVMS give a much worse performance than the CP/CMS structure when the workload is not near saturation point?

A general study of the Functional Virtual Machine System was done by Stuart Cox[9]. In his study the general communication requirements of the FVMS were specified and a communication subsystem built on the facilities provided by VME/B

system on an ICL 2988 was developed for a simulation model to obtain some statistics. However due to the unique architecture of the VM/370 system where virtual machines are logically isolated entities, FVMS on VM/370 has its unique problems. One salient problem is how the SVMs communicate with one another so that users will get the processing they request in appropriate SVMs. If the communication among the SVMs proves to be costly and inefficient because it consumes substantial amount of system resources, the FVMS is unlikely to be able to operate efficiently. Hence a real implementation of the communication subsystem on VM/370 must be built for studying the FVMS behaviour on VM/370.

A simulation model of the SVMs is required for this study. A model provides the flexibility for experiments. It should be able to simulate the behaviour of SVMs offering different facilities. When a SVM offers full facilities, it is a model of an independent virtual machine supported by the CP/CMS.

The study of a FVMS model also had the proliferation of networks of relatively low-cost, high-performance micro/minicomputers or computer workstations in mind. This contrasts with the personal computer approach in which each machine is self contained and every service is supported by the machine alone. It also differs from systems like the V distributed system[10], [11], the Medusa system[12], the Cambridge distributed system[14] and others discussed in [19] in which the operating system services is distributed among the workstations and user application processes can be run on any processor in the processor/workstation pool. In the FVMS(a Functional Processor System is a more proper name here) the user application services(compiling, editing, execution, etc) are partitioned and assigned to different functional processing units which are the machines in the network. The cost of loading application programs such as compilers or editors from disk drives or file servers as in an ordinary network of machines is removed during the time a processor is dedicated to be a certain service machine and the software providing the

service is 'permanently' resident in the processor memory.

A potential advantage of the Functional Processor System lies in the advance of the VLSI technology. If a group of machines each with specialised capability can be available, for example, the machines have different instruction sets oriented towards providing certain services more efficiently, the Functional Processor System would show greater efficiencies than the existing systems. In fact, even with the existing machines, a network of workstations and other more powerful machines, with the workstations managing the highly interactive programs or the interactive parts of the programs and other more powerful machines providing the computationally intensive services, is already a beginning of the functional partitioning of services among a group of machines. Software[46] has begun to emerge in which the graphical interfaces of applications and the application programs execute on different machines connected through a network. When the processing power of certain services in such a system is saturated, it is a simple matter of adding on more functional processing units to get the system upgraded.

R. P Bird[13] reported an attempt to build a network of machines in which there is a compiler server node dedicated to provide compilation service to the other machines in the network. In this project only compilation service is put into dedicated machines and it was still in design stage.

A study of distributed application programs was described in [25]. It was a graphics application and it showed that distributed application can run faster than local ones despite the communication cost incurred.

A group of logically isolated virtual machines of the VM/SP system are ideal models of a group of microcomputers or workstations which do not share memory or any other resources, and of any groups of loosely-coupled computers. The study of the FVMS model will be able to provide some information about a network of loosely-coupled machines each specialised in some user services.

In summary, this research has two aims: (1) To develop a model which includes a real communication subsystem and a submodel of SVMs; the model should also be useful for the study of a Functional Processor System on a network of real loosely-coupled machines. (2) To study, using the model built, the performance behaviour of the FVMS from the user point of view and how it is compared with the virtual machines in the CP/CMS structure.

Chapter 2 of this thesis analyses the CP/CMS system, called Independent Virtual Machine System (IVMS) in this thesis, in details and then formally specify the FVMS. Chapter 3 discusses the design and the implementation of the communication subsystem which is an important component of the FVMS. This is followed by the description of the SVM model built for this study in Chapter 4. In Chapter 5, the experiments which were carried out to study the IVMS and the FVMS is described and the results are presented and analysed. Before the conclusions of this research in Chapter 7, Chapter 6 relates the FVMS model to a network of real machines.

Chapter Two

The VM Systems on VM/370

In this chapter, two components of the VM/370 system, the Control Program(CP) and the Conversational Monitor System(CMS) are discussed. This helps to understand what has to be done to support the virtual machines and how a single-user independent virtual machine organises its work. Following this it will be easy to see the potential advantages of the Functional Virtual Machine System(FVMS) which is described in detail later in the chapter. The last section of the chapter is the choice between the experimental and the analytical approaches in the study of the FVMS.

2.1 The CP/CMS System

This section describes the CP/CMS system. More detailed information may be found in [15], [16], [17].

2.1.1 The System Overview

The VM/370 system has four components:

The Control Program (CP)

CP controls and manages the real machine hardware, and creates and supports multiple virtual machines which replicate the architecture of the real machine.

The Conversational Monitor System (CMS)

CMS is a single user operating system. It manages the virtual machine resources and provides the user with an interactive computing environment which includes file storage, file utilities, language processors, editors and loaders.

Remote Spooling Communication Subsystem (RSCS)

RSCS is a special-purpose operating system which runs in a virtual machine and supports remote readers and printers and is responsible for file transfer to and from another computer.

Interactive Problem Control System (IPCS)

IPCS is a set of programs which facilitate problem reporting to help software support staff to analyse and track problems in software subsystems.

Fig 2.1 gives an illustration of a system where multiple virtual machines run CMS as the operating system to support a single interactive user in each virtual machine and RSCS runs in another virtual machine to provide its functions to the whole system.

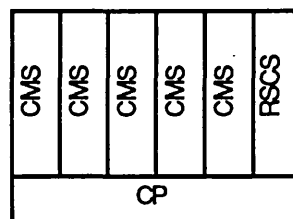


Fig 2.1 A system with 6 virtual machines.

2.1.2 The Control Program

CP is an operating system which provides an architecturally complete replica

of a System/370 (processor, storage and peripherals), called a virtual machine, for each one of its users. Hardware and software are employed to support each virtual machine. The techniques of multiprogramming, time slicing and storage management are used to organise the concurrent operations of the virtual machines.

2.1.2.1 The Simulation of One Virtual Machine

In a VM system, each virtual machine is represented by a control block: the VMBLOK, which maintains

- the status of the virtual CPU, which includes a virtual Program Status Word and the virtual registers;
- a set of tables to control the virtual storage;
- a set of control blocks to store the description of the I/O structure of the virtual machine.

Simulation of the CPU

CP manages the real CPU resource by a CPU sensitive, preemptive dispatcher and a storage sensitive, timer driven scheduler. The real CPU has two states: supervisor state and problem state. Virtual machines run in problem state and CP functions run in supervisor state. The virtual CPU also has two states: virtual supervisor state in which the operating system of the virtual machine executes, and virtual problem state in which the user's application programs execute. The machine instructions are classified as privileged and non-privileged instructions. A privileged instruction is one that interrogates or changes the states of the real system. Other instructions are non-privileged.

When a virtual machine is dispatched, the non-privileged instructions are executed directly by the real CPU, that is, no simulation of these instructions.

Therefore there is no overhead supporting the virtual CPU executing non-privileged instructions. A privileged instruction will be trapped by CP via a program interruption. The virtual machine is suspended and CP gains control. CP's interruption handler determines the nature of the interruption. The status of the virtual CPU is checked. If it is in (virtual) problem state, CP simulates the program interruption to the virtual machine so that its operating system, after it is resumed, will handle the interrupt just as the operating system of any other computer system will. If the virtual CPU is in (virtual) supervisor state, CP will analyse the operation to be simulated and do the actual simulation, possibly testing and altering the status of the virtual CPU, the I/O structure, storage key array and other elements. The privileged instruction handling ends with the invocation of the CP's dispatcher to dispatch the next virtual machine to run.

Simulation of the Store

Dynamic address translation hardware is employed to establish and maintain a virtual address space for each virtual machine. The set of segment, page and swap tables, which can be located through pointers in the VMBLOK of each virtual machine are used for virtual address translation. To start running a virtual machine, CP first loads a control register with the address of its segment table. Through the tables, a virtual address used in the execution of an instruction is mapped to the real address. A translation exception interruption occurs when (1) the virtual machine is trying to access an address which is outside its virtual address space; (2) the page is not currently in the main memory of the real machine, that is, a 'page fault'. In the former case, an address exception interrupt is reflected to the virtual machine and the operating system of the virtual machine will handle it. In the latter case CP has to page in the relevant page from secondary storage and then translate the address as normal. Bringing in the page involves finding a page frame in the real storage to

allocate to the virtual machine, organising the paging subsystem to transfer the page from secondary storage, suspending the virtual machine and later when the transfer of the page has finished, putting it back to the multiprogramming list so that it will be resumed when its turn comes. If enough pages for the virtual machine are in the main memory, the virtual machine will be able to execute its instructions for substantial periods of time without incurring a page fault. This will result in only small overhead of CP to support the virtual storage of the virtual machine.

Simulation of the I/O devices

In a virtual machine, the following I/O devices are normally supported by CP: a number of 'mini-disks' (each is a portion of a real direct access storage device such as a 3370), the operator's console (the user's terminal), and a number of unit record devices such as printers, card readers and punches. Corresponding to the channels and the control units of the real machine, a virtual machine has virtual channels and virtual control units. Some virtual I/O devices have direct counterparts on the real machine and others do not have.

In the VMBLOK of a virtual machine, a set of control blocks describe its I/O structure. They record the existence of each (virtual) I/O device, their status (for example, busy or free) and the real I/O devices to which they correspond.

When a virtual machine issues a SIO (Start I/O) instruction, CP gains control and checks if the virtual I/O device address is valid in the virtual machine's I/O structure. It then makes any necessary adjustments in mapping between the virtual and real devices and builds an I/O task control block called IOBLOK for the real I/O devices. This may involve the translation of the virtual channel program to the real channel program which uses real storage addresses and reflects the real storage page frames to be used. In the cases of mini-disk I/O virtual track addresses have to be mapped to real track addresses to get to the proper mini-disk space. If it is an I/O

operation involving a unit record device CP has to simulate it by a spooling operation. CP also checks the real device, control unit and the channel that compose the virtual I/O path are free. CP then marks them as busy and starts the I/O operation on the real device before resuming the virtual machine's execution. If any part of the I/O path is busy (another virtual machine is doing I/O on it), the IOBLOK will be queued until the whole path is free and then I/O is started. When the I/O operation is finished, the real device interrupts the CPU and CP will update the relevant virtual I/O device control blocks, indicating an I/O completion interruption is pending, and release the I/O path by marking them free again. As control returns to the virtual machine, its operating system will handle the I/O completion interrupt as if the real I/O device had interrupted the virtual machine.

2.1.2.2 The Organisation of Multiple Virtual Machines

The sharing of the CPU among the concurrent virtual machines is managed by the dispatcher and the scheduler. The multiprogramming list is the list of virtual machines (VMBLOKs) which actually share the CPU by time slicing and the main memory by the storage management. The eligible list is a list of virtual machines which are executable and waiting to enter the multiprogramming list but their working sets are not in the memory.

The dispatcher each time schedules the first virtual machine in the multiprogramming list to use the CPU. The virtual machine runs for a period of time until one of the following circumstances arise: (1) the user in the virtual machine logs off; (2) an interrupt occurs indicating the virtual machine has used up its current time slice of the CPU; (3) an interrupt occurs from within the virtual machine because it wants to do I/O, or because of a page fault, or a supervisor call to request a CP function to be performed on the virtual machine's behalf, or an error has

occurred; (4) an interrupt occurs from one of the I/O devices signalling the completion or error condition of an I/O activity. These conditions may result in the changes in some virtual machine's scheduling status. The current virtual machine may be removed from the multiprogramming list. Some other virtual machine may be inserted into the multiprogramming list by the scheduler, which is the CP component to control the content of the multiprogramming list. Whatever happens the dispatcher simply finds the next virtual machine to run when control returns to it.

The virtual machines in the multiprogramming list and the eligible list are ordered by their deadline priority done by the scheduler. A virtual machine enters the multiprogramming list after finishing a terminal I/O operation, or it is the virtual machine in the eligible list with the highest deadline priority whose working set can be accommodated by the number of memory pages available for allocation. The deadline priority is calculated from a formula involving the current resource utilisation of the virtual machine, the time the virtual machine last dropped from the dispatch list, the user priority and the current load on the system. An interactive virtual machine's deadline priority is better than a non-interactive virtual machine's. Thus the interactive virtual machines are normally considered for dispatching before non-interactive virtual machines.

In the dynamic paging area(where virtual machines share the real memory) of the main storage, each page frame may be in one of the following lists: (1) the USER list if it is allocated to a virtual machine currently in the multiprogramming list or occupied by a page of a pageable CP routine(an active CP routine); (2) the FLUSH list if it is allocated to a virtual machine which is not in the multiprogramming list(in the eligible list or unrunnable); (3) the FREE list if it is currently unassigned. If a virtual machine or a CP routine needs a page, one is taken from the FREE list. CP attempts to keep a number of page frames in the FREE list so that when a page is needed, 'stealing' from the FLUSH list or the USER list would not be necessary. When

there are not sufficient page frames for all the concurrent virtual machines, a virtual machine may lose some or all its pages in the main memory (paged out and assigned to other virtual machines) when it becomes unrunnable and all these pages have to be paged in the next time it executes on the CPU. In the case when the scheduler underestimates the working sets of the virtual machines and the multiprogramming list becomes too long, 'thrashing' occurs and even the virtual machines in the multiprogramming list need frequent paging in activities.

2.1.2.3 System Overheads

From the descriptions above, it is noted that the overhead to support virtual machines comes from:

- (1) the software simulation of privileged instructions issued by a virtual machine. It involves CP's interrupt handling, instruction decoding, instruction simulation and the dispatching of the next virtual machine. Efforts have been made to reduce the overhead by replacing software execution of these steps with hardware emulation of some privileged instructions. After these privileged instructions have been hardware emulated control returns to the virtual machine without recourse to CP. Examples are the VM Assist, Extended Control Program Support:VM/370(ECPS:VM), which are included in VM/SP[15], [16]. These products improve the performance of the virtual machine and the overall system. However there are still instructions that must be simulated by software.
- (2) The paging activities caused by the sharing of one real storage by the virtual storages of all the concurrent virtual machines. One virtual machine's pages may be paged out to secondary storage to make room for the pages of some other virtual machines when the former is not running, for example, waiting for a disk input. At some later time all these pages have to be brought back into the main storage

when they are referenced by the virtual machine. This type of overhead increases with the increase in the contention for storage among concurrent virtual machines. This results in less CPU time and paging device time available for production work.

(3) The housekeeping of the status records of all the components of the virtual machines. The more concurrent virtual machines there are the more work is involved.

The amount of productive work achieved by the machine is proportional to the amount of time the system is engaged in production work. Therefore the system would be more productive if the overhead to support the virtual machines is not consuming system resources that would otherwise be used by the virtual machines if not for too many of them are sharing the resources simultaneously.

2.1.3 The CMS

CMS is the operating system which organises the resources of a virtual machine and provides an interactive computing environment to a single user. The resources of a virtual machine includes the CPU, the main storage, the file storage which is distributed over a number of 'mini-disks' and the I/O devices. According to [17] the functions performed by the CMS routines can be divided into nine areas and they are

- (1) initialise the CMS virtual machine environment;
- (2) process and execute CMS files;
- (3) process commands that manipulate the file system;
- (4) manage the CMS file system;
- (5) handle I/O operations;

- (6) handle interruptions;
- (7) manage CMS storage;
- (8) simulate non-CMS operating environments;
- (9) perform miscellaneous CMS functions.

The following description is concentrated on how CMS manages a user session and serves his commands. That is, the areas of (2) and (3). Obviously many functions in other areas are implicitly involved in the processing of (2) and (3). Then a discussion on the efficiency of CMS in the environment of multiple CMS virtual machines is presented.

2.1.3.1 How CMS Works

CMS is an one-user operating system. It reads a command from the user, has it executed, updates the user's file directory and then asks the user to input the next command. CMS does not allow parallel or concurrent command executions in the virtual machine. Therefore there is no background execution or forking of one execution into several executions.

When CMS loads a program module from disk to the storage of the virtual machine, it loads it into one of the two areas, one called the *user program area*, the other called the *transient area*. The transient area is 8 Kbyte long and is generally used for short system command modules which may be invoked by the programs executing in the user program area. The user program area is used to hold user programs and other system command modules.

The way CMS decodes a command. The file store of the virtual machine is searched for an EXEC file with the command name as the file name. If one is found, EXEC is invoked

to load the EXEC interpreter which processes the EXEC file. If no EXEC file is found, a check is made to see if the command module is currently in the transient area or the command is one of the few residing in the CMS nucleus. If so, it will be executed from there. If the check does not find the command module, CMS then searches in the user's disks for a MODULE file with the command name. If such a file is found, it is loaded into the main storage and executed. When all these fail, CMS assumes this is a CP command and passes it over to CP. CP returns with a condition code indicating successful execution or that the command is unknown. CMS then responds to the user with the result of the execution by a return code and an error message when appropriate.

The way CMS compiles a program. There is usually an interface module between the CMS command interpreter and the compiler program and it is of a MODULE file type. After the CMS interpreter loads the interface module into the virtual machine storage, this interface will check the existence of the program to be compiled. It also parses the compiler options that the user has specified and sets up the parameter list to the compiler. The interface then loads the compiler into the storage and control is passed to it. When control returns to the interface module at the end of the compilation, it releases the compiler pages and all other relevant pages and passes the return code from the compiler to the user.

The assembly of a program in an assembler language works in a similar way to the compilation. For the System/370 assembly language, the interface module is also responsible for defining the work files used by the assembler and closing them afterwards.

The way CMS runs a user program. For a user program written in a high level language like Pascal, Simula 67 or Cobol, an EXEC file is usually used to specify the

sequence of the CMS commands to execute it. It checks if the program to be executed exists and defines the program library or libraries which is(are) to be searched by the loader for any unresolved external references during the loading process. Then the EXEC file calls the loader to load the program and the library routines used by it into the virtual machine storage and starts the execution of the loaded program. For some languages like Cobol or Simula 67, the EXEC file also defines the files used by the program for input/output with the CMS FILEDEF command before loading it into the storage.

The way CMS provides system facilities. In the virtual machine configuration, there are a number of 'mini-disks' connected to the virtual machine. Among them, there are some system disks -- read-only disks where some of the system modules are stored. These system modules provides the facilities like copying a file, browsing a file, printing a file and so on. When commands requesting these tasks to be done are issued by the user, the appropriate module is loaded either into the transient area or the user program area of the virtual store and is executed. Loading is not necessary if the module is already in the transient area and it is serially reusable. Not all the system services are supplied from the system disks, some system modules are resident in the CMS nucleus. To execute these system modules loading is obviously not required.

2.1.3.2 Efficiency of a System with Multiple CMS Virtual Machines

In a CP/CMS system, all application software resources like the language processors, loaders, library routines are supplied by CMS. This presents an efficiency problem when each CMS in a multiple CMS virtual machine environment loads a copy of a certain software module like a compiler into its virtual storage during the same span of time. From the real machine point of view it is a waste of time to load

and a waste of physical storage to hold multiple copies of the same software in the machine storage.

Due to the consideration of efficiency some software resources supplied to the users can be shared concurrently by multiple virtual machines. When a shared version of a software routine is available, loading only takes place if no other user has already loaded it into the real storage (and mapped it onto his own address space) and is still using it. Once the routine is in the storage and before the last user releases it, any new user will just be given the entry point address and loading is skipped. Therefore there is only one physical copy of the routine in the real storage (including the paging area of the secondary storage). The shared named systems are used in this way by the virtual machines. In the Department of Computing, Imperial College, the Pascal compiler was shared. There was also a shared run time support system for Pascal programs.

Sharing software resources among concurrent virtual machines improves the real system's efficiency. However this depends on whether the shared version of a software system is available or not. In reality, a large number of the software modules such as language processors are not shared.

2.1.4 Virtual Machines as a Vehicle for System Services

RSCS, a component of the VM/370 system, runs in a virtual machine and provides file transmission facilities to and from remote computers which the users in all other virtual machines on the system may use. This is an example of using a virtual machine to provide a service (resource) which is available to all the system users.

Another example is the provision of the batch facility using a virtual machine running either CMSBATCH or some other operating system on a CP/CMS system. All

the CMS users on the system may send jobs to the batch virtual machine(s) and the batch virtual machine(s) will return the results of the executions to their initiators. Here a batch virtual machine is providing extra processing power to that of the individual virtual CPUs of the CMS users.

In the local installation[18] of the Department of Computing, Imperial College, virtual machines were more widely used to provide various system services. For example, system accounting was the responsibility of one virtual machine identified as ACCOUNTS and file archive and backup was handled by another called FAB.

Virtual machines need some mechanism to communicate with each other to request the system services and get response from the service provider. Spooling facility is used to transfer job files to and from batch virtual machines. Communication with the RSCS employs the same facility. A more advanced mechanism called Virtual Machine Communication Facility(VMCF: discussed in the next chapter) was used between a CMS virtual machine and a FAB or ACCOUNTS in the local installation.

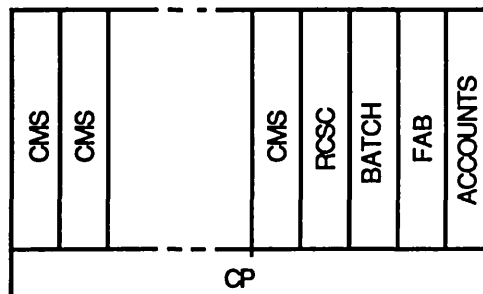


Fig 2.2 A system with a number of CMS virtual machines and some other service virtual machines.

These virtual machines are usually referred to as service virtual machines. When their services are not required by the users they are in a dormant state and thus are not competing with other virtual machines for resources.

The advantages of using virtual machines to provide system services are:

- (1) it prevents the CP from developing into a complex system. The CP is managing the real machine resources and time-sharing them among all the concurrent virtual machines. Adding services to it would increase its complexity, the chances to introduce inefficiency in CP and the system overhead.
- (2) it makes good use of the isolation between virtual machines and between virtual machines and the CP. The system service in a virtual machine is protected by the isolation and its error and failure is contained for the same reason.

Fig 2.2 shows the system with a number of CMS users and several service virtual machines mentioned above.

2.2 The Functional VM System(FVMS)

In this section, the Functional VM System(FVMS) is described. Its merits and disadvantages are then discussed.

2.2.1 The System Structure

The CP

There is no change to the functions and implementations of the CP. It still manages the real hardware, creates and multiplexes virtual machines on it. In other words, it still supports a time-sharing environment for multiple virtual machines.

The Virtual Machines

Each virtual machine no longer supplies the whole range of software resources to a single user but is dedicated to provide a subset of the user-oriented software resources. The software modules are loaded in the virtual storage throughout the lifetime of the virtual machine(this could change if there is dynamic functional

repartitioning of the services into the virtual machines). Users do not have their individual virtual machines but travel to specialised virtual machines where software resources are resident and execute their tasks there.

The basic virtual machine configuration is a virtual processor and a virtual memory as illustrated in Fig 2.3(a). This basic configuration is the same as that of the processing elements of some distributed network of machines[41]. Depending on the facilities offered by the virtual machine, it may have various I/O devices, control units attached to it. For example, one that is in charge of handling the users' terminal inputs have all the real terminals 'dedicated' to its virtual terminals like in Fig 2.3(b) but these terminals do not have the functions of the virtual machine console. Users' files do not attach to a particular virtual machine but are stored the same way as in other time-sharing system and all virtual machines which do input/output have access to the whole file storage. Protections to the user files is done so that a virtual machine serving one user cannot access the files of another user.

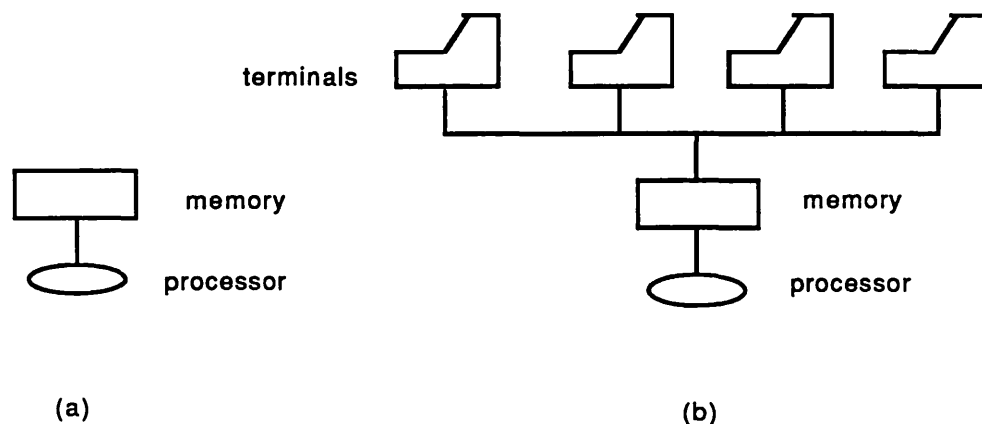


Fig 2.3 Two different configurations of the service virtual machines in a FVMS.

The operating system of the virtual machine will still perform those CMS functions listed in Section 2.1.3 which manage the virtual machine resources. For example, handle interruptions, manage CMS storage, and handle I/O operations if the

virtual machine provides I/O service.

In summary, a virtual machine of this type is a specialised processing unit with its private memory designated to provide a subset of the user services offered by the whole system. It is called a Service Virtual Machine(SVM).

A SVM may be a single user server, capable of offering service to one user at a time. This would lockout other users when the SVM is being occupied with the service of one user. To alleviate this problem, several SVMs could provide a certain service, or alternatively several users could be multiplexed in one SVM. The second solution would almost certainly be effective if the SVMs are real machines like micro or minicomputers. That is, if the FVMS is composed of a set of cooperative real computers, multiplexing will help to achieve the full utilisation of the processing power of the real machines. This approach may be compared to the servers of distributed operating systems which are composed of multi-processes or multi-tasks to provide the same service to several users concurrently[19]. For a FVMS, even with multiplexing, several SVMs may be needed to satisfy the demand on the processing power for a certain service. In the virtual machine case however, multiprogramming users in a virtual machine is often counter-productive with the multiprogramming of virtual machines by CP. There is one situation where multiplexed SVMs might be useful: during the service of a user by a certain SVM, the service of another SVM is required before the current service can be continued. The request is made to the other SVM and before the reply returns the current SVM may serve another user. An example is an input operation during the execution of a program while the SVM does not have I/O capability therefore the service of another SVM is required. Another case is a SVM with I/O capability which is offering highly interactive services like editing. When the SVM is waiting for a terminal input to complete, it may switch to another user to serve.

Whether using multiplexed SVMs contributes to a better performance of the

FVMS, or whether using a group of single-user servers to handle the workload of a type of service is better needs to be answered by the experiments with the FVMS.

To summarise, a FVMS is a system of cooperative groups of SVMs where the user services are distributed among the groups and the workload of each service is shared by the members of the group. Fig 2.4 is an illustration of such a system where the unit of a user service is one user command. Thus the command interpretation is one type of service. Obviously it is not possible to distribute the user services strictly according to the functions of the user commands. Some trivial commands have to be grouped together to form a service type handled by some SVMs. Users flow through servers(SVMs) to have their tasks done and along the directed lines to reach the appropriate servers. A user may be served by any one of the SVMs which offer the required service. The grey area in Fig 2.4 will be explained later.

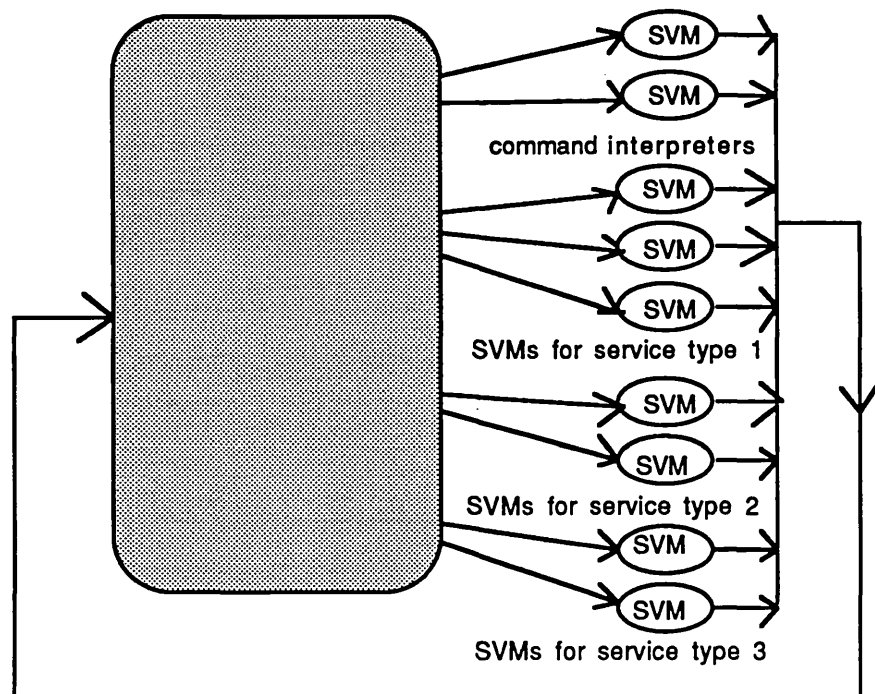


Fig 2.4 A FVMS with 9 SVMs.

The number of SVMs supporting a certain type of service should be such that the total workload put on the real machine by the SVMs allows the machine to offer

optimal throughput. The FVMS configuration could be relatively static in an environment like a department of Computer Science in a university. In such an environment, when there is a laboratory session for a certain class of students, most of them would likely be using the same editors, compilers and other software resources and the proportion of the demand for these services would not change drastically within the session. The number of SVMs in various types of services should be proportional to the demands for them. Of course these demands would change in a matter of days or weeks, or when there is a change of sessions for a different class of students. The FVMS should be able to adapt its configuration according to the demands of the users, that is, to increase the number of SVMs whose services are very much in demand and decrease the number of those not needed. In some cases what is needed is to change the system software loaded in the storage of a virtual machine and possibly also change its configuration so that it becomes a SVM offering a new service.

The Inter Virtual Machine Communication

As software resources are distributed among SVMs, an inter virtual machine communication mechanism is required to transfer users to the various SVMs where their requests are served. The shaded area of Fig 2.4 represents roughly the functions of such a mechanism, that is, it delivers users to the right SVMs. A SVM obtains a user from the communication system, carries out his job and returns the user to the communication system through which the user gets to the next appropriate SVM. A single user server then repeats the whole procedure. A multiplexed server may get more user(s) before the service of the current user(s) is finished.

Though all SVMs run on the same processor and all their virtual storages share the same physical storage, they are logically separated entities -- the components of the virtual machines are independent and separate from one another. Therefore the problems or difficulties that arise in the communication among SVMs are the same as

those between the components in a distributed system. The only difference is that physical communication lines are not involved in transmitting information between SVMs. It also follows that similar to distributed systems with loosely coupled but closely cooperative units, the communication system of the FVMS may become the bottleneck of the whole system and make it an ineffective way to provide user services. The inter virtual machine communication is studied in the next chapter.

2.2.2 The Potential Advantages of FVMS and Its Disadvantage

The FVMS has the following potential advantages:

- (1) It reduces the overhead of the CP to support the virtual machines when more users are served by less SVMs instead of one user one virtual machine. This advantage might be significant when the system is very heavily loaded.
- (2) By loading the SVMs with the software routines for user services such as compilers and run time support systems 'permanently' during the lifetime the SVM is designated to offer that service, repeated loading is avoided. For software systems that are not shared between virtual machines, concentrating them in the storages of a few SVMs not only saves the loading efforts but also keeps the number of copies of a system simultaneously in the real storage down.
- (3) In the FVMS it would be possible for a user to have parallel executions of several commands. These commands can be sent to the appropriate SVMs and be carried out concurrently. FVMS also has the potential ability to provide an environment for parallel executions of the different phases of a single command by different SVMs.
- (4) It retains the advantages of the virtual machine system as a FVMS is also composed of multiple virtual machines. The protection between the virtual machines protects each SVM from the interference from the outside and contains any errors

of its own within its own boundary. It is still easy to test new systems or changes to any old system in a virtual machine or a group of virtual machines not involved in the FVMS but running currently with the FVMS.

The disadvantage of the FVMS is that users do not have their own virtual machines. As a result, they cannot choose their own machine configurations and the operating systems to run.

2.2.3 Previous Developments and Research

Using virtual machines to provide specialised services to all the users on the system is not new.

Section 2.1.4 has listed a few such examples. Basically virtual machines like RSCS and the batch virtual machines operate asynchronously with the operation of the CMS user who calls upon their services. Response time, that is, the time between a CMS user issues a command which needs the service of the RSCS or a batch virtual machine, and the time the user is notified of the finish of the job, is not critical. The communication between a CMS virtual machine and these service virtual machines can be considered infrequent when compared with the frequency with which the user is using the software resources provided in his own virtual machine.

Experiments have been done at IBM Thomas J. Watson Research Centre[21] to use one master virtual machine and a few slave virtual machines to support a centralised program library management service for a group of interdependent users. Through these virtual machines users may read from, write to and do processing on the centralised data base. A user request is given to the master virtual machine which creates a slave virtual machine to serve the user's request. After serving one user, the slave virtual machine may be assigned to another user for service by the master

virtual machine. The master virtual machine may delete a slave virtual machine when it is not needed. Here the master virtual machine is the data base service call reception centre and the slave virtual machines are providing the service. The multiprogramming of the slave virtual machines(and other virtual machines) provides the concurrent services of more than one user. This research is a step forward towards using specialised virtual machines to provide a centralised service for all users.

In these previous developments and research, service virtual machines run alongside with users' CMS virtual machines. The majority of the user services are offered locally within users' individual virtual machines by CMS while a few not so frequently used or very specialised functions are provided by a group of 'specialist' virtual machines. In the FVMS, all user services are distributed among service virtual machines(SVMs).

A general study of the FVMS on various systems with the virtual machine structure did by Stuart Cox[9] has been described in the last chapter. The centre of his study is the idealised communication subsystem for a FVMS, the existing inter virtual machine/process communication mechanisms on various systems and a simulation on the VME/B of the ICL 2988 where there is a shared memory between the virtual machines.

On a different level, a lot of work has been done to distribute the operating system services on a network of loosely-coupled machines[11], [12], [14], [19], [24], [27]. Some of them have been mentioned in the last chapter. A number of solutions to the problem of mapping clients to servers in a distributed environment have been proposed[26], [34]. A scheme to build a shared virtual memory on a group of loosely-coupled multiprocessors to facilitate communication has also been studied[48].

Also relevant, [47] presented a comprehensive study to the problem of load

sharing in distributed systems. Queueing analysis is employed to compare a number of algorithms to the global scheduling problem, that is, which job is processed by which server. In these algorithms, either the server decides from which source the next customer is picked up for service, or the source decides to which server the current customer is going for service. The study points out that the former class of algorithms is superior than the latter if the communication overhead is not excessive.

2.3 Experimental Approach or Analytical Approach?

In the introduction chapter it was stated that a model of the FVMS consisting of a real implementation of the inter-SVM communication subsystem and a model of the SVM should be built. But first it has to be established why experimental method is chosen rather than analytical methods. An analytical model employing queueing theory[22] or using Petri Net method[23] could be constructed for performance evaluation and these approaches save a lot of programming efforts and experiment time. However the analytical model was rejected for the following reasons.

Firstly, the VM/SP system was available for experiments. This means the 'real' performance of CMS virtual machines under various workload may be measured. This is an advantage when one of the goal of the research is to compare the performance of the CP/CMS system and the FVMS. It also means, as can be seen in Chapter 4, the real hardware and the software base on which the FVMS runs is available.

Secondly, whether logically isolated SVMs of the FVMS can communicate with one another effectively is unknown and has to be investigated by experiments. The problems in the construction of the communication system will only arise and be solved if possible during the process of building the system itself. Even if it is assumed that an effective communication system can be built, the distribution of the inter virtual machine communication delay for an analytical model to use is unknown.

In some studies of the distributed systems, the communication costs are assumed to be negligible. However, on the VM/SP system the load put on the system by the inter SVM communication and that from the SVM operations affect each other. The impact of the workload in SVMs on the communication delay *and* the communication operation's impact on the performance of the SVMs are hard to estimate. Realistic figures can only be obtained through experiments with the communication system and the virtual machines.

Therefore the experimental approach using a FVMS model is employed in this research.

Chapter Three

The Communication Subsystem

This chapter describes the design and the implementation of the communication subsystem for the FVMS. First, the introduction section of the chapter points out the ways the communication subsystem could improve or hinder the efficiency of the whole FVMS. The functional requirements of the communication subsystem are then specified. It is followed by an examination of all the means by which information may be shared/passed among concurrent virtual machines on VM/SP. Finally as no facility on the VM/SP meets the requirements of the communications in the FVMS, also to study the feasibility of a communication subsystem for the FVMS on VM/SP, the design and implementation of the communication subsystem using the basic facilities available on VM/SP were done and their descriptions are presented here.

3.1 Introduction

The communication subsystem in a FVMS plays a crucial role in it. It could be the cause of an inefficient FVMS.

As described in the last chapter, a FVMS is a confederation of cooperating virtual machines that work together to process the users' job requests. One of the advantages of the FVMS is that having less number of the virtual machines than the number of users reduces the overhead of the control program (CP) supporting the virtual machines, leaving more system resources to the virtual machines to carry out user's work. However, the CPU time saved is not the net gain in a FVMS as the call in each SVM to the communication subsystem to pass a user to another virtual machine and the actual transfer of a user from one virtual machine to another by the

communication subsystem take time. If the communication cost offsets the savings in the overhead of CP supporting the virtual machines and the savings of loading some application software, the FVMS would be less efficient. On the other extreme, if the communication subsystem is very efficient, more system resources would be available to the SVMs for user tasks, thus a system in which resources are better utilised than the independent VM system would be achieved.

The communication subsystem is the passage in the FVMS through which users go from one SVM to another to have their jobs done. This leads to the second criteria the communication subsystem must satisfy for the FVMS to be an efficient system: it not only has to be efficient in consuming system resources but also it must not be the bottleneck of the FVMS. If the communication subsystem is efficient in using the resources but because the amount of information being passed through is too much for it to keep pace with the SVMs, the FVMS would not be efficient as a whole entity.

There is another way in which the communication subsystem could affect the efficiency of the FVMS: if it could have the function of providing a more balanced distribution of the workload[49] among the SVMs and maximize the parallelism/concurrency of them, the throughput of the FVMS would be increased. An example of the kind of situations where the communication subsystem could make a difference is: a virtual machine which is able to offer multiple kinds of services tries to get a user to serve when there are users waiting for different services. The communication subsystem could just distribute one user randomly from all the users this machine can serve, or distribute one from a service type with the heaviest workload among all the types of the services performed by the machine. As it is part of a heaviest workload, more users would be waiting for this service than other services. As a result, the quicker this service can be offered to the users, the amount of total waiting time of the users reduced would be greater.

Obviously trying to balance the workload would increase the system resources

like CPU time consumed by the communication subsystem, increasing the probability of it becoming the bottleneck of the FVMS. Therefore there is a limit to how much the communication subsystem can do in this aspect.

All these should be remembered when a communication subsystem is to be built or when an available system is to be evaluated.

3.2 Functional Requirements

In this section the functions required of the communication subsystem are specified. It concerns with what should be done by the communication subsystem and not how these functions are achieved or what resources are needed to carry out these functions.

Basically each SVM is working in a cycle of three steps:

1. get a user to serve (from the communication subsystem);
2. do the service (either from the beginning to the end without communicating with other SVMs, or if an external service is needed during the current service, send the user to be delivered to another SVM and wait for the user's return then continue the service);
3. send the user away (to the communication subsystem so that the user will be distributed to a SVM where the next service of the user can be provided);

When a SVM wants to get a user, there are two possible situations if it is a multiplexed SVM. The machine may have got other user(s) it is currently serving and just wants more user(s), that is, the machine can serve more users concurrently. Or the machine has not got any user. In the former case, the machine would like to obtain one more user if there is one but should continue to serve the user(s) it is already

serving if there is no other user waiting to be served. In the later case the machine is trying to get a user to serve thus if there is no user available at the moment, it has to wait until a user who needs its service is returned by some other virtual machine. For a non-multiplexed service virtual machine it is like a multiplexed machine with no user being served: it has to wait if no user is available.

When a machine wants to send a user to a new service machine, it is not concerned with when the user will get to the destination. After sending the user away, or even after making the request to send a user away, the machine can continue its next step of operation, that is, try to get another user to serve.

After this brief description of the situation of the SVMs when they want to get or send a user the functional requirements of the communication subsystem are given below.

3.2.1 Sender Point of View

After a SVM finishes serving a user, it needs to send the user to another SVM where a different service may be provided. The receiver SVM is one of the several SVMs which can offer the service required and have been waiting to serve a user. The communication subsystem will choose one among these several and transmit the user to it. If no such SVM is idle at the time of the sending request, the user should be directed by the communication subsystem to the first SVM which becomes available at a later time. It is obvious that the communication is a one to one-among-many type of communication and the receiver SVM's identification is unknown to the sender SVM.

The communication is asynchronous in the sense that the sender only uses the facility for passing some information(a user) to the receiver and not for the synchronisation of the activities of the communication partners involved. It is the communication subsystem's responsibility to get the user transmitted to a suitable

destination SVM. The sender SVM will just continue to serve other users.

3.2.2 Receiver Point of View with Potential Waiting Involved

When a SVM tries to get a user to serve and it is not serving any other user, it has nothing to do before the next user arrives. This is the case when a single user server wants a user or a multiplexed server with no user in it. Here getting some information(a user) and synchronising the receiver's activities with the sender's are both achieved through the communication. A blocking receive is required, that is, if there is a suitable user for the SVM the communication subsystem will distribute the user to it; however if there is no user the SVM should be blocked until a user is returned by some other SVM and then the user will be given to it and the receiver SVM resumes its operation.

A receiver SVM issues a receive request and is unaware of which SVM has got a user who wants the receiver's service. For example, a compiler SVM wants to get a user who intends to compile a program. The user would be from one of the several command interpreter SVMs. The compiler SVM is not aware of which command interpreter has first tried, or will first try, to send a user to compile.

So this is an one to one-among-many synchronous communication. When the receive operation ends the SVM has got a user to serve.

3.2.3 Receiver Point of View with No Potential Waiting

The addressing form of this type of communication is the same as the previous receive, that is, an one to one-among-many communication. The difference is this is a non-blocking operation. If there is a user waiting for a service which the receiver can offer, the receive operation returns with the user. If this is not the case, the receive

operation returns with an indication that no user has been obtained by the receiver.

This facility is useful to a multiplexed SVM. After such a SVM sends away a user, it tries to top up its list of users being served by a receive request. If the SVM still has other user(s) in its list, a blocking receive will prevent the SVM from continuing the service of the remaining users when a new user is not already waiting for the service.

3.2.4 Concurrent Communications

The SVMs should be able to conduct concurrent communication among themselves.

After a SVM A sends a user X away and before this user X gets to his new SVM B, the communication system should be able to allocate another user Y to machine A who may be from some other machine or from machine B. In other words, machine A may get its new user Y from any SVM before, after, or at the same time as user X is allocated and passed to his new SVM B. This is an example of concurrent activities of sending a user away and obtaining another one. In other word, this is an example of an incoming communication being concurrent with an outgoing communication for a SVM.

As SVM A gets user Y, it starts processing user Y and user X may or may not have been passed to SVM B. After SVM A finishes the service of user Y, it should be able to pass user Y away even before user X gets to his destination. User X and Y may have different destinations. This is the case when X and Y needs different services and SVM B does not provide both. This is also the case if SVM B is a single-user SVM if it is assumed that users do not wait in the server for it to finish the previous user. Therefore concurrent outgoing communications are required for A. User X and Y may have the same destination if SVM B is a multiplexed machine and the service(s) X and Y requires can both be offered by B. This shows that the ability to have concurrent

incoming communications is also needed by the multiplexed SVMs, in addition to concurrent outgoing communications.

On a real communication network, this means two things. First, it is preferred that the network will accept further requests before a particular request is finished. Second, it is not essential for the operations of the FVMS that the messages arrive at their destinations strictly in the order of their delivery requests.

3.3 Facilities Available

Communications among different entities can be achieved by having a common area where information can be accessed by all communication partners or by passing information around the interested parties. This section describes all the facilities on VM/SP that apparently can be used to share/pass information among concurrently operating virtual machines. Full descriptions may be found in [15], [16], [28].

3.3.1 Shared Segments

There is only one part of the real storage that can be accessed concurrently, that is, shared by different virtual machines on VM/SP: the storage occupied by the reentrant portions of saved systems or discontinuous saved segments. A saved system is a system that is saved under a unique name in pageable format on a CP-owned disk volume. It is the control program portion of an operating system which is IPLed frequently and used by many virtual machines. For example part of the CMS nucleus is reentrant and designed to be shared by the concurrently operating virtual machines with CMS as its operating system. Discontinuous saved segments are one or more discontinuous 64K byte segments saved with a given name on a CP-owned volume so that the named segments can be dynamically attached to and detached from a virtual

machine according to the machine's requirements during its operations. These discontiguous saved segments are those parts of code which are required only sometimes by virtual machines like the compilers.

The shared segment facility is used for reducing the real machine resource requirements when multiple virtual machines are supported. There is only one copy of the shared segments present in real storage and in external page area, regardless of the number of virtual machines sharing these segments concurrently. The page table and the swap tables of the shared segments are also shared, thus reducing the real storage required further.

Shared segments may be optionally protected or unprotected. The protection of the shared segments is provided by CP. When a shared segment is protected, CP ensures that a shared segment modified by one virtual machine is not accessed by another machine. When a shared segment is not protected, all the sharing virtual machines will be affected by the modifications made by one of the virtual machines.

With the protection of the shared segments on, CP checks if any part of the shared segments has been altered by the current virtual machine every time before CP dispatches a new virtual machine. There are several ways that a virtual machine may change any pages within the shared segments. If the changes are made by commands like TRACE, ADSTOP or STORE, the virtual machine issuing these commands will get an error message and CP will place the modified page of the shared segment with its page table and swap table in nonshared mode exclusively accessible by the virtual machine itself. All other users of the shared segment will continue to access the original copy of the shared segment and its newly created page table and swap table. If a virtual machine alters a shared segment by some commands other than TRACE, ADSTOP or STORE, it will get a warning message from CP and the modified page will be returned to the free storage by CP so when that page is referenced by a virtual machine later, CP will page in the original copy, that is, every virtual machine, including the one that

made the change, continue to access the unmodified segment.

For unprotected shared segments, CP does not check whether any page of these segments is altered by the current virtual machine before dispatching a new virtual machine. Therefore the changes will not be detected and the modified pages will be accessible by all the sharing virtual machines.

However if the command STCP is used to alter the storage of a shared segment, the alteration is made to the storage and the page changed is written to the CP-owned volume. All the virtual machines, including the one issuing the STCP command will access the modified segment in shared mode just as before the modification is made.

3.3.2 The Spooling Facility

The spooling facility simulates the existence of a virtual unit record device, such as card reader, card punch, printer or console, using the direct access disk space on CP-owned volumes that is allocated for spooling. The card input for a spooled virtual card reader is transmitted from the real card reader to the spool input files on disk by CP. The programs that are requesting input from the spooled card reader will obtain data from the spool input files when CP processes these input requests. The output from programs for a spooled printer, punch or console is directed by CP to spool output files on disk, which will be written to the real device by CP at a later time after these files are closed.

When a spool file is created, it is assigned certain attributes. Among these attributes are the owner identification which is the identification of the virtual machine to which the spool file belongs, and the device type which associates this spool file with one type of spooled unit record device.

Spool output files for spooled virtual punches, printers and consoles of any virtual machine can be transferred to the spooled virtual card reader(s) of the same

or any other virtual machines. No physical copying of files from one disk area to another is needed. The owner identification assigned to the spool output file is changed to that of receiver virtual machine to which the file is to be transferred, and the file is identified as a card reader spool file instead of output (punch, printer or console) spool file, that is, the device type of the spool file is changed to card reader. After these changes a spool output file of a certain virtual machine becomes the spool input file of another (or the same) virtual machine.

3.3.3 Message Passing

There are three facilities by which concurrent virtual machines may pass messages to one another, namely, the Inter Program Communication Facility(IPCF), the Virtual Machine Communication Facility(VMCF) and the Inter-User Communication Vehicle(IUCV). IPCF is a facility locally built in the department utilising the IUCV and VMCF communication systems. The functions provided by these facilities are examined here.

To unify the terminology, 'the sender' or 'the sender virtual machine' means the virtual machine which is referred to as 'the source virtual machine' or 'the source communicator', and 'the receiver' or 'the receiver virtual machine' refers to the virtual machine which is named as 'the sink virtual machine' or 'the target communicator' in the VMCF, IUCV documents.

VMCF

A standard facility of CP, called Virtual Machine Communication Facility(VMCF), provides a method by which one virtual machine may communicate and exchange data concurrently with any other virtual machines operating on the same VM/SP system.

When VMCF is used for communication between two virtual machines, data is transferred directly from the sender's virtual storage to the receiver's virtual storage. There is no internal buffering of the data within CP, the data involved is always retained by either the sender or the receiver virtual machine. The amount of data that can be moved in a single request is limited only by the virtual storage sizes of the transaction parties. VMCF locks one data page for the sender and one for the receiver each time during the entire transfer operation performed for a request. The transfer does not involve devices because VMCF uses userids to logically address data from one virtual machine to another virtual machine.

VMCF is designed to support a communication facility that can be used concurrently by multiple virtual machines.

The virtual machine communication using VMCF is invoked via a DIAGNOSE instruction with a 40 byte parameter list, in which the VMCF subfunction like send or receive and information like the address and length of the message area are given. A special external interrupt is used to notify a receiver virtual machine of a pending transfer of data, which the receiver will get after a sender's send request. The external interrupt is also given to the sender virtual machine to report the completion of a communication transaction which may be an one way data transfer or a two way exchange of information.

Each transaction is an one-to-one communication, where only two virtual machines, the sender and the receiver, are involved. When making a request to send some data, the sender specifies the receiver virtual machine by giving its identification. And similarly when making a receive request, the receiver specifies the sender virtual machine's identification.

All communication requests including the send, receive requests are asynchronous. For the sender of a message, the send request initiates a data transfer operation and the sender may continue its processing without waiting for the

completion of the physical transfer. Obviously it is the sender's responsibility to ensure the integrity of the message area until it receives the transfer completion interrupt. For a receiver, the receive request will finish either with the data transferred into the receive area or with an error code indicating no message is found (no message has been sent to it). There is no waiting (suspension of a VM) until a message comes. Normally a receiver issues a receive request after getting the interrupt notifying a pending message.

IUCV

Similar to VMCF, IUCV is a facility, supported by CP, to allow virtual machines to pass data from one virtual storage to another. The maximum amount of data that may be transferred by a single request is 32767 bytes.

Each IUCV communication takes place between a sender and a receiver (thus an one-to-one communication) over a predefined linkage called a path. A virtual machine may define multiple paths and any two virtual machines may have multiple paths between them. Communication is initiated through IUCV macro instructions in programs running in virtual machines. An external interrupt is used in a similar way as in VMCF to coordinate the virtual machines' communication activities. However external interrupt handling may not be necessary in at least one circumstance. By using the TEST MESSAGE subfunction of IUCV a receiver virtual machine may check for any pending messages. In the case when no message is pending the virtual machine enters a wait state till a message becomes pending, then the TEST MESSAGE function will be re-executed. This provides a virtual machine a means to wait for a message.

There is no synchronous send operation provided by IUCV. The sender does not wait for the completion of the transfer operation of its message from its virtual storage to the receiver's virtual storage. The potential receiver's receive request terminates with the message received or a code indicating no message is found. In other

words the receive request is also asynchronous. The synchronous receive operation, nevertheless, may be achieved by the combination of the TEST MESSAGE function and the RECEIVE function.

Like VMCF, CP and IUCV do not buffer the data transferred from the sender to the receiver.

IPCF

The VMCF and IUCV facilities form the basis of the IPCF system. Therefore basically IPCF is also a facility to transfer data directly from storage to storage among concurrently executing virtual machines. There are, however, a few features of IPCF which are not available in VMCF or IUCV and its interface in the form of a Pascal procedure call with parameters is easier to use than both the DIAGNOSE instruction for VMCF and the IUCV macro instruction.

The important features of the IPCF are summarized here.

. Routes

- programs wishing to communicate with each other using IPCF first define 'routes'.

A virtual machine may define multiple routes. Each route may be 'connected' to several virtual machines. Multiple routes may exist between any pair of virtual machines.

- a route may be defined such that messages only flow in one direction, or in both directions.

. Synchronisation

- the virtual machine that requests a message to be sent may be suspended until the receiver virtual machine has received the message, or it may continue its other work and check at a later time if the transmission has been completed or be informed automatically when the transmission completes.
- a virtual machine may try to receive a message at any time. In asynchronous

mode, the receive request returns with a non-zero return code indicating no message is received if no message has been sent to it. But in the synchronous mode, the receiver is suspended until a message comes and is received.

. Three levels of operations

- the IPCF requests issued by a virtual machine may apply at one of the three levels. At the lowest level an operation request only concerns the virtual machine itself and one specific virtual machine connected to one of its routes specified by the request. Next level operations concern the virtual machine itself and all the virtual machines connected to a specific route of this virtual machine. For example, a send request at this level will send a copy of the message to all the virtual machines connected to the specified route, i.e. a route-wide broadcast. A receive request at this level will receive the first message sent along this specific route by any one of the virtual machines connected to this route. At the widest level of IPCF operations, a request will be applied to all the virtual machines connected to all/any routes of the request initiator. These three levels of operations in fact provide facilities of one-to-one and one-to-many communications.

3.3.4 A Preliminary Assessment

The shared segment facility with the protection on in fact provides a storage area which is concurrently readable but not writeable. This cannot be used to pass information from one virtual machine to another during concurrent execution of the machines. The shared segment facility with the protection switched off may provide multiple write access, but at the same time removes the safe guard of the integrity of this storage area therefore cannot be used.

The spooling facility can be used to pass information between virtual machines

but the information is passed through files on CP-owned disk volumes. Apart from the disk I/O involved, a file has to be opened and closed before and after some information is written to it. All these have to be done every time a communication is necessary. Although spooling files are used for passing batch jobs from virtual machines of CMS users to the batch virtual machines and back it is not efficient for FVMS's purpose for the following three reasons: (1) concurrent SVMs in a FVMS communicate with each other with much higher frequency than CMS users send batch jobs; (2) the amount of data involved in each communication between the SVMs is in general much smaller than the job files that are sent to and out from the batch virtual machines; (3) the response time for a batch job requested from a CMS virtual machine is not critical but the response time for a service request between two SVMs directly affects the user satisfaction with the system. Under this circumstance passing information through a disk file is much too slow.

Using message passing with VMCF as the basis for a communication system in FVM is feasible. Work needs to be built on top of it to provide the functions not available in VMCF. An example is VMCF does not provide synchronous receive of messages.

IUCV can provide asynchronous receive operations as well as synchronous receive operations. However using IUCV (or VMCF) directly in the SVMs of the FVMS involves interrupt handling which has already been done by the IPCF .

By far IPCF seems the most suitable facility. It supports more functions that are required by the cummnications in the FVMS than other inter-virtual machine communiucation facilities with the low level implementation details like interrupt handling hinden below its Pascal procedure call interface. In the following sections, IPCF is examined in more details and how it is used to provide the FVMS communication is presented.

3.4 The Inter Program Communication Facility (IPCF)

IPCF is based on the VMCF and IUCV communication systems but provides additional features and a simpler, easier to use interface. When implementing IPCF functions, IUCV is used in preference to VMCF. This is explained later.

Messages are sent along routes and multiple virtual machines(users) may be connected to a route. In this section where IPCF is described, 'virtual machine' and 'user' are used to mean the same object.

Communication operations may apply to all the users connected to any/all route of a virtual machine(**system wide requests**), or to all the users connected to a given route(**route wide requests**), or to a single user connected to a particular route(**user wide requests**). There are ten IPCF functions. Their characteristics and effect are described here. Some of them are not documented in the IPCF document[28] and are given here as the results of the experiments done on IPCF. Pascal programs request IPCF functions using the external procedure IPCFPAS:

```
PROCEDURE IPCFPAS(VAR RPL : IPCREQ;  
                  VAR ErrorCode : INTEGER);
```

The first parameter is a Pascal RECORD type variable which specifies the parameter list for the request. The parameter list contains information like the IPCF function requested and any parameters required by the function. The detailed structure and its description of the parameter list is given in [28]. The second parameter is set by IPCF to the error code resulting from the request.

The Open Request. The system wide open request initiates IPCF processing and has to be done before any other IPCF requests may be issued by a virtual machine. After this

request, a virtual machine may define its communication routes one by one with route wide open requests. Each request defines the name and the type of a route. The type may be one directional(send only, or receive only) or two directional(send and receive). Messages may only flow in the directions defined by these requests. The user wide request connects a given route to another user(virtual machine) by specifying its own route name, the other virtual machine's identification and the route name as used by the other virtual machine.

Each virtual machine may define multiple routes, has multiple users connected to a route. Any pair of virtual machines may set up multiple routes between them. Fig 3.1 shows several possible connections between virtual machines. In Fig 3.1(a) each virtual machine has routes to and may communicate with any other virtual machines. In Fig 3.1(b) virtual machine A has a route which is connected to B,C and D. A may communicate with any of them individually(user wide requests) or all of them in one transaction(route wide requests). B,C and D may communicate with A but no communication may take place among themselves. Fig 3.1(c) allows many to many communications where both A and B may send messages to C or D or both and C,D to A,B but no communication between A and B, nor between C and D.

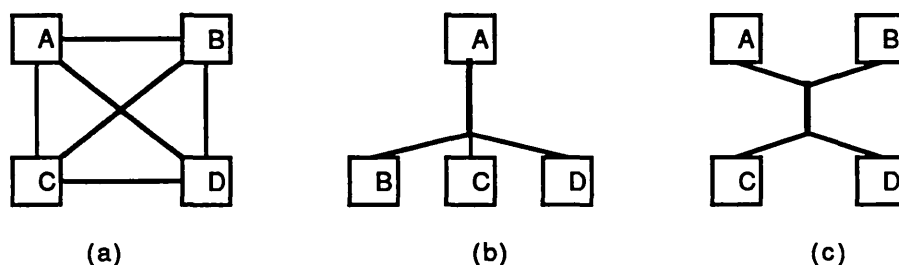


Fig 3.1 Several possible ways of setting up communication routes among virtual machines.

The Send Request. After communication routes have been established between any two virtual machines, messages may be transmitted along them. The sent request initiates such transactions. As in VMCF and IUCV the physical transfer of a message happens

only after the receiver virtual machine has made a receive request. A user wide send request specifies the transfer of a message from the sender to a single receiver connected on a given route. A route wide request specifies the transfer of a copy of the message from the sender to all the users connected on a given route. A system wide send is intended for all the users connected on all of the sender's routes to get the same message.

A send request may be synchronous or asynchronous. A synchronous request returns control to the sender virtual machine when the receiver(s) have got the message. An asynchronous request finishes after IPCF has validated and accepted it. The sender has the choice of using a system buffer or not using it. IPCF only buffers one message between a particular pair of virtual machines along a particular route. There may only be one outstanding message between any particular pair of virtual machines along a particular route in a particular direction at any one time. In other words, after a message is sent by a sender to a receiver along a route and before the receiver finishes receiving the message, any attempt by the same sender to send another message along the same route to the same receiver will generate a "route busy to the user" error message from IPCF and the send request will be rejected. However the sender may still send messages along other routes to the same receiver or to other receivers, and the sender may even send messages along the same route but to different receivers. Therefore concurrent communications may take place along different routes from one virtual machine to another or along the same route from one virtual machine to different virtual machines.

The Receive Request. This request can be used to receive one message and it may be done synchronously or asynchronously. A synchronous receive may involve the suspension of the virtual machine until a message arrives and is received. An asynchronous receive does not wait for a message if one cannot be found.

The system wide request receives one message from any connected user, that is, any virtual machine who is connected to a route of this virtual machine. The route wide request receives one message from any user connected on a given route. Finally the user wide request receives one message from a given user connected on a given route.

The Await Request. This request is used to wait for a message to arrive without receiving it. A system wide await request waits until a message which may be sent by any user connected to any route of the virtual machine arrives. A route wide request waits for a message sent by any user connected to a specific route. A user wide await request only waits for a message from a specific user connected to a specific route.

The virtual machine which gives the request will be suspended only when there is no message already waiting to be received in the scope concerned (system wide, route wide or user wide). If there is such a message, the request immediately ends.

The Waiting Request. This is to check for the virtual machine if there is a message waiting to be received from any user connected on any route(system wide request), or from any user connected on a particular route(route wide request), or from a particular user connected to a particular route(user wide request).

The request returns with the information of the existence of a pending message or otherwise. No waiting is involved.

The Connected Request. This request determines if any user has been connected to any route(system wide request), if any user has been connected to a given route(route wide request), if a given user has been connected to a given route(user wide request).

The Check Request. This request enables a virtual machine to check if a previous asynchronous send request has been completed, that is, if that message has been

received by the receiver.

A system wide request finds out if any asynchronous send request on any route has been completed. A route wide request finds out if any asynchronous send request on a given route has been completed. A user wide request finds out if the previous asynchronous send request to a named virtual machine on a named route has been completed.

The Cancel Request. This request gives a virtual machine the ability to cancel a previous asynchronous send request before the intended receiver receives the message. A system wide request cancels all outstanding messages sent on all the routes. A route wide request cancels all the messages sent on a named route. A user wide request cancels the message asynchronously sent to a named virtual machine on a named route.

The Close Request. The close request terminates IPCF processing with a given user on a given route(user wide request), all IPCF processing with all the users connected to a given route(route wide request), all IPCF processing along all the routes(system wide request). A route wide close request must have been issued for every route of the virtual machine before a system wide close request will be accepted.

The Exit Request. This request enables a virtual machine to define a routine to be used as an exit routine in one of the four circumstances:

- (1) a validation routine is called whenever another virtual machine tries to do a user wide open request specifying this virtual machine as the destination. This routine serves the purpose of deciding to accept or reject a connection.
- (2) a journal routine is called for every IPCF request made by the virtual machine. This offers the virtual machine the opportunity to record all its IPCF activities.
- (3) a completion routine is called whenever an asynchronous send request is completed. This allows the virtual machine to do some automatic processing after

an asynchronously sent message has been transferred to the receiver.

(4) an arrival routine is called whenever a message is sent to this virtual machine.

It can be used by a program which wishes to perform some other work when waiting for a message and be able to do something immediately after a message arrives.

One point should be noted that Await request cannot be used in a certain scope (system, route or user wide) if an arrival exit routine has been defined in that scope. Apparently the interrupt notifying the arrival of a message to a virtual machine would not terminate its state of waiting initiated by an IPCF await request but cause the activation of the arrival exit routine when such a routine is defined. And an exit routine always returns control to where it was before its execution thus the virtual machine will be in a permanent suspension state. This is a situation that should not happen and a point where IPCF could be improved.

3.5 The Assessment of IPCF

How suitable the IPCF is as an inter-SVM communication facilities will be examined from several aspects in this section. These include the functions provided by IPCF and those not provided, the errors encountered during the experiments with IPCF and some performance considerations.

3.5.1 Functions Provided and Those Not Provided

Near the beginning of this chapter, the functional requirements of the communication system for the FVMS are specified. Basically, it is a system with one to one-among-many communication. Suspension of a communication partner from the

time it sends some information away to the time the receiver receives it is undesirable. Suspension of the receiver during the process of its trying to receive a message is sometimes required but in other times not required.

IPCF provides some facilities that can be used by the SVMs for communications in the FVMS. The asynchronous SEND request can be used for sending a message from one virtual machine to another virtual machine without suspending the sender. The asynchronous RECEIVE is a facility to receive a message sent by another virtual machine with no waiting if the message is not there. The synchronous RECEIVE will wait for a message and then receive it. Alternatively, the AWAIT request can be used to wait for any message coming from any route before an asynchronous RECEIVE to get it. The SEND and RECEIVE requests work together in such a way that a single virtual machine may accept and process concurrent messages originating from many different virtual machines. A virtual machine may also initiate concurrent message sending to many different virtual machines. The message queueing mechanism of the IPCF even allows concurrent message traffic from one particular virtual machine to another provided these messages travel on different routes which were established between the two virtual machines. However, no concurrent communications between two virtual machines are accepted along the same route in a particular direction.

The route wide (and system wide) receive requests are one to one-among-many communications which is what is required in the FVMS. However, the route wide (and system wide) send requests are one to many communications as a copy of the message is sent to all the virtual machines connected to a given route (or any route for system wide sends). This results in all the virtual machines connected receiving a copy of the message. Therefore the send and receive working together is in fact an one to many communication and is not directly suitable for communications in FVMS.

The user wide IPCF requests are one to one communications between virtual machines. The initiator of an IPCF request directly names the virtual machine engaged

in the communication transaction. To use IPCF as a basis of the communication system for FVMS, the problem of how to use one to one communication to achieve one to one-among-many communication has to be solved.

3.5.2 Errors

As the behaviour of the IPCF functions is not fully documented, a few programs were written to send or receive messages from one another so as to find out the exact behaviour about what can be done and what cannot be done with IPCF. These behaviour were described in the previous sections. In the process of sending and receiving messages among several virtual machines, some messages received appear to have part of the contents corrupted. Sometimes an error message "unknown IPCF error, code 2936280" was issued by IPCF. There was no apparent relationship between the occurrence of this unknown error and the corruption of the message content as sometimes they appeared together but other times only one of them occurred.

3.5.2.1 Unknown IPCF Error Code "2936280"

IPCF is based on IUCV and VMCF to achieve the communication facilities it offers to the virtual machines. When trying to set up a route to a certain virtual machine, IPCF tries to establish an IUCV connection(path) first and will use VMCF only when an IUCV connection is refused. An IUCV connection will be refused after the maximum number of IUCV paths allowed for the virtual machine have already been set up. The maximum number of IUCV paths that a virtual machine may have is defined by a statement in the virtual machine DIRECTORY with its default value being four.

When a route to a certain virtual machine is implemented as an IUCV path, all the communication along this route between the two virtual machines are achieved by

IUCV operations. Otherwise VMCF is used to carry out the communication requests.

It appears that when the unknown IPCF error occurred, it was always when IPCF was receiving a message via VMCF. That is, an VMCF interrupt arrived indicating a pending message, later on when this virtual machine was trying to receive the message, the return code was 2936280. This error did not occur every time when a virtual machine was receiving a VMCF message and there was no indication as whereabout this erroneous code was generated.

One way to find out if the error was in VMCF and not in IUCV is to force the IPCF to use IUCV connections solely by authorizing the number of IUCV paths to be more than what was required. This way VMCF would not be used by IPCF at all. IPCF could also be forced to rely on VMCF communications alone by limiting the number of IUCV paths to zero.

The directories of the virtual machines concerned were changed to include the statement defining the number of IUCV paths to a value greater than each virtual machine might possibly need. As a result, IPCF was using IUCV solely to do communications. After this change, tens of thousands of messages were passed between virtual machines and this IPCF unknown error never surfaced again. This seems to be the proof that the error code was originated in VMCF rather than IUCV. It is at least an indication that using IUCV alone would not cause the error.

However, the occasional corruption of part of a message still existed and seemed to happen once in every several thousands of messages passed among the virtual machines. An experiment was done to investigate this problem.

3.5.2.2 The Experiment to Investigate the Occasional Corruption of Messages

Four virtual machines were the senders of messages and one virtual machine

the receiver.

Two routes were set up between the senders and the receiver and four sender virtual machines were connected on these routes to the receiver virtual machine. The set up was shown in Fig 3.2.

The four sender virtual machines were concurrently executing copies of the same program. Basically a sender's action was a loop with a predetermined number of iterations:

```
loop n times:
    send a message of type P to the receiver along route 1;
    send a message of type Q to the receiver along route 2
end of the loop
```

Before a sender virtual machine sent each message, the CHECK request with the receiver virtual machine as the communication partner was issued, repeatedly if necessary, to make sure the previous message along the same route had been received.

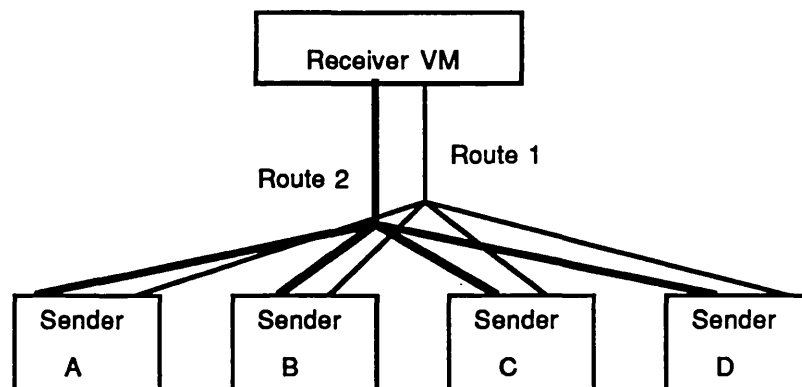


Fig 3.2 The set up of IPCF routes among the virtual machines in the experiment to find out why some message was corrupted.

The receiver virtual machine was executing concurrently with the four sender virtual machines. After setting up the two routes and having all the four senders connected to each of the two routes, the receiver entered into a loop:

```

repeat
    a system wide AWAIT to wait then decide from which
    route and which sender the next message is pending;
    initialise the message receiving area with contents
    totally different from the contents of the incoming
    message;
    RECEIVE the message;
    check each and every field of the message; if something
    is not what is expected, print that field;
until all the messages from all the senders are received

```

The senders and the receiver were run a number of times. Each time from four hundreds to one thousand messages were passed from each sender to the receiver along each route. The types of the messages along the two routes had message length of approximately 600 bytes and 4k bytes respectively.

In the 4k byte long message there was a field which was an array of 132 characters. Before each message was received the array part in the receive area was initialised to 132 '>' characters. In the incoming messages sent from various senders the corresponding array part was filled with blank characters. Once during the checking of a 4k byte long message, this array was found to contain the blank character in the first 32 characters and ">" characters in the remaining part of the array. Obviously if the message had been received correctly, the array should have been occupied by 132 blank characters. This error suggested that when control returned to the program in the receiver virtual machine after the RECEIVE operation, the message had not been completely transferred.

If the message was only partially received when it was being checked, would the data transfer be finished at a later time? To test this, code was added after the checking of the message, just before the end of the receiver's loop, that if an error was detected in the first checking, the same checking would be done again. In the subsequent executions of the programs, errors were detected in four messages among the twenty thousands or more messages that were sent and received. One error occurred in a 600 byte long message and the other three in the 4k byte long messages. In all

cases, the messages were found to contain the correct contents in every field on the second checking. Also when an error was detected the erroneous part of the message was printed out immediately after the detection, in some cases that part printed out was already correct by the time of the output.

These results confirmed that sometimes when control returned from the RECEIVE procedure, part of the message space still contained the values filled in before the current message was received. Therefore the actual data had not been fully transferred into the receive area when the checking of the message was carried out. However, this transfer of the message was completed sometime later.

One possible cause of this kind of situations can be explained as this: when the receiver virtual machine makes a RECEIVE request, after validating the request IPCF passes control to CP(IUCV) to transfer the message. Meanwhile the receiver virtual machine is not 'officially' in a suspended state. If the CP's operation to transfer the message was interrupted for some reason, for example a page exception, and at the end of the interrupt handling, the receiver virtual machine was permitted to continue execution as if the message receiving operation had been completed, part of the message would contain the original content and seemed corrupted. Somehow CP managed to finish the message transfer at a later stage.

3.5.2.3 The Effects of These Errors

The aims of this research are to find or build a feasible communication subsystem for FVMS in the VM/SP environment and do some experiments to compare the FVMS and the independent virtual machine system.

In the previous sections the errors found when running IPCF were described. The likely places where the errors occurred and the likely reasons that caused them were suggested or identified. However the exact places of the errors were not

pinpointed precisely and the causes of the errors not proven. Much more effort is needed to achieve these. How serious do the effects of these errors have on the communications between virtual machines?

Among the two types of the errors encountered, the unknown IPCF error can be avoided by IPCF using IUCV facilities solely. This seems a perfect solution as there is no reason to believe that relying on IUCV would incur any greater communication cost to the virtual machines than using a mixture of IUCV and VMCF facilities or using VMCF only. The difference between using IUCV and VMCF is briefly discussed in the following section.

As to the second type of the errors, the occasional corruption of part of the messages, or in fact the incomplete receiving of a message, it only happens once in every five or six thousands of messages. Therefore this type of errors only has a very low probability of causing the incorrect functioning of the FVMS unless the communications among the various virtual machines are so frequent that five, six thousands or more messages are going to be passed around during one experiment of the FVMS.

So the errors will not really affect this research. With the consideration of this, it was felt that the investigation of the errors should end here and it was more important to move in the direction of the main research goal. In all the subsequent work VMCF did not play any part in the communications between the virtual machines.

3.5.3 Some Performance Considerations

It has been stressed at the beginning of the chapter that a shorter communication delay and a smaller communication cost means that the FVMS has more opportunities to achieve better performance. Here some performance considerations about the IPCF are described.

The performance of IPCF depends on a lot of factors. It is affected by the way IPCF is used, the size of the storage of the real machine on which the virtual machines run, and the length of the code to be executed to serve the IPCF requests.

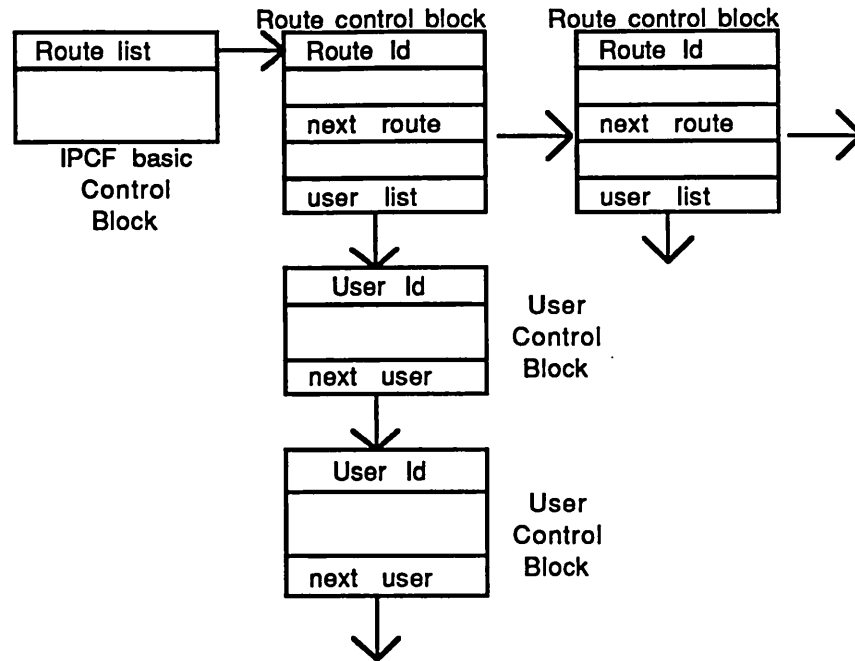


Fig 3.3 IPCF control blocks for all the connections of a virtual machine with other virtual machines.

The length of the code to perform IPCF operations can be divided into the length of the IPCF code[29] and the length of the IUCV code[30] as all IPCF requests are eventually carried out by IUCV operations. IUCV requests are processed by one of the three modules, DMKIUC, DMKIUE and DMKIUG with another module DMKIUA as the common entry point for all requests. When IPCF issues an IUCV macro call, the IUCV macro body is executed. It checks the parameters of the macro call and sets up IUCV work area then passes control to the entry point of the IUCV processing modules DMKIUA. In contrast, the path between the DIAGNOSE instruction in IPCF to the VMCF processing module DMKVMC seems considerably longer[31], [32]: (1) handling and interpreting a privileged instruction exception resulted from the execution of the DIAGNOSE instruction; (2) passing control to a privileged instruction interpreter;

(3) passing control to a DIAGNOSE instruction interpreter; (4) passing control to the VMCF processing module. However the VMCF processing module itself is much shorter than the IUCV processing modules put together. Thus on the whole there are no substantial advantages nor disadvantages in using IUCV rather than VMCF.

The length of the IPCF code to be executed is a function of the number of connections a virtual machine has with other virtual machines. This is not simply the number of IPCF routes but the number of route, user combinations which correspond to IUCV paths. Conceptually, all the connections of a virtual machine are recorded in a structure by IPCF as shown in Fig 3.3.

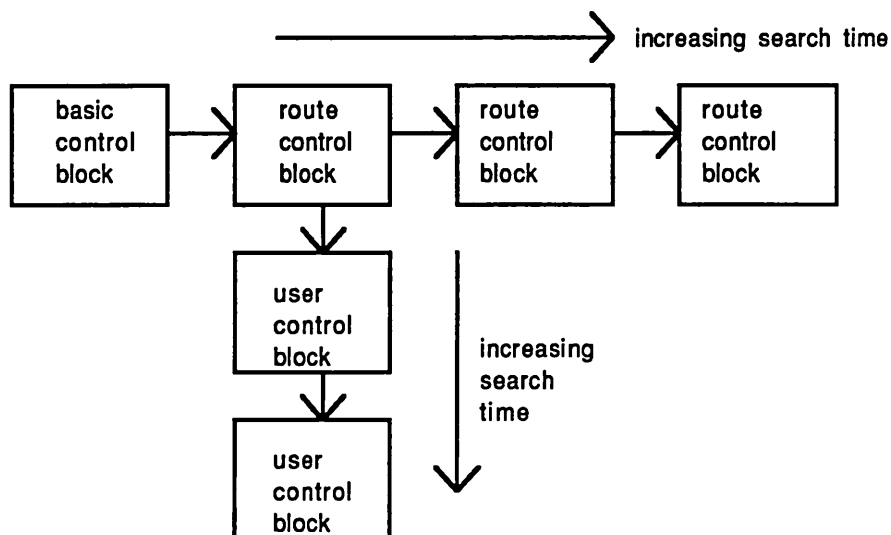


Fig 3.4 The relationship between the search time and the position of a user control block.

The route control blocks correspond to the routes a virtual machine has opened and are linked together through the 'next route' pointer. The first route control block is pointed by the 'route list' pointer in the overall system control block, the basic control block, which is built when a virtual machine issues a system wide OPEN request. The route control blocks are inserted into the list in such a way that the block pointed by the basic control block belongs to the route last opened by the virtual

machine. Similarly, a **user control block** is added into the 'user list' of a route to represent a connection between two virtual machines when either this virtual machine issues a user wide **OPEN** request to connect the virtual machine identified by the 'user id' to the route or this other virtual machine tries to establish the connection. The order of the user control blocks is also the reverse to that they are added in. The three types of the control blocks, namely, the basic control block, the route control block and the user control block, are the areas where system specific, route specific or user specific information are stored respectively.

To handle each user wide request, whether it concerns with an incoming or an outgoing message, IPCF searches through the route list then the user list to find the right user control block. To handle a route wide request like **SEND** or **CANCEL**, IPCF goes through the route list to find the route control block then acts on every user control block on that route. The handling of each route wide request, like **CHECK**, **CONNECTED** or one concerning incoming traffic, like **RECEIVE**, **WAIT** or **AWAIT**, looks through the route list to find the route in question, then through the user list of the route. The search stops when the first user control block giving a positive answer is found or when all the user control blocks on the user list are exhausted. Similarly, the handling of each system wide request concerning incoming traffic searches through the user list of each route starting from the first route at the head of the route list and stops when a positive answer is found or all the user control blocks on all the routes of the virtual machine are exhausted. Therefore two factors affect the search time to the control block(s) concerned:

- (1) the earlier the route was set up, the earlier the connection on a route was established, the longer the search time is; Fig 3.4 shows the increase of the search time with respect to where a user control block is.
- (2) the search time increases with the number of control blocks a virtual machine has, that is, the number of connections the virtual machine has with other virtual

machines. But if the total number of connections is not high, the search times and their differences are not significant.

Apart from the difference in search time to reach the right control block, IPCF also favours the more recently established connections among other connections along a particular route in another way. An obvious example is a route wide RECEIVE request which receives a message sent along a route by any user connected to that route. Suppose both virtual machines A and B are connected to route X of virtual machine C. Its IPCF control blocks would be linked like that shown in Fig 3.5, assuming A connects to the route before B does.

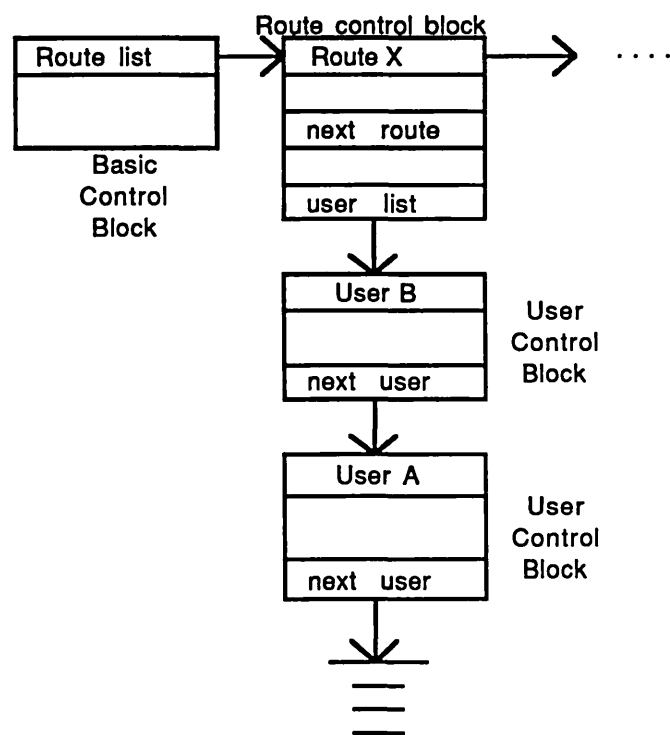


Fig 3.5 The IPCF control blocks for an example connection.

When C does a route wide RECEIVE on route X, IPCF looks through the user control blocks to see if any connection on that route has a pending message. Suppose both A and B have sent a message to C along route X, the message from B will be

received first even in the case when A sends its message before B does, because the user control block for B is found before A's in the search through the list. Suppose after this, B sends another message to C on route X. If C does a route wide RECEIVE again the message from B will be received again and the message from A will still be pending. This message from A will be received only when there is no pending message from B when C tries a route wide RECEIVE.

In the above case, the communications from B to C is heavily favoured than that from A. If another virtual machine D is connected to C on route X before A is, the user control block concerning D will be pointed by the 'next user' pointer of the user control block concerning A. As a result the communications from A to C, though not as favoured as the communications from B, will be more favoured than those from D.

Generally speaking, for route wide requests concerning incoming traffic, this First Found First Serve policy creates the phenomenon that, the earlier a connection is established along a route, the more the communications from that connection is 'discriminated against', the longer it takes a message to come through that connection. The same policy applies to system wide requests for incoming messages. The result is the earlier a route is set up, the longer it takes a message along it to come through. The lengthening in the communication times as a consequence of the First Found First Serve policy is certainly not linear and increases with the increases of the incoming flow of messages. This would cause 'blockage' in some connections if a virtual machine is not processing its messages faster than its communication partners are sending them.

The uneven search time for a user control block and the First Found First Serve policy both favour the connections more recently established and discriminate against those less recently set up. However in the case when the frequencies of incoming messages to a virtual machine from several communication partners differ, it may improve the IPCF efficiency by setting up connections with lower communication

frequencies before setting up connections with higher communication frequencies provided that on average the virtual machine is processing its messages faster than the messages are coming in. This may be difficult as the relative frequencies between different pairs of virtual machines may not be known and the frequencies may change with time. Nevertheless if the order of the connections can reflect the frequencies of the communications, more messages would flow from one virtual machine to another with shorter time and less messages would be penalised by the unequal treatments of messages travelling through different connections.

If the frequencies of incoming messages from various communication partners do not differ significantly or the frequencies change with time, it would be fairer if the route list and the user list are implemented as circular lists and the starting point for a search is moved to the next member of the list after each search is done through the list. The modification required are very simple: the 'next route/user' pointer of the last member points to the first member and each search would be followed by moving the 'route/user list' pointer to the next member. Obviously the condition for end of search should also be changed slightly. These changes to the control blocks would be necessary if there are a large number of SVMs in the FVMS.

3.5.4 Conclusion

IPCF appears to be a feasible basis for a communication subsystem to provide the communication facilities for the SVMs of a FVMS. IPCF offers facilities to concurrently send, receive, and wait for messages. The major problem is how to provide one to one-among-many communications using one to one communication facilities. The efficiency of the IPCF decreases with the increase of the number of connections a virtual machine has with other virtual machines. But when the number of connections is small, this decrease in efficiency is negligible. As to when the

efficiency, which is affected by the number of connections as well as by the frequency the virtual machines are communicating, would become too low for the FVMS, can only be decided by the experiments with the FVMS. The following section describes the design and the implementation of the communication subsystem of the FVMS on VM/SP. In chapter 5, the performance of the communication subsystem, using the communication primitives of IPCF, will be presented and discussed.

3.6 The Design and Implementation of the Communication Subsystem

The central problem is how to use one to one communication to achieve one to one-among-many communication. Several possibilities were considered. One was implemented and experiments done on it. In this section, 'server' is used as a synonym of SVM.

3.6.1 Round Robin

Consider the FVMS as consisting of several groups of SVMs. All the SVMs in one group offer a specific service. The workload of one service are entirely shared by the SVMs in that group. There are n groups of SVMs if there are n kinds of services recognized in the system. If a SVM provides more than one type of service, it appears in more than one group thus shares the workload of more than one service.

Each SVM in the FVMS keeps one list of the group member identifications for each service. There are n lists with n types of services in the system. This could be simplified so that a SVM only keeps those lists of servers whose services are ever going to be needed by its users. This varies with the nature of the service(s) each SVM is providing. For example, for a compiler SVM with I/O capability the command interpreter service is all its users ever need after a compilation is done.

A SVM sends its users requiring a particular service round the members of the server list for that service. For example, there are three SVMs to run all the user programs of all the users. Suppose these SVMs are identified as Loader1, Loader2 and Loader3. Whenever a command interpreter has a user who wants to run his program, the user will be sent to one of the loaders in a round order. That is, if the last user requiring loader service was sent to Loader1, this user would be sent to Loader2. If the last user went to Loader3, this user would go to Loader1. All the SVMs follow the same policy regardless of where the other SVMs are sending their users, with no consideration as to whether their destinations are busy or idle. If a destination is busy the user will queue in the SVM until it is his turn to be served.

This communication strategy is probably the simplest. A SVM is responsible for deciding which SVM should be its current user's next server. All the SVMs may be connected to each other along one IPCF route. Fig 3.6 shows the logical configuration. All the communications rely directly on the facilities provided by IPCF and nothing more needs to be built. A SVM to SVM (user wide) send request will send a user to the intended receiver and a route wide receive enables a SVM to receive a user wherever it comes from. All the SVMs are equal in the sense that there is no master-slave relationship between any two SVMs. For this reason there will be no shut-down of the system if one SVM crashes therefore the system is more reliable.

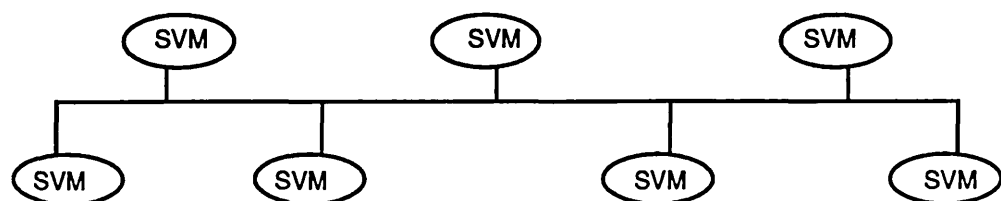


Fig 3.6 The connection of SVMs for a round robin way of distributing users to servers.

A variation of this round robin scheme is a SVM sends its current user to one

randomly chosen from the group of SVMs offering the required service using a hash function with the user identification as the key. This is in some way similar to the scheme used in [34]. In this scheme the process who requires a service from a server uses a hash function to decide which service allocator to ask for a server and each server is initially allocated to one of the service allocators using a hash function. The difference is here service allocators are not used.

One of the main disadvantages of these communication strategies is the possibility of overloading some of the SVMs while others become under-loaded. This situation arises when several SVMs happen to send their users to the same SVM to be served in a relatively short period and in the meantime there are other idle SVMs offering the same service. The users may also accumulate in the SVMs in which the users' jobs are taking longer to process than those in other SVMs. The uneven distribution of the workload would considerably slow down the traffic of users trying to go through this service. As a result, the performance of the FVMS is very likely to be limited by these bottlenecks appearing here and there from time to time.

Another disadvantage is its inflexibility to dynamic reconfiguration of the system. When there is a sudden high demand on one of the services, bottlenecks can be avoided by increasing the number of SVMs offering that service (within a certain limit). When this surge of demand passes, there is no need to keep a comparatively high number of SVMs in an active state (they are half active as the demand is less than the supply now). As every SVM holds a list of the servers offering the service, or at least every server whose users may ever need the service holds this list, all these SVMs have to be informed and their lists changed, every time there is a change in the number of SVMs sharing the workload of this service. The difficulty is who is able to detect these ups and downs in the demand of a certain service, given that there is no way to tell if the queue in a certain SVM is caused by a bottleneck mentioned earlier or by the saturation of the SVMs for that service. And who will inform the relevant SVMs

to change their lists and create, initialise (and later on destroy) these extra SVMs, once a group of servers are saturated.

The first disadvantage could be overcome with the employment of some sort of dynamic load balancing strategies where a heavily loaded SVM could pass some of the jobs to a less loaded SVM. However the second problem is difficult to solve so it was decided not to use this type of communication strategy.

3.6.2 Virtual Machines as Service Controllers(SCs)

To solve the problem of where to send a user when a certain user needs a certain type of service, a virtual machine may act as the access point for the service. Such a virtual machine is called a service controller(SC). There will be n service controllers as the access points for n types of services. All the users will be sent to the SC representing the service he wants. All the SVMs will ask the relevant SC for users.

Users sent to a SC queue there if no server is available to serve them. The SC then dispatches each user to the first available server at a later time. When a SVM sends a message to request for a user and there is none, the identification of the machine is queued in the SC. The SVM may then do a synchronous receive to wait and receive a user to serve. So SCs are the rendezvous for users and their SVMs.

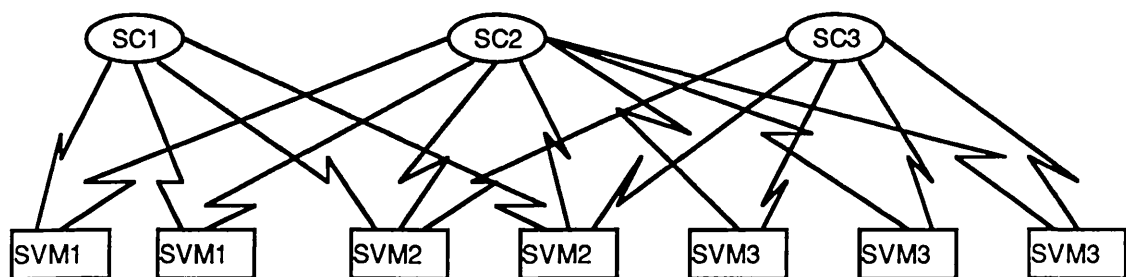


Fig 3.7 The connections among the SVMs and SCs in a FVMS.

Fig 3.7 shows an example of the connections between the SVMs and the SCs. In the example there are three types of services covered by three types of SVMs. Each SVM is connected to its SC and to any other SC(s) that its user may need access to as their 'sink' after the service in the current SVM has finished. For each SC, there are connections to each of the SVMs that provide the service to which the SC distributes the users, and connections to any other SVMs whose users may likely need the service.

Using virtual machines as access points to services seems the right solution. However, there are some disadvantages or difficulties in the operations.

- (1) In the example diagram, there are three SCs and seven SVMs. This ratio of the number of SCs and the SVMs are considered very high. Therefore unless there are comparatively large number of SVMs providing various types of services the overhead of supporting these service controllers might be too high for the FVMS to show its advantages over the present VM system. ---- the first disadvantage.
- (2) When a SVM offers more than one type of services it may have to ask more than one SCs for the next user to serve. If it makes the requests concurrently it may end up with more than one user being sent to it. If the requests are issued sequentially and only after the previous SC has given a negative reply, it may take a while before the SVM gets a user. If all the SCs give negative replies, the SVM has to make requests to all the SCs to queue itself there. The result of this action would be the same as making the requests concurrently in the first place. ---- the second disadvantage.
- (3) A system reconfiguration would be very difficult in the sense of converting some redundant SVMs to provide more popular services. When a long queue of requests builds up for a certain type of service, the SC has no information about the loads of other types of services so does not know whether there are some idle SVMs that may be converted. The only way to reconfigure the system is for SCs with

redundant SVMs to destroy some and SCs with long request queues to create some new SVMs and this is not as economic as converting SVMs. ---- the third disadvantage.

Therefore on a FVMS where the number of SVMs offering each service is small this scheme is certainly not efficient. And using SCs is not feasible or is very difficult when a FVMS has SVMs offering multiple services. But in systems where all SVMs provide a single service, or the services can be grouped into several service groups and SVMs do not offer services belonging to more than one group, and there are large number of them covering each service(or service group), and system reconfiguration is infrequent, this scheme could be suitable.

3.6.3 Virtual Machines as Communication Controllers(CCs)

A scheme similar to the one just described is shown in Fig 3.8. The virtual machines that were the service controllers in the previous scheme extends their role of distributing the users of a certain service to controlling all the incoming and outgoing messages of the SVMs offering that service. These virtual machines are called communication controllers(CC). Instead of sending a user directly to the SC which would distribute it to a server, the SVM sends its user back to its own CC which relays the user to another CC where it is distributed to the server.

The extra work a CC does than a SC makes the connections between a SC and those SVMs which provide the sources of the users for the service unnecessary. The connections between CCs replace these connections. As a result, the number of connections among CCs and SVMs decreases considerably, thus reducing the chances of those messages being delayed when the CCs are trying to receive them. Nevertheless, the three disadvantages of using SCs remain in the system and there is one addition to

the list: an extra level of indirection in the passage of a user from its current SVM to the next, that is, longer time and more system resources (the work of the two CCs) are required for the communication.

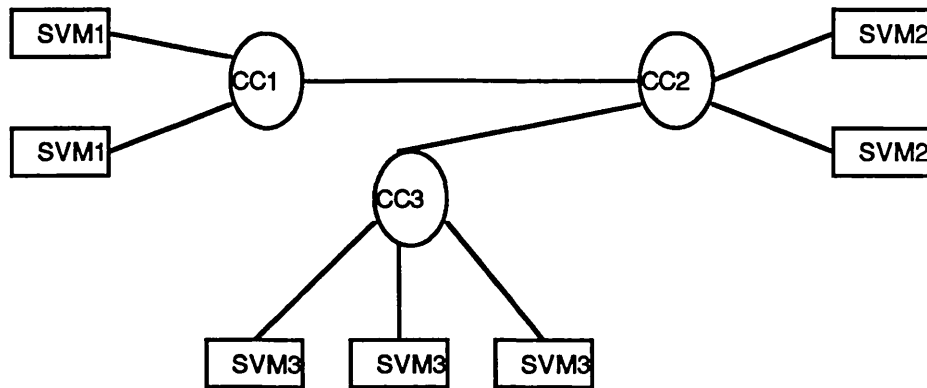


Fig 3.8 The connections among the SVMs and the CCs.

In summary, this scheme might be the only feasible solution in systems with the properties suitable for the SC scheme with an extra characteristics: the number of connections/communication paths between the SVMs is limited.

3.6.4 One Central Communication Controller(CCC)

The disadvantages of the SCs and the CCs would all be overcome by having a centralised controller, the Central Communication Controller(CCC), which manages the communications among all the SVMs.

Fig 3.9 shows the connections when a CCC is used. All the SVMs make their requests for users to the CCC and pass their users to the CCC to be distributed to the next suitable SVM. Users queue in the CCC when no server is available. The CCC is also the place where the identifications of the SVMs which are waiting for users are queued. Thus the CCC plays the role of matching service requests to their servers.

Having such a central control has a few dangers. There is the possibility of

the CCC becoming the bottleneck of the system. All the SVMs are connected to it. All the SVMs are concurrently sending messages to it to be dealt with. However bottleneck at the CCC would not materialise provided that the messages from the SVMs arrive at intervals longer than the time required to process them. While whether this is true can only be found out from experiments, the message arrival rate and the message processing rate are affected by the following factors:

- (1) the number of SVMs connected to it;
- (2) the frequency at which the SVMs are sending messages to it;
- (3) the time required by the CCC to process a message.

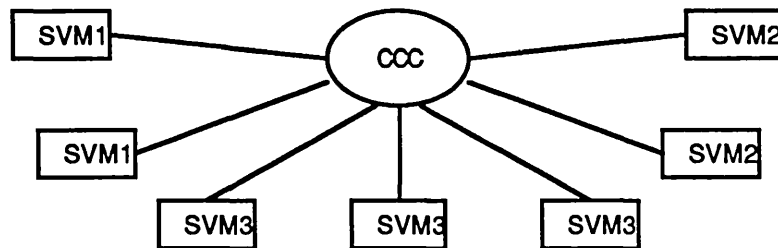


Fig 3.9 The connections between the CCC and the SVMs.

The number of SVMs connected to the CCC is the number of SVMs in the system. There is a limit to how many of them the CCC could efficiently manage. Obviously this number is related to the frequency the SVMs are sending messages, the higher the frequency the smaller the number.

The message frequency depends the nature of the tasks handled by each SVM. Generally speaking,

- (1) the finer the user's jobs are divided into different services, the more frequent the communications would be. Take the example of a user job compilation. If the compilation of a high level language program into a machine code program is a whole service, the job will be done by one SVM from the beginning to the end without communication with other SVMs. If the compilation is divided into two

types of services representing the two phases of a compilation, with one phase translating the program into an intermediary language and another from the intermediary language into machine code, the frequency of communications by the SVMs doing a single phase would increase from $1/(\text{time to do the whole compilation})$ to $1/(\text{time to do a single phase})$. Consider one user compilation as one unit, previously it involves the CCC passing the user to the compiler SVM and when the compilation finishes, the SVM passing the user back. If the compilation is done as two services in two SVMs the number of messages for the whole compilation would double.

- (2) The message frequency is also drastically affected by whether a SVM has I/O capability, that is, if the input, output required when a user is being served is done by the server itself or it has to be done in a different I/O server. More discussion of the service division will be given in the next chapter.
- (3) The smaller the size of the user job, the higher the communication frequency is. If two SVMs are offering the same service and one is given a sequence of small jobs while the other a sequence of long ones, the machine with short jobs will certainly have a higher user turnover, that is, more sending and receiving of users with the CCC per unit time. Also if all trivial commands are sent by the command interpreters to various SVMs to be answered, this would increase the communication frequency a great deal.

The amount of work involved in performing the basic task of the CCC on receiving a message from one of the SVMs is trying to match a user to a server or a server to a user. This would consume little CPU time compared with the CPU time required by a SVM to finish a (non-trivial) service. However, if the CCC tries to do the matching intelligently, the processing time of a message would somewhat lengthen. The intelligent behaviour which could be built into the CCC includes:

- (1) If no intelligence is built into the CCC, multiplexed servers and single-user

servers will be treated in the same way. That is, a server asks for a user, the CCC sends one to it, maybe after some waiting of the server. After this, the CCC has no record of the existence of this server and does not consider it again until it communicates with the CCC next time. This is perfect for a single-user server. However, after a user is dispatched to a multiplexed server which has not reached its full capacity, the server has to ask the CCC for another user and again till it has got all the users it can handle concurrently. Hence if the CCC retains the information about it so that when there is another user requiring the same service, it would be able to send this user to it automatically, thus saving the server's time to send the request(s), the system time to transmit the request(s) and possibly the time of the user to wait in the CCC before the request gets to it. Obviously the CCC has no way of knowing whether a multiplexed server has reached its capacity of serving users concurrently unless a server tells it.

The extra work involved in dealing with multiplexed servers differently from single-user servers is mainly the recognition of the fact that some servers need more than one user at any time and other servers only need one. Therefore queueing a machine means recording the SVM's identification as well as the fact that a server is multiplexed or not multiplexed and keeping the relevant information about the spare capacity of the server. After a user is sent to a server, nothing more remains to be done if the server is not multiplexed. When it is multiplexed, there are two possible cases: (a) the server has more spare capacity. The server needs to be queued if it is not already queueing in the CCC. The information about its spare capacity needs updating if it was already queueing. (b) the server has no more spare capacity. The server must be dequeued as it was certainly queueing in the CCC because of its spare capacity before obtaining the last user.

(2) take into consideration of possible work load balancing when matching users to

their servers.

- A SVM offering multiple services thus capable of serving users of different demands should be sent a user requiring a more popular service than others (more users are waiting for that service).
- If there are more than one service virtual machines waiting for users of a certain service, some are single-user server or multiplexed server with no user yet and some are multiplexed servers with at least one user being served but capable of accepting more user(s), a user should be sent to a server belonging to the first category so that users are spread across more servers.

Deciding which user to send to a server or which server a user should be sent to intelligently would lengthen the processing time of the CCC, probably only by a small amount. Different considerations for single-user and multiplexed SVMs under various situations would also lengthen its message processing time. However the amount of work an intelligent CCC does is still less than a program of a couple of hundred lines even without excessive looping in it. Therefore the likelihood of a bottleneck at the CCC is not very high. And if the frequency the incoming messages of the CCC is not such that the minimum processing time for a message has to be achieved, an intelligent CCC would considerably help the efficiency of the whole FVMS as mentioned in the introduction of this chapter. Whether minimum processing time for a message is demanded by the frequency of the messages can only be answered by the experiments of the FVMS using CCC.

Unlike the SC and CC schemes the concentration of information in CCC removes the difficulty in obtaining a user for SVMs offering multiple services. There is only the CCC virtual machine apart from the SVMs to support by the CP. Messages just go through one CCC instead of two CCs on the journey from one SVM to another. The intelligence of treating single-user and multiplexed SVMs differently and balancing the

workload of the servers for a service can be achieved by SCs and CCs as well as in the CCC. However the intelligence of choosing a user among several possible ones for a multiple-service server can only be put in the CCC and not in SC or CC.

Dynamic reconfiguration of the FVMS is very easy to achieve with the CCC as the central control of the user distribution. If the CCC finds that a queue of service requests is building up for a certain type of service, it can either create (or log on) some more SVMs providing that service or convert some SVMs whose service is not busy to offer the much demanded service. When there are redundant SVMs for some services all the time, the CCC may destroy(log off) some SVMs. To create a SVM for certain service(s) the AUTOLOG facility of the VM/SP system may be used under the CCC's instruction. To destroy a redundant SVM, the CCC may send a normal message to it with a NULL user. When a SVM receives such a message it will log off itself. To change the service a SVM provides has the advantage of avoiding the cost of virtual machine creation so if there is a redundant SVM on one hand and comparatively long service request queue on the other hand, such step is preferred than destroying and creation of SVMs. Naturally if it is a network of real machines and the machines are specialised from the hardware point of view, reconfiguration is not very sensible.

The use of the CCC is in some way similar to the scheme called mailboxes[33] in message based concurrent programming where a mailbox is the destination designator for sender processes and the source designator for receiver processes. But this is a mailbox with intelligence. One disadvantage of using the CCC for communication in the FVMS is its vulnerability to errors. A crash in the CCC would shut down the whole operation of the FVMS. This is in a way similar to a malfunction of the communication network shutting down a distributed system. This is also the main objection after the possibility of bottleneck researchers of distributed systems have against centralised control. However with so many advantages the CCC is considered best suited to FVMS' requirements if it is able to manage all the

communications passing through it.

In this section, the factors that affect the functions and the efficiencies of the CCC have been discussed. The implementation of it bearing in mind these factors will be given now. It is assumed here that the IPCF facility that transfer messages between virtual machines never fails and messages accepted by the IPCF will be received without any corruption of it by the receiver SVM.

3.6.5 The Implementation of the CCC

The messages between SVMs and the CCC

There are three types of messages being transferred between a SVM and the CCC.

- (1) The message contains the set of services provided by a SVM and its capacity as far as how many users it can accept. This is the information required by the CCC to decide what user(s) can be given to the SVM.

When a SVM has just been initialised it needs to send a request for user to the CCC in this type of messages.

- (2) The message is the user control block of a user, which includes, among other information, the type of service this user wants. The structure of the user control block will be given in the next chapter.

When in the process of serving a user, a service is required from another SVM, the current SVM sends the user in this type of message to the CCC and waits for the return of the user. An example is an I/O operation in the process of a compilation which has to be done by an I/O server. When the CCC is assigning a user to a SVM to be served, this type of messages is also used.

- (3) The message is the combination of the type 1 and type 2 messages just described.

This message combines the user and the SVM's capability into one communication.

The reasons for this will be given in the next chapter.

The Interface to the CCC

To pass a message to the CCC, a procedure

```
PROCEDURE IPCFPAS(VAR RPL : IPCREQ;  
                  VAR ErrorCode : INTEGER);
```

is called. One component in the parameter list(RPL) of the IPCF request is the address of the message(a pointer type variable in Pascal) and another component is the length of the message area. One of the three types of messages described above can be sent to the CCC.

- (1) When the message is of type 1. This is a request made by a SVM for one user (or more if it is a multiplexed server). The CCC finds a user waiting for the service (or one of the services) offered by the server and sends the user to the SVM. If it fails to find a user or after sending all the users the (multiplexed) server can serve it still has the capacity to receive more, relevant information about the SVM is retained by the CCC so that when users come they can be sent to it.
- (2) When the message is of type 2. A user is returned by a SVM. The CCC finds out the type of service this user needs. If there is a SVM with the capacity to serve him, it sends the user to it. If no SVM offering the service is available, the user is retained by the CCC so that when one is ready he could be directed to it.
- (3) When the message is of type 3. This is a message which returns a user AND requests for more user(s). The actions taken for type 1 message and those for type 2 message are both taken.

After passing a message to the CCC, a synchronous RECEIVE in the form of an IPCFPAS procedure call can be done to wait for the CCC's response. If the (multiplexed) SVM does not need to wait for the response, an asynchronous RECEIVE can be done at any time convenient to it.

The main control loop of the CCC

The CCC loops through the following three steps:

- (1) Receive a message;
- (2) If a user is passed through the message, deal with the user;
- (3) If machine capabilities are contained in the message, deal with the SVM.

The Queues maintained by the CCC

There are two types of queues, one for the users waiting for servers and one for the SVMs waiting for users. The user control blocks are put into the user queues. In a machine queue, the SVM's identification and its capacity(whether it is multiplexed and how many users it has currently got), are recorded for each SVM waiting. For each type of the services recognised by the system, there are these two types of queues. Therefore users queueing for different services are in separate queues and SVMs waiting for different users are also in separate queues. Those offering multiple services are in more than one machine queues. Needless to say, at least one of the user queue and the corresponding machine queue is empty at any one time.

Insertions into the queues. A user is inserted into a user queue at the end of it so as to have a First Come First Serve policy. All users have equal priority. A single-user SVM is inserted as the head of a machine queue while a multiplexed SVM is inserted at the end. This ensures that when more than one SVM is waiting, the single-user SVMs get their users before the multiplexed servers, increasing the chances of users being distributed evenly across the SVMs providing a service.

Requeueing a multiplexed SVM. When a user is distributed to a queueing multiplexed server and this does not make the server reach its full capacity, the machine is requeued to the end of its machine queue. This aims at spreading the users more evenly among the multiplexed SVMs. When there are more than one multiplexed SVMs

waiting, users arriving at the CCC consecutively would go to different SVMs.

The order the user queues are searched for a user. It has been pointed out that a SVM offering multiple services should be given a user waiting for a more popular service. A static way to implement this policy would be fixing the order the user queues for various services are searched, with the user queues for the 'more popular' services before those for the 'less popular' ones. For example, put the I/O service before the command interpreter service which is in turn before the compilation service. A more objective (and dynamic) way to judge the popularity of the services at different times would be by the lengths of the queues. A multiple-service server will get a user from the longest queue among the ones that are of its concern. This requires the number of users in each queue is maintained. However, more effort is needed to find out which queue concerned by a SVM is the longest. To keep the CCC's task simple and not to make it slow, the static method is considered enough.

Dealing with a user returned by a SVM

The algorithm in a pseudo Pascal language is simply:

```
IF the particular SVM queue is empty y
  THEN insert the user into the user queue for that service
ELSE BEGIN
  send a message containing the user control block to the SVM at the head of
  the SVM queue;
  IF the SVM is multiplexed THEN increase the number of users it is serving
  by one;
  IF this SVM has got its maximum number of users (one for single-user
  servers )
    THEN remove it from the queue(s)
    ELSE insert or requeue it in the SVM queue(s)
END
```

Dealing with a SVM requesting for job(s)

The steps in dealing with a SVM requesting for job(s) are:

- (1) Find how many users this SVM may accept(one for single-user server);
- (2) try to send as many users as possible to the SVM, searching the user queue(s) in the predefined order; dequeue the users sent out from the user queue(s);
- (3) If necessary, insert the SVM into the machine queue(s) with information about its capabilities and capacity.

Chapter Four

The Service Virtual Machine Submodel

Before the design of the SVM submodel, a question that why such a model is chosen has to be answered and this is the topic of the first section of the chapter. In the following section, some design issues concerning the model in general are discussed. The design of the individual components of the model is then presented. After an assessment of the model in which characteristics and scope of the model are discussed, the chapter ends with the validation of the model.

4.1 Why a Model ?

The communication subsystem for the FVMS has been built. To design and implement a Service Virtual Machine(SVM) subsystem would involve:

- (1) building the interface between the programs which offer the services of the SVMs and the communication subsystem. This interface would be responsible for sending and receiving users for the SVM, that is, the communication of the SVM with other components of the FVMS including the CCC.
- (2) building the application programs of the SVM. These are the software resources provided by the SVMs. The present application programs used by the CMS such as compilers are not suitable for the fact that they do everything within the virtual machine. For example, all the I/O transactions required by an application program are done via the I/O statements of the programming language used and are handled by the I/O routines of the CMS system and then the actual I/O is arranged by the CP. To modify these programs so that performances of the SVMs with I/O

capabilities and those without them can both be measured is no simple task. Besides, a whole range of application programs which offer the experimental flexibility of different combinations of services in a SVM and contain all the application details would have been needed.

Obviously, to build the SVM system would require substantial amount of design and implementation efforts. On the other hand, a SVM model would be a performance prediction tool with the convenience of experimenting different ways of distributing user oriented system software among the SVMs but without the need for considering application details. The major problem with a simulation model of a non-existent software system is its validation, as there is nothing to validate the model against. This is not entirely true for a SVM model. If a model of the SVM could be built such that

- (1) the SVM model is executing in the same environment as a real SVM would;
- (2) different software services could be combined into the SVM freely and when combining all the software services into a single SVM, the model behaves like a CMS virtual machine

the model is good enough for predicting the performance of the SVMs offering only subsets of the software services.

The environment for a real SVM is the lower level software and the (virtual) hardware of the virtual machine. The lower level software controls the virtual hardware. The virtual hardware is supported by time-sharing with other SVMs on the real machine, managed by the CP. The SVMs communicate with one another using the primitives and the protocols of the IPCF through the communication subsystem of the FVMS. By building the SVM model on top of CMS, running instances of the model in multiple CMS virtual machines, and using the communication subsystem designed and implemented for the FVMS(the CCC), each instance of the model has got the lower level

software of the SVM which is managing the resources of the virtual machine, the same multiprogramming environment as the real SVMs would be in, and the real communication support for the SVMs in a real FVMS. Therefore the SVM model would be executing in the same environment as the target SVMs being simulated.

If various routines or sets of routines carrying out software services are viewed as the building blocks of a virtual machine, the components simulating various services offered by a SVM are the building blocks of a SVM model. By putting all these simulating building blocks in the same instance of a SVM, a model of a SVM providing comprehensive user services is produced. In fact, it becomes a model of a CMS virtual machine. As it is a software model, it could be coded efficiently or inefficiently and this would affect its performance. However, if this model behaves like a real CMS virtual machine, if it is consuming the same (real) system resources in the same proportions as a real CMS virtual machine when performing the services and giving the end user the same performance figures, the efficiency of the coding of the model is irrelevant and it is a valid model of the CMS virtual machine, bearing in mind the model is executing in the same environment as a real CMS machine. When these building blocks are distributed among different instances of SVMs the software routines simulating various services would require the same amount of system resources to carry out the simulation. Each instance of the SVM model is executing in the same multiprogramming environment as the real SVMs would be in and employing the same inter-SVM communication mechanism as the real SVMs would. It follows that the SVM model would be a reliable performance prediction tool.

In this chapter a SVM's 'capabilities' refers to the services it provides to the rest of the FVMS.

4.2 Some Design Issues

In this section several decisions made during the design of the model and the reasons behind these decisions are presented. They include how the simulation model of a SVM is driven, how to provide flexibility in simulating SVMs with different capabilities and the types of services included in the model.

4.2.1 Simulation Controlled by Resource Consumption Definitions

In a real SVM, there are service routines (software packages) which carry out specific tasks, that is, services supported by the SVM. As stated in the last section, in order to produce a valid model of the SVM, the service routines in the SVM model have to consume the same system resources in the same proportions as the real service routines in a CMS virtual machine. For example, a compiler service routine would consume the same resources in the same proportion as a real compiler in a CMS virtual machine. It was decided that these service routines within the model would be driven by parameters which specify the resource requirements for serving each call on the service.

These parameter driven service routines of the SVM submodel should be constructed in such a way that the same routine can be used to simulate the services whose major 'characters' are the same. For example, the same service routine should be able to simulate a Pascal compiler and any other block structured language compilers. The only difference in the simulation is the difference in resource consumption in the individual cases. Naturally during the operation of the FVMS, a Pascal compiler service is considered as a different service to the compiler service of another language. But this does not prevent the model from using the same code to simulate these different services.

The resource requirements relevant are the CPU time for doing a service, the memory requirement in terms of the size of the working set, and the amount of input and output on various peripheral devices. The aim of choosing these resource requirements is to cause each SVM model which runs on top of the CMS in a virtual machine to put the same workload on the real machine as a real SVM would. As a result, the same resource sharing and contention environment in various components of the CP and the real hardware would be created when a complete FVMS model with the appropriate number of SVMs(models) is run. The CPU time requirement guarantees the SVM's share of the control of the real CPU for carrying out the service. The working set size would give the memory management part of the CP and the paging devices the same amount of work as the real service the model is simulating would do. By doing the same amount of I/O on various devices, the SVMs collectively would produce the same contention on the I/O devices and individual SVM would generate the same number of I/O requests which are handled by the CP as in the real service case. As I/O requests constitute the majority of the system requests a CMS virtual machine makes to the CP, the result is the overhead to support these virtual machine requests by the CP is generated.

It is assumed that the size of the working set of a service routine in a CMS virtual machine does not change considerably during the whole execution process of the routine. This is close to truth as in the majority of the cases, the contents of a working set varies with time but the size does not change significantly, at least for the major part of the execution duration. This is particularly true considering the specific subtasks carried out in a virtual machine such as editing or compiling.

A tolerance of five percent on model accuracy calibrated against the real services provided by the CMS was aimed for in total run time.

4.2.2 Simulation Flexibility

The model would provide flexibility and ease of distributing services among the SVMs. A SVM would be able to offer any subsets of the services available on the FVMS, including the full set of services. To achieve this flexibility, a model which includes the code for all the services would be more suitable than a number of different models each capable of performing a fixed subset of the services. The drawback for this model is its bigger size. However, only the routines which offer the services of the SVM will be called upon and the virtual machines supported by the CP generally has enough memory to hold the model code and its data. The whole model would be loaded into the virtual machine memory at model initialisation time and those routines not needed by a certain SVM would be paged out sometime and stay in the disk paging area without any further effect on the performance of the SVM model or the system as a whole.

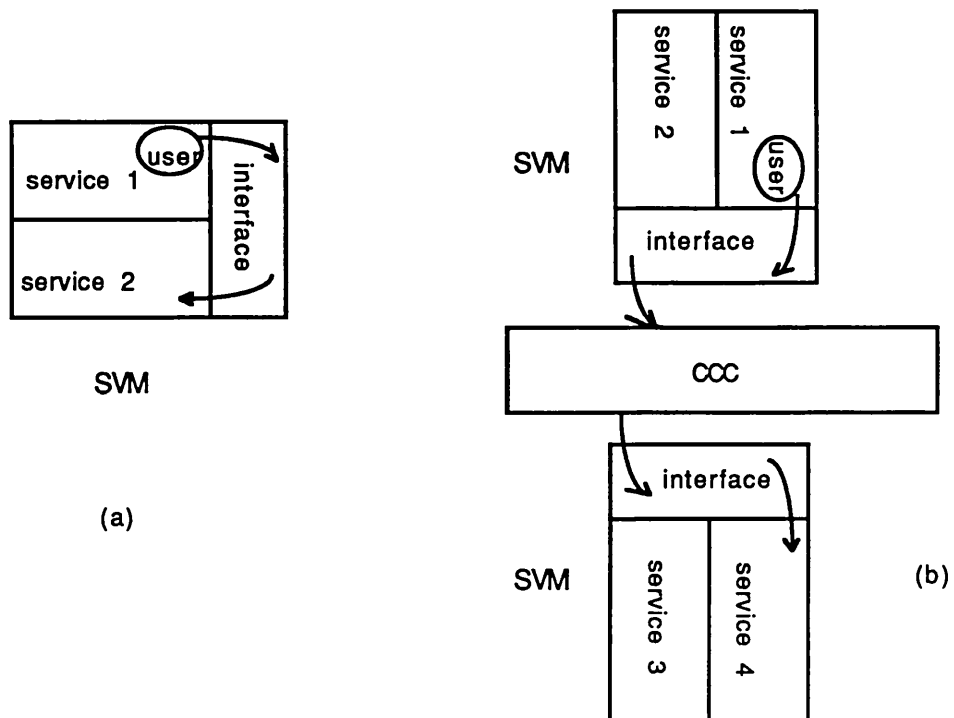


Fig 4.1 The transfer of users between services.

There is an interface between the different service routines which users pass through from receiving one service to receiving another. To achieve model flexibility, this interface is also the interface between the SVM's service routines and the FVMS' communication subsystem. The interface would maintain the transparency of the locations of other services so each service is unaware of whether they are available in the current SVM or not. If a required service is provided in the current SVM, the user would be passed to the appropriate service routine without involving the communication subsystem, as illustrated in Fig4.1(a). If it is not, the communication interface would send the user to the CCC where he is delivered to another SVM for service. Fig 4.1(b) shows the transfer of the user. This approach requires the interface of each SVM to have the knowledge of its capabilities -- what services it is providing. It has the advantage that it is making the minimum usage of the communication facility and therefore pays the minimum communication cost during the functioning of the SVM.

Another approach which also achieves simulation flexibility is whenever there is a change of services for a user, the interface would pass him to the CCC, even in the case where the service is available in the same SVM and he might be passed back if there is no other suitable SVMs to serve him. This approach has the merit that the code executed when the SVM is providing the full set of services simulating a CMS virtual machine would be exactly the same as the code executed when the SVM is providing a subset of the services as part of the FVMS. The main advantage is its potential load balancing effect between different SVMs. As the SVM chosen for the new service by the CCC would be the one which has been waiting for work for the longest period the approach would keep the maximum number of SVMs busy. For example, a SVM, identified as SVM1, provides the command interpreter service and the I/O service. Another SVM offers the I/O service and probably some other service, for instance the compiler service. The latter is referred to as SVM2. A user in SVM1

wants to input a command. SVM1 sends him to the CCC instead of serving him locally. Suppose SVM2 is idle at the moment and the CCC passes him to SVM2. Then SVM1 will be able to serve the next user who wants the command interpreting service. Now both SVMs are busy and two users instead of one are being served.

However this strategy was rejected due to the following reasons. Firstly unnecessary communication cost would be incurred making the SVMs in a FVMS less efficient. Secondly it would introduce extra communication cost in between every service in the SVMs simulating CMS machines which does not exist in the CMS machines. The difference between the code executed by a SVM simulating a service machine of a FVMS and that simulating a CMS machine would reflect part of the extra communication overhead in the SVMs' part of the FVMS. Moreover the load balancing effect of the first strategy may not bring efficiency in every case. Take the same example described above. Suppose after the user(called User A) is sent from SVM1 to SVM2 by the decision of the CCC for inputting the next command, SVM1's next user(called User B) wants the compiler service. While SVM2 is waiting for User A to type in his command, SVM1's User B is delivered to SVM2 via the CCC. Now SVM2 starts to serve User B and SVM1 is idle. When User B in SVM2 needs to open his compiler files, he is passed to the CCC which relays him to SVM1. This whole procedure ends up with the users of both SVMs, User A and B, have their I/O service done in a different SVM. The communication costs paid by both SVMs are not negligible.

4.2.3 The Scope of the Model and Its Limitations

The SVM model is to simulate the user-oriented software services provided by the CMS. Therefore those CMS functions listed in 2.1.3 that manage the resources of the virtual machine and handle the interrupts should still be done in the CMS when the SVM model is running on top of it. The user-oriented software services could be

roughly divided into the following areas:

- (1) the language processors -- compilers, assemblers.
- (2) the processing and execution of user files;
- (3) the editing service;
- (4) services that concern the user's file storage like COPY, MOVE, BROWSE, ERASE, FLIST, PRINT or RENAME;
- (5) the text formatting service;
- (6) the I/O service;
- (7) the command interpreter service.

4.2.3.1 The Language Processing Service

The language processing service is the software service where user programs are translated into machine code programs. Compilers and assemblers are the software routines. These routines would need to input the program to be compiled/assembled and output the results of the compilation/assembly. There are two choices:

- (1) The routines will always do I/O by themselves like the present compilers/assemblers. The I/O statements are issued by the routines, passed to the appropriate CMS I/O routines which request the CP to start(simulate) the I/O operations. In this way the I/O service is embedded in the language processing service and it is not possible to do experiments with the model where a language processing SVM could have another SVM to manage the I/O for it.
- (2) The language processing routines in the model are coded so that I/O is a distinct service and it may be done by another SVM or by the same SVM.

If a language processing SVM has an I/O SVM to do its input and output, all the input/output data needs to be transferred between them via messages. From the whole

system point of view this is not efficient as more work has to be done(apart from the I/O to be done by the I/O SVM, the data has to be transferred). From the language processing SVM point of view time is saved if the time taken to communicate with the I/O SVM is less than the time needed for the I/O operation. This is only useful if the language processing SVM is serving more than one user concurrently -- a multiplexed SVM. The conclusion is if the SVMs are virtual machines sharing one physical machine I/O should not be done in separate SVMs; if the SVMs are running in separate real machines having an I/O server(SVM) may be advantageous. To make a flexible SVM model it was decided to keep the I/O service as a distinct service so that I/O could be done locally or in another SVM.

As said before, the model driven by different resource consumption characteristics will be able to simulate the behaviour of different language processors.

4.2.3.2 The Editor Service and the Command Interpreter Service

The editor service is a highly interactive service which does not put a heavy load on the system. A user of the editor types in the change(s) he wants to make on a file then the editor makes the change(s). These are repeated until the user quits the editor. In terms of real time, only a very small proportion of time is taken by the CPU to make the changes and a large amount is spent by the user to think of and type in the editor commands. It seems natural to multiplex several users in one editor SVM so as to reduce the number of virtual machines CP has to support -- one of the potential advantages of the FVMS.

The command interpreter service is very similar to the editor service in that both are highly interactive and both have similar pattern of CPU time consumption, that is, a short period of processing followed by a comparatively long period for an input/output operation. Therefore it is reasonable to assume that both services would

behave in a similar way in a SVM of the FVMS. For this reason only the command interpreter service is included in the SVM model.

4.2.3.3 The Execution of User Files

There are basically two types of user files executed by CMS. One is the machine code(object code) file output from a compiler or an assembler. Another is the EXEC file interpreted and executed by the EXEC processor.

The execution of EXEC files is not included in the model for simplicity reason. As an EXEC file is a sequence of CMS commands that can come from the user's keyboard, nothing important is lost in the study of the FVMS by omitting this side of the service.

The user programs that are object code may be classified as interactive or non-interactive according to whether they do I/O to the user's terminal. The simulation of the execution of interactive programs requires human resources to communicate with the model. When there are several SVMs executing interactive programs the human resources are very difficult to obtain. If a virtual machine is used to simulate the human input it would interfere with the performance of FVMS as (1) this virtual machine would compete with other SVMs for machine resources; (2) it would add burden to IPCF as the 'human input' has to be transmitted to the appropriate SVM. Therefore only executions of non-interactive programs are considered in the model. The CP command SLEEP may be used to simulate the idle state of the virtual machine when it is waiting for an input but it does not simulate the work required from CP in handling a virtual machine's I/O request. Also the SLEEP command cannot be used if a SVM is multiplexed to execute several user programs concurrently with one program waiting for an input and another running.

With the increasing wide spread usage of computers, interactive processing is more common than ever in numerous professions. However, user need for batch

processing grows proportionally to interactive computing[8], a fact that increases the need for efficient non-interactive processing service. So the exclusion of the execution of interactive programs is not such a big handicap as it looks.

4.2.3.4 Other Services

Apart from the archive and tape facilities, services concerning the user's file storage are mainly the COPY, MOVE, ERASE, RENAME, PRINT, FLIST and BROWSE services. These services are not considered in the model. However, if the services of BROWSE and FLIST are included in the model a command interpreter SVM with I/O capability would be the suitable place to offer these services. The reason is these commands generally involve a little CPU's processing time on serving the command and updating the user's file directories, it would not be worthwhile to have the command interpreter to pass the user to a different SVM, have the command served and return the user to the command interpreter. The BROWSE service is similar to the EDIT service in that it accepts commands to display different portions of a file so it is not implemented for the same reason as for the EDIT service. The FLIST service displays a user's file directory and accepts CMS commands which work on files. As it is just a more user-friendly environment for a user to give commands it is not included in the SVM model.

The text formatting service is not very frequently used as a user will not do text formatting in every one of his sessions. Therefore it is excluded from the services provided in the SVM model.

The command interpreter service and the I/O service are simulated in the model.

4.2.4 Error Recovery in a SVM

If during the service of a user a service routine detects an error which makes the continuation of the service impossible, the service routine should terminate the service and return the user with a description of the error encountered. This should not have any effect on the other user(s) the SVM is concurrently serving, if there is any, or on the subsequent users the SVM will have after the error detection.

If a service routine fails completely during the service of a user, for example, a user program crashes the compiler service routine, the least the SVM must do is to return the user with a failure message, return other users being concurrently served and re-initialise itself to a state that it is ready to serve the next user. These recovery procedures obviously cannot be started by the service routines themselves but the operating system of the SVM should in this case be able to start a recovery routine which does the job just mentioned. A primitive way to achieve this is whenever a SVM receives a new user, it leaves a copy of the user record with the recovery routine. When a user service is finished the user record is removed from the recovery routine register before the user is returned to the CCC. In this way the recovery routine knows what users a SVM has got and what service is being done for them at any moment. If the recovery routine is activated it knows what messages to send. To restore the SVM to a ready state for the future users would be a simple job of restarting the SVM.

4.2.5 A Guidline for the Design of the Model

One way the SVMs may improve the efficiency of the whole system is to minimize the inter-SVM communication traffic subject to constraints on the maximum concurrency among them[20]. In the design of the model, this principle is behind

putting several I/O requests into one communication with an I/O server. Also buffering input and output data in blocks of 4K byte between a SVM and its I/O server is also in this spirit.

The service units of the SVM model is chosen to be at the command level of the present system. Therefore one compilation is one service and the execution of a user program is another service. These services could be divided into smaller service units. For example the compiler service could be divided into several units according to the phases of the operation. Or there may be a pre-compilation service which is to coordinate the setting up of the I/O files for the compilation (this would be an efficient way if the compiler server is a high power machine and the pre-compilation server is a front end minicomputer). Although dividing these services into smaller services may increase the concurrency of the services but it also increase the communication frequencies among the SVMs. On the other hand a compiler server would have no reason to communicate with another compiler server when they are compiling different user programs.

4.3 The Model Components

This section starts with an overview of the model. This is followed by the description of the data structures used in the model. Then the major components of the model are described one by one.

4.3.1 The Overview

The model consists of three major components. Each of them corresponds to one of the three steps in a SVM's work life, as shown in Fig 4.2. These three components are:

(1) the model initialisation routines

These routines initialise the model's global data, queues, communication channels(IPCF routes) so that it is in the right state to start functioning in the FVMS.

(2) the communication interface

The interface is responsible for transferring jobs(requests for services) between service routines with or without crossing the boundaries between SVMs.

(3) the service routines

The service routines simulate various services. All the service routines are included in the model as all SVMs are running the same copy of the code. Only those routines which simulate the service(s) of a particular SVM are called upon.

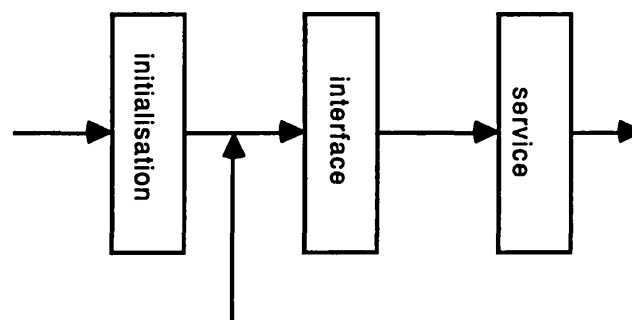


Fig 4.2 The three major steps in the life of a SVM.

Before the one by one discussion on the design of these components, the data structures used by the model to manage its data and the reasons for choosing them are presented in the next section.

The model was implemented in Pascal language. Therefore in the following descriptions the data structures and the algorithms are sometimes introduced in pseudo Pascal language.

4.3.2. The Data Structures

Various data structures used in the model to represent a user in the model, or to store information during its operation, or to represent an interaction with other components of the FVMS are described here.

4.3.2.1 The File Status Record (FSR)

This is a record which records the information about a user's I/O structures, that is, the files a user is using for his input and output. A file may be connected (assigned) to any I/O devices such as the terminal screen or keyboard, a disk file, a printer or a card reader and so on. A file after being connected may be opened. It may be open for reading or writing. After the input/output has finished with a file, it may be closed and then detached from the device. All the status information about each and every file a user is using is stored in the file status record(FSR). The model has the limitation on the users that at any time a maximum of six files may be used simultaneously by one user. This is because the FSR is actually an array of six records in Pascal terms. It includes the two files connected to the terminal screen and the keyboard respectively. It was thought that this limitation would only exclude for experiments the extreme cases where a user needed more than six I/O channels to do I/O during a period of time.

4.3.2.2 The I/O Request Record(IORR)

The following is the data structure used for holding all the I/O request(s) from one SVM to an I/O SVM at any point of the execution of a program:

```

IORequestBlock = RECORD
    Requests : SET OF IORequests;
    CASE GroupOfRequest : IOGroup OF
        Group1 : (AssignReq : IOAssignReq;
                  OpenReq : IOOpenReq;
                  PutReq : IOPutReq);
        Group2 : (CloseReq : IOCloseReq;
                  DetachReq : IODetachReq;
        Group3 : (WriteReq : IOPutReq;
                  GetReq : IOGetReq);
        Group4 : (OutPutReq : IOPutReq)
    END;

IORequests = ( Assign, Detach, Open, Close, Put, Get );

```

Each 'AssignReq' is a record able to hold up to six assign(connect to an external device or file) requests. Each 'OpenReq' is also a record in which a maximum of six open file requests can be stored. Similarly the 'DetachReq' and the 'CloseReq' can have up to six disconnect or close requests in them respectively. 'PutReq' (also 'WriteReq' and 'OutputReq') is a request to output a line of text to a file. This line of text could be considered as the content of the output buffer between the program in execution and the file concerned. 'GetReq' specifies a request to read a line of text from a file. The line of text simulates the contents of the input buffer between the program and the file. Six is the maximum number of files a user may use simultaneously at any time.

'GroupOfRequest' indicates one of the frequent sequence of requests made by programs. Group1 may be used in the initialisation process of a program execution, where association of files defined in a program with external devices and the opening of them for reading/writing, followed by a message indicating the execution (of a compilation or something else) has started, are often done in sequence. Group2 is for closing files and then detaching them from peripherals or disk files. Group3 covers the situations where a message is output and a response is wanted. Group4 is used to output a message only. There are other situations where a program's need is different

from the above four sequences of I/O requests. The exact type of request(s) that a program would make is specified in 'Requests', in which one or more types of requests can be given. Consider the following example. When a program wants to close a file opened earlier for writing then open it later for reading on the same device(a disk file), it may specify in one I/O request: 'Close' for 'Requests', 'Group2' for 'GroupOfRequest' and give the file name in 'CloseReq'. After an I/O SVM has done the job it may choose 'Open' for 'Requests', 'Group1' for 'GroupOfRequest' and give the appropriate values in 'OpenReq'.

The structure of the I/O request record was designed with two aims in mind. First the efficiency of the I/O operations and second the flexibility. It was recognized that usually a program, whether it is a system application program like a compiler or a user program, will have one file for input and another for output. Quite often, some other file is also used for either input or output. In fact, in some programming languages, there is one standard input file and one standard output file. At the beginning of the execution of these programs the run time support system of the programming language will connect these files to the appropriate devices or disk files, open them for reading or writing according to the requirements. Therefore two requests to connect(assign) them followed by two open requests are done in sequence during the initialisation of almost every program execution. If these requests are issued by a SVM to an I/O SVM separately, four communication transactions have to be done between the two SVMs. It would save a lot of communication costs if a number of requests of different types(like connect and open) and concerning different files(both the input and output files) can be put into one data structure when the sequence of the I/O operations are known. In situations where only one I/O operation is required, only one request concerning one file needs to be put in the data structure.

4.3.2.3 The User Control Block(UCB)

This is the data structure used to represent a user within the system. Apart from the user identification, it carries mainly the following information:

- (1) the type of the service required. When a user is sent to the CCC by a SVM, the CCC decides the type of SVMs the user can go to from the service requested. When a SVM capable of offering multiple services receives a user, it also has to find the exact service demanded.
- (2) the name of the subroutine required within a service. Usually there are a number of subroutines that will make up one service, just like an editor would include various subroutines to carry out its subfunctions. This acts like the address of the next instruction that the user will be executing in a conventional computer system. When a SVM is processing a user, that is, simulating the service of the user, this information indicates what stage the service is in.
- (3) the user's file status record(FSR). Here is stored the status of the I/O files being used.
- (4) the I/O request record(IORR). This carries the I/O requests a SVM would like to make on behalf of the user. An I/O SVM receives a user and identifies the operations to be done from here. It then updates the user's FSR.
- (5) the identification of the SVM which is serving a user(for a service other than I/O service) and that of the SVM which is the I/O server for the main service. The reasons for having these SVMs' identifications are explained later.

This UCB is different from the process control block(PCB) in the conventional computer system or the virtual machine control block(VMBLOK) in the VM system. The main difference is that in the FVMS the UCB is the information carrier of the user across the SVMs and the CCC while the PCB and the VMBLOK represent a scheduling unit within the real machine. For individual SVMs there is a data structure

representing a job unit while a user is being served. It is called a Job Record.

4.3.2.4 The Job Record(JR)

This is a record type structure storing the amounts of various resources to be consumed in the simulation of a job like a compilation or a user program execution. The resources concerned, as described earlier in this chapter, are the CPU time, the number of lines of input and output on various I/O devices and the working set size. The record also holds a number of pointers to the data of the simulated service. Among the data there is a data structure for buffering the large amount of data transfers between a SVM and its I/O server. It is a Data Block.

4.3.2.5 The Data Block(DB)

For a SVM without I/O capability, it has to make request to an I/O SVM for input and output and data has to be transferred between them. To do this transaction on a line by line basis would be extremely expensive as far as communication is concerned if the input output media are disk files as it is possible to buffer the I/O operation in larger quantities.

The physical transfer of a message from one virtual machine to another is done in units of page frames. Each time one page in the sender's virtual memory and one in the receiver's virtual memory will be locked in the real memory. Obviously communication between virtual machines in blocks of one physical page would be more efficient than consecutive communications in smaller units. Not much will be gained by using larger units, specially when the demands on memory space are high and pages not being referenced may be paged out, which delays a message transfer.

For the benefit of the I/O SVM the data block has a header which records the

name of the file and the name of the user.

4.3.2.6 The Data Block Request(DBR)

Corresponding to the DB message there is the message to request it. It is used when a SVM asks an I/O SVM to do some input from a certain disk file for it. It is not used for input from an asynchronous device like the user's keyboard or from any other devices, in which cases the I/O Request Record(IORR) described earlier is used.

This DBR only contains (1) the identification of the SVM making the request so that the data block input may be sent back by the I/O SVM directly without going through the CCC; (2) the name of the user; (3) the name of the input file and (4) the number of lines actually required. (4) is necessary because the SVM which is simulating the service and asking for input knows how much input needs to be done, that is, the size of the input is not decided by the size of the dummy file being read by the I/O SVM but is determined by the resource consumption specification of the service. Therefore each time a full data block will be requested but the last data block may be of any length and any data block may be the last.

4.3.2.7 The User-Machine Message(UMM)

This data structure is used by the messages between the SVMs and the CCC. It carries the User Control Block(UCB), the specification of the capabilities of the SVM and an indication whether the SVM is multiplexed or not. The combination of the UCB and the information about the SVM is due to the recognition of the following fact.

After a SVM finishes serving a user, it needs to pass the user(UCB) back to the CCC. Therefore the first design of the message structure was the UCB. A more careful look at the way the SVMs work in the FVMS reveals that a SVM will follow the message

sending the UCB to the CCC with another message telling the CCC what capabilities this SVM has so that the CCC can send the next user requiring its service. In order to reduce the number of communications that take place between the SVM and the CCC the structure containing both type of information was designed so that only one message is necessary under these circumstances. This has the advantages of increasing the system's efficiency for the reasons:

- (1) both the SVMs and the CCC would spend less time actually sending and receiving messages;
- (2) the communication routes in the system would be less congested, specially at the CCC end therefore reducing the chances that the CCC would become the bottleneck of the FVMS.

Such a message is also used for a FVMS reconfiguration when the CCC decides to change the service(s) a SVM offers from less demanded ones to more demanded ones. The CCC can simply send a user(UCB) to the SVM who requests a service not currently done by the machine and include in the same message the new capabilities the CCC designates to the SVM. The SVM would then load the new service routine(s) and start serving the user received.

Obviously, there are times that not all the information which can be carried in the message are required by the receiver in the communications between the CCC and a SVM or between the SVMs themselves. For example, when the CCC passes a user to a SVM(not during a FVMS reconfiguration), only the UCB part of the message carries meaningful information. However the rest of the message is only eight bytes long and thus little overhead would be incurred by including these in the message. The only penalty is when a SVM has to send its capabilities alone to the CCC as some situations requires. However these situations only arise in an I/O SVM waiting for the completion of an asynchronous I/O operation. During this time, the SVM would request

for the next user to serve without sending the current user back at the same time. Given that Pascal is a language with strict type checking, it is not worth having three different types of messages for three different situations as each type of messages needs a corresponding set of routines to send and receive them.

4.3.2.8 Lists

There are four types of linked lists used by the SVM models.

(1) File Status Record List

This is a list of file status records(FSRs). It is a list maintained by any SVM with the I/O capability. It records all the file information of the users it is providing the I/O services to. More details of the use of this list will be given when the operations of the I/O SVM is explained.

(2) Input Request Record List

This is a FIFO list containing the data block requests(DBRs). It is also a list maintained by any SVM with the I/O capability. It is the list of all the input requests made to the SVM for it to process.

(3) The Data Block List

This FIFO list contains blocks of data. Each member of the list is of the DB type described earlier. Such a list for an I/O SVM is the list of data blocks to be output, sent by the SVMs to which it is providing I/O service. For a SVM without the I/O capability, it is a list of data blocks sent by the I/O SVMs as input for the program(s) it is running. For a multiplexed SVM without the I/O capability which may be running several programs concurrently(serving several users concurrently), the data blocks in the input list may be for different programs. For a SVM which is providing the I/O service for itself, the input and output data blocks are passed directly between the 'client' routine

and the 'server' routine without being queued in and dequeued from the list.

(4) The User Control Block List

This is a FIFO list of User Control Blocks(UCBs). It is a queue of the users under the control of the SVM. Obviously for a non-multiplexed SVM this list has at most one member. For a multiplexed SVM there may be more than one member.

4.3.2.9 The User Command List

This is an array which contains a list of CMS commands. It simulates a sequence of commands a user issues during his terminal session. A SVM with I/O capability will have this array read in at the model initialisation time. The way it is used by the SVM will be explained later.

4.3.3 The Initialisation of the SVM Model

The initialisation of a SVM has three tasks to do

- (1) identify the SVM's capabilities. That is, the types of the services a SVM is designated to provide to the rest of the FVMS.
- (2) initialise the global data of the SVM.
- (3) according to the SVM's capabilities, set up the communication routes with other SVMs and with the CCC.

4.3.3.1 Identify the SVM' Capabilities

A SVM's capabilities are represented by a variable of the SET type in the Pascal language. A SET variable is considered suitable as it allows the user of the model the

freedom to choose any combination of the services so a SVM may provide any subset of the services(including the full set) on the system. If a SVM offers a particular service, that service will be included in this set.

To identify a SVM's capabilities, each type of the services is represented by a number. Then it is a simple matter of reading in a line of numbers from the virtual machine's console or a system configuration file to set up the value of the SET variable accordingly.

As may be seen earlier, the capabilities of the SVM will change if the SVM is instructed by the CCC to do so.

4.3.3.2 Initialise the Global Data of the SVM

This is to set the various queues(lists) of the SVM to empty, and to connect and open the input, output files used to simulate the input and output of the SVM. In addition, if the SVM has I/O capability, the user command list is read in.

4.3.3.3 Set up the Communication Routes

The number and kind of communication routes a SVM requires depend on its capabilities. If it has full capabilities, that is, it simulates the operation of a CMS virtual machine, no communication routes are needed. Otherwise, every SVM at least has a route connected to the CCC to pass the user it has finished serving back and to obtain the next user to serve. This route is referred to as NEXTUSER. The connection between the various SVMs and the CCC would be like that illustrated in Fig 4.3.

Consider the case when a user wants to do a compilation, the command interpreter, assuming it does not have compilation capability, would pass the user to the CCC along NEXTUSER. The CCC would then search for a compiler SVM and pass the

user to the first one available, also through NEXTUSER. If the compiler SVM cannot do I/O, one of the first actions taken after getting the user is to make a request to an I/O SVM to connect and open the input, output files for the compilation. The request(s), contained in the UCB, will go through the CCC and arrive at the first idle I/O SVM. After this I/O SVM has assigned and opened the appropriate files for this user and updated his FSR, it may pass the UCB back to the compiler SVM via the CCC again. This means the return journey of the UCB would involve two Sends and two Receives (the message will be sent by the I/O SVM and received by the CCC then sent by the CCC and received by the compiler SVM). This is not really necessary as the identification of the compiler SVM can be known to the I/O SVM. If there is a direct communication route between the compiler SVM and the I/O SVM, the UCB may return on it involving only one Send and one Receive operation, that is, one message transfer. This route is called USERLINE.

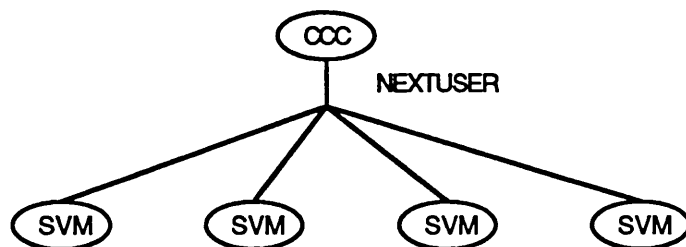


Fig 4.3 The communication route between the CCC and each SVM.

Moreover if the identification of the I/O SVM is passed to the compiler SVM in the returning UCB, the compiler SVM may make the subsequent I/O requests direct to this I/O SVM without going through the CCC for the whole duration of the compilation. An alternative is to pass every I/O request to the CCC which relays it to a suitable I/O SVM. The I/O SVM chosen by the CCC on each I/O request is able to serve the request immediately after receiving it while the I/O SVMs with requests directly coming from its clients may be busy serving another request and the load of I/O requests may

become unbalanced among the I/O SVMs. However the extra message transfer(two message transfers instead of one) and the CCC's message processing also take time. As a result the direct communication approach is no worse off than going through the CCC so long as the load on one I/O SVM is not so heavy that a lot of requests reach it and have to wait and queue for service. It has to be pointed out that the situation of unbalanced load(one I/O SVM is responsible for the I/O service of two programs in other SVMs while another I/O has no responsibility at all) may occur from time to time. The extreme case of requests queueing up in one I/O SVM and some other I/O SVM being idle is possible, but has very small chance of happening. And direct communication with the I/O SVM during the service of one job greatly reduces the load on the CCC and the communication overhead for I/O operations. Therefore two more communication routes are set up for this purpose. A one-directional route called ASKFORDA is to pass the data block requests(DBRs) to an I/O SVM and a two-directional route called DATALINE is to pass both the input and output data blocks(DBs) between an I/O SVM and its client. As a loader SVM which executes user programs need the I/O services the same way as a compiler SVM, routes just described are also set up between the loader SVMs and the I/O SVMs.

This interconnection of the CCC and the SVMs and between the SVMs is somewhat similar to the Cyber series system. In the Cyber series [36], [37] there are one or two central processors(CPs) and a pool of the peripheral processors(PPs). Communications between a CP and a PP is done by messages placed in the input or output buffer of the PP concerned. A dedicated PP (PP0), responsible for coordinating the activities of all the other PPs, periodically polls all the output buffers to see if any PPs have some communication to a process on one CP. Direct communication between the PPs is done along an I/O channel connecting the two PPs without the management of the PP0.

In summary, all SVMs have the route NEXTUSER connected to the CCC. If a SVM

provides the I/O service, or compiler/loader service, but not both services, three more routes, namely USERLINE, ASKFORDA and DATALINE are connected between the I/O SVMs and the compilers/loaders for direct communication. USERLINE carries UCBs, ASKFORDA transmits DBRs and DATALINE is for DBs.

As an example, the interconnection among the SVMs and between the SVMs and the CCC in a system with two compilers(C), two loaders(L), two I/O SVMs(IO) and a command interpreter with I/O capability(CI/IO) would be like the one in Fig 4.4.

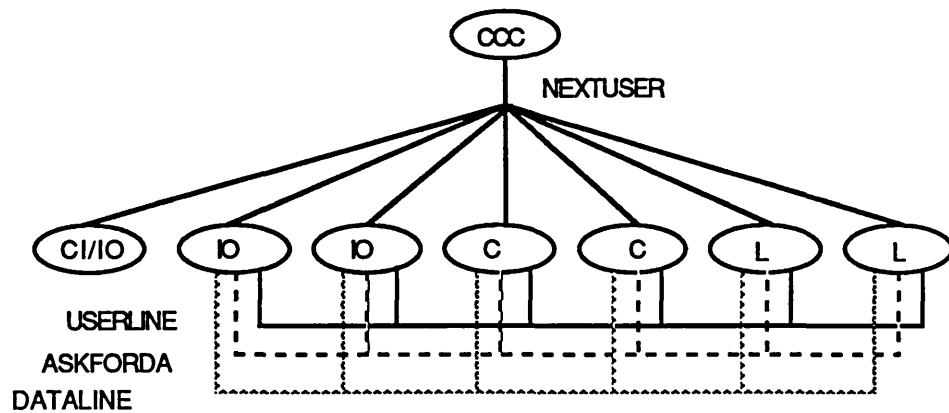


Fig 4.4 The communication routes in an example FVMS.

The IPCF OPEN requests (System-wide, Route-wide and User-wide) are used to set up these communication routes in the FVMS.

During a system reconfiguration in which the capabilities of the SVM changes, some communication routes may be closed using the IPCF CLOSE request and some new routes may be open. Sometimes there is no need to change the interconnection even the services offered changed.

4.3.4 The Communication Interface

This section discusses the design of the communication interface of the SVM model. It is the part of the model which lies between the communication subsystem of

the FVMS and the service routines that carry out the work of the SVM. The discussion starts with the statement of the functions of the communication interface. It is followed by the study of the problems of deadlock in the communications among the CCC and the SVMs. Then the data managed by the interface is presented. The section ends with the design of the main routines in the interface.

In this section the communication interface will simply be referred to as the interface.

4.3.4.1 The Functions of the Interface

The functions of the interface of a SVM can be compared to the Post Room of an institution. There are internal mails: information transferred between departments. The interface of the SVM is responsible for delivering information between services within the same SVM. There are also external mails to be handled: information going to an external address or coming in from an external address. These are the information to be sent to a service not in the same SVM or to the CCC, and information received from another SVM or from the CCC. The interface is responsible for the 'packaging' and sending of all the outgoing messages and receiving of all the incoming messages.

The types of information handled by the interface, internal or external, are

- (1) the users (UCBs)
- (2) the data block requests (DBRs)
- (3) the data blocks (DBs).

It can be seen from Fig 4.4 that UCBs may travel between the CCC and a SVM along the IPCF route NEXTUSER and between two SVMs along USERLINE. To be more precise the messages on NEXTUSER are of the User-Machine Message(UMM) type. DBRs and DBs would go between two SVMs along ASKFORDA and DATALINE respectively.

4.3.4.2 The Possibilities of Deadlocks and How to Avoid Them

In the last chapter it has been stated that due to the restrictions of the IPCF there may only be one outstanding message at most between any pair of virtual machines on a particular route in a particular direction. This would not produce any problem for single user servers(SVMs) as they would receive one user to serve, complete the service and then return the user and ask for the next one to come. Therefore there is no need to send a message before the previous one has been received.

However for multiplexed SVMs the following situation may arise. A multiplexed SVM has sent a message to the CCC which has not received it. The CCC also has an outstanding message waiting for the SVM to receive it. If at this point both the SVM and the CCC try to send to each other another message both of them will be blocked because the route is busy in both directions. This situation could be depicted by Fig 4.5(a). The long arrow line represents an outstanding message and the short one the message the SVM/CCC is trying to send. When this situation occurs no message from any other SVMs would be received by the CCC and such SVMs could be blocked as a result. The mutual blocking like the one in Fig 4.5(a) could be extended to involve more SVMs. Fig 4.5(b) is an illustration of the situation where SVM2 is the I/O server for SVM1 while offering its I/O service to other SVMs at the same time. In this figure SVM2 is trying to send data blocks consecutively to SVM1. SVM1 is trying to send User-Machine Messages to the CCC. The CCC is trying to pass I/O requests in UCBs to SVM2. Again more SVMs may be blocked because the CCC is blocked.

The blocked entity(SVM or CCC) could resolve the problem by storing away the message, doing something else including receiving some message before trying to send again. The situation is analagous to that encountered in an Ethernet communication system[38], [44]. When a message is not transmitted successfully through an Ethernet network because the communication route is busy, that is, another message is

using the communication route at the same time, the message will be sent again after a random delay. However, a SVM or the CCC 'doing something else' may produce more messages to be sent through the same busy route and it is not clear how often 'trying again' is suitable without delaying the flow of messages in the FVMS.

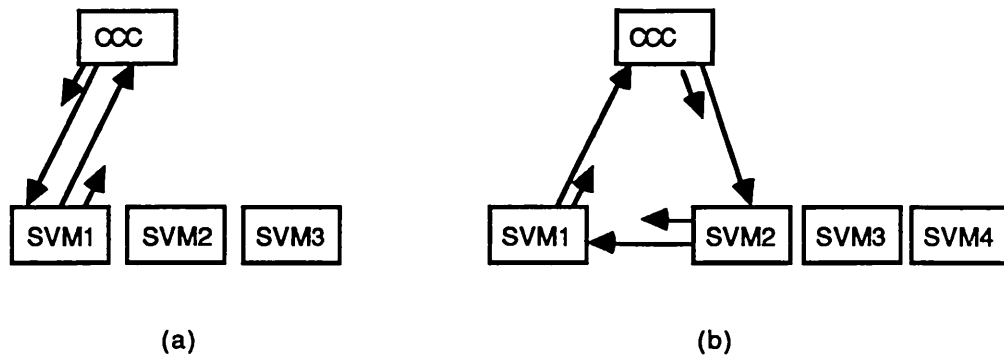


Fig 4.5 Possible deadlock or mutual blocking in the FVMS.

Another solution is to have an interrupt handling routine which is entered whenever a SVM receives a message pending interrupt. This routine could be responsible for receiving the incoming message so the route is cleared for the next message. This solution is ruled out as the interrupt handling routine cannot coexist with the IPCF Await request which waits for the arrival of the next message by putting the virtual machine into idle state. As explained in Chapter Three the virtual machine would be in a permanent waiting state if the message arrival interrupt comes after the SVM issued the Await request.

To make a simple solution a routine is designed to receive all the outstanding messages every time before a SVM tries to transmit a message. This would enable messages to go through as quickly as possible without using an interrupt handling routine. This solution does not remove the possibility of deadlocks completely but was considered acceptable because of its simplicity and because the chances of deadlock happening are very remote.

As messages are received possibly before the SVM would process them, they

have to be stored by the SVM. The way these messages are stored is described below.

4.3.4.3 The Job Lists of the SVM

(1) the User Queue

This is a list of UCBs. It contains the users sent to the SVM to be processed. For a non-multiplexed SVM, this list would have not more than one member at any time obviously. For a multiplexed SVM, the users in this list are served in a first come first serve order. No priority policy is implemented in the SVM model. In both multiplexed and non-multiplexed SVMs a user(UCB) would stay in this list after being received by the interface and before being picked up for service. This is the only time the UCB is in this list. Every SVM has this queue of users.

(2) Data Block Queue

This is of the Data Block List type described in the Data Structure section(4.3.2) of the chapter. Its usage has been described there. So it is only necessary to point out that this is a queue of data blocks waiting to be processed by the SVM. An I/O SVM and any other SVMs which receive their data blocks input by another SVM would have this queue of data blocks.

(3) Data Block Request Queue

This is a queue also described in the Data Structure section(4.3.2). It is a list of data block requests waiting for service from the I/O SVM. Only I/O SVMs have this queue.

(4) Event Queue

This is a list of UCBs as well. It represents the queue of users waiting for the completion of an input operation from an asynchronous device. The users in this queue are not put into it by the interface as a result of receiving them from

outside of the SVM. However after the asynchronous input operation finishes a user in this list has to be dealt with by the SVM so it is also one of the job lists of the SVM. Only I/O SVMs would have this job list. More description of this list will be given later when the I/O service of the SVM is discussed.

4.3.4.4 The Implementation of the Interface

The interface consists of a set of routines to carry out its functions stated in Section 4.3.4.1. Conceptually there are two groups. One group of routines are responsible for the basic sending and receiving of individual messages. The second group has two routines concerning the selection of the next user to serve. One routine uses the first group routines to receive messages and puts them into the appropriate job lists of the SVM. The other examines the job lists and extracts a job(user) from them.

The basic sending, receiving routines

There is one routine to send a DBR along the route ASKFORDA and another one to send a DB along DATALINE. To send a UMM there are different circumstances handled by different routines:

- (1) To send a UMM carrying full information in the sense it contains the UCB as well as the SVM's capabilities when the current service of a user by the SVM has finished. The message goes to the CCC along route NEXTUSER.
- (2) A UMM without the SVM's capabilities is sent. This happens in three situations. Firstly, when during the service of a user a different service is required. The SVM is asking for the new service but would not want to include the SVM's capabilities in the UMM as it would continue serving the present user after the new service is finished. So it does not want the CCC to send the next user yet. Secondly, after the

service of a user, the SVM only needs to pass the user back to the CCC as the request for another user has been sent earlier(see below). Thirdly a UMM without the SVM's capabilities is also used along the route USERLINE between one SVM and its I/O server when they are communicating directly with each other.

(3) A UMM with only the SVM's capabilities is sent. when an I/O SVM has initiated a terminal input operation, the UCB would be put into the Event Queue to wait for the arrival of the input. The SVM becomes available to serve the next user so a UMM with only the SVM's capability is sent to the CCC.

When the service required by the user is available in the same SVM, the UCB would simply be put into the User Queue and no message is sent. As described in the previous section the User Queue is the list of users waiting to be served by the SVM.

There is one routine to receive each type of the messages(UMM, DBR and DB). In particular, if the routine receiving a UMM finds that the machine capability part of the message is not empty, it calls the reconfiguration routine to change the service(s) this SVM is offering before putting the new user into the appropriate job list. This means changing the service routine(s) loaded in the SVM's memory in addition to a few administration changes to attend to.

The selection of the next job

As described earlier, the sending procedures may put the UCB seeking for a different service into one of the job lists if the service is available in the same SVM. Another source of jobs for a SVM is from outside of the SVM. One routine checks for any pending messages on all the routes of the SVM and receive them by calling the appropriate receiving routines. The information received(users, data block requests or data blocks) are put into the correct job lists.

This routine is called in every sending procedure before an outgoing message is sent and in the user selection routine before the SVM tries to select the next job to do.

It has two modes in checking for pending messages to receive: with waiting or without waiting. Waiting mode means if no messages have arrived on any routes, the SVM has to wait for a message before it has a job to do. This happens when the routine is called from the user selection procedure and all its job lists are empty. No waiting mode applies that when the routine is called from a sending procedure or from the user selection procedure at least one job is in one of its job lists.

The user selection routine is called whenever the SVM wants to find the next user to serve. It simply examines the job lists of the SVM one by one and dequeues the first job it finds from its list.

4.3.5 The Simulation of the Services

This is the part of the SVM which simulates the various software services provided by the CMS to the user of the virtual machine. The model has provided the following services:

- (1) the I/O service;
- (2) the command interpreter service;
- (3) the compiler service;
- (4) the user program execution service referred to as the loader service;

The service routines of the model could be divided into four parts corresponding to the four types of services. In addition, there is a part of the code to generate the I/O requests needed by services (2), (3) and (4). All these parts are described in this section.

4.3.5.1 The I/O Services

The I/O operations required by the other services of the FVMS are:

- (1) attach an internal file referred to in other service routines to an external device or disk file(Assign)
- (2) open a file(Open)
- (3) read a line from a file(Get)
- (4) write a line to a file(Put)
- (5) close a file(Close)
- (6) detach an internal file from the external device or disk file(Detach)
- (7) read a block of lines from a file
- (8) write a block of lines to a file.

The first six I/O operations are requested from other service routines in an I/O Request Record(IORR) in the UCB. Operation (7) is requested in a Data Block Request(DBR) and (8) in a Data Block(DB).

The I/O services are provided by two layers of routines. The lower layer carries out the actual simulation of the I/O operations while the upper layer processes the I/O requests mentioned above.

I/O service lower layer

The routines which simulate the I/O operations requested in an IORR(operation

(1) -- (6)) follow a common pattern:

- (1) check the File Status Record(FSR) of the user for any inconsistency or error this I/O operation would cause, for example, to open a file which has not been attached to any device or disk file, or to write to a file which was open for reading.
- (2) do the operation. The Assign/Detach operation is simulated by the CONNECT statement in Pascal. This CONNECT statement is a local extension to the standard Pascal. It connects a file to a device during the execution of a program. The Open/Close operation is simulated by the RESET/REWRITE statement of Pascal and

the Get/Put operation by the READ/WRITE.

(3) updates the user's FSR when necessary.

The devices on which these I/O operations are done in a CMS environment are very likely to be the user's terminal or the virtual machine's disk. Additionally the compiler may have an in STORE file which is a file kept in the virtual machine's memory. The DISK, STORE and TERMINAL are the three devices used in the simulation of I/O operations by the model.

When an operation concerns a disk file, one of the three files are used. A file READSCRATCH in the model is connected to a disk file with some dummy data in it and is opened for reading during the initialisation of the SVM model. Similarly a file WRITESCRATCH is connected also to a disk file and is opened for writing. All the disk reading operations are done on READSCRATCH and all disk writing operations on WRITESCRATCH. A third file called SCRATCHFILE is used for CONNECT, RESET and REWRITE operations.

When a STORE file operation is needed the SVM model performs the operation on a file which is connected to a file in STORE.

An output operation to the user's terminal is simulated by a write to the console of the SVM. However the simulation of the input of a line from the user's keyboard is not done by a real input from the SVM's console. Unlike writing to the SVM's terminal screen, reading from the terminal means someone has to respond in a reasonable time with an appropriate line typed in. Because of the difficulties in assuring someone's availability on the terminal and in controlling the time taken to type in the line, the method described below is used.

The input from the keyboard can be classified into two categories: (1) a user command requested by a command interpreter; (2) other things like the login command with the user identification or his password. The first category of input is predetermined before each experiment with the SVMs and read into an array called the

User Command List(UCL) when the model is initialised. The UCL was described in the Data Structures for the SVM model. When a user's command is requested a line is extracted from the UCL; no real input from the keyboard is done. Similarly the second category of input is prepared in another array and a line is taken from it each time an input is required.

The user thinking time before each terminal input is entered is simulated by setting up an I/O event of 10 seconds. After obtaining an input line of either category, the time the input is assumed to have arrived (10 seconds from the current time) is recorded in the UCB. The UCB is put aside into a list of UCBs called Event Queue which is a list of users in the process of doing terminal input. This actually simulates the waiting for the completion of the asynchronous terminal input. The SVM would check for the end of any input event by searching through the Event Queue and examining the end-of-event time recorded in each UCB and take the UCB out of Event Queue when the time is due. Meanwhile the SVM would get another user to serve through its communication interface. The checking for any end of input event by the SVM is like the software polling done by the peripheral processors on the Cyber series machines to detect end of I/O operations[36].

A constant of 10 seconds was chosen as the user thinking and typing time. It cannot be accurately simulating the reality. However the system performance is mainly affected by the number of active users, not those taking coffee break or having lunch and therefore these idle users are excluded from the simulation. For active users, usually they have a rough idea of the sequence of the things they want to do during their sessions so thinking would not take a long time. Ten seconds was considered close to reality.

Two routines would simulate the input and output of a data block(operations (7) and (8)) respectively. Depending on the type of device the I/O file concerned is connected to, that is, DISK or STORE, the input/output operation would be done on

READSCRATCH, WRITESCRATCH or the STORE file of the SVM.

I/O service upper layer

This layer of I/O service routines are responsible for processing the three types of I/O requests(IORR, DBR and DB) and one type of input events mentioned above.

(1) To process the Event Queue

Users waiting for the completion of an asynchronous terminal input are queueing in the Event Queue. A routine would look through the Event Queue and perform a polling of the end-of-event time recorded in each UCB. If the time has passed, the UCB would be dequeued and returned to the caller of the I/O service through the communication interface of the SVM.

(2) To process IORR

As explained in the Data Structure section of the SVM model, an IORR may contain more than one type of I/O requests and/or more than one request of the same type. A routine in this layer of the I/O service would examine the IORR and pass each individual request to the corresponding routine in the lower layer to simulate the I/O operation. If the UCB was not put into the Event Queue by the lower layer service routine because it is not an asynchronous input operation the routine would return it to the caller of the I/O service through the communication interface of the SVM.

(3) To process DBR

This service routine processes the request for the input of a block of data from a DISK or STORE file. It first checks for any error that might exist from the FSR in the UCB, for example the file has not been connected or is not open for reading. If no error is detected the routine in the lower layer for reading a data block is called. Finally using the communication interface the data block read is passed to the SVM which issued the input request.

(4) To process DB

This procedure is similar to the processing of the DBR: it checks for errors first and then calls the lower layer routine to output the data block.

4.3.5.2 The Making of the I/O Requests

This is a set of routines used by command interpreters, compilers and other service routines to set up the I/O requests so that the I/O service routines described in the last section may process them. Because the I/O operations in the FVMS may be done in the same SVM or by another SVM, the Pascal I/O statements like READ or WRITE cannot be used directly in the service routines where I/O is needed. Instead I/O requests are built and delivered to the I/O service routines through the communication interface of the SVM.

The I/O requests concerned here are those contained in IORR. DBR type of requests are constructed in the communication interface before the transfer of the request and DBs are filled in by the service routines which want to output the data blocks.

When building the I/O requests in an IORR, each request is added into the current IORR. This is because the IORR of the user(UCB) is intended to contain a number of requests maybe of different types. For example two consecutive Assign requests and two Open requests would be added one by one to the IORR.

4.3.5.3 The Command Interpreter

This service is responsible for the continuity of the operation of the FVMS command environment. In the model the command interpreter

(1) controls the user logon procedure. This includes obtaining the user's login name

and his password, opening the I/O files which are connected to the user's terminal.

(2) reads user commands from the terminal and executes them.

(3) controls the user logoff procedure. This includes the closing of those files connected to the user's terminal.

On obtaining a user command the command interpreter decodes it and records the type of the service required and the name of the first subroutine that starts the service in the UCB. Then the UCB is passed to the communication interface to be sent to the correct service routine in the same SVM or in another SVM. After a command has been executed the command interpreter will get the UCB back through the communication interface. A prompt will be output to the user terminal and the command interpreter would issue the input request for the next command. The prompt message simulates the 'ready' message/error message that the CMS interpreter types after the execution of each command.

All the I/O operations are done through the I/O service routines in the same SVM, or in another SVM if the command interpreter SVM does not have I/O capability. The command interpreter sets up the I/O requests in the I/O Request Record(IORR) of the UCB and the UCB is passed to the I/O service routines by the SVM's communication interface.

4.3.5.4 The Compilations

The real compilers that are running in the CMS virtual machines cannot be used here for three reasons:

- (1) real compilers do not do their I/O in a separate virtual machine or interact with programs running in other virtual machines;
- (2) present compilers do not multiplex users. They handle the compilation of one

program at a time.

- (3) present compilers are not clearly divided into stages which could be executed in different SVMs.

The third point is within the consideration for the following reason. Although in the operation of the FVMS, a user command was chosen as the service unit so a compilation would not be split into smaller service units, a compilation service could be divided into smaller units when the FVMS model is simulating a network of real machines.

Before the design of the simulation was started, a study of a real compiler was done. The compiler chosen was the Pascal compiler on the departmental VM/SP machine which had been the main compiler for the undergraduate students until the main teaching language changed to Modula-2 several years ago.

The P4 Pascal Compiler

This is a two-pass compiler. In the first pass the source program is read in line by line, parsed, and an equivalent program in P4 code assembly language is output to an intermediate file. A file listing the source program with line numbers is also produced in the first pass. This file is called the listing file. Error messages if there is any will be put in this file. In the second pass the intermediate file is read in and translated into IBM machine code program called the object program. The module which does the first pass is referred to subsequently in this section as the compiler and that for the second pass as the P4 assembler.

The source file is a CMS file of variable record(line) length. So are the listing file and the intermediate file. The object file(machine code) is a fixed record length(80 characters) CMS file. Usually the source file, listing file and the object file are all input from or output to the virtual machine's disk. The intermediate file is

output to the virtual store by the first pass of the compiler and input from there by the second pass. However one of the compilation options allow it to be stored on the disk. It is erased after the compilation unless specific request is made to retain it.

There is an interface between the CMS command interpreter and the actual compiler and the P4 assembler. The interface checks the existence of the source file and sets up the parameter list including all the compilation options the user has specified to the compiler. As mentioned in Chapter 2, there are 'shareable' versions of the compiler and the P4 assembler on the departmental machine. Therefore at the point where the interface would load the compiler into the virtual machine's storage, it checks if the version currently on the machine is shared or not. If it is, the interface would just pass control to the entry point of the compiler. If it is not, the interface would load the compiler into the virtual memory and run it. The same procedure applies at the point where the loading of the P4 assembler is needed. After the P4 assembler has finished, an 'EXEC' file is produced by the interface. This 'EXEC' file contains one command calling another EXEC program which specifies the standard sequence of CMS commands to run a Pascal program. The sequence of commands will set up the run time support system, load the object file produced by the compilation and start its execution.

The Simulation of the Compilation

The aim of the simulation is to consume the same resources in the same proportions as would the real compilations. The resources concerned are the CPU time, the working set size and the number of input source lines, the number of lines of the listing file, object file and the intermediate file involved in the real compilation.

The simulation of the first pass of the compilation is shown as an example by the flowchart in Fig 4.6. For the second pass the algorithm is similar to the first pass thus it is not discussed here. In Fig 4.6 the initialisation of the simulation mainly

includes

- (1) the setting up of the amount of various resource consumptions according to the parameters specified for the simulation and the recording of the compilation options such as whether the intermediate file should be output to a STORE file or a DISK file;
- (2) a real request to load the real Pascal compiler is made. This has the following effect: If the real compiler on the VM/SP machine is not shared between the virtual machines, the loader will load the real compiler code into the SVM's store then returns with the address of the compiler's entry point. If the real compiler is shared, the loader will simply returns with the entry point. This is exactly what happens at the beginning of a real compilation.
- (3) a linked list with members each being a 4K byte structure is created. The purpose of this list is explained later.

The working set of a real compilation would consists of some pages from the compiler code and some pages from the data of the compiler. To make the simulation as close as possible to the real compilation, the compiler code part of the working set of the simulation are pages from the real compiler. This is possible as the entry point of the real compiler is known from the initialisation stage of the simulation. When the model accesses the compiler code part of the working set what it does is to reference a number of pages starting from that entry point. The number of pages referenced is specified by one of the parameters of the simulation. The compiler data part of the working set is represented in part by the pages of the working set of the SVM model. Any remaining part is supplemented by the linked list of 4K byte blocks which is created at the initialisation time. The number of the 4K byte blocks in the list is also specified by a simulation parameter. By accessing the pages of the real compiler and each member of the linked list at every step of the simulation, the model is able to keep

these pages in its working set. Note the real compiler code is accessed but not executed; the code of the model is executed.

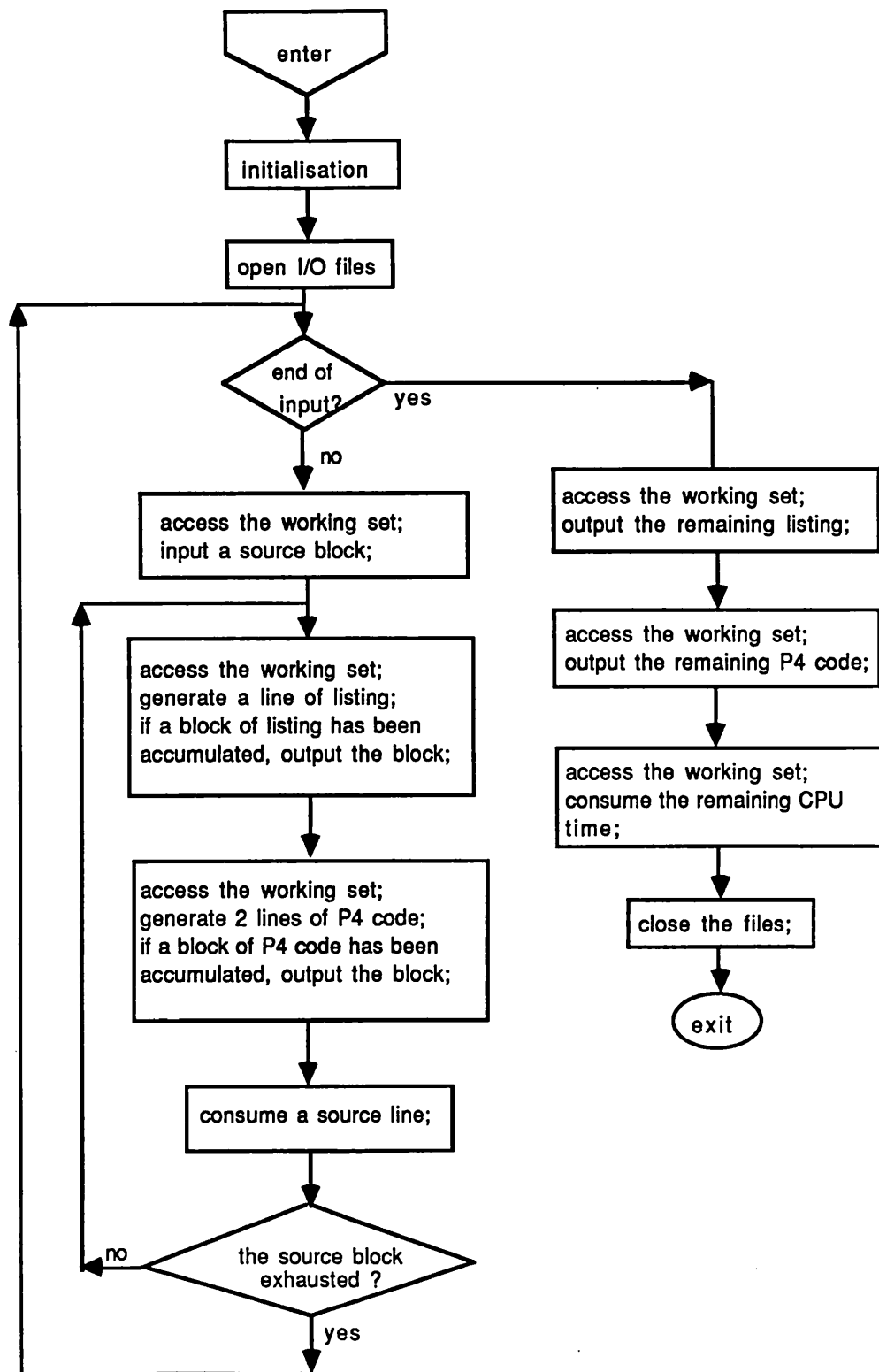


Fig 4.6 The flowchart for the simulation of the first pass of compilation.

The amount of input and output are controlled by the simulation parameters. When the resource requirements of the compilation are set up during the initialisation of the simulation, the number of lines of I/O concerning various files are read into the variables of the simulation. The model loops through its steps as shown in Fig 4.6 until all the I/O have been done. It has already been shown that for the efficiency of the SVMs in FVMS the disk I/O are buffered between a SVM and its I/O server(SVM) in blocks. Therefore each time input is needed, a block with the structure of the Data Block(DB) described earlier is read in by the I/O service routine of the same SVM or by another I/O SVM on the request of the compiler SVM. After this the compiler SVM would just get a line from its DB until the source block is exhausted. The output of the compilation is also accumulated in a DB before being sent to the I/O service routine.

To cater for the case when the model is simulating a non-Pascal compiler where a listing file is not produced, the steps to generate a line of listing would first check whether the number of lines to produce is still positive. If not, it will go to the next step of the simulation straight away. In this way, if the model is simulating a compiler with no listing file, it is only necessary to set the simulation parameter for the number of lines in the listing file to zero. If it is simulating a compiler whose listing file only contains the error messages but not the listing of the source program, the simulation parameter for the number of listing lines can be set to a realistic small number.

The consumption of the CPU time is monitored by the VCPU timer of the virtual machine. It includes the time spent on initialising the simulation. As the I/O operations may be done by the same SVM or by a different SVM and the amount of time taken would not be the same, the actual time involved in these operations are not counted in as the CPU time of the simulation. However the CPU time requirement of the compilation includes the time for the I/O operations. As a result the time taken to

input a block of source line is measured separately by a test program and the same is done for the output. Then each time before a block input/output is done a fixed amount of CPU time is added to the appropriate variable. As the last step of the simulation the amount of CPU time taken so far is compared with the CPU time requirement and any remaining time is consumed by accessing the working set and doing some dummy calculation repeatedly.

An example of a compilation command specifying the resource requirement parameters is given below.

```
COMPILE E+ 18 4 2 660 776 -2272 622 7788 -150 1808 -50
```

- (a) 'COMPILE' is the command given to the command interpreter of the FVMS to call a compiler. 'E+' indicates that the EXEC file is to be generated at the end of the compilation and 'E-' indicates otherwise.
- (b) '18' is the number of pages of the compiler code in the working set, '4' the number of 4K byte blocks in the linked list as part of the compiler data in the working set and '2' is the number of pages of the P4 assembler code in the working set of the second pass.
- (c) '660' is the number of lines the source program has. '776' is the number of lines of the listing file produced. '2272' is the number of lines of P4 code to be generated, '-' means the intermediate file containing the P4 code is to be stored in the virtual machine's store and '+' would mean it is to be on the virtual machine's disk. '622' is the number of lines the object file should have.
- (d) '7788' and '1808' are the CPU time in milliseconds the first and second pass of the compilation need respectively. Last '-150' and '-50' are the calibration parameters for the CPU times of the two passes.

The values of these parameters would be taken from student programs during

the validation stage of the model. The calibration parameters provide a means to adjust the model so that the CPU time consumed by the model is close enough to the values set by the command.

The first pass and the second pass of the compilation are separately constructed in two sets of small procedures. Only very minor modification to the model is required to make the two passes as two different services so that they can be done in different SVMs concurrently, if this is desired.

4.3.5.5 The Loader Service

This is the service where users' programs are loaded into the virtual machine's store and executed. The way CMS offers this service has been described in enough detail in Chapter 2. There are two main tasks to be done: (1) the loading; (2) the execution; and these are the tasks to be simulated for this service.

From the virtual machine point of view, compilers and user programs are both programs to be executed. They both do input and output, both consume CPU time and both need physical memory space to store their code and data. If the compilers are not shared among virtual machines there will be no difference between the execution of a compiler and the execution of a non-interactive user program. The only difference is the amount of machine resources these executions need. With the sharing of a compiler, the difference between the two executions will be the place where the code to be executed is stored. The shared compiler code is stored outside the individual virtual machine's address space and the executions of the compiler program by concurrent virtual machines are accessing the same copy of the code. For executions of user programs the code is inside each virtual machine's storage.

From these considerations a conclusion was drawn that the same piece of code to simulate the compilations in the model(the first pass of the compilation) could be

used to simulate the executions of non-interactive user programs. The code was modified slightly so that the model does different things at two points depending on whether the SVM is a compiler server or a loader server. The first point is at loading time. A compiler server would load the compiler as described in the last section. A loader server would read in a file of machine code program whose execution is being simulated. The second point is when the model is accessing its working set(see Fig 4.6). The compiler server still accesses pages of the real compiler as the code part of its working set. The loader server would access a number of pages starting from the address which is the starting point in the virtual machine memory where a user program would have been loaded. Naturally as a result it is the starting point in the virtual address space where the code for the SVM model is loaded.

Below is an example of the command given to invoke the loader service in the FVMS.

```
EXECUTE PCBLIST 15 20 456 378 3506 -45
```

- (a) 'EXECUTE' is the command to the FVMS' command interpreter to call the loader service.
- (b) 'PCBLIST' is the name of the machine code file which is read at the beginning of the execution simulating the loading process.
- (c) '15' and '20' are the numbers of the pages for the code and data parts of the working set to be accessed.
- (d) '456' and '378' are the number of lines for input and output respectively.
- (e) '3506' is the CPU time required for the execution and '-45' its calibration parameter.

4.4 The Assessment of the Model

This section discusses the model characteristics in the aspects of the storage requirements of the model and the time requirement for simulation experiments. The limitation on the model because of its characteristics is then stated.

4.4.1 The Model Characteristics

The whole SVM model consists of a little more than 180,000 bytes of code. However not all the code is required by any SVM. SVMs offering different services execute different parts of the code. Even a SVM providing full services will not need the part of the code that is for communication with other SVM or with the CCC. The table shown in Fig 4.7 gives the storage requirements of several services as examples.

	Storage requirements
the compiler service including the building of the I/O requests, but not I/O service routines	44K
the loader service including the building of the I/O requests, but not the I/O service routines	34K
the I/O service	44K
the communication interface	25K

Fig 4.7 The storage requirements of some of the service routines of the SVM model.

According to the table the compiler service has 44K bytes of code and the I/O service 44K byte code. A compiler SVM with I/O capability would have about 88K plus

the 25K communication interface code. Therefore such a SVM would contain 113K bytes of code. This is an estimate of the maximum amount of code because

- (1) there is a small difference between the code executed when the I/O service is offered within the same SVM than it is offered in another SVM. The code in the former case is slightly simpler;
- (2) the active loop in the compiler service part is at least 20K smaller than the whole of the service code.

As a result 60K bytes would be a reasonable estimate of the active code executed by such a SVM.

The model execution time is of the same length as the time simulated if the model is valid. At the beginning of the chapter it is stated that the SVM would give the same response time as a CMS virtual machine when providing the full service. This guarantees that the model execution time is the time taken for the same events to happen in the real system. This is better than those models with the simulation time being a multiple of the time being simulated. But it is still a time consuming job to do experiments with the model.

4.4.2 More on the Scope of the Model

One of the consequence of the size of the storage requirements for various services is that it limits the type of the programs suitable for simulation. For example, if the storage requirement for the active code of the compiler service is 60K bytes, no compilation with a working set of less than 60K could be accurately simulated. The same applies to the loader service. This limitation of the model must be remembered when experiments are conducted on the model and conclusions drawn from the results.

4.5 The Validation of the Model

Performance figures from a model which has not been validated are of no use. In this section the process of the validation of the model is described and several examples of the validation results are given.

4.5.1 Validation Method

It has been stated in the beginning section of the chapter that if the model behaves like a CMS virtual machine when all the software services are combined into a single SVM, it would be a valid model of the CMS machine. And because the SVMs in the FVMS are built with the same building blocks as the SVM simulating a CMS machine, the model is valid in predicting the performance of the FVMS.

In measurable terms if the SVM model offering full set of services is consuming the same system resources in the same proportions as a CMS virtual machine it is a valid model. The system resources concerned are the CPU time, the size of the working set and the number of lines of text for input and output on various input, output devices. Therefore to validate the SVM model, the resource consumption figures of various jobs on the CMS system were first collected and these were used to validate the model against.

Programs were compiled and executed first in a CMS virtual machine using the compiler and the loader provided by the CMS. The CPU time consumed and the size of the working set were recorded for each compilation and execution. The averages are taken from several runs of each compilation and execution. The number of lines of input and output on various I/O devices were also recorded for these jobs.

The SVM model was then run simulating each compilation and execution. The CPU time taken to simulate each job is collected. Any significant discrepancy between

the figures obtained from the model and those from the CMS virtual machine system would be an indication that the model was not simulating the CMS system properly and some changes to the model were required. Small differences in CPU time could be tuned by the calibration parameter in the job resource requirement specification for the SVM model.

It was pointed out in the last section that the model was not able to simulate jobs that had working size smaller than 60K bytes. However for other jobs, the difference in the working set sizes between the particular jobs on the CMS system and their simulations by the model can simply be solved by adjusting the appropriate resource specification parameter for the simulations. And this is what was done during the validation periods of the model. The model will be doing the exact number of lines of input and output on various devices as the jobs on CMS system will so no validation is needed in this aspect.

Several examples of the validation results concerning CPU times are presented in the next section.

The figures for the CMS system were collected when that CMS virtual machine was the only user on the physical machine and those for the SVM model were obtained when the model was running independent mode, that is, offering full set of services and was the only virtual machine on the system. It was assumed that the model calibrated and validated at this workload level would produce the same performance as the CMS machines at higher workload levels.

A five percent error was allowed during the validation of the model.

4.5.2 Examples of Validation Results

Several samples of the student programs from a laboratory exercise of the undergraduate course in the department were compiled and executed on a CMS virtual

machine and their simulation by the model was done. The model was first found consuming a lot more CPU time than the jobs were supposed to require. After some modifications to make the model more efficient in terms of CPU time it was able to consume the right amount of CPU time with some tuning using the calibration parameters in the job resource requirement specification. Table 4.1 and Table 4.2 show the CPU times obtained from the CMS system and the SVM model respectively.

Table 4.1 VCPU time in seconds for compilations.

job1	job2	job3	job4	
5.24	5.93	4.96	4.78	
5.25	5.95	5.37	4.81	
5.25	5.94	4.97	4.80	
5.25	5.94	4.97	4.95	
5.25	5.94	4.97	4.80	
5.25	5.94	4.97	4.81	
5.25	5.94	4.97	4.80	
5.249	5.943	5.026	4.821	mean

(a) VCPU times for compilations from the CP/CMS system at command level.

job1	job2	job3	job4	
5.269	5.922	4.984	4.828	
5.268	5.932	4.995	4.824	
5.259	5.941	4.997	4.803	
5.275	5.917	5.005	4.821	
5.267	5.937	5.008	4.821	
5.268	5.930	4.998	4.819	mean

(b) VCPU times for compilations from the SVM model in independent mode at command level.

job1	job2	job3	job4
+0.36%	-0.22%	-0.56%	-0.04%

(c) % differences of the model's results from the CP/CMS system's.

Table 4.2 VCPU time in seconds for executions of user programs.

job1	job2	job3	job4	
0.80	1.14	0.76	1.03	
0.81	1.14	0.75	1.04	
0.81	1.15	0.75	1.03	
0.81	1.15	0.75	1.04	
0.81	1.15	0.76	1.03	
0.81	1.15	0.75	1.03	
0.808	1.147	0.750	1.033	mean

(a) VCPU times for executions from the CP/CMS system at command level.

job1	job2	job3	job4	
0.786	1.143	0.737	1.043	
0.791	1.150	0.752	1.043	
0.795	1.147	0.734	1.046	
0.806	1.133	0.741	1.041	
0.844	1.138	0.741	1.040	
0.804	1.142	0.741	1.043	mean

(b) VCPU times for executions from the SVM model in independent mode at command level.

job1	job2	job3	job4
-0.50%	-0.42%	-1.2%	+0.97%

(c) % differences of the model's results from the CP/CMS system's.

All the CPU times were collected at the command level thus in fact they were the CPU times to process the compilation or execution commands. This means the CPU time used by the command interpreter was also included. It ensured that the command interpreter and the service routine involved in executing the command were consuming the right amount of CPU time when they were working together in the SVM

model. VCPU time means Virtual Central Processor time on the VM/SP system. It is the CPU time used by the virtual machine's CPU.

As can be seen in Table 4.1 and 4.2, the percentage difference of the CPU time from the model when compared with the CMS system was below 1% for compilations and below 2% for the executions. This indicated that when doing these compilations and executions, the model in independent mode was accurately simulating a CMS virtual machine as far as consuming CPU time is concerned.

Chapter Five

Experiments, Results and Analysis

After the model is built and validated, the next question is how to obtain some simulation results from the model. In this chapter the experiments conducted on the model are described and the results analysed to answer the questions posed at the beginning of this research. The first section of the chapter introduces the experiments. The second section comments on the performance of the communication subsystem which is a crucial part of the FVMS. Then the comparison of the performance of the model in FVMS mode and in IVMS mode is presented in the third section and the last section is a study of the FVMS in different configurations.

5.1 The Experiments

This section describes the experiments performed on the model. It starts with the environment in which the experiments were carried out because the environment affects the system performance and therefore the accuracy of the results. After this is the discussion of the choices of the experimental data. The experimental data were the jobs requested by the users simulated in the experiments on the model. This is followed by the experiment plan which describes how the experiments were conducted on the model. The last part of the section is about how data was collected from the experiments.

5.1.1 The Environment of the Experiments

Appendix A describes the configurations of the real machine and the virtual

machines supported by the real machine on which the experiments were done. As can be seen, it is not a particularly powerful machine and its memory size may easily be smaller than some minicomputers of the present time. However this does not upset the main subjects of the study which are the relative performances of different configurations of the FVMS and the comparison of the performance of the model in IVMS mode with that in FVMS mode.

Being a system of virtual machines sharing the resources of one real machine, all virtual machines actively running on the system have impact on the performance of the whole system. Therefore to get meaningful results from the experiments great care was taken in the environment in which the experiments were conducted:

- (1) Apart from the virtual machines which were the components of the model, there was no other users (thus user virtual machines) logged on to the system. And there were no system activities such as VM directory updating or daily backups of disk files, which means the system virtual machines attached to the system were idle. All the virtual machines that were sharing and competing for the resources of the real machine were those involved in the experiments.
- (2) To avoid delays that might distort the results of the experiments the terminals(consoles) of all the virtual machines involved in the experiments were directly connected to the IBM. These terminals did not go through any terminal connecting devices such as Gandalf or any forms of network to reach the IBM machine.

5.1.2 Experimental Data

These are the resource specifications for the jobs used in the experiments on the model. The aim is to study the performance of the FVMS when the users with different resource requirements are using the machine. Therefore jobs demanding

different resources in different proportions formed the experimental data for the model. The types of the resources considered include the CPU time, the working set size and the amount of I/O. Some jobs use one type of resources heavily and other types moderately or lightly. Some jobs put heavy demands on more than one type of resources while do not need some other type of resource very much. And there are jobs which require every type of resource in large quantities. All these are represented in the sample jobs to be experimented on the model and their resource requirements are listed in Table 5.1 and Table 5.2. They are:

- (1) the compilation of a small program.

This is a compilation job with moderate resource requirements. It is referred to as J1 and the resource requirements are listed in Table 5.1. The CPU time, the working set size and the amount of I/O required for the job may be compared to a medium-sized student programming exercise in the computer science department of a university.

Table 5.1 Resource requirements of the two compilations simulated in the experiments.

job type	J 1	J 2
CPU time (first phase)	7788 milliseconds	39069 milliseconds
CPU time (second phase)	1808 milliseconds	9724 milliseconds
working set size	128 Kbytes	136 Kbytes
source file size	660 lines	1980 lines
listing file size	776 lines	2328 lines
Pcode file size	2272 lines	6816 lines
object file size	622 lines	1866 lines

- (2) the compilation of a big program.

The resource requirements of this compilation is similar to that for a large student programming exercise. The amount of I/O is three times of the small program specified in J1 and the CPU time is about five times of J1's. However the increase in the working set size is much less than the increase of the other

resource requirements. This reflects the resource requirements of real compilations: the working set size of the compilation is not proportional to the size of the program to be compiled. In fact the working set size does not vary dramatically whether it is compiling a three hundred lines program or an one thousand lines program. This is because the size of the immediate data structure a compiler is working on at any time does not change very much with the size of the compiler's input. The resource requirements of this compilation are given in Table 5.1 as J2.

(3) The execution of a big program.

This job is characterised by long CPU processing time, large working set size and large quantity of I/O, that is, a lot of every type of system resources. It is labelled as J3 and the resource requirements are listed in Table 5.2.

(4) the execution of a CPU bound program.

Obviously this job requires long CPU processing time but does not do numerous I/Os and does not require a lot of computer memory. The resource requirements are listed under job type J4 in Table 5.2.

(5) the execution of an I/O bound program.

This presents the type of jobs which do large amount of I/O, have a medium-sized working set but do not do a lot of processing. The job is referred to as J5 and the resource requirements are in Table 5.2.

(6) A CPU time and memory bound execution.

The execution of a program needs CPU intensive processing and a large working set but does little I/O. The resource requirements of this job(J6) are in Table 5.2.

(7) the execution of a program which is CPU and I/O bound.

The characteristics of this job are it large amount of CPU time and more than usual I/O operations but the memory requirement during execution is not

particularly huge. The column J7 in Table 5.2 specifies its resource requirements.

Table 5.2 Resource requirements of the user programs used in the experiments.

job type	J3	J4	J5	J6	J7
CPU (milliseconds)	44856	28835	26913	70084	47628
working set (Kbytes)	115	83	91	120	99
input file (lines)	2500	198	4500	100	1250
output file (lines)	3500	420	5500	120	1750

5.1.3 Experiment Plan

This is about how experiments were organised. First is how the experimental data discussed in the last section were used in the eight groups of experiments. Then the changes in the number of users in the model and the changes in the configurations of the model in each group of the experiments are described.

5.1.3.1 The Groups of the Experiments

There were eight groups of experiments carried out on the model. In seven groups of them all the users simulated in the system repeatedly did one of the seven types of jobs listed as J1 -- J7 in Table 5.1 and Table 5.2. These groups of experiments are denoted by the jobs run in it, that is, group J1 group J7. In the eighth group of experiments referred to as group J8, small size compilations (J1) mixed with big size executions (J3). Half the users on the system would do the small compilations (J1) and the other half would do the big executions (J3).

The seven groups of experiments where all users did the same job simulated a situation it was often seen in a student laboratory session in a computer science

department of a university. At any time when there was a student laboratory session the majority of the users on the computer would be students from the same class and they would be doing programs with roughly the same length of code. This means the compilation jobs and execution jobs they put to the computer would require similar system resources in similar amounts. Obviously when users were working on programs with more or less the same size they would be doing editing, compiling and execution with their programs. However these seven groups of experiments created the situations where the users on the system made similar demands on the software and hardware resources and the model showed how it would behave in this kind of situations. It should be noted that each one of these seven groups of experiments demanded the system resources in different proportions.

The eighth group of experiments generated the situation where there were two classes of students on the system competing for resources, one class was doing programs each of which did not put heavy workload on the system and the other class was doing more complicated jobs which were more resource demanding.

5.1.3.2 The Different Number of Users on the System

In each group of the experiments increasing number of users was simulated to see the performance of the model under increasing workloads. The number of users experimented ranged from one to eighteen. From the experiences of student laboratory sessions, it appeared that the machine often began to respond very slowly once the number of active users reached 12 or 14. This suggested that around these numbers of users the machine was near saturation point. Therefore experiments with the number of users up to 18 would provide the information about the system behaviour from light loaded state to saturated state. The users simulated in the experiments were all active users with only ten seconds to respond to the system (this process is simulated). This

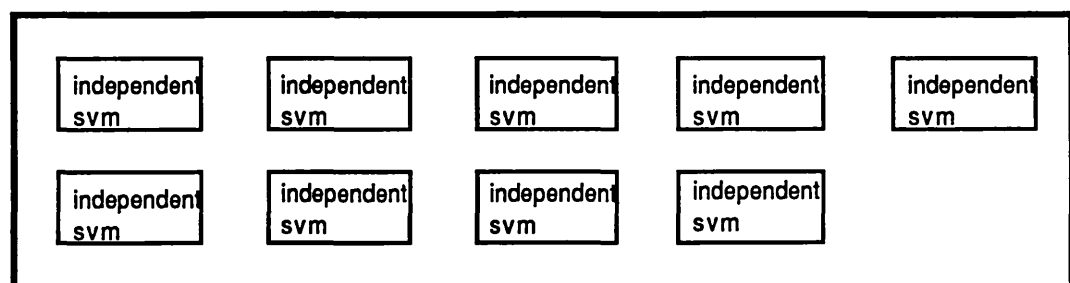
included from seeing the system prompt the time to decide on the next command and the time to type it in.

5.1.3.3 The Different Configurations of the Model

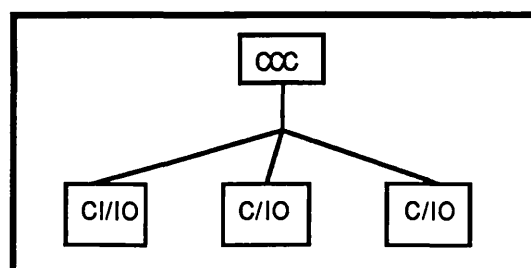
For each group of the experiments, several configurations of the model were experimented to study the response of each configuration to the workload. The changes in configuration can be classified as:

The IVMS mode and the FVMS mode of model

In the IVMS mode of the model each SVM provided the full set of services. There was no need to send a user to another SVM so the CCC was not involved in the operation. Each SVM served a single user, simulating a CMS virtual machine. The number of SVMs was the number of users on the system.



(a) 9 users running in 9 independent SVMs.



(b) 9 users served by 3 functional SVMs.

Fig 5.1 Sample configurations of the model in IVMS mode and in FVMS mode.

In the FVMS mode of the model there was the CCC, responsible for the inter-SVM communications, and a number of SVMs each offering a subset of the services of the system.

Fig 5.1 shows an example of the two configurations where there were nine users on the system. In Fig 5.1(b), CI/IO was a SVM offering the command interpreter and the I/O services. CI/IO was a compiler SVM with I/O capability, or generally, a SVM with I/O capability and responsible for executing a user command.

Multiplexed SVMs and non-multiplexed SVMs

In FVMS mode, the SVMs may be multiplexed or non-multiplexed. Multiplexed SVMs served more than one user at a time. They continued to accept users to serve before the service of the current user was finished and returned to the CCC. Non-multiplexed SVMs only served one user at a time. The next user would not be sent to such a SVM until the present one was returned to the CCC.

For SVMs which offered highly interactive services like command input and interpretation, multiplexed mode of operation was considered most suitable as they could interpret one user's command while another user was thinking of or typing in his next command. Therefore in all the experiments command interpreter SVMs always operated in multiplexed mode. Such SVMs with I/O capabilities were labeled as CI/IO in all the figures in this chapter.

For non-interactive type of services experiments began with multiplexed SVMs only offering one type of service. So there was no need of switching the application code a SVM executed, only the switching of users when several users were being served concurrently. A multiplexed compiler SVM or Loader SVM would do the input/output in another SVM so when during a compilation or execution some I/O operations needed to be done, I/O requests were made to an I/O SVM, the compiler

or loader then switched to the service of another user.

Experiments were done where each multiplexed SVM would concurrently serve not more than three users. When the results were negative when compared with non-multiplexed SVMs, experiments with multiplexed SVMs serving two users concurrently were carried out. Experiments with multiplexed SVMs offering more than one type of non-interactive service were not carried out when single-service multiplexed SVMs did not show favourable results.

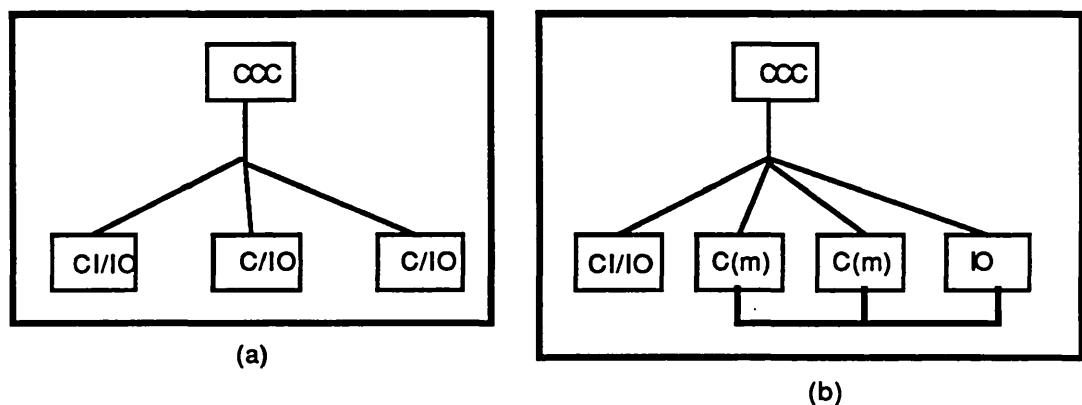


Fig 5.2 Sample configurations of the model in FVMS mode with non-multiplexed SVMs (C/IO) in (a), or with multiplexed SVMs (C(m)) in (b).

Fig 5.2(a) depicts a configuration where two non-multiplexed SVMs with IO capability executed all the user commands on the system. This was compared with the configuration where two multiplexed SVMs without IO capability executed all the user commands while an IO SVM did the I/O operations for them as shown in Fig 5.2(b). When each multiplexed SVMs(C(m)) served up to three users at a time, the configuration was labelled as M3. When up to two users were multiplexed in each of them, M2 was the label of the configuration. When an extra I/O SVM was added to the M2 configuration to see any speed up of the executions M2IO2 was the label of the configuration.

Varying the number of SVMs in the FVMS

When there are less number of SVMs offering some type of service than the number of users requesting it at a certain time these requests(users) have to queue and wait for that service. On the other hand if the number of SVMs offering this type of service is increased, the waiting time for the service will decrease but each service of a user by a SVM will take longer to finish as there are more SVMs competing for the real machine resources and the CP overhead of supporting these SVMs becomes greater(the service time will not lengthen if it is a real machine instead of a virtual machine). To study the behaviour of the FVMS with different number of SVMs, the number of the users on the system and their job requests were kept unchanged and varying number of SVMs were used in the experiments. The aim is to find out whether an optimal configuration for a particular workload exists or not and if it does, what it is.

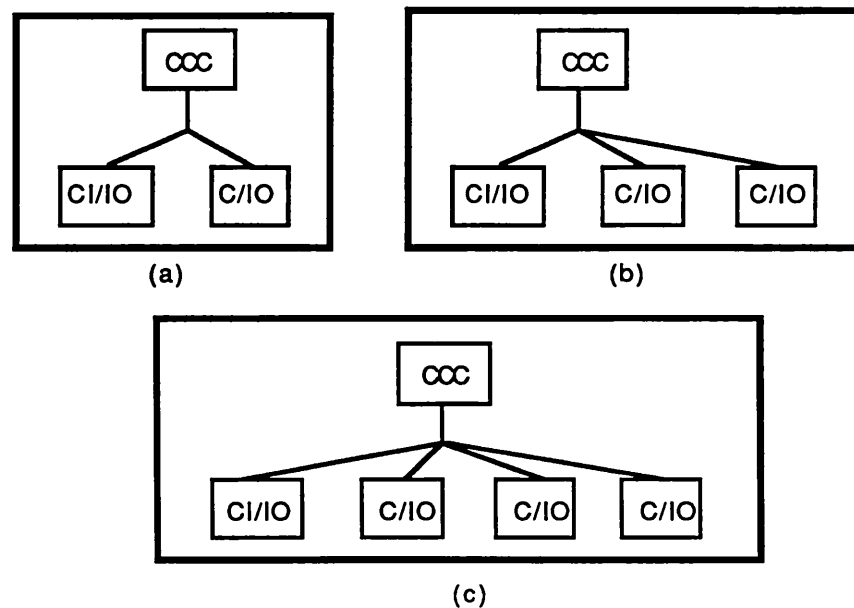


Fig 5.3 Sample configurations of the model in FVMS mode with different number of SVMs. (a) configuration C1; (b) configuration C2 or C1L1; (c) configuration C3 or C2L1 or C1L2.

Fig 5.3 depicts the changes made to the configuration of the FVMS while the

workload of the system was kept constant. C/IO was a SVM with I/O capability responsible for executing user commands. The number of the command interpreter SVM with I/O capability(CI/IO) was kept to one as the experiment results showed that the waiting time of users for the command interpreter service was close to zero. In experiments group J1 to J7, configurations shown in Fig 5.3(a), (b) and (c) were labelled C1, C2 and C3 respectively. For the eighth group of experiments in which half the users were doing compilations and half doing executions, the configuration shown in Fig 5.3(b), called configuration C1L1, meaning one compiler SVM and one loader SVM was experimented. Then the configurations, both represented by Fig 5.3(c), one labelled as C2L1 with two compiler SVMs and one loader SVM, another labelled as C1L2 with one compiler SVM and two loader SVMs were experimented.

5.1.4 The Collection of Data

In order to study the detailed behaviour of the FVMS on VM/SP, performance figures have to be collected at various points from the model in different configurations under different workloads. Fig 5.4 indicates the main stops a user made in having one command executed when the model was in the FVMS mode. The numbered arrow lines give the order of the events. They show that the user was first passed in a message from the command interpreter SVM(CI) to the CCC after the command was input by the user and interpreted there. The CCC would then find a suitable SVM(S) to carry out the command. So in the next step the user was sent to S and had the job executed. After this the user was transferred back to the CCC from S. Again the CCC found the first available CI (though in the experiments only one CI/IO was used, the model was designed to cope when more than one CI/IO was needed) and distributed the user to it and at this point the whole cycle was repeated.

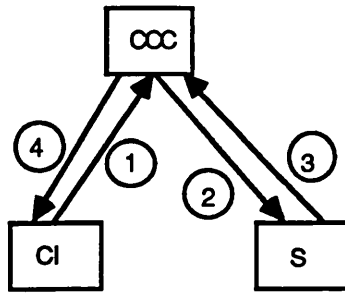


Fig 5.4 A user's journey during one command execution.

Therefore the data collected for the model in FVMS mode were as the follows:

- (1) the time waiting to be served in a CI (W_{SCI}).

This was the time between a user being accepted into a CI after being sent from the CCC and the CI actually picked him up to serve. As CIs were dealing with several users concurrently there was a gap between the two events. If the time between them was not small a CI was having too much load and was delaying the user. This is an indicator that more SVM(s) should be used to provide the command interpreter service.

- (2) the time waiting to leave a CI (W_{LCI}).

After the user entered a command, a CI might not be able to deal with this user immediately because it might be serving some other user. This time showed how long a user had to wait after the command was entered and before the CI started to send the user to the CCC. It included the little processing to decode the user's command. For SVMs like a CI or an editor which does not do much processing after a terminal input, it was an approximation of the response time such a multiplexed interactive SVM would produce.

- (3) the time taken for the message to travel from a CI to the CCC (C_1).

This was the time taken to transmit the message containing the user from a CI to the CCC. It was the time span which started from the moment the CI initiated the send procedure and ended at the time when the CCC had finished

receiving the message.

- (4) the user waiting time in the CCC (W_S).

This was calculated from the time the CCC had finished receiving the user, to the time the CCC started to send the user to a SVM. It was an approximation to the time spent on waiting before a suitable SVM(S) could be found to serve the user. As the actual processing, involved in dequeuing a SVM when one was immediately available, or queueing the user, or dequeuing a user when one SVM became free, was minimal compared with the actual waiting, this approximation was considered accurate enough.

- (5) the time taken for a message to travel from the CCC to a S (C_2).

Similar to (3), but this was the transmission time of the message from the CCC to S, allocated by the CCC to serve the user.

- (6) the time waiting to be served in S (W_{SS}).

Like in a CI, the user may not be served immediately after being received into S. For a multiplexed compiler SVM or loader SVM this waiting might be significant.

- (7) the service time in S (R).

This was the time taken to carry out the user's command. The changes in the configuration of the FVMS model would affect this time more than the other times collected.

- (8) the message transmission time from a S to the CCC (C_3).

- (9) the waiting time in the CCC for a CI (W_{CI}).

This was similar to (4). The difference was this was the queueing time in the CCC for a CI's service.

- (10) the message transmission time from the CCC to a CI (C_4).

These individually recorded times are actually the complete components of the response time to a user command. It was found that the response time collected before it was broken into these individual components was very close to the sum of them (less than 0.5% difference). This was also an indication that the calls to the system clock and the instructions to record them did not significantly affect the system performance. As a result this sum was accepted as the system response time of the model in FVMS mode while these components provided detailed behaviour of the FVMS. Equation 5.1 expresses this relationship where T denotes the response time.

$$T = W_{SCI} + W_{LCI} + C_1 + W_S + C_2 + W_{SS} + R + C_3 + W_{CI} + C_4 \quad (5.1)$$

When the model was operating in the IVMS mode, a user stayed in the SVM throughout the whole session. The information collected about the model behaviour in IVMS mode was for comparison with that of the FVMS mode. Therefore the response times to user typed commands were collected.

The data mentioned above was collected from the beginning to the end of each run of the experiments. However the statistics used in the performance study only included the measurements when the system had reached its equilibrium state. This means the figures from the 'transient periods'[39] when the system had just started and the simulated users were just logging into the system or executing their first few commands were excluded from the statistics.

5.2 The Performance of the Communication Subsystem

Before analysing the overall performance of the FVMS, a study of the performance of the communication subsystem will be done and this is the topic of the current section. In chapter three it was stated that the communication subsystem

could be the cause of an inefficient FVMS. The study on how this component of the FVMS perform will give some indication whether the communication overhead of a FVMS, composed of logically separate virtual machines of the VM/SP, is too heavy to make it a feasible system.

Using the send/receive procedures in the communication interface of the SVM submodel to pass a message from one virtual machine to another under controlled environment takes 15 to 18 milliseconds. The controlled environment refers to the situation where the sender has nothing to do after the send request, the receiver is waiting for the message and there is no other virtual machine running on the VM/SP system. Both the messages of 600 bytes and 4K bytes long (the size of a UMM message and a DB message described in chapter 4) takes the same time to be transmitted. This indicates that the 15 to 18 milliseconds are mainly the overhead imposed by the communication interface of the SVM and the IPCF in message packaging/unpackaging, message being put into one of the job lists of the SVM(Section 4.3.4.4) and being taken out ready for service, message buffer allocation and connection checking and so on.

The performance of the communication subsystem is examined from three aspects:

(1) the communication costs to the users.

This is the time spent by the user, during the execution of one user command, on message transfers among various SVMs and the CCC in order to get the software service provided by these SVMs.

(2) SVM's time not used in serving users

This refers to the time interval between the end of the service of one user and the beginning of the service of another in a single-user server SVM when there are users queueing in the CCC for the service. The events that happen in this interval are the passing of one user from the SVM to the CCC, CCC's dealing with the user and the transfer of another user to the SVM.

(3) how busy the CCC is

As the CCC is the centre of user distribution, there is the possibility that it becomes the bottleneck of the FVMS. The experiment results will show whether it is a significant factor in performance.

All these aspects of the performance of the communication subsystem will be discussed in the following subsections.

5.2.1 The Communication Costs to Users

Obviously the communication costs incurred during the input, the interpretation and the execution of a user command depend on where the software services are provided. The command input and interpretation are done in a command interpreter SVM (CI/IO). If the whole process of executing the command is carried out in a single SVM which is not the CI/IO, the communication cost will be the sum of the time taken for the user to go through the four message transfers as depicted in Fig 5.4, that is, if C denotes the total communication costs to the user

$$C = C_1 + C_2 + C_3 + C_4 \quad (5.2)$$

where C_1 , C_2 , C_3 and C_4 are the individual communication costs in (5.1).

If more than one SVM is involved in the execution of a user command, for example a multiplexed compiler SVM with I/O operations carried out in an I/O SVM, the communication costs will also include the message transfer time for the I/O operations. Because in the operation of the FVMS a user command is one service unit, there is no other cases of more than one SVM involving in the service of one command.

Table 5.3, Table 5.4 and Table 5.5 give the actual total communication costs(C) in executing the eight groups of experiments(J1 to J8) under different configurations

of the FVMS and with varying number of users on the system. J8(short) and J8(long) in Table 5.4 and Table 5.5 refer to the short compilation jobs and the long executions respectively in the J8 group of experiments. The figures in brackets in all the tables present the percentages of the communication costs in the response times to user commands. In other words they are obtained from the following expression

$$\frac{\text{Total Communication Cost (C)}}{\text{The Response Time (T)}} \times 100\%$$

T and C are as defined in (5.1) and (5.2) respectively.

In order to do comparison with non-multiplexed configurations, the communication times shown in the tables for the configurations involving multiplexed SVMs do *not* include the communication costs for I/O operations. This extra communication cost will be commented on later.

Table 5.3(a) The total communication costs in seconds for job types J1 to J7 where n is the number of users on the system. Configuration : C1.

n	1	2	3	6	9
J1	1.6 (8.6)		4.8 (8.8)		4.3 (2.4)
J2	0.6 (0.9)		3.4 (1.6)	4.7 (1.0)	5.6 (0.8)
J3	0.3 (0.5)	3.3 (2.7)	3.3 (1.7)	1.9 (0.4)	
J4	0.3 (0.8)		2.4 (2.3)		2.4 (0.7)
J5	0.3 (0.5)		3.8 (2.3)	2.5 (0.7)	
J6	0.8 (1.0)		3.4 (1.5)		2.9 (0.4)
J7	0.5 (0.8)	2.0 (1.7)	3.2 (1.8)	3.1 (0.8)	

Table 5.3(b) The total communication costs in seconds for job types J1 to J7 where n is the number of users on the system. Configuration : C2.

n	3	6	9	12	15	18
J1		2.9(2.6)	3.1(1.8)	3.0(1.3)	3.3(1.1)	2.5(0.6)
J2	3.1(1.5)	2.1(0.5)	2.1(0.4)	4.0(0.5)	2.2(0.2)	2.1(0.2)
J3		1.9(0.6)	1.8(0.3)	1.8(0.3)	1.7(0.2)	
J4		3.4(1.6)	3.5(1.0)	3.3(0.8)	3.4(0.6)	
J5	2.8(2.0)	3.4(1.3)	2.5(0.5)	3.5(0.6)	3.5(0.5)	2.6(0.3)
J6		1.7(0.4)	1.8(0.3)	1.6(0.2)	2.0(0.2)	
J7		2.4(0.7)	2.4(0.5)	2.3(0.3)	2.6(0.3)	

Table 5.3(c) The total communication costs in seconds for job types J1 to J7 where n is the number of users on the system. Configuration : C3.

n	6	9	12	15
J1		2.2 (1.2)		
J2	4.8 (1.1)	4.8 (0.7)		6.4 (0.6)
J3	6.5 (1.7)			
J4		2.2 (0.6)		
J5	2.0 (0.7)			
J6		4.3 (0.6)	5.3 (0.5)	
J7	1.8 (0.5)			

All the figures in Table 5.3-5 have been rounded to two significant digits and should be taken as order of magnitude estimates only, due to the facts that they are in seconds. However, as it can be seen, even these very rough measurements can be helpful to determine the degrees of the communication delay in the functioning of the FVMS.

Table 5.4 Total communication costs in seconds for J8. n = number of users. Configuration : C1L1.

n	J8 (short)	J8 (long)
2	0.5 (2.2)	1.0 (0.6)
4	2.8 (3.9)	2.5 (1.0)
6	2.7 (2.4)	2.9 (0.8)
8	2.4 (1.5)	2.5 (0.5)
12	2.4 (1.0)	2.1 (0.3)
16	2.3 (0.7)	2.2 (0.2)
18	2.5 (0.7)	1.9 (0.2)

All the tables in Table 5.3-5 show that generally the communication time was only a small percentage in the response time to user commands. Among the 90 readings from all the configurations with all types of jobs the communication times in 88 cases were less than four percent of the response time. In 87 readings it was less than three percent while in almost two third of the experiments, 57 readings to be

exact, it was less than one percent of the response time.

Table 5.5 The total communication costs in seconds in other configurations. j = job type. c = configuration. n = number of users. t = total communication time.

j	c	n	t
J1	M3	9	1.9 (0.5)
J1	M2	9	2.0 (0.9)
J1	M2IO2	9	1.6 (0.5)
J4	M3	9	3.9 (0.7)
J4	M2	9	2.9 (0.8)
J8 (short)	C2L1	12	2.3 (1.2)
J8 (short)	C1L2	12	3.8 (1.0)
J8 (long)	C2L1	12	2.6 (0.2)
J8 (long)	C1L2	12	3.4 (0.5)

These figures show that in the configurations of the FVMS experimented, with the job types and the number of users tried, the transfers of users among the SVMs did not significantly lengthen the response time. This could be interpreted as the insignificance of the communication delay in the response time formula(5.1). In fact the two most significant factors in the response time formula are the job service time and the time waiting for a suitable server to execute the job. These results are similar to the conclusions drawn by the research into the distributed graphical application[25]. That research put the speed of the workstation and the remote host as more important to performance than the speed of the communication.

However if the absolute value of the communication time is considered, the communication cost would be a substantial overhead on the execution of the commands which would have only a few seconds of response times from the IVMS under the same workload.

In the following sections the figures displayed in Table 5.3-5 are studied to see whether the number of users on the system, the way the FVMS is configured and the types of jobs executed on the system are significant factors that affect the

communication costs of the FVMS.

5.2.1.1 The Communication Costs and the Number of Users on the System

Examining Table 5.3 column by column and Table 5.4 row by row, two observations can be made:

- (1) the first column of Table 5.3(a) and the first row of Table 5.4 were consistently lower than the majority of the other readings from the same configuration in all groups of experiments;
- (2) when the first column of Table 5.3(a) and the first row of Table 5.4 are excluded, the absolute communication times did not apparently increase or decrease with the increase of the number of users in a particular configuration of the FVMS.

These observations are no surprises. After a FVMS with fixed number of SVMs reaches equilibrium state, the number of users does not affect the speed of the system, that is, the throughput of the system. Because the communication subsystem and the message transfer are run on the same physical system as the rest of the FVMS, the communication cost in terms of time is not affected by the number of users on the system. The difference between the low communication times of the first column of Table 5.3(a) and the first row of Table 5.4 and the rest of the readings is because they were obtained from two different levels of equilibrium states. The former were obtained when the number of users in the FVMS was equal to the number of SVMs which actually executed the users' jobs like compilers or loaders. For example: the first column of Table 5.3(a) where there was only *one* user in the system and C1 represents the configuration with one command interpreter and *one* compiler or *one* loader plus the CCC. The comparatively higher communication costs were obtained when there were more users than the number of SVMs which executed the users' jobs.

Table 5.6 Comparison of individual message transfer times (in seconds).

job	no. of users	C ₁	C ₂	C ₃	C ₄
J1	1	0.1	0.7	0.4	0.3
	mean	0.1	4.3	0.1	0.1
J8(short)	2	0.0	0.1	0.2	0.3
	mean	0.1	1.7	0.3	0.4
J8(long)	2	0.3	0.3	0.3	0.0
	mean	0.2	1.6	0.2	0.3

The difference the two levels of equilibrium states brought may be found when the components of the total communication costs, that is, the individual message transfer times, are examined. Table 5.6 lists these individual message transfer times in two groups of experiments (J1 and J8) as examples. In the table 'mean' refers to the average message transfer time for the particular component (C₁, C₂, C₃ or C₄) when the number of users was greater than one for configuration C1 and job J1 and when the number of users was greater than two for configuration C1L1 and job J8.

In Table 5.6 one phenomenon is very noticeable: the mean communication time for C₂, that is, the communication time for the messages from the CCC to a SVM other than the C1, is considerably longer than that where one user was in experiment J1 and two in experiment J8.

The substantial increase in C₂ may be explained by the following fact: when the number of users was greater than the number of SVMs which actually executed the users' commands, there were always some users waiting for the service of the SVM in the CCC when a SVM finished the job of one user. And because the actual execution of a user's command was longer than the input and the interpretation of a command, the C1 was waiting in the CCC for a user most of the times when a user needed C1's service.

When a user arrived at the CCC from the SVM at the end of a service, the CCC would pass the user to the CI and then send a new user to the SVM to serve. Because the user to the CI was sent first and the CCC had to dequeue a new user from the user queue before sending to the SVM, the CI would almost certainly get scheduled to run before the SVM. Also because CI dealt with the input and output of user commands it was an interactive virtual machine which always got scheduled before non-interactive virtual machines like the SVMs doing compilation or execution of user programs in the experiments. Therefore from the time the CCC sent a user to the SVM to the time when the SVM got scheduled to receive the message the CI had received its user and most likely had processed the user. This lengthening in message transfer time for one message out of two would always happen when the CCC has to distribute two users(messages) to two SVMs (including the CI) at the 'same' time, more so if the first receiver had more favourable treatment from the CP than the second receiver because of interactiveness.

On the other hand, when the number of users was equal to the number of SVMs which executed users' commands, the above situation would not rise(this was true in the experiments because each user always issued the same command). There would be no user waiting for the service of the SVM which executed user commands. As a result the CCC would not need to deliver two messages at the same time. Of course if all SVMs separated their calls for next user to the CCC from the return of the current user, the CCC would never have to send two messages at once and no individual message transfer time would be lengthened. However this was ruled out for two reasons. First, the lengthening of individual message transfer time would be avoided but the cost for transferring the extra message had to be added. Secondly, the number of incoming messages to the CCC would double and this would greatly increase the system resources consumed by inter-virtual machine communications and the chances of the CCC becoming a bottleneck.

In summary, two classes of communication times exist. One with the shorter magnitude relates to the situation where the number of users is so small that the CCC does not have to distribute two users to two SVMs at once. If this is not the situation, the other class with the longer magnitude is the communication cost to the users.

The communication time has a decreasing percentage in the response time as the number of users increases in a particular configuration of the FVMS. This is natural following the fact that when more users are served by the same configuration of FVMS the waiting time of users for a suitable server(SVM) is bound to increase. Thus even though the absolute communication time does not appear to change very much its percentage in the response time falls.

5.2.1.2 The Communication Costs and the Different Configurations of the FVMS

The question to be answered is whether there is significant difference between the communication times collected from the experiments with different configurations. It has just been shown that if the two classes of the communication times are not mixed up the number of users on the system is not a factor which affects the communication cost. Therefore the communication times from experiments with different number of users may be considered as within a relatively homogeneous block of experimental material when the users in the experiments were running the same jobs in varied configurations. For example, disregarding the figures from Table 5.3(a) concerning one user on the system, each row in Table 5.3(a) may be examined against the figures from the corresponding row from Table 5.3(b). Appendix B presents the results of the calculations according to the process called the analysis of variance[40]. The *null hypothesis* is that there is no difference in the communication times of the different

configurations. The alternative hypothesis is significant differences exist among the communication times from different configurations. The summaries drawn from these calculations consist of the following points:

- (1) There was evidence that as the number of SVMs were increased from configuration C1 to C2 then to C3 the communication times from these configurations were different, at least when certain types of jobs were executed in the experiments.
- (2) There was no evidence that configurations C2, C1L1 and C2L1 produced different communication times. However, significant difference did exist between communication times from configurations C1L1 and C1L2;

The conclusion is that the increase of the number of SVMs which are executing user commands may or may not affect the communication times, depending on what type of jobs are executed in these SVMs. Though no analysis of variance was done involving configurations with multiplexed SVMs, that is, configurations M3, M2 and M2IO2, based on the conclusion drawn from above it is induced that when the number of SVMs and the type of jobs they serve are kept unchanged the fact that whether these SVMs are multiplexed or not makes little difference to the communication costs. The figures presented in Table 5.5 concerning the multiplexed SVMs apparently support this statement.

5.2.1.3 The Communication Costs and the Types of Jobs on the System

The results of the analysis of variance in Appendix B also point out that significant differences exist between the communication times when different jobs are executed in configuration C2 (Table 5.3(b)). The same procedure indicates that running job J3 and job J6 in configuration C2 yield the same communication times and running the rest of the job types(J1, J2, J4, J5 and J7) yield the same communication times. Table 5.1 and Table 5.2 confirm that both J3 and J6 had big

working sets. Though the working set sizes for J1 and J2 looked big, they were compilation jobs and the part of the working set which was the compiler code was shared among the compiler SVMs, so the real memory requirements were less. The CPU time and the I/O requirements did not seem to have dominating effect.

In configuration C1 the analysis of variance concludes that there is no difference among the communication times when different jobs were on the system.

In the above cases the users were all doing the same job in each experiment. Table 5.4 contains the communication times when half the users were doing one job(J8(short) which is actually J1) and the other half were doing another job(J8(long) which is J3) in configuration C1L1. The results of the analysis of variance in Appendix B state that there was no difference between the communication times of the users doing different jobs in one system. Note the two different jobs in J8 produced different communication times when separately executed in different experiments in configuration C2, which had the same number of SVMs as configuration C1L1.

The conclusion from these would be when jobs, which produce different communication times if run separately, are run together, the resultant communication times are closer to each other and the difference disappears. In other words, the communication times depend on the total workload of the system. So jobs which put different workloads on the system experience different communication costs. When they are part of a total workload, though they may be contributing differently towards it they experience the same communication cost.

5.2.1.4 The Conclusion about the Total Communication Costs

The total communication costs to users in the input and the execution of a user command is about three percent or less of the response time. The communication speed

seems less important than other factors like the speed of the service to the response time.

Generally speaking, the number of users on the system does not affect the communication costs. However, lower than normal communication costs is obtained when the number of users is so low that whenever a user wants the service of a certain type of SVMs, there is always one. Therefore users never need to wait for a SVM and the CCC never has to distribute two messages at the same time.

The increase in the number of SVMs providing a certain service may or may not affect the communication costs. This depends on the service(s) the extra SVM(s) are to perform. In other word, it depends on the load the extra SVM(s) put on the system.

There is no significant difference between the communication costs from a FVMS which employs multiplexed SVMs and that which does not, provided that in both systems, the number of SVMs are the same and the services are distributed among them in the same way.

The type of jobs executed in the system and thus the whole workload on the system affects the communication costs.

5.2.2 The Communication Costs to Non-Multiplexed SVMs

A relevant measure of the efficiency of the communication subsystem is the percentage time a SVM spends in communicating with the CCC and therefore is not able to serve a user. It is defined as

$$\text{communication factor} = \frac{\text{time in communication between services}}{\text{time in communication between services} + \text{service time}} \times 100\% \quad (5.3)$$

When there is a constant stream of jobs to be processed, the only time a

multiplexed SVM is not serving a user is when a message containing a user is being transferred into it as the time to initiate the procedure to send a user back to the CCC asynchronously is negligible. Therefore the communication factor is very small. For a non-multiplexed SVM, it has nothing to do after sending a user asynchronously to the CCC and before the next user has been received. This time period includes the time spent in the two message transfers(each includes the passing of the message interrupt from the sender to the receiver, the scheduling of the two communication parties and the physical transfer of the message) and the processing time of the CCC. Given that the processing time of the CCC is so short that it is reasonable to ignore it, the sum of the two message transfer times is an accurate estimate of the time gap between its serving two users in a non-multiplexed SVM. This time gap is the communication cost to a non-multiplexed SVM.

Table 5.7(a) SVM idle time and its communication factor(configuration : C1).

no. of users	2	3	6	9
J1		4.6(21.8)		4.1(19.7)
J2		3.0 (4.2)	4.0(5.0)	5.1 (6.4)
J3	3.1 (4.7)	2.8 (4.2)	1.1 (1.5)	
J4		2.3 (6.1)		2.3 (6.1)
J5		3.5 (6.1)	2.3(3.9)	
J6		3.2 (3.9)		2.5 (3.1)
J7	1.9(3.0)	3.1 (4.9)	3.0(4.7)	

Table 5.7(b) SVM idle time and its communication factor(configuration : C2).

no. of users	3	6	9	12	15	18
J1		2.6 (6.5)	2.6 (6.6)	2.5 (6.0)	3.0 (8.1)	1.6 (4.1)
J2	2.3 (1.6)	1.2 (0.9)	1.0 (0.8)	2.6 (1.9)	1.0 (0.8)	1.1 (0.8)
J3		0.8 (0.7)	0.5 (0.4)	0.7 (0.6)	0.7 (0.5)	
J4		3.1 (4.1)	3.0 (3.9)	2.9 (3.9)	3.0 (4.0)	
J5	2.4 (2.5)	3.1 (3.3)	2.1 (2.0)	3.0 (3.2)	3.0 (3.2)	2.2 (2.3)
J6		1.0 (0.6)	1.0 (0.6)	0.8 (0.5)	0.7 (0.4)	
J7		2.1 (1.8)	2.1 (1.8)	1.9 (1.6)	2.2 (1.9)	

Table 5.7(c) SVM idle time and its communication factor(configuration : C3).

no. of users	6	9	12	15
J1		1.6 (2.6)		
J2	2.1 (0.9)	2.5 (1.1)		2.6 (1.2)
J3	4.1 (2.1)			
J4		1.6 (1.4)		
J5	1.4 (1.0)			
J6		1.2 (0.5)	1.0 (0.4)	
J7	1.2 (0.7)			

Table 5.7(a) to (e) display the time in seconds a non-multiplexed SVM spent in communicating with the CCC between two services in the experiments with the FVMS.

The figures in brackets are the communication factors.

**Table 5.7(d) SVM idle time and its communication factor(configuration : C1L1.
S = J8(short). L = J8(long).).**

no. of users	4	6	8	12	16	18
S	2.4 (6.0)	2.2 (5.4)	1.9 (4.5)	1.8 (4.4)	1.7 (4.1)	2.1 (5.1)
L	2.0 (1.6)	2.5 (2.0)	2.0 (1.6)	1.5 (1.2)	1.6 (1.3)	1.4 (1.1)

Table 5.7(e) SVM idle time and its communication factor.

configuration	job type	no.of user	idle time
C2L1	J8(short)	12	1.1 (1.7)
C2L1	J8(long)	12	1.1 (0.6)
C1L2	J8(short)	12	2.3 (3.6)
C1L2	J8(long)	12	1.4 (0.6)

From the tables it is observed that the time gap between the service of two users in a SVM ranges from 0.5 seconds to 5.1 seconds. The distribution of these time gaps is given in the histogram in Fig 5.5. It can be seen clearly that the majority of

them is below 3.4 seconds. The mean of the seventy six time gaps is 2.2 seconds.

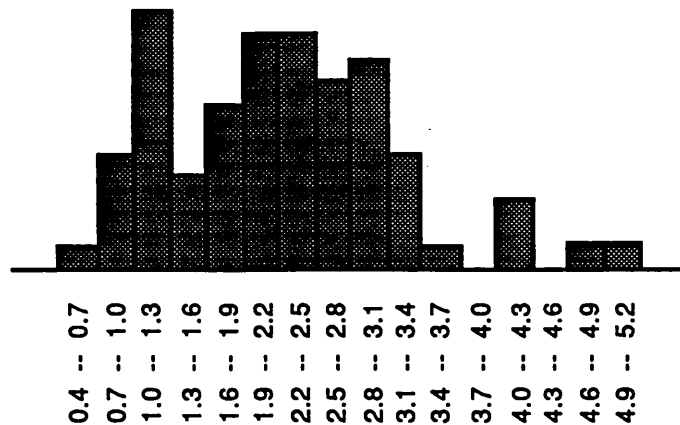


Fig 5.5 Distribution of the time gaps between services in SVMs.

Naturally for a small job like J1 which needs about 10 seconds of CPU time (Table 5.1) the communication factor is higher than those cases where jobs take longer to finish. For configuration C1 the average communication factor for J1 is 20.8% while that for all the other job types ranges from 1.5% to 6.4%. For configuration C2 the average communication factor for J1 is 6.3% while that for all other job types ranges from 0.4% to 4.1%. In configuration C3 the average communication factor for J1 is 2.6% (the only measurement) while that for all other job types ranges from 0.4% to 2.1%. Even when the jobs are mixed in configuration C1L1, the average communication factor for J8(short), which is J1, is 4.9% while that for J8(long), which is J3, is 1.5%. These figures also mean except the servers doing short services ('short service' means response time of 20 seconds or less -- 10 times the average length of the time gaps is considered appropriate) the non-production time of the servers because of communication is short.

5.2.3 Is the CCC a Bottleneck?

The most serious form of a bottleneck at the CCC in a FVMS is when the service rate of the CCC is lower than the sum of the rates at which all the SVMs are sending messages to the CCC. The service rate of the CCC is the rate at which the CCC accepts messages and processes them. The rate a SVM sends messages to the CCC is the rate it accepts users and serve them. Obviously when the system is in an equilibrium state the first rate and the sum of the rates from all the SVMs are controlling each other and settle at the slower rate. If the CCC produces the slower rate, the SVMs would be constantly waiting for the CCC to receive their messages and send them new ones. The CCC becomes the bottleneck of the system. However this was considered unlikely because the amount of processing the CCC does to process a message would be negligible when compared with the amount of processing a SVM had to do to execute a user command. And from the results shown in the previous section, the SVMs were not spending a lot of time waiting for the CCC. The fact that the time gaps in Table 5.7(a) to (e) did not change with the increase of the number of users or the number of SVMs is a proof of that.

A less serious form of bottleneck at the CCC is the periods of time when more than one or even all SVMs happen to send messages to the CCC in small intervals and the collective rate of messages is faster than the CCC's service rate. This happens when by chance more than one SVM finish the service of their users in close intervals. On a time sharing system this could happen when SVMs are producing messages more than the CCC can receive and process them in a couple of time slots when the CCC is scheduled to use the CPU.

To estimate how busy the CCC reached in the experiments, Table 5.8 lists the probabilities of the CCC having a less than one second long interval between incoming messages in various configurations. The message intervals were calculated from the

moments each time a SVM tried to send a message to the CCC in the experiments concerning the relevant configurations. '2 SVMs' refer to the configuration C1; '3 SVMs' cover the C2 configuration and the C1L1 configuration; '4 SVMs' include the C3 configuration, the C1L2 and the C2L1 configuration; '3 SVMs(M3)' refer to the M3 configuration and '3 SVMs(M2)' the M2 configuration. The probability is obtained by dividing the number of intervals which is less than one second by the total number of message intervals.

Table 5.8 Probability of having less than one second of message interval to the CCC.

2 SVMs	3 SVMs	4 SVMs	3 SVMs(M3)	3 SVMs(M2)
0.0000	0.0018	0.0037	0.1255	0.0492

A message interval of less than one second happened when two different SVMs tried to send to the CCC within one second or when a multiplexed SVM (maybe a CI or an ordinary SVM executing a user command) had more than one user finishing the service within a second.

Table 5.8 indicates that the probability of a less than one second message interval is extremely low in the configurations which do not use multiplexed SVMs to execute user commands. The probability of having two messages inside one period of time when the CCC does not have control of the CPU would be even lower. Therefore the probability of the CCC becoming a bottleneck in the configurations experimented is very remote. However some temporary short delay or congestion could occur at the CCC if more than one SVM send messages to it before it gets control of the CPU though the probability of its happening is very low. This was stated at the beginning of the section.

There is another type of delay. If a multiplexed SVM or CI tries to send more

than one messages during the time span when it has control of the CPU the second message cannot be sent out because of the limitation of the communication subsystem. In other words this SVM is blocked. The probability of this happening is small and could almost be avoided if the CCC is classified as a priority virtual machine which has the control of the CPU whenever it is runnable.

5.3 The Performance of the FVMS and That of the IVMS

When the performance of the FVMS is compared with that of the IVMS, two aspects are considered. First is the average response time of all requests made to the system. Second is the average response time each user is receiving. This section discusses them one by one.

5.3.1 Comparison of the Average Response Time from the FVMS and the IVMS

To compare the average response times from the FVMS and the IVMS mode of the model, regression analysis[40] is used and the first step is to develop prediction equations from the experimental results of the model. These prediction equations express the mean response time as a function of the number of users on the system. By comparing the prediction equations for the FVMS and the IVMS, conclusion may be drawn with certain confidence level that whether there is significant difference between the two systems' performance and whether one system is better than the other.

Obviously the dependent (or response) variable in the prediction equations is the mean response time. There are two independent variables. They are one

quantitative variable, the number of users on the system, and one qualitative variable, whether it is FVMS or IVMS mode of the system. The workload of the system in terms like the CPU utilisation and other similar terms is not considered here because the experiments were conducted under strictly controlled environments so that the performance of the two systems may be compared under the same workload in terms of the number of users and the jobs they do on the systems. To show directly how the mean response time of all the users on the system changes with the increase of the number of users in both the IVMS and the FVMS, the dependent variable is expressed as a multiple of the mean response time in an one-user IVMS from the same group of the experiments (thus the users are doing the same jobs). In group J8, it is the multiple of the average response time in the two-user IVMS (one doing job J1 and the other doing job J3). Subsequently they are referred to as the normalised average response time.

An inspection of the data obtained from the experiments would lead to the hypothesis that a second-order prediction equation would provide a good fit to them. Therefore if y represents the normalised average response time, $E(y)$ is the expected mean of y , x_1 represents the number of users on the system and x_2 the mode of the model, that is, FVMS or IVMS, the following prediction equation is to be constructed by the method of least squares:

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 + b_3x_2 + b_4x_1x_2 + b_5x_1^2x_2 \quad (5.4)$$

$$x_2 = 1 \quad \text{if IVMS is the system being used}$$

$$0 \quad \text{if FVMS is the system being used.}$$

Therefore for FVMS,

$$E(y) = b_0 + b_1x_1 + b_2x_1^2$$

and for IVMS,

$$E(y) = (b_0 + b_3) + (b_1 + b_4)x_1 + (b_2 + b_5)x_1^2.$$

By assuming the presence of b_0 and b_3 in (5.4), the prediction equation to be constructed would fit the experimental results best but it may not pass the origin. This may not seem logical. However, strictly speaking the value of the response time is not zero but undefined when the number of users is zero. In order to get the curve or straight line which fits the results best, b_0 and b_3 were not taken out from the equation and it is noted that the equation does not apply when there is no user on the system.

Individual test is carried out on each of the b-parameters in (5.4). The t test with the null hypothesis $b_i = 0$ ($i = 0, 1, \dots, 5$) against the alternative hypothesis $b_i \neq 0$ is conducted. If only the hypothesis $b_2 = 0$ is not rejected in all the tests for b_i , it indicates that the experimental results provide evidence that the prediction equation for the FVMS is a straight line but that for the IVMS is a parabola.

Another type of tests, the F test, is conducted in the process of the prediction equation building. To test that not both of the prediction equations for the FVMS and the IVMS are straight lines, the null hypothesis is $b_2 = b_5 = 0$ and the alternative hypothesis is either b_2 or b_5 is not zero. To find out whether there is sufficient evidence to support the statement that there is significant difference in the performance between the FVMS and the IVMS is to test whether two prediction equations instead of one are required for predicting the response variable for the two systems. In terms of the b-parameters, the null hypothesis is all the b-parameters for terms involving the x_2 variable are zeros against the alternative hypothesis that at least one of them is not zero.

Appendix C shows the results of the regression analysis done on the data obtained from the eight groups of experiments. Fig 5.6(a) to (h) depict the response curves or lines whose equations are also listed below and they are discussed here.

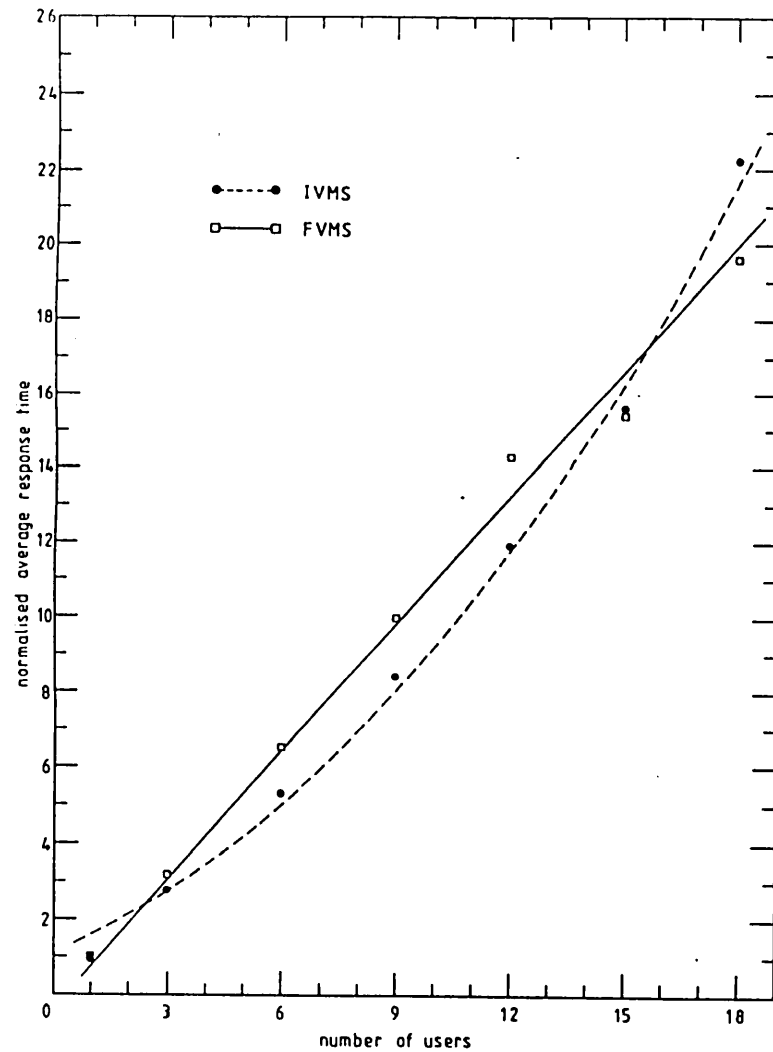


Fig 5.6(a) The relationship between the normalised response time and the number of users on the system for group J1.

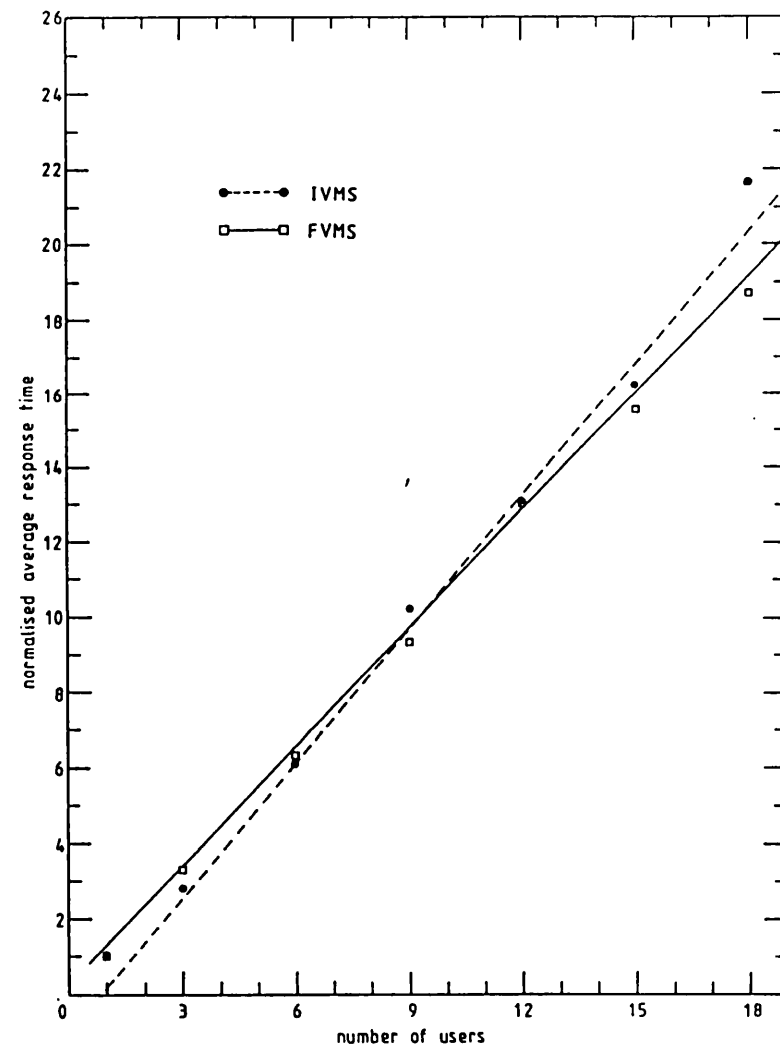


Fig 5.6(b) The relationship between the normalised response time and the number of users on the system for group J2.

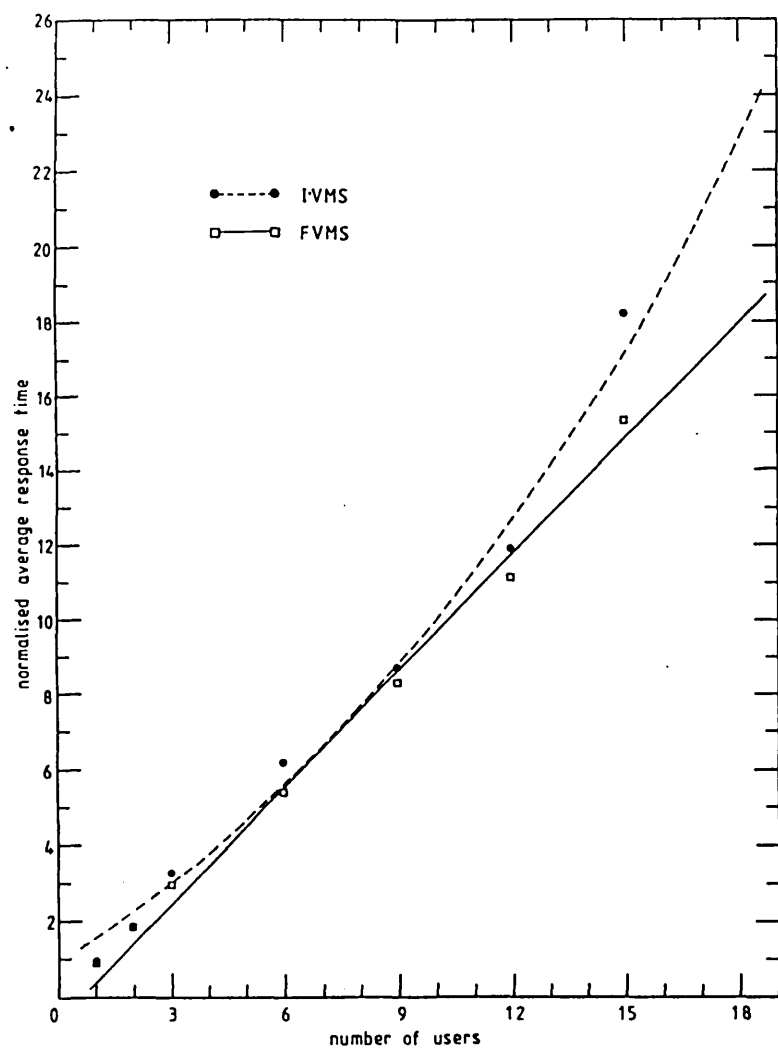


Fig 5.6(c) The relationship between the normalised response time and the number of users on the system for group J3.

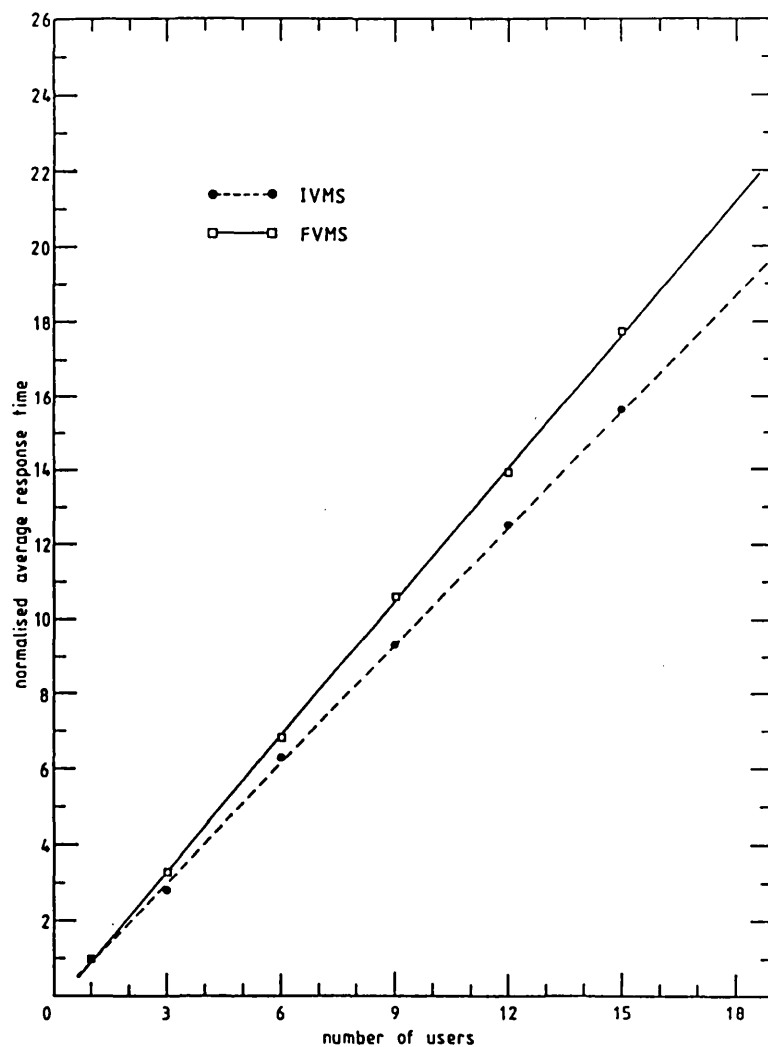


Fig 5.6(d) The relationship between the normalised response time and the number of users on the system for group J4.

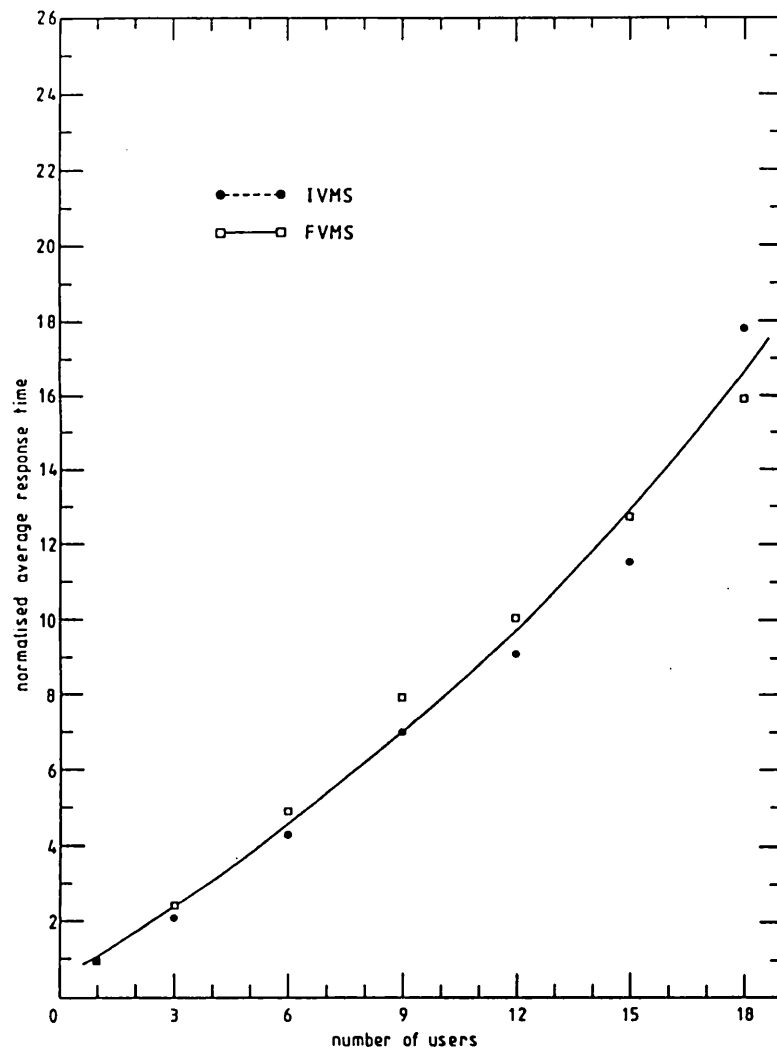


Fig 5.6(e) The relationship between the normalised response time and the number of users on the system for group J5.

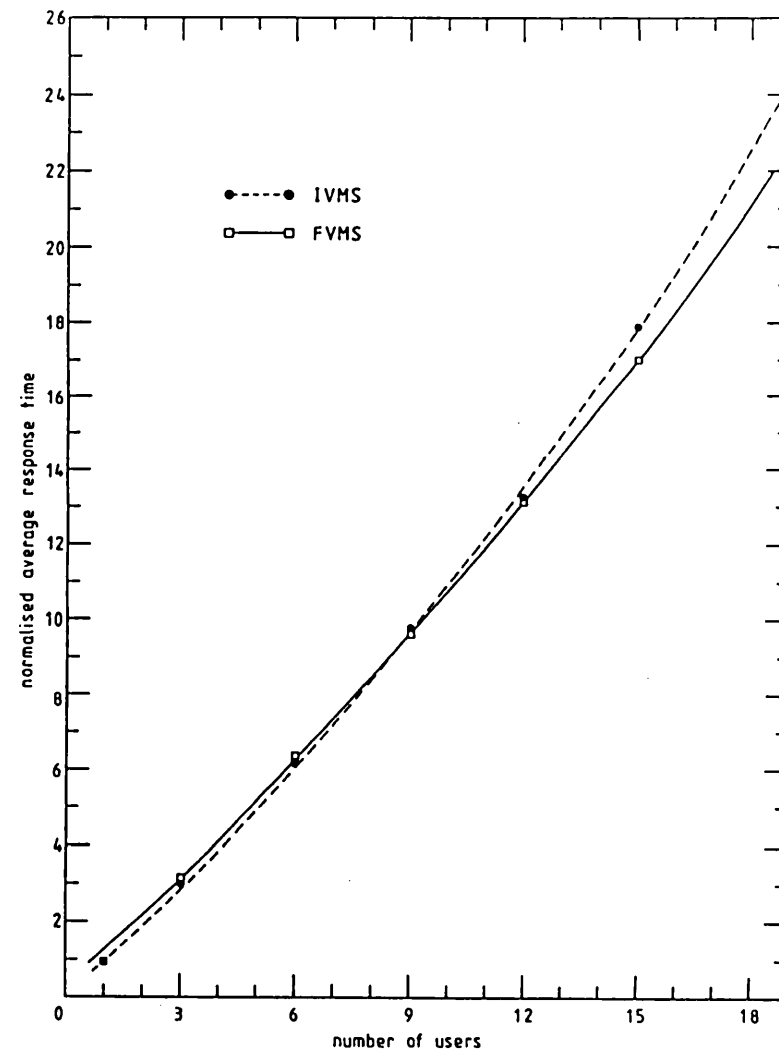


Fig 5.6(f) The relationship between the normalised response time and the number of users on the system for group J6.

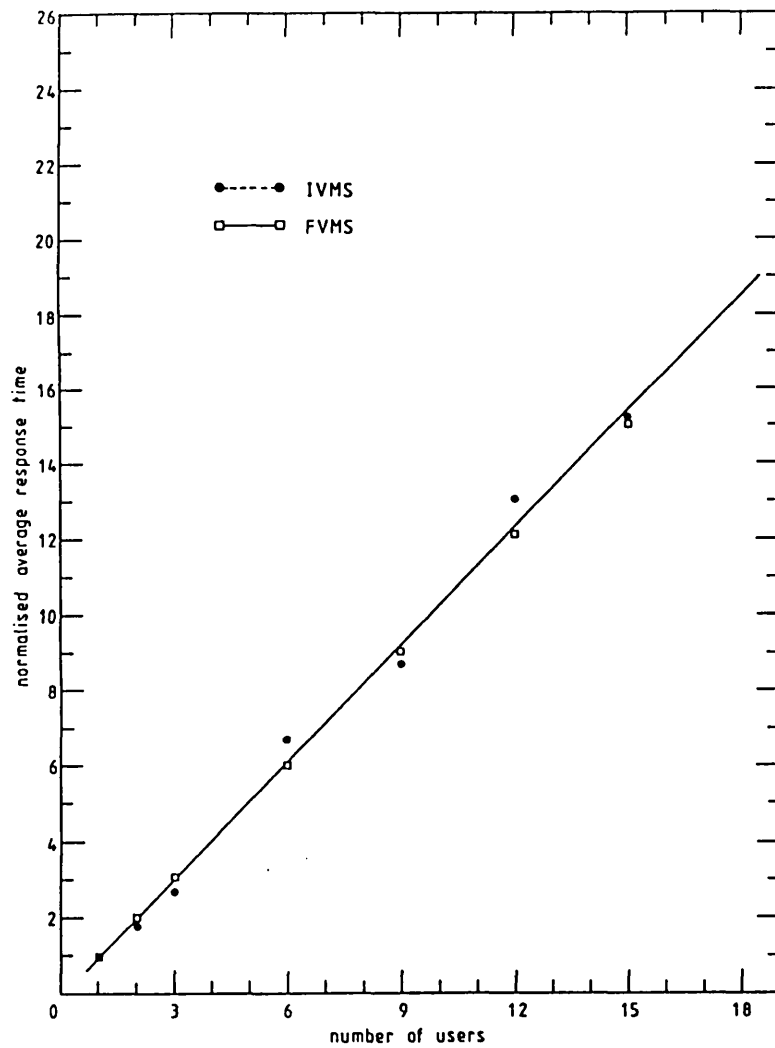


Fig 5.6(g) The relationship between the normalised response time and the number of users on the system for group J7.

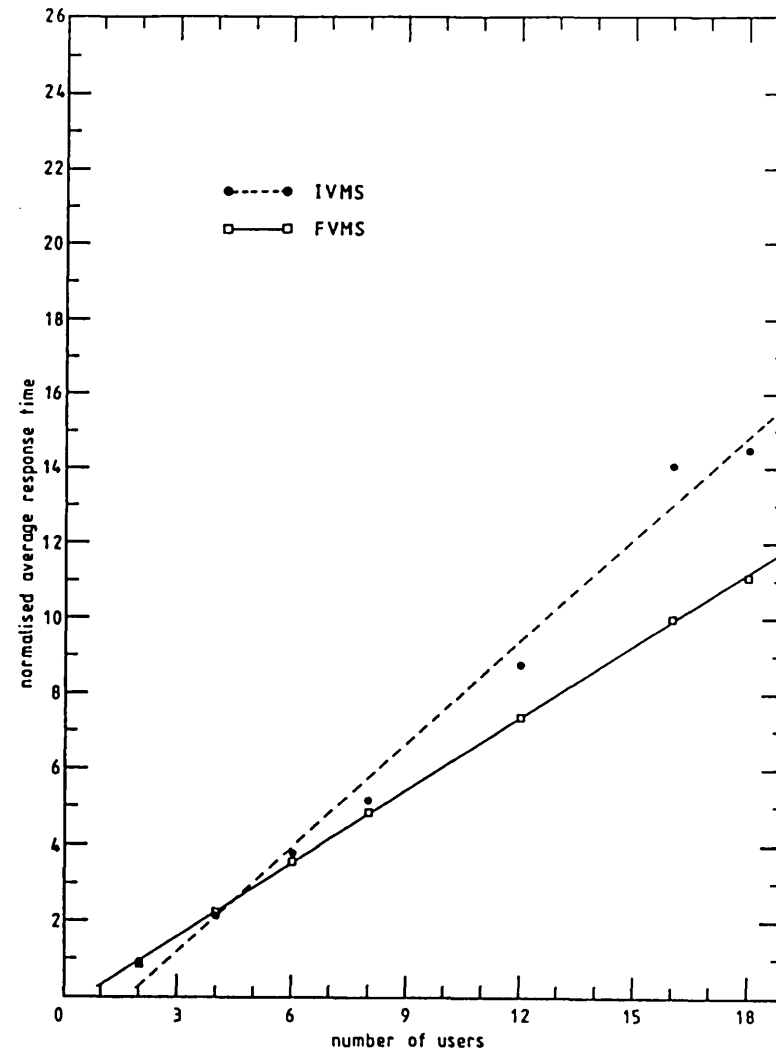


Fig 5.6(h) The relationship between the normalised response time and the number of users on the system for group J8.

<u>Group J1</u>	
IVMS:	$E(y) = 1.3185 + 0.3953x_1 + 0.0400x_1^2$
FVMS:	$E(y) = -0.2957 + 1.1261x_1$
<u>Group J2</u>	
IVMS:	$E(y) = -0.9356 + 1.1822x_1$
FVMS:	$E(y) = 0.3609 + 1.0404x_1$
<u>Group J3</u>	
IVMS:	$E(y) = 1.1967 + 0.5142x_1 + 0.0375x_1^2$
FVMS:	$E(y) = -0.5658 + 1.0283x_1$
<u>Group J4</u>	
IVMS:	$E(y) = -0.1453 + 1.0515x_1$
FVMS:	$E(y) = -0.2455 + 1.1909x_1$
<u>Group J5</u>	
IVMS & FVMS:	$E(y) = 0.5829 + 0.5317x_1 + 0.0192x_1^2$
<u>Group J6</u>	
IVMS:	$E(y) = 0.1030 + 0.9133x_1 + 0.0176x_1^2$
FVMS:	$E(y) = 0.1151 + 0.9661x_1 + 0.0105x_1^2$
<u>Group J7</u>	
IVMS & FVMS:	$E(y) = -0.0597 + 1.0224x_1$
<u>Group J8</u>	
IVMS:	$E(y) = -1.4086 + 0.9009x_1$
FVMS:	$E(y) = -0.2691 + 0.6376x_1$

Among these equations which relate the normalised average response times to the number of users on the system, group J4 is the only one where the IVMS provided a better performance than the FVMS(Fig 5.6(d)). The IVMS and the FVMS provided equivalent average response times in groups J5 and J7(Fig 5.6(e) and (g)). In groups J1, J2 and J8(Fig 5.6(a), (b) and (h)), the FVMS had better average response times than the IVMS when the number of users on the system exceeds a certain value. Below this value, the FVMS produced average response times slower than or equivalent to the IVMS'. The FVMS had better performance in group J3 and J6(Fig5.6(c) and (f)) no matter what the number of users on the system was.

If the working sets of all these jobs are examined(Table 5.1 and 5.2), it is noted that job J4 had a working set of size 83 Kbytes. It was followed by job J5(91 Kbytes) and J7(99 Kbytes). All the remaining jobs had working sets bigger than 100 Kbytes(in group J8 half the users were doing J1 and half J3). It should be noted that J1 and J2 were compilation jobs. On the VM/SP system on which the comparison of

IVMS and FVMS were done, the compiler code was shared by all the virtual machines. As a result, although the working sets of J1 and J2 appeared to be bigger than those of J3 and J6, the demand on the physical memory in group J1 and J2 was not as big as the numbers indicated.

It points to the conclusion that the size of job working set is a determining factor in deciding the relative efficiency of the FVMS and IVMS. If the users on the system have high memory demands the FVMS offers more efficient service than the IVMS except when the number of users is very low. One of the reasons is because 'thrashing' occurs on the IVMS when the number of users exceeds a certain value. In Fig 5.6(a), (c) and (f), the lower ends of the response curves for the IVMS approximated straight lines and these curves turned upwards making parabola shapes as the number of users increased. However 'thrashing' is not the only reason for FVMS' better performance over IVMS'. As Fig 5.6(b) and (h) showed FVMS was able to produce better performance before 'thrashing' takes place. In these two diagrams the IVMS response curves were straight lines with steeper gradients than FVMS' straight lines. It is expected that as the number of users increases beyond the ones experimented, the IVMS response lines would turn upwards and follow the patterns showed in Fig 5.6(a), (c) and (f). The point is the FVMS was making more efficient use of the system resources when the user demands on memory are heavy, even before 'thrashing'.

In group J4 the prediction equations for the FVMS and the IVMS suggest that the IVMS produces better average response times whenever the number of users is greater than one. Furthermore the more users there are the better performance the IVMS would have over the FVMS. Table 5.2 indicates that job J4 had 83 Kbytes working set and did about 600 lines of I/O but consumed nearly 29000 milliseconds of CPU time -- this is a CPU-intensive job. It appears that under this kind of workload the FVMS is not making full use of the system resources because it is trying to serve more users

using a lot less virtual machines.

5.3.2 The Fairness of the FVMS and the IVMS to Users

The experiments showed that 'thrashing' was still likely to occur on the IVMS. The CP had tried to prevent it from happening by allowing only some virtual machines to concurrently execute on the real machine while denying some other virtual machines access to the system resources for quite some time when the workload is heavy. The scheduler's policy is to favour interactive jobs. However one consequence of this is: after being denied the machine resources a virtual machine of course has not done any terminal I/O in a time period; when the scheduler makes decision about who should be in the runnable list and who should not this virtual machine will have less chance of going into the runnable list than a virtual machine which was executed for a time slice and had done once or twice terminal I/O. When there are a lot of active users on the system a virtual machine may be delayed in this way for a long time. This is not to say that such virtual machines will be forgotten by CP. However the users in these virtual machines put aside by the CP face intolerably long wait and in real life they would almost certainly give up and terminate their sessions. Table 5.10 displays the minimum and maximum of the normalised average response times of individual users from both the IVMS and FVMS for job types J1, J2, ..., J7 with the normalised average response time as defined before. In the same table, for job type J8(mixture of J1 and J3), the users doing J1 have their average response time presented as multiples of the user average response time of the user doing job J1 in a two-user IVMS where the other user is doing J3. Similarly the response times of the users doing J3 in group J8 is expressed as multiples of the response time of the user doing J3 jobs in the same two-user IVMS.

As can be seen from Table 5.10, in the experiments involving J1, J2,... J7,

users are doing the same jobs repeatedly and on the IVMS they experience similar response times when the number of users is low. However when the number of users is high, the user with the fastest response time may have the system responding to him in less than one tenth of the time that the user with the slowest response time is getting. Sometimes the slowest user average response time may be as high as 50 times the fastest user average response time. On the other hand, the FVMS responds to the user requests in a very fair manner and all users experience similar response times when they are doing similar jobs. Obviously the fastest user average response time on the IVMS is shorter than that on the FVMS but on the other hand the slowest response time on the IVMS is much longer than that on the FVMS. The experiments involving mixed jobs(J8) manifest similar fairness of the FVMS and unfairness of the IVMS. It should be noted that the figures for the FVMS here are obtained from configuration C1L1 where equal number of SVMs(one) are serving each one of the two types of the job requests.

Table 5.10 The minimum and maximum of user average response time from the IVMS and FVMS.

job	no of user	IVMS		FVMS	
		min	max	min	max
J1	2	1.5	1.5		
	3	2.8	3.0	3.2	3.2
	6	5.1	6.1	6.6	6.6
	9	7.1	11.5	10.1	10.2
	12	7.3	23.7	14.5	14.6
	15	6.5	395.7	17.9	18.9
	18	6.5	342.8	23.3	24.3
J2	3	2.7	2.8	3.3	3.3
	6	5.1	9.8	6.3	6.3
	9	7.1	26.9	9.1	9.6
	12	7.9	23.9	12.9	13.1
	15	10.5	66.2	15.5	15.9
	18	7.5	237.2	18.7	18.7

Table 5.10(continue..) The minimum and maximum of user average response time from the IVMS and FVMS.

job	no of user	IVMS		FVMS	
		min	max	min	max
J3	2	1.9	1.9	1.9	1.9
	3	3.2	3.4	3.0	3.0
	6	5.1	8.0	5.4	5.5
	9	5.0	34.8	8.2	8.4
	12	5.4	61.9	11.0	11.1
	15	4.5	83.8	15.3	15.3
J4	3	2.6	3.2	3.3	3.3
	6	5.7	6.6	6.6	7.0
	9	8.1	13.3	10.6	10.7
	12	10.4	46.2	13.6	14.9
	15	13.5	19.5	17.4	17.8
J5	3	2.1	2.1	2.4	2.5
	6	3.8	4.9	4.9	5.0
	9	6.9	11.5	7.9	8.0
	12	7.6	12.2	10.0	10.1
	15	8.8	18.7	12.6	12.7
	18	5.9	236.2	15.9	16.0
J6	3	2.9	3.1	3.2	3.2
	6	5.3	8.6	6.4	6.5
	9	8.0	19.6	9.7	9.7
	12	7.9	22.4	13.1	13.3
	15	10.6	28.5	16.9	17.2
J7	2	1.8	1.8	2.0	2.0
	3	2.6	2.8	3.1	3.1
	6	5.7	8.2	5.9	6.0
	9	7.2	13.1	8.9	9.2
	12	9.4	15.8	11.9	12.3
	15	7.7	85.8	14.9	15.1
J8	4	1.8	2.7	2.3	2.5
	6	2.2	7.5	3.7	3.8
	8	2.8	11.9	5.0	5.1
	12	2.7	111.5	7.6	7.8
	16	1.6	473.3	10.2	10.6
	18	3.8	118.3	11.5	11.7

5.4 The Performance Comparison Between the Different Configurations of the FVMS

As described in the earlier section of the chapter there are mainly two types of variations in the different configurations of the FVMS which were experimented: the variation in the number of SVMs providing a particular type of service and the variation in the number of users concurrently served by a SVM at a time. In the latter case it is a non-multiplexed SVM when only one user is served at a time and a multiplexed SVM when more than one user can be served at a time. In this section the performance of the FVMS in these different configurations are examined.

5.4.1 The Performance of the FVMS with Different Number of SVMs

Table 5.11 displays the percentage of improvement in the response times obtained from the several configurations of FVMS with different number of SVMs, namely configurations C1 and C3, over configuration C2 in the experiments involving job types J1, J2, ..., J7. The users were all doing the same job as specified in the 'job' column. The 'user' column gives the number of users on the system. A negative percentage indicates a longer response time.

Table 5.11 shows that configurations C1 and C3 cannot give better response times than configuration C2. Often C1 and C3 provide slower responses than C2 and C3 offers quicker responses than C1.

There are two dominating components in the response time from a FVMS concerning the configurations experimented. One is the waiting time in the CCC for a SVM to execute the user job after the user command is input. The other is the job execution time. The main factor that affects the job execution time is the number of SVMs that have heavy resource demands on the real machine. As the CCC and the CI/IO

SVM are not using the system resources heavily, the main competitors for the system resources are the SVMs executing user jobs. When this type of SVMs is increased, the job execution time lengthens. At the same time the waiting time for a SVM shortens for there are more SVMs to do user jobs. For example, in group J2 of the experiments with 6 users on the system, in configuration C1 the waiting time was 392.9 seconds and the execution time was 76.6 seconds. The waiting time decreased to 257.3 seconds and the execution time increased to 132.5 seconds in configuration C2. The more substantial decrease in the waiting time than the increase in the execution time resulted in the quicker response time in configuration C2. In configuration C3 the waiting time was 203.4 seconds while the execution time was 221.8 seconds. This indicated that adding one more SVM from C2 to C3 caused greater increase in the execution time than the decrease in the waiting time.

Table 5.11 Percentage of improvement in the response time from configurations C1 and C3 of FVMS with different number of SVMs over configuration C2 in experiments involving job types J1 to J7.

Job	User	C1	C3
J1	9	-6.4%	-2.4%
J2	3	0.8%	
	6	-20.8%	-11.1%
	9	-22.1%	-11.9%
	15		-10.2%
J3	6	-26.4%	-9.8%
J4	9	0.5%	-7.4%
J5	3	-20.0%	
	6	-29.1%	-0.9%
J6	9	-1.5%	0.4%
	12		-0.7%
J7	6	-7.4%	-1.1%

The above comparisons tell one thing about the FVMS: there is an optimal configuration for a particular workload and it is not the more SVMs there are the better.

Table 5.12 Normalised average response time from FVMS. The J8(short/long) columns give the number of the type of jobs executed in the measurement period.

configuration	J8(short)	J8(long)	response
C2L1	131	23	6.7
C1L1	152	46	7.4
C1L2	127	74	9.7

However when there are requests for different types of software services, it would be very difficult to find the optimal balance of the number of each type of SVMs so that the whole system would provide a fair and efficient service for all users. For example in the group J8 experiments, it appears according to Table 5.12 that configuration C2L1 provided shorter average response times than C1L1 which in turn was better than C1L2. But the same table also lists the number of the short and long jobs which were executed by the two types of SVMs during the measurement periods of the experiments. It is clear that C2L1 produced the better performance only because more short jobs(compared with the number of the long ones) were processed than the long ones. Naturally the ratio of SVMs processing various types of services should correspond to the ratio of the demands for these services but this ratio is constantly changing and accurate predictions may not be easy.

5.4.2 The Performance of the Multiplexed SVMs

There are multiplexed non-interactive SVMs like the compiler or loader SVMs and multiplexed interactive SVMs like the command interpreter SVMs in the experiments. First the performance of the multiplexed non-interactive SVMs is compared with that of the non-multiplexed SVMs offering the same service. Then the performance of the multiplexed interactive SVMs is examined.

5.4.2.1 Comparison of Multiplexed (Non-Interactive) and Non-Multiplexed SVMs

The way multiplexed non-interactive SVMs work has been described in the last chapter. To summarise, there are mainly two points: one is a multiplexed SVM will serve up to a fixed number of users concurrently; another is such a SVM only switches to a different user when it sends the current user to another SVM for some input/output operations.

The performance of non-interactive multiplexed SVMs as compared with non-multiplexed SVMs is displayed in Table 5.13. The table lists the job execution times of the multiplexed SVMs in configurations M3, M2 and M2IO2 as multiples of the job execution time of the non-multiplexed SVMs in configuration C2. Configurations C2, M3, M2 and M2IO2 all have the same number of SVMs executing user commands. To obtain equivalent performance, job execution time on M3(multiple of that on C2) should be close to three as three users were served concurrently in the SVMs. Similarly equivalent performance requires the job execution time on M2 and M2IO2 to be close to two. Any figures greater than these would indicate less efficiency for the multiplexed SVMs. And figures less than three for M3(or two for M2 and M2IO2) indicate superiority of multiplexed SVMs over non-multiplexed ones.

Table 5.13 The job execution times of the multiplexed SVMs expressed as multiples of the job execution time of the non-multiplexed SVMs in configuration C2.

job type	no. of users	job execution time		
		M3	M2	M2IO2
J 1	9	7.4	2.8	3.6
J 4	9	5.1	2.2	

As can be seen from Table 5.13 in M3 the SVMs were a lot less efficient than those in C2. M2 and M2IO2 performed comparatively better than M3 but were still behind C2. One reason for this is because in the experiments the multiplexed SVMs did not do I/O operations but had an I/O SVM to serve its users' requests, the input and output data had to be passed between the I/O SVM and the SVM it served. The inter-SVM communication to transfer blocks of data consumed too much extra system resources which would otherwise be shared among the SVMs to serve their users. This is in spite of the direct data communication route between the two SVMs concerned to bypass the CCC. Another reason is multiplexed SVMs had to switch the work space when switching between users. When this switching was done frequently the number of 'page faults' to be dealt with by the paging subsystem would greatly increase, increasing the system overhead to support the virtual machines. However there is no reason to suggest that switching users within a virtual machine would increase more overhead than switching virtual machines as the IVMS would have to do.

So the experimental results seem to suggest that multiplexing users in SVMs is not a way to reduce the number of users waiting for a service. This is true in the configurations experimented. However in a network of real machines where the inter machine communication does not use the servers' resources to transmit messages, multiplexing is far from doomed to be ineffective.

5.4.2.2 Multiplexed Interactive SVMs

The way a multiplexed interactive SVM responds to a terminal input is by polling a flag in each of the UCBs(User Control Blocks) in the Event Queue, as described in the last chapter. Table 5.14(a) to (e) present the time taken in seconds for the CI/IO SVM to respond to a terminal input. It is the time as denoted by W_{LCI} in section 5.1.4 "The collection of Data".

Table 5.14(a) The response time of interactive multiplexed SVM. Configuration C1.

no. of users	1	2	3	6	9
J1	0.4		0.1		0.1
J2	0.2		0.1	0.4	0.3
J3	0.2	0.1	0.3	0.4	
J4	0.0		0.1		0.1
J5	0.0		0.1	0.1	
J6	0.0		0.2		0.2
J7	0.0	0.0	0.1	0.1	

Table 5.14 shows that the interactive multiplexed CI/IO SVM performed satisfactorily in most configurations. In configuration C1 the response time of the CI/IO was less than half of a second in every experiment. In configurations C2, C1L1, C2L1 and C1L2, with the exceptions of group J4 and two readings in group J1 involving 15 and 18 users on the system, CI/IO was able to respond within two seconds. In every experiment with various configurations and workloads, except the M3 configuration with 9 users doing job J4, the response time was less than 6.5 seconds. The response time of 26 seconds on configuration M3 processing J4 indicates that when there were multiplexed non-interactive SVMs processing CPU-intensive jobs the performance of multiplexed interactive SVMs would be substantially affected and could not operate efficiently.

Table 5.14(b) The response time of interactive multiplexed SVM. Configuration C2.

no. of users	3	6	9	12	15	18
J1		0.2	0.2	0.3	4.3	5.6
J2	1.1	0.6	0.7	0.7	0.6	0.6
J3		0.2	0.6	0.6	0.6	
J4		2.7	0.1	4.3	2.5	
J5	0.2	0.1	0.2	0.1	0.2	0.2
J6		0.4	0.4	1.0	0.7	
J7		0.1	0.2	0.1	0.1	

Table 5.14(c) The response time of interactive multiplexed SVM. Configuration C3.

no. of users	6	9	12	15
J1		0.4		
J2				6.1
J3	5.7	3.9		
J4	4.3	6.2		
J5	0.5			
J6		2.4	2.2	
J7	0.5			

Table 5.14(d) The response time of interactive multiplexed SVM. Configuration C1L1.

no. of users	2	4	6	8	12	16	18
J8(short)	0.2	0.2	0.2	0.2	0.2	0.3	0.2
J8(long)	1.0	0.2	0.2	0.3	0.2	0.2	0.2

Table 5.14(e) The response time of interactive multiplexed SVM in the remaining configurations.

configuration	job	no. of users	response time
C2L1	J8(short)	12	1.0
C2L1	J8(long)	12	1.0
C1L2	J8(short)	12	0.9
C1L2	J8(long)	12	1.7
M3	J1	9	5.6
M2	J1	9	4.4
M2IO2	J1	9	2.0
M3	J4	9	26.0
M2	J4	9	5.8

The figures in Table 5.14 were collected from the CI/IO SVM in the experiments. They were indications of how a multiplexed editor SVM would perform in the same configuration under the same workload, because both CI/IO SVMs and editor/IO SVMs would follow the same work pattern: a little processing following each terminal input. The encouraging figures in Table 5.14 suggest that a number of users

can be multiplexed in one SVM efficiently and this would enable the FVMS to work with substantially reduced number of virtual machines.

Chapter Six

How the Model Relates to a Network of Real Machines

In this chapter the usefulness of the FVMS model towards the study of a network of real machines operating also under the principle of functional partitioning of the application services is discussed. First such a system called Functional Processor System(FPS) is introduced. Then the suitability of the SVM submodel as a model for a real service machine in a FPS is assessed. Finally the communication subsystem of the FVMS model is examined.

6.1 Introduction

This section introduces the FPS and the question of using the FVMS model as a tool to study it is raised.

6.1.1 The Functional Processor System (FPS)

With the growing availability of microcomputers and computer workstations, a network of such machines, usually in the same building, has gained acceptance as an efficient way to provide computer service to a user community like a department or an institution. At present such a network of real machines are usually configured as a group of identical processors on which user processes run, plus a limited number of machines dedicated to special services for the whole system like the file server or the printer server[10], [11], [12], [14], [35]. It is possible to configure the system in

the way of the FVMS in which each machine provides a subset of the system services. Such a system will be called a Functional Processor System(FPS) in this chapter to distinguish it from the FVMS. The real machines which provide services in the FPS will be called Service Machines(SMs) as opposed to SVMs in the FVMS.

As mentioned in the Introduction chapter, the FPS has the advantage of saving the cost of loading service programs like compilers or editors into the memory because they are resident in the SM's memory. When loading programs involves reading from disk drives, this saving will not be negligible. If files are provided from a file server, the FPS approach saves the moving of the service program from the file server into where it is required through the communication network, not to mention the reading from the file storage into the file server's memory if it is not already in its memory or cache.

A potential advantage of the FPS lies in the advance of the VLSI technology. Even now it is not unthinkable to have a system with the number of processors almost always higher than the number of users. And if the processors in the FPS have specialised capability like different instruction set on different machines to carry out certain services more efficiently than others, and services are only offered on the processors which are designed to support them, the service provided by the whole system will be more efficient than what can be achieved on a network of identical processors.

6.1.2 The FVM Model for the FPS?

In chapter 3 and 4, the two components of the FVMS model, namely the communication subsystem and the SVM submodel, are described respectively. The communication subsystem is not a simulation but a real implementation to support the inter-SVM communication, based on the inter-virtual machine communication facility

available on the VM/SP system. The SVM submodel is a simulation model of the SVMs in the FVMS. Undoubtly a model for the FPS will also consist of these two subsystems, one to provide the inter-SM communications and the other to simulate the operations of the SMs.

Because of the nature of a virtual machine, it being the simulation of a complete real computer, it seems possible to use the FVMS model run on a group of virtual machines to simulate the FPS. Another way the FVMS model might be used is to port it onto a network of real machines, maybe after some modifications. In the following sections the FVMS model is examined in the context of a network of real machines. The aim is to demonstrate how suitable the model is in the two possible ways of being a tool for studying the FPS and what the modifications to the model are necessary to make it a suitable model.

6.2 The SVM Submodel to Simulate a Service Machine(SM)

A group of logically isolated virtual machines may be viewed as a model of a group of loosely-coupled real machines. Each virtual machine has its own processor and memory which are not accessed by other machines, just like its real counterpart. When these real machines are linked together by a communication network and cooperate like the SVMs do in the FVMS, they will interact with each other in the same way as the SVMs do and have the same communication requirements as those defined in chapter 3 for the FVMS. The processing of the user services in these SMs of the FPS is the same as in the SVMs of the FVMS. The operating system of the SM will manage the resources of the machine and the user service will run on top of it. To summarise, the SMs consume the same resources during their operations as the SVMs do, the SMs perform the same functions (services) in the FPS as the SVMs do in the FVMS and each SM has the same functional requirements for communication towards the rest of the

system as each SVM does.

The difference between a SM and a SVM is the real time required to carry out a user service. In the FVMS the SVMs share the same real machine resources while in the FPS the SMs do not share anything when processing their jobs. Because the SVMs are time-sharing the real machine, the partition of the CPU and other resources slows down the individual service times. For example, assuming the processing speed of a SM is the same as the real machine time-shared by the SVMs and a certain compilation needs x time units of service. On a SM it will take x time units to complete the compilation but because of time-sharing it takes a SVM $5x$ time units. So the SVM is not modelling the service of this particular job by the SM. However there is no reason why this service by the SVM cannot be viewed as a simulation of the SM doing a service which actually requires $5x$ time units. As far as the communications relating to the job scheduling is concerned the interaction between this SVM and the rest of the system in content and in frequency is the same as that of the SM doing a $5x$ time unit job. It can also be viewed as a simulation of the SM doing a service of $2.5x$ time units of the real machine time-shared by the SVMs if the speed of the SM is half of the real machine for FVMS.

With the control of the simulation parameters, experiments with the SVM submodel may be used to simulate SMs of various speed. For example, it is to find out whether two SMs with speed v will give better performance than four SMs with speed half of it. Experiments may be done with two SVMs providing throughput appropriate to the speed v first and then with four SVMs with throughput half of the other case. The performance figures obtained may be used to draw conclusion about the real machine situation.

Therefore the SVM submodel running on a virtual machine is a perfectly valid model of a SM for a FPS. The only requirement is the careful adjustment of the resource consumptions of jobs which are the simulation parameters of the FVMS

model.

If the SVM submodel is ported onto a real machine in a network of them to become part of the FPS model, even the adjustment of the resource consumption mentioned above is not necessary. The modifications needed would be the communication requests made to the IPCF of the VM/SP system. They would be replaced by the corresponding local communication requests.

6.3 The Communication Subsystem

It has been pointed out in the last section that the communication requirements of the SMs in the FPS are the same as those defined in chapter 3 for SVMs of the FVMS. However, there is the difference in the communication medium between the FVMS and the FPS. In the FPS a physical communication network with one of a number of possible interconnection topologies links the machines together. Messages are transmitted across the network from machine to machine. In the FVMS inter-virtual machine communication is done through data transfers within the same real memory which is shared by multiple virtual memories of the communicating SVMs.

This difference in the communication media leads obviously to the difference in the message transfer time between the communication partners and possibly the way inter-SM communication should be handled differently. In this section the need of having a Central Communication Controller(CCC) in a FPS is first established and the communication networks in multiprocessor systems with no central or shared memory are examined, then the suitability of the communication subsystem of the FVMS model as a model for those systems is discussed.

6.3.1 A CCC for the FPS

In chapter 3 various ways of satisfying the communication requirements of the SVMs utilising the Inter Program Communication Facility(IPCF) available on the departmental VM/SP system were discussed. Several designs were rejected and a Central Communication Controller (the CCC) implemented on a virtual machine was chosen. The functions of this CCC are to provide a storage area where messages queue for a receiver and to act as a decision centre where the mapping of a sender to a receiver is made for the one to one-among-many communications. The questions are whether these two functions should still be played by a CCC and whether it should still be running on a single machine in a FPS. The first question could be answered by reconsidering the schemes rejected in chapter 3, and the second by taking the interconnection topology of the network into consideration.

The first was the Round Robin way of communication among the SVMs. When a SVM finishes the service of a user, it sends the user to one in a group of the SVMs offering the next required service in a round robin manner or in a random manner as a variation of the scheme(Fig 3.6). The disadvantages of this scheme are the possibility of overloading some SMs and underloading others when the work load is neither very light nor very heavy, and the inflexibility to dynamic reconfiguration(this would not be a factor if the SMs have specialised hardware and therefore there is no dynamic reconfiguration). These disadvantages make this type of schemes undesirable for the FPS, just as in the FVMS.

The second scheme was using Service Controllers(SCs). Each SC runs on a dedicated virtual machine and is responsible for mapping users to SVMs for a particular service(Fig 3.7). After a service a SVM sends its user to the appropriate SC for finding the next server and itself asks the SC(s) to send the next user to server. The high ratio of the number of SCs and the SMs would not degrade the performance of

the SMs as in the case of a FVMS because the overhead of supporting the SCs does not affect the SMs. However the difficulty with those SMs offering multiple services and the difficulty with reconfiguration(if there is reconfiguration) would still be the reasons of rejecting this scheme for the FPS.

The last scheme rejected was the usage of Communication Controllers(CCs). A CC controls all the incoming and outgoing messages of the SVMs offering a particular service(Fig 3.9). It would not be suitable for the FPS for the same reasons as the case of the SCs.

As it was said in chapter 3, under the following circumstances the SC and CC scheme may be a suitable structure for the inter-SM communications: (1) There are a large number of SMs offering each service; (2) SMs do not offer multiple services, or SMs only offer clearly partitioned group of services where there is no overlap of services between groups; (3) and the SMs are specialised service machines beginning at the hardware level so reconfiguration is out of the question. However, considering the fact that the processing at the CCC is so short compared with that of a service at a SM, the CCC scheme will be a more cost-effective way to provide the communications among the SMs in a FPS. This is except the number of SMs has reached such a level that the collective rate of incoming messages is higher than the rate the CCC can process them.

A distributed scheme for communication on a network of machines was proposed in the research[26]. It uses a global buffer which is distributed. However the implementation of that scheme would involve several message transfers for each communication and thus incur heavy communication cost to be avoided for a FPS if possible.

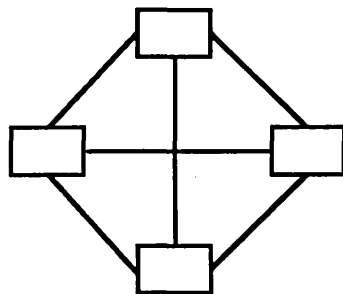
The disadvantage in the vulnerability to a failure in the CCC software or in the CCC machine itself is the same for the FPS as for the FVMS. A backup of the CCC machine which has the same system state information and is constantly monitoring the

working CCC could provide some safety.

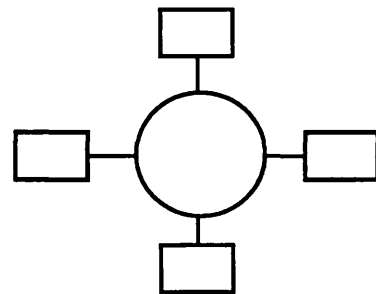
So the CCC will still be responsible for the inter-SM communications and the job distributions in the FPS. Also there are a number of SMs with the CCC machine communicating with every SM in the FPS. There are also some direct data transfers between a SM and its assigned I/O server(another SM) if the former does not have I/O capability as described for the SVMs in chapter 4. The interconnection of these machines will be considered next.

6.3.2 Multiprocessor Interconnection Schemes

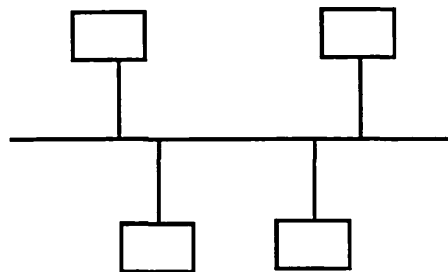
As outlined in [41] and [42] the interconnection schemes in multi-microprocessor(they may be computer workstations) systems can be classified into direct transfer strategy and indirect transfer strategy.



(a) complete interconnection.

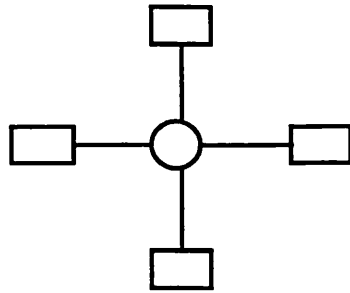


(b) a ring connection.

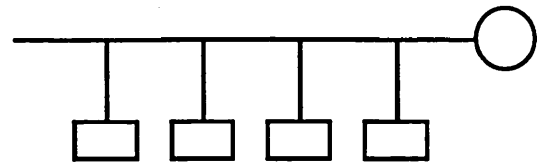


(c) a bus.

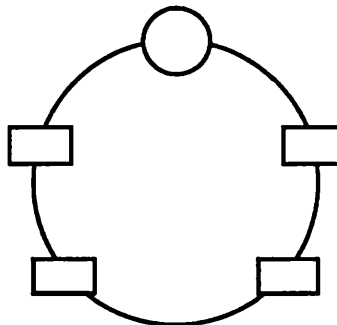
Fig 6.1 Direct interconnection for communication.



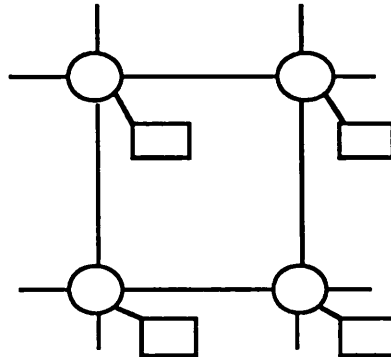
(a) Star configuration with a central switch.



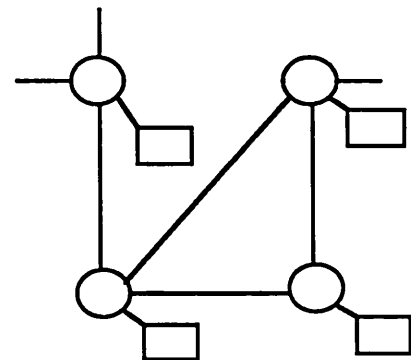
(b) Shared bus with controller.



(c) Loop with central switch.



(d) A regular topology.



(e) Irregular network (Variable Topology Multicomputer).

Fig 6.2 Indirect interconnection for communication. Rectangles represent microcomputers or workstations where user jobs are processed. Circles are the switching devices which are microcomputers dedicated to routing messages.

The direct transfer strategy is characterised by the absence of intermediary stage(s) whereby routing takes place. Under this strategy a sending machine specifies

the location of the receiver in the message and selects the correct path. Several examples of interconnection schemes under this strategy are given in Fig 6.1. The Cambridge ring[14] is an implementation of Fig 6.1(b) and many sites use Ethernet to link their machines together like Fig 6.1(c).

In indirect transfer strategy, intermediary stage(s) exists which decides the mapping of the sending and the receiving machines. A one-stage device may exist between the sender and the receiver to provide centralised routing as in Fig 6.2(a), (b) and (c) or a number of switching units may be responsible for decentralised routing, as in Fig 6.2(d) and (e).

6.3.3 The CCC In These Interconnection Topologies

Among all these possible interconnection topologies some are suitable for the FPS and others are not.

6.3.3.1 The Direct Transfer Strategy

It is possible for a FPS to use any one of the three interconnection schemes depicted in Fig 6.1 where the position of the CCC machine in the network is no different from the SMs. This will allow the CCC to communicate with every SM in the system and any SM to communicate directly with another SM, for example for an I/O operation, once the CCC has designated it. In this case the FVMS model may be ported to the network of real machines and become a model for the FPS. Both the internal working of the CCC and the SVM(SM) submodel remain the same. The only change needed would be the requests to IPCF to send/receive messages provided that asynchronous communication is available.

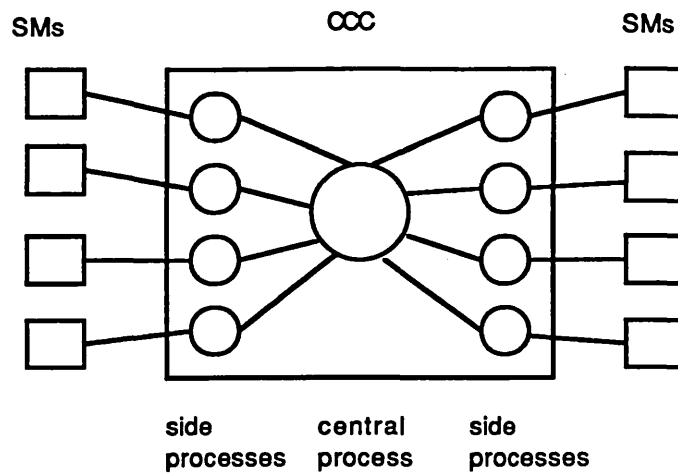


Fig 6.3 The CCC structured into several concurrent processes to communicate synchronously with the SMs.

A variation in the interconnection given in Fig 6.1(a) can be found in the Transputer[43] interconnection scheme. Messages are directly transferred from source to destination but the interconnection is not complete in that a transputer is only connected to four other transputers(4 physical channels). In addition, senders and receivers can only communicate synchronously. To solve the problem caused by the synchronous communication, the CCC could be divided into a number of processes. One process, called central process, is to deal with the mapping of the users to the servers that includes the queueing of the users and server identifications. In addition there is one process for each SM in the system, named as a side process, dedicated to the synchronized sending/receiving of messages to/from the SM and relaying them through messages to the central process to be dealt with. Fig 6.3 shows the relationship within the processes of the CCC and their relationship with the SMs. Assuming SMs are not multiplexed this configuration works because the CCC does not send the next user to a SM before the current user has been sent back. As a result the central process does not send the next user to the side process responsible for the communication with the SM before the current user comes back from this process. Even with multiplexed SMs this scheme would still be adequate as the chances of

communication traffic jam would not be high and frequent. Even when a temporary traffic jam does develop it would be for a very short time. The central process may use the alternative construct in the OCCAM language to receive messages from one of the side processes. So the central process would not be blocked because a certain SM or the side process dealing with it is not ready for communication.

The problem caused by the limited number of connections between the transputers could be solved by structuring the side processes into a hierarchy in the form of a tree, like that shown in Fig 6.4. The root of the tree is the central process. Each other node in the tree is a side process running in a separate machine. To fully utilise the machine's processing power, the machine will also run services, that is, the side processes are distributed to the SMs. To reduce delay in the communication procedure, the side processes in the SMs will be high priority processes. This means messages have to be relayed through the different hierarchy levels of the tree and the communication cost would be greater than the systems where this is not necessary. However, in systems with limited number of connections between the processing units, relaying is unavoidable.

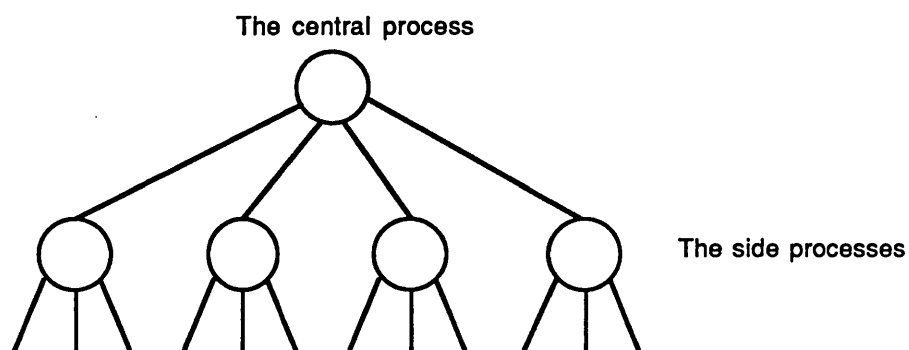


Fig 6.4 The CCC restructured into a tree of processes with each node running in a different processing unit.

Among the three interconnection topologies, the ring structure and the bus structure have the limitation that the messages might saturate them. For the complete

interconnection scheme each communication path is dedicated and the message transfer time is not affected by the number of connections a machine has with the rest of the system. In fact some kind of irregular interconnection among the machines(SM or CCC) of the FPS would be enough because not every SM communicates with every other SM.

When the FVMS model is run on the virtual machines of the VM/SP system, the CCC with the IPCF could simulate the inter-SM communication of the FPS provided that asynchronous communication is available on the FPS being modelled and that the inter-machine message transfer time of the FPS is known. Because cross machine message transfer almost certainly takes longer than the message transfer within a real memory, the inter-virtual machine message transfer of the FVMS model could be lengthened in a controlled manner by adding some code in the receiver's receiving procedure in its communication interface with the IPCF.

The communication subsystem of the FVMS model which includes the CCC and the IPCF is capable of simulating those FPSs with only synchronized communications, as IPCF provides synchronous sending of messages. This would block the sender if the receiver is not immediately ready to receive the message, thus reduce the parallelism among the machines. The obstacle is the operating system of the virtual machines on VM/SP, namely CMS, which does not support executions of concurrent processes within the virtual machine. Therefore the proposed structural change of making sending/receiving procedures a separate process cannot be simulated on the VM/SP's virtual machines.

6.3.3.2 The Indirect Transfer Strategy

In the interconnections with indirect transfer strategy, switching device(s) exist between the communicating processors to map the messages to their receivers.

In a FPS, this job is done by the CCC. It would be natural to use the interconnection schemes with a centralised switching control like those shown in Fig 6.2(a), (b) and (c) if the intelligence of the CCC can be built into this central switching device and if there is no direct communication between the SMs. The second condition is met when all the SMs have I/O capability so there is no I/O SM to serve another SM's data input/output and no data to be transferred between them. The central switch normally contains a microcomputer with its local memory. Such a device is capable of carrying out the CCC's job. Thus the FVMS model may be run on a group of machines with such interconnections. The only changes needed would again be the communication requests to the IPCF to be replaced by the local communication calls to send/receive messages. If direct communication is required between the SMs these interconnection topologies would not be suitable for the FPS.

Assuming no direct communication is required, the FVMS model running on the virtual machines could simulate the FPS with a centralised switch to control all the communications among the SMs. The virtual machine running the CCC simulates the central switch. Again the communication time may be adjusted by adding some code in the receiver's communication interface with the length of the execution of these added code controlled.

Fig 6.2(d) and (e) are the interconnection topologies with decentralised switching devices to route the messages from the senders to the receivers. In the earlier sections it has been decided that the mapping of the messages to the receivers has to be centralised in the CCC and any decentralisation of the job in the form of SCs or CCs has been rejected except under special circumstances. If the CCC's job is built into one of the switching devices while other switching devices simply relay the messages, there is the waste of the intelligence of a lot of switching devices and the communication overhead as messages are passed from switching device to switching device before reaching their destinations. So the FPS would be less cost-effective on

systems with such interconnection topologies. The communication overhead would be much less in the irregular interconnection than in the regular one. Nevertheless there is still the waste of switching devices.

Chapter Seven

Conclusions

In this research the ways of operating a Functional Processor System on a group of logically isolated virtual machines were investigated. A FVMS model consisting of a real communication subsystem and a SVM submodel was built.

The communication subsystem in the model is a centralised intelligent agent which matches the messages(application service requests) to their servers. The CCC maximizes the concurrency/parallelism among the service machines and dynamically reconfigures the system according to the service demands. The communication scheme incurs lower communication costs than a number of alternatives by minimizing the number of messages to be transmitted in the process of delivering a service request from one service machine to another, under the constraints load balancing and the ease of reconfiguration.

The service machine submodel provides the flexibility of experimenting various ways of partitioning the application services among the service machines. To optimize the FVMS, a user command is a service unit in the submodel so that the inter-service machine communications are reduced to a minimum, with the exception that the I/O service may be done by another service machine. However, the service routines in the submodel are in a large number of small procedures so that only minor changes are needed to split the present service units into smaller ones for concurrency or parallelism of the services of a higher degree. The submodel also reduces the inter-service machine communication by batching the I/O service requests and buffering the data transfers between the service machines. By using the resource consumption specifications as simulation parameters allows the submodel to simulate

application services with different resource requirements easily. It even allows the same code to simulate different application services like different compilers or a compiler service and a loader service, just by varying the simulation parameters.

The service machine submodel also offers the means of comparing the FVMS and the CP/CMS performance by running in two modes, the FVMS mode and the IVMS mode. In FVMS mode, the service machine operates as a component of the FVMS. In IVMS mode, it provides full application services, simulating a CMS virtual machine, or a real stand-alone single-user computer.

The study of the FVMS model on the VM/SP system demonstrated:

- (1) The communication cost to users in having their commands executed on a FVMS, in terms of the time spent in message transfers, is only a minute percentage of the response time and has less effect on it than the speed of the service machines does.
- (2) The workload of the system (what the SVMs are doing and how many of them there are) is the single factor which affects the communication times on the VM/SP system. The communication times are most sensitive to the total memory requirements of the SVMs.
- (3) The centralised control of message distributions works well in the FVMS. The CCC is not a bottleneck in the FVMS. Though the number of SVMs used in the experiments was small, with the message frequency seen in the experiments, it is believed that the CCC would be able to operate smoothly even the number of SVMs triples the number experimented.
- (4) With the exception of CPU intensive jobs, the FVMS is able to provide better or equivalent response times to the users as the CP/CMS system when the machine is heavily loaded, though the improvement, when there is, is not enormous. When under light workload, the FVMS is not very much worse than the CP/CMS. The FVMS is also fairer to the users than the CP/CMS system.
- (5) Multiplexing in a SVM with I/O done in another SVM is not an efficient way to

provide concurrent services to the users on VM/SP.

- (6) Multiplexing several highly interactive users of the same service in one SVM is an effective and efficient way to facilitate concurrent services of this type.

The model built also provides a tool in the study of a Functional Processor System on a network of loosely-coupled real homogeneous or heterogeneous machines. The SVM submodel is also a model for a real service machine in the network. With the control of the simulation parameters, the SVM submodel is able to simulate the services by machines with different speeds. This is much easier than experimenting with a real network of machines. Minor changes to the submodel is what is required to make the model simulate services smaller than a command.

The FVMS model running on VM/SP can simulate a network of real machines with a direct transfer strategy. With no limitation on the number of connections, the CCC running in one real machine matches the service requests to service machines. Some code added to the receiving procedures of the CCC and the SVMs with controlled length of execution would compensate the difference of message transmission time between the IPCF and the real network. With the limitation on the number of connections, the CCC needs to be restructured as described in the last chapter.

The FVMS model can also simulate a FPS with indirect transfer strategy where a central switch is used. The FPS does not require direct SM to SM communication and the CCC runs in the central switch.

Another way the FVMS model may be used is to port it onto a network of real machines, all identical or otherwise. The IPCF requests will be replaced by the local communication calls. If only synchronous communication is available, the sending/receiving routines of the CCC and the SM should be made a separate process running on the same machine.

Research in the field of the distributed applications and functional processors

is just beginning. A lot of questions still need answers. What is the behaviour of a FPS on a network of real machines? How small and in what way can the applications be divided into when the communication costs and the parallelism balance each other? Is there a way to construct a distributed communication subsystem which survives the machine crash while keeping the communication costs down?

A final point to be made here: Virtual Machine, though an 'old' concept, is still a valuable experimental tool. It provided 'the machine' for the study of the operating systems in the 'mainframe' computer age. Now, when distributed and parallel computing is considered the way to get more power out of the computers, it is also useful in the study of the distributed systems.

Appendix A

The Machine

The Physical Machine

- - a 4331 model group 2 processor, with a 2 Mbyte processor store;
- - 3 channels: one byte multiplexor and two block multiplexors;
- - 1140 Mbytes of storage on two 3370 disk drives (with 4 actuators). These are fixed disks with a transfer rate of 1.86 Mbyte/sec;
- - micro-coded support for VM:ECPS:VM (Extended Control Program Support for VM) consisting of the (1) Virtual Machine Assist I; (2) Virtual Machine Assist II; (3) CP Assist; (4) Virtual Internal Timer Assist;
- - VM/370 is the system running on the machine. CP supports the concurrent operations of users' virtual machines by multiprogramming them on the real machine.

The Virtual Machine

The virtual machine provided for each user of the system by CP are architecturally complete. They have

- - a store of 1 Mbytes (virtual, can be extended up to 16 Mbytes);
- - a processor (by time slicing the real machine);
- - unit record devices, a reader, a punch, a printer (by spooling);
- - disks (by dividing real disks into minidisks);
- - a console (the user's terminal);
- - CMS is the operating system in each virtual machine.

Appendix B

The Analysis of Variance on the Communication Times

Using a method called the analysis of variance[40], this appendix is to provide evidence to support the statements made in chapter 5 about the factors which affect the communication costs of the FVMS model running on the IBM VM/SP machine.

In the following presentations of the results of the calculations involved in the analysis of variance, Q_b is the sum of squares between treatments(configurations or types of job) and Q_w is the sum of squares within treatments. $F_{a,b}$ is the test statistics with a, b degrees of freedom.

B.1 The Communication Times and the Different Configurations of the FVMS

As said in chapter 5, if the figures from the first column of Table 5.3(a) are not included, the communication times from experiments with different number of users may be considered as within a relatively homogeneous block of experimental material when the users in the experiments were running the same jobs in a configuration to be compared with other configuration(s).

Calulations were done to check whether there was evidence to support the hypothesis that significant difference existed between communication times from two different configurations of the FVMS. Therefore the null hypothesis was there was no difference .

First the configurations C1 and C2 are considered. Then configurations C2 and C3 are checked. After this configurations C2 and C1L1 are examined. Configuration C3

has one more SVM to execute user commands than configuration C2 which in turn has one more than configuration C1. Configuration C1L1 is compared with C2L1 before it is compared with C1L2. Configuration C2 and C1L1 have the same number of SVMs but the two SVMs executing user jobs in C2 are doing the same job requests while those in C1L1 are doing different job requests. Configuration C2L1 has one more SVM to serve the J8(short) job requests than C1L1 and configuration C1L2 has one more SVM to serve the J8(long) job requests than C1L1.

B.1.1 Between Configurations C1 and C2

(1) J2 is the job executed by the system.

calculation result: shown in Table B.1.

conclusion: significant difference between the communication times of these configurations as $F > F_{0.05}$ which is 7.71.

(2) J5 is the job executed by the system.

calculation result: shown in Table B.2.

conclusion: no evidence to suggest that significant difference existed between the two configurations when J5 was the job on the system because $F < F_{0.05}$ (18.51).

B.1.2 Between Configurations C2 and C3.

(1) J2 was the job on the system.

calculation result: shown in Table B.3.

conclusion: significant difference between the communication times of these

configurations as $F > F_{0.05}$ which is 7.71.

(2) J6 is the job executed by the system.

calculation result: shown in Table B.4.

conclusion: significant difference between the communication times of these configurations as $F > F_{0.05}$ which is 18.51.

B.1.3 Between Configurations C2 and C1L1

(1) For users executing J1 in C2 and J8(short) in C1L1 (J1 and J8(short) are actually the same job).

calculation result: shown in Table B.5.

conclusion: no significant difference between the communication times of these configurations as $F < F_{0.05}$ which is 7.71.

(2) For users executing J3 in C2 and J8(long) in C1L1 (J3 and J8(long) are actually the same job).

calculation result: shown in Table B.6.

conclusion: no significant difference between the communication times of these configurations as $F < F_{0.05}$ which is 18.51.

B.1.4 Between Configurations C1L1 and C2L1

For the configurations concerned here there were 12 users on the system, half doing J8(short) and the other half doing J8(long).

calculation result: shown in Table B.7.

conclusion: no significant difference between the communication times of these configurations as $F < F_{0.05}$ which is 18.51.

B.1.5 Between Configurations C1L1 and C1L2

For the configurations concerned here there were 12 users on the system, half doing J8(short) and the other half doing J8(long).

calculation result: shown in Table B.8.

conclusion: significant difference between the communication times of these configurations as $F > F_{0.05}$ which is 18.51.

B.2 The Communication Times and the Types of Jobs on the System

This is to seek evidence that communication times differed when different jobs were processed on the FVMS. As in the last section the communication times listed in the first column of Table 5.3(a) and the first row of Table 5.4.

In the first subsection below when configuration C1 is considered, because of the availability of data the calculations are done in two groups of jobs with J2 in both groups. And because of this overlap between the groups, when no evidence is found to suggest a difference between the communication times in both groups it is interpreted as no evidence suggests a difference among all these job types on configuration C1.

In the subsection concerning configuration C2, first the data from all job types(J1 to J7) are included in the calculation. The result is that there is evidence showing differences among the communication times of these job types. Following this various groupings of the job types are done and two largest groupings are found to have similar communication times within the groups.

The last subsection is to see whether the communication times for different jobs (J8(short) and J8(long)) processed on the same system are different.

B.2.1 On Configuration C1

(1) Among the communication times for jobs J1, J2, J4 and J6.

calculation result: shown in Table B.9.

conclusion: no significant difference among the communication times when these jobs were executed on the system as $F < F_{0.05}$ which is 6.59.

(2) Among the communication times for jobs J2, J3, J5 and J7.

calculation result: shown in Table B.10.

conclusion: no significant difference among the communication times when these jobs were executed on the system as $F < F_{0.05}$ which is 6.59.

B.2.2 On Configuration C2

(1) Among the communication times for job types from J1 to J7.

calculation result: shown in Table B.11.

conclusion: significant difference among the communication times when these jobs were executed on the system as $F > F_{0.05}$ which is 2.57.

(2) Among the communication times for job types J, J2, J4, J5 and J7.

The communication times taken from those listed in the rows labeled as J1, J2, J4, J5, and J7 in Table B.11:

calculation result: shown in Table B.12.

conclusion: no significant difference among the communication times when these jobs were executed on the system as $F < F_{0.05}$ which is 3.06.

(3) Among the communication times for job types J3 and J6.

The communication times taken from those listed in the rows labeled as J3 and J6 in Table B.11:

calculation result: shown in Table B.13.

conclusion: no significant difference among the communication times when these jobs were executed on the system as $F < F_{0.05}$ which is 5.99.

B.2.3 On Configuration C1L1

The communication times taken from those listed in Table 5.4 excluding the first row, the result of the calculation for job types J8(short) and J8(long):

calculation result: shown in Table B.14.

conclusion: no significant difference among the communication times when these jobs were executed on the system as $F < F_{0.05}$ which is 4.96.

Table B.1

	configuration						
	C1			C2			
communication times	3.4	4.7	5.6	3.1	2.1	2.1	
mean	4.5667			2.4333			
overall mean				3.5000			
Q_b				6.8271			
Q_w				3.1130			
$F_{1,4}$				8.7724			

Table B.2

	configuration			
	C1		C2	
communication times	3.8	2.5	2.8	3.4
mean	3.15		3.10	
overall mean	3.125			
Q_b	0.0025			
Q_w	1.0250			
$F_{1,2}$	0.0049			

Table B.3

configuration						
communication times mean	C2			C3		
	2.1	2.1	2.2	4.8	4.8	6.4
	2.133			5.333		
overall mean	3.7333					
Q_b	15.3600					
Q_w	1.7148					
$F_{1,4}$	35.8290					

Table B.4

configuration				
	C2		C3	
communication times	1.8	1.6	4.3	5.3
mean	1.7		4.8	
overall mean			3.25	
Q_b			9.61	
Q_w			0.52	
$F_{1,2}$			36.96	

Table B.5

configuration						
communication times mean	C2			C1L1		
	2.9	3.0	2.5	2.7	2.4	2.5
	2.8			2.5333		
overall mean	2.6667					
Q_b	0.1068					
Q_w	0.1872					
$F_{1,4}$	2.2820					

Table B.6

configuration						
communication times mean	C2			C1L1		
	1.9	1.8		2.9	2.1	
	1.85			2.5		
overall mean	2.175					
Q_b	0.4224					
Q_w	0.325					
$F_{1,2}$	2.599					

Table B.7

configuration						
communication times mean	C1L1			C2L1		
	2.4	2.1		2.3	2.6	
	2.25			2.45		
overall mean	2.35					
Q_b	0.04					
Q_w	0.09					
$F_{1,2}$	0.889					

Table B.8

configuration				
communication times	C1L1		C1L2	
	2.4	2.1	3.8	3.4
	mean	2.25	3.6	
overall mean	2.925			
Q_b	1.8224			
Q_w	0.125			
$F_{1,2}$	29.158			

Table B.9

job type									
	J1		J2		J4		J6		
communication times	4.8	4.3	3.4	5.6	2.4	2.4	3.4	2.9	
mean	4.55		4.50		2.40		3.15		
overall mean			3.65						
Q_b			6.69						
Q_w			2.67						
$F_{3,4}$			3.341						

Table B.10

	job type							
	J2		J3		J5		J7	
communication times	3.4	4.7	3.3	1.9	3.8	2.5	3.2	3.1
mean	4.05		2.60		3.15		3.15	
overall mean			3.2375					
Q_b			2.1638					
Q_w			2.6750					
$F_{3,4}$			1.0785					

Table B.11

job type	communication times				mean
J1	2.9	3.1	3.0	3.3	3.075
J2	2.1	2.1	4.0	2.2	2.600
J3	1.9	1.8	1.8	1.7	1.800
J4	3.4	3.5	3.3	3.4	3.400
J5	3.4	2.5	3.5	3.5	3.225
J6	1.7	1.8	1.6	2.0	1.775
J7	2.4	2.4	2.3	2.6	2.425
overall mean	2.6143				
Q_b	10.4243				
Q_w	3.5900				
$F_{6,21}$	10.160				

Table B.12

overall mean	2.945
Q_b	2.767
Q_w	3.4825
$F_{4,15}$	
2.979	

Table B.13

overall mean	1.7875
Q_b	0.00125
Q_w	0.1075
$F_{1,6}$	0.070

Table B.14

	J8(short)	J8(long)
mean	2.5167	2.3500
overall mean	2.4334	
Q_b	0.0834	
Q_w	0.8223	
$F_{1,10}$	1.0146	

Appendix C

The Regression Analysis on the Response Times

The aim is to find out the relationship between the average response time and the number of users on the system in both the FVMS mode and the IVMS mode using regression analysis[40]. The technique of regression analysis is used to build the performance prediction equations so that the conclusions drawn from the results may be expressed with certain confidence level. For example, if two distinct equations are needed for the FVMS and the IVMS respectively, it may be stated with, say 95% confidence, that there is significant difference between the performance of the FVMS and that of the IVMS. Then it is with the same confidence when a statement is made that the performance of one system is better than the other.

In this appendix the first section presents the experimental results based on which the prediction equations are built. Then the calculation steps and results for each group of experiments are presented in the following section.

C.1 The Raw Data

Table C.1(a) The response times from the IVMS expressed as multiples of the average response time from a one-user IVMS(the normalised response times).

no. of user	Job Type						
	J1	J2	J3	J4	J5	J6	J7
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.5		1.9				1.8
3	2.8	2.8	3.3	2.8	2.1	3.0	2.7
6	5.3	6.1	6.2	6.3	4.3	6.2	6.7
9	8.4	10.2	8.7	9.3	7.0	9.8	8.7
12	11.9	13.1	11.9	12.5	9.1	13.3	13.0
15	15.6	16.2	18.2	15.6	11.5	17.9	15.2
18	22.3	21.6			17.8		

Table C.1(b) The response times from the FVMS expressed as multiples of the average response time from a one-user IVMS(the normalised response times).

no. of user	Job Type						
	J 1	J 2	J 3	J 4	J 5	J 6	J 7
1	1.1	1.0	1.0	1.0	1.0	1.0	1.0
2			1.9				2.0
3	3.2	3.3	3.0	3.3	2.5	3.2	3.1
6	6.5	6.3	5.4	6.8	4.9	6.4	6.0
9	10.0	9.3	8.3	10.6	7.9	9.7	9.0
12	14.2	13.0	11.1	13.9	10.0	13.2	12.1
15	15.4	15.6	15.3	17.7	12.7	17.0	15.0
18	19.6	18.7			15.9		

Table C.2 The response times from the IVMS and the FVMS expressed as multiples of the average response time from a two-user IVMS doing the same job(J8) (the normalised response times).

system	no. of user						
	2	4	6	8	12	16	18
IVMS	1.0	2.2	3.8	5.2	8.8	14.1	14.5
FVMS	0.9	2.3	3.6	4.9	7.4	10.0	11.1

C.2 Regression Model Building

The dependent variable in the regression model is naturally the (normalised) response time. The independent variables are the number of users and the mode of the system. An inspection of the data obtained from the experiments would lead to the hypothesis that a second-order prediction equation would provide a good fit to the data collected. Therefore if y represents the (normalised) response time, $E(y)$ is the expected mean of y , x_1 represents the number of users on the system and x_2 the mode of the model, that is, FVMS or IVMS, the following prediction equation is to be constructed by the method of least squares:

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 + b_3x_2 + b_4x_1x_2 + b_5x_1^2x_2 \quad (C.1)$$

$$x_2 = \begin{cases} 1 & \text{if IVMS is the system being used} \\ 0 & \text{if FVMS is the system being used.} \end{cases}$$

Therefore for the FVMS,

$$E(y) = b_0 + b_1x_1 + b_2x_1^2$$

and for the IVMS,

$$E(y) = (b_0 + b_3) + (b_1 + b_4)x_1 + (b_2 + b_5)x_1^2.$$

Because x_1 and x_1^2 are highly correlated: the coefficient of correlation r between the variables x_1 and x_1^2 listed in Table C.3 is 0.97, and because the likelihood of rounding errors in the regression coefficients $b_0, b_1, \dots$ is increased in the presence of multicollinearity, the values of x_1 is coded in the regression analysis to take the values

$$\frac{x_1 - \text{mean of } x_1}{\text{the standard deviation of } x_1}$$

When the prediction equation is fixed, the above expression is substituted back into it to express $E(y)$ in terms of x_1 , the number of users on the system.

Table C.3 The values of the number of users and their squares.

x_1	1	3	6	9	12	15	18
x_1^2	1	9	36	81	144	225	324

C.2.1 The Job Group J1

The complete second-order model is given by (C.1). Table C.4 gives the values

of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J1 were used to fit the model. The results of the calculations are shown in Table C.5.

Table C.4 The values of x_1 and their coded values.

x_1	1	3	6	9	12	15	18
coded x_1	-1.3015	-0.9819	-0.5023	-0.0228	0.4567	0.9362	1.4157

Table C.5 Results of the calculations for the complete model (J1 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	10.21442413	26.34661048		
x_1	6.78347349	25.29851874		
x_1^2	-0.25027302	-0.71774322		
x_2	-1.90846932	-3.48081585		
x_1x_2	0.54622090	1.44044359		
$x_1^2x_2$	1.77651656	3.60254429		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	3.41455078	288.871	0.99449170

As the T tests for b_2 and b_4 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_2 = b_4 = 0$$

is tested against the alternative hypothesis

$$H_a: \text{at least one of them differs from 0.}$$

Calculations were done to fit the model after dropping b_2 and b_4 , that is,

$$E(y) = b_0 + b_1x_1 + b_3x_2 + b_5x_1^2x_2 \quad (C.2)$$

The results of the calculations are shown in Table C.6.

The F statistics, based on $6-4=2$ and $14-6=8$ degrees of freedom is

$$F = \frac{(4.59094238 - 3.41455078) / (6 - 4)}{3.41455078 / (14 - 6)} = 1.3781$$

Because the computed value of F does not fall in the rejection region, that is, it does not exceed $F_{0.05} = 4.46$, the null hypothesis that both b_2 and b_4 equal zero is accepted.

Table C.6 Results of the calculations for (C.2) concerning J1 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	9.99989700	39.04753519		
x_1	7.04516649	35.92365738		
x_2	-1.72660351	-3.62596129		
$x_1^2 x_2$	1.56434643	4.33729887		
DF	$T_{0.025}$	SSE	F value	R^2
10	2.228	4.59094238	446.749	0.99259394

Table C.7 Results of the calculations for (C.3) concerning J1 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	9.80704117	34.28277584		
x_1	7.10710144	23.00175465		
DF	$T_{0.025}$	SSE	F value	R^2
12	2.179	13.74780273	529.081	0.97782218

To determine whether data provide sufficient evidence to indicate a difference between the performance of the FVMs and that of the IVMS, the terms involving x_2 in (C.2) are dropped. Therefore the null hypothesis is

$$H_0: b_3 = b_5 = 0$$

against the alternative hypothesis

H_a : at least one of them differs from 0.

The data is used to fit the model:

$$E(y) = b_0 + b_1x_1 \quad (C.3)$$

The results of the calculations are shown in Table C.7.

The F statistics, based on $4-2=2$ and $14-4=10$ degrees of freedom is

$$F = \frac{(13.74780273 - 4.59094238) / (4 - 2)}{4.59094238 / (14 - 4)} = 9.9727$$

The null hypothesis is rejected as $F_{0.05}$ for the relevant degrees of freedom is 4.10.

Following the above calculations (C.2) is accepted as the performance model for J1 group of jobs. After substituting the values of x_1 (uncoded) and x_2 into (C.2), the equation for FVMS is

$$E(y) = -0.29568076 + 1.12607354x$$

and for IVMS

$$E(y) = 1.31852657 + 0.39527459x + 0.03996538x^2$$

x being the number of users on the system.

C.2.2 The Job Group J2

The complete second-order model is given by (C.1). The values of x_1 and their coded values in Table C.4 also applies here. The data in Table C.1(a) and (b) under the column J2 were used to fit the model. The results of the calculations are shown in Table C.8.

As the T tests for b_2 , b_3 and b_5 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them

contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_2 = b_3 = b_5 = 0$$

is tested against the alternative hypothesis

$$H_a: \text{at least one of them differs from 0.}$$

Calculations were done to fit the model after dropping b_2 , b_3 and b_5 from (C.1), that is,

$$E(y) = b_0 + b_1x_1 + b_4x_1x_2 \quad (C.4)$$

The results of the calculations are shown in Table C.9.

Table C.8 Results of the calculations for the complete model (J2 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	9.68604374	36.45338712		
x_1	6.51696157	35.46244864		
x_1^2	-0.09548982	-0.39956989		
x_2	0.02014368	0.05360622		
x_1x_2	0.83922875	3.22915503		
$x_1^2x_2$	0.60480946	1.78953108		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	1.60388184	583.036	0.99726325

Table C.9 Results of the calculations for (C.4) concerning J2 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	9.87347317	64.92395174		
x_1	6.50939703	28.02149170		
x_1x_2	0.88712829	2.70036147		
DF	$T_{0.025}$	SSE	F value	R^2
11	2.201	3.56164551	899.509	0.99392271

The F statistics, based on $6-3=3$ and $14-6=8$ degrees of freedom is

$$F = \frac{(3.56164551 - 1.60388184) / (6 - 3)}{1.60388184 / (14 - 6)} = 3.255$$

Because the computed value of F does not exceed $F_{0.05} = 4.46$, the null hypothesis that b_2 , b_3 and b_5 equal zero is accepted. The T test for each individual parameter in Table C.9 shows that all three parameters b_0 , b_1 and b_4 contribute information for the prediction of the response time. It follows that there is evidence that b_4 is not zero, that is, the performance of the FVMS does differ from that of the IVMS.

The results of the calculations above lead to the acceptance of (C.4) as the performance model for the group J2 jobs. Substituting the values of x_1 (uncoded) and x_2 , the performance equations for the FVMS and the IVMS are

$$\text{FVMS:} \quad E(y) = 0.36085152 + 1.04043812x$$

$$\text{IVMS:} \quad E(y) = -0.93556899 + 1.18223344x$$

respectively, with x being the number of users on the system.

C.2.3 The Job Group J3

The complete second-order model is given by (C.1). Table C.10 gives the values of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J3 were used to fit the model. The results of the calculations are shown in Table C.11.

Table C.10 The values of x_1 and their coded values.

x_1	1	2	3	6	9	12	15
coded x_1	-1.0976	-0.9102	-0.7228	-0.1606	0.4016	0.9638	1.5259

As the T tests for all the parameters except b_0 and b_1 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_2 = b_3 = b_4 = b_5 = 0$$

is tested against the alternative hypothesis

$$H_a: \text{at least one of them differs from 0.}$$

Table C.11 Results of the calculations for the complete model (J3 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	6.11269951	16.96194296		
x_1	5.06917095	19.49259650		
x_1^2	0.53509792	1.58911834		
x_2	0.27657619	0.54267808		
$x_1 x_2$	0.64894360	1.76451227		
$x_1^2 x_2$	0.54398471	1.14233798		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	2.60791016	241.718	0.99342424

Table C.12 Results of the calculations for (C.5) concerning J3 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	6.94277668	27.37859554		
x_1	5.67003775	20.70093888		
DF	$T_{0.025}$	SSE	F value	R^2
12	2.179	10.80322266	428.529	0.97276002

Calculations were done to fit the model involving only b_0 and b_1 , that is,

$$E(y) = b_0 + b_1x_1 \quad (C.5)$$

The results of the calculations are shown in Table C.12.

The F statistics, based on 6-2=4 and 14-6=8 degrees of freedom is

$$F = \frac{(10.80322266 - 2.60791016) / (6 - 2)}{2.60791016 / (14 - 6)} = 6.285$$

As the computed value of F exceeds $F_{0.05} = 3.84$, the null hypothesis is rejected, that is, (C.5) cannot be the performance equation for the group J3 jobs.

An attempt is made to fit the data to (C.6), after dropping only b_3 from (C.1).

The results of the calculations are shown in Table C.13.

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 + b_4x_1x_2 + b_5x_1^2x_2 \quad (C.6)$$

Table C.13 Results of the calculations for (C.6) concerning J3 group.

parameter		estimate	T for H_0 : parameter = 0		
intercept		6.25098801	25.55327879		
x_1		5.10460711	21.12413288		
x_1^2		0.43161681	1.62006974		
x_1x_2		0.57806528	1.75148949		
$x_1^2x_2$		0.75095338	2.74325756		
DF	$T_{0.025}$	SSE	F value	R^2	
9	2.262	2.70373535	327.788	0.99318260	

As the T tests for b_2 and b_4 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_2 = b_4 = 0$$

is tested against the alternative hypothesis

H_a : at least one of them differs from 0.

Calculations were done to fit the model after dropping b_2 and b_4 from (C.6), that is,

$$E(y) = b_0 + b_1x_1 + b_5x_1^2x_2 \quad (C.7)$$

The results of the calculations are shown in Table C.14.

Table C.14 Results of the calculations for (C.7) concerning J3 group.

parameter		estimate	T for H_0 : parameter = 0		
intercept		6.48532343	32.69817479		
x_1		5.48726702	29.88039285		
$x_1^2x_2$		1.06739688	4.16111421		
DF	$T_{0.025}$	SSE	F value	R^2	
11	2.201	4.19689941	514.234	0.98941767	

The F statistics, based on $5-3=2$ and $14-5=9$ degrees of freedom is

$$F = \frac{(4.19689941 - 2.70373535) / (5 - 3)}{2.70373535 / (14 - 5)} = 2.485$$

The null hypothesis is not rejected as the computed value of F is below $F_{0.05} = 4.26$. From Table C.14 it can be seen that no more terms can be dropped from (C.7) as the T test for each parameter is greater than $T_{0.025}$. In particular, the T test confirms that b_5 is not zero so the performance of the FVMS and the IVMS differ. (C.7) is accepted as the performance prediction equation for the J3 group of jobs. After substituting the uncoded x_1 and x_2 , (C.7) becomes for

$$\text{FVMS:} \quad E(y) = -0.56576790 + 1.02829058x$$

$$\text{IVMS: } E(y) = 1.19672117 + 0.51422806x + 0.03748396x^2$$

C.2.4 The Job Group J4

The complete second-order model is given by (C.1). Table C.15 gives the values of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J4 were used to fit the model. The results of the calculations are shown in Table C.16.

Table C.15 The values of x_1 and their coded values.

x_1	1	3	6	9	12	15
coded x_1	-1.2452	-0.8716	-0.3113	0.2490	0.8093	1.3700

Table C.16 Results of the calculations for the complete model (J4 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	8.84425449	94.59659710		
x_1	6.37131166	98.02998808		
x_1^2	0.04837285	0.55473040		
x_2	-0.91587025	-6.92681282		
x_1x_2	-0.73991108	-8.05020170		
$x_1^2x_2$	-0.06265551	-0.50807268		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	0.12451172	3514.194	0.99965864

As the T tests for b_2 and b_5 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_2 = b_5 = 0$$

is tested against the alternative hypothesis

H_a : at least one of them differs from 0.

Calculations were done to fit the model after dropping b_2 and b_5 from (C.1),

that is,

$$E(y) = b_0 + b_1x_1 + b_3x_2 + b_4x_1x_2 \quad (C.8)$$

The results of the calculations are shown in Table C.17.

Table C.17 Results of the calculations for (C.8) concerning J4 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	8.88457394	169.76351532		
x_1	6.37608910	111.22249713		
x_2	-0.96809441	-13.08008203		
x_1x_2	-0.74609894	-9.20302310		
DF	$T_{0.025}$	SSE	F value	R^2
10	2.228	0.13146973	7395.918	0.99963957

The F statistics, based on $6-4=2$ and $12-6=6$ degrees of freedom is

$$F = \frac{(0.13146973 - 0.12451172) / (6 - 4)}{0.12451172 / (12 - 6)} = 0.1676$$

The null hypothesis is not rejected because the computed value of F does not exceed $F_{0.05} = 5.14$.

To confirm that the performance of the FVMS and that of the IVMS differ, the null hypothesis

$$H_0: b_3 = b_4 = 0$$

is tested against the alternative hypothesis

H_a : at least one of them differs from 0.

Calculations were done to fit the model after dropping b_3 and b_4 from (C.8), that is,

$$E(y) = b_0 + b_1x_1 \quad (C.9)$$

The results of the calculations are shown in Table C.18.

Table C.18 Results of the calculations for (C.9) concerning J4 group.

parameter		estimate	T for H_0 : parameter = 0	
intercept		8.40053272	44.19636080	
x_1		6.00303078	28.83311658	
DF	$T_{0.025}$	SSE	F value	R^2
10	2.228	4.33532715	831.347	0.98811430

The F statistics, based on $4-2=2$ and $12-4=8$ degrees of freedom is

$$F = \frac{(4.33532715 - 0.13146973) / (4 - 2)}{0.13146973 / (12 - 4)} = 127.9034$$

With $F_{0.05} = 4.46$, the F test rejects the null hypothesis and confirms that the performance of the FVMS and IVMS do differ. Thus (C.8) is accepted as the prediction equation for the J4 group of jobs. Substituting the uncoded x_1 and x_2 , the prediction equations for the FVMs and the IVMS are

$$\text{FVMS:} \quad E(y) = -0.24554360 + 1.19087972x$$

$$\text{IVMS:} \quad E(y) = -0.14527605 + 1.05152876x$$

C.2.5 The Job Group J5

The complete second-order model is given by (C.1). Table C.4 gives the values

of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J5 were used to fit the model. The results of the calculations are shown in Table C.19.

Experience tells that when the T tests for a majority of the parameters have values below $T_{0.025}$, dropping all these parameters would not be a proper next step. Instead, one of these parameters is dropped (b_5) and the data is used to fit (C.10). The results of the calculations are shown in Table C.20.

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 + b_3x_2 + b_4x_1x_2 \quad (C.10)$$

Table C.19 Results of the calculations for the complete model (J5 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	7.64853954	19.72194272		
x_1	5.40764856	20.16098443		
x_1^2	0.22827846	0.65445579		
x_2	-1.19842577	-2.18507988		
x_1x_2	0.23097655	0.60891427		
$x_1^2x_2$	1.04647386	2.12143030		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	3.41674805	178.180	0.99110025

Table C.20 Results of the calculations for (C.10) concerning J5 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	7.20003986	18.79007240		
x_1	5.36621761	17.02056937		
x_1^2	0.75151682	2.58533805		
x_2	-0.30143327	-0.73217004		
x_1x_2	0.31383979	0.70576360		
DF	$T_{0.025}$	SSE	F value	R^2
9	2.262	5.33911133	159.539	0.98609298

The test statistics F based on 6-5=1 and 14-6=8 degrees of freedom is(though this calculation is not really necessary, as only one parameter is dropped, nevertheless) as given below while $F_{0.05} = 5.32$.

$$F = \frac{(5.33911133 - 3.41674805) / (6 - 5)}{3.41674805 / (14 - 6)} = 4.501$$

As the T tests for b_3 and b_4 are below the value of $T_{0.025}$, F test is used to see whether the data provide sufficient evidence to indicate that at least one of them contribute information for the prediction of the response time. The null hypothesis:

$$H_0: b_3 = b_4 = 0$$

is tested against the alternative hypothesis

$$H_a: \text{at least one of them differs from 0.}$$

Calculations were done to fit the model after dropping b_3 and b_4 from (C.10), that is,

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 \quad (C.11)$$

The results of the calculations are shown in Table C.21.

Table C.21 Results of the calculations for (C.11) concerning J5 group.

parameter		estimate	T for H_0 : parameter = 0		
intercept		7.04932690	22.83759276		
x_1		5.52313709	25.87143842		
x_1^2		0.75151694	2.70698377		
DF	$T_{0.025}$	SSE	F value	R^2	
11	2.201	5.95227051	349.244	0.98449588	

The F statistics, based on 5-3=2 and 14-5=9 degrees of freedom is

$$F = \frac{(5.95227051 - 5.33911133) / (5 - 3)}{5.33911133 / (14 - 5)} = 0.517$$

The null hypothesis is accepted as the computed value of F does not exceed $F_{0.05} = 4.26$. Also no more terms can be dropped from (C.11) as the T test results in Table C.21 indicated. Therefore (C.11) is accepted as the performance model for the J5 group of jobs and it confirms that no difference in performance existed between the FVMS and the IVMS. Substituting the uncoded x_1 , the prediction equation for both the FVMS and the IVMS is

$$\text{FVMS \& IVMS: } E(y) = 0.58292921 + 0.53171992x + 0.01919949x^2$$

C.2.6 The Job Group J6

The complete second-order model is given by (C.1). Table C.15 gives the values of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J6 were used to fit the model. The results of the calculations are shown in Table C.22.

Table C.22 Results of the calculations for the complete model (J6 group).

parameter		estimate	T for H_0 : parameter = 0		
intercept		8.21093178	91.19294514		
x_1		6.04135466	96.52036301		
x_1^2		0.25034589	2.98109146		
x_2		-0.14143389	-1.11072748		
$x_1 x_2$		0.28949854	3.27060518		
$x_1^2 x_2$		0.30588961	2.57564390		
DF	$T_{0.025}$	SSE	F value	R^2	
6	2.447	0.11547852	4042.315	0.99970323	

Only one T test for the parameters is below $T_{0.025}$. The calculations, using the data, are done again to fit (C.12) after dropping b_3 . The results are shown in Table C.23.

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 + b_4x_1x_2 + b_5x_1^2x_2 \quad (C.12)$$

Table C.23 Results of the calculations for (C.12) concerning J6 group.

parameter		estimate	T for H_0 : parameter = 0		
intercept		8.14021397	125.85686546		
x_1		6.03629303	95.18447741		
x_1^2		0.30162024	4.23229275		
x_1x_2		0.29962045	3.34984258		
$x_1^2x_2$		0.20334315	2.67941134		
DF	$T_{0.025}$	SSE	F value	R^2	
7	2.365	0.13903809	4895.686	0.99964267	

Table C.24 Results of the calculations for (C.13) concerning J6 group.

parameter		estimate	T for H_0 : parameter = 0		
intercept		8.14021492	72.61629562		
x_1		6.18610859	79.38592410		
x_1^2		0.40328994	3.85729341		
DF	$T_{0.025}$	SSE	F value	R^2	
9	2.262	0.53698730	3256.167	0.99861991	

To confirm that there is evidence that the performance of the FVMS does differ from that of the IVMS, a F test is done to see whether terms involving x_2 may be dropped from (C.12). The equation to be fitted is (C.13) and the null hypothesis is

$$H_0: b_4 = b_5 = 0$$

and the alternative hypothesis is

H_a : at least one of them differs from 0.

$$E(y) = b_0 + b_1x_1 + b_2x_1^2 \quad (C.13)$$

The results of the calculations are shown in Table C.24.

The F statistics, based on $5-3=2$ and $12-5=7$ degrees of freedom is

$$F = \frac{(0.53698730 - 0.13903809) / (5 - 3)}{0.13903809 / (12 - 5)} = 10.018$$

The null hypothesis is rejected as $F_{0.05} = 4.74$. As a result (C.12) is accepted as the prediction equation for J6 group of jobs. After substituting the uncoded x_1 and x_2 into it the equations for the FVMS and the IVMS are

$$\text{FVMS:} \quad E(y) = 0.11511066 + 0.96608102x + 0.01052174x^2$$

$$\text{IVMS:} \quad E(y) = 0.10301469 + 0.91327556x + 0.01761517x^2$$

C.2.3 The Job Group J7

The complete second-order model is given by (C.1). Table C.10 gives the values of x_1 and their coded values. The data in Table C.1(a) and (b) under the column J7 were used to fit the model. The results of the calculations are shown in Table C.25.

After some calculations it is found that all the parameters which have the computed T test value below $T_{0.025}$ can be dropped from the prediction equation. The F test to confirm this is as follows. The null hypothesis

$$H_0: b_2 = b_3 = b_4 = b_5 = 0$$

is tested against the alternative hypothesis

H_a : at least one of them differs from 0.

Calculations were done to fit the model involving only b_0 and b_1 , that is,

$$E(y) = b_0 + b_1 x_1 \quad (C.14)$$

The results of the calculations are shown in Table C.26.

Table C.25 Results of the calculations for the complete model (J7 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	6.89433241	28.28249220		
x_1	5.33900452	30.35120577		
x_1^2	-0.00680915	-0.02989500		
x_2	0.23672850	0.68669036		
$x_1 x_2$	0.28315741	1.13822615		
$x_1^2 x_2$	-0.12951937	-0.40209212		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	1.19323730	479.372	0.99667341

Table C.26 Results of the calculations for (C.14) concerning J7 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	6.95135117	74.22448582		
x_1	5.45607185	53.93667538		
DF	$T_{0.025}$	SSE	F value	R^2
12	2.179	1.47351074	2909.164	0.99589205

The F statistics, based on $6-2=4$ and $14-6=8$ degrees of freedom is

$$F = \frac{(1.47351074 - 1.19323730) / (6 - 2)}{1.19323730 / (14 - 6)} = 0.4698$$

The null hypothesis is accepted as the computed F value does not exceed $F_{0.05} = 3.84$. This means there is no difference between the performance of the FVMS and that of the IVMS. The uncoded x_1 is substituted into (C.14) and the prediction equation for

the J7 group of jobs is:

$$\text{FVMS \& IVMS: } E(y) = -0.05965464 + 1.02244474$$

C.2.8 The Job Group J8

The complete second-order model is given by (C.1). Table C.27 gives the values of x_1 and their coded values. The data in Table C.2 were used to fit the model.

The results of the calculations are shown in Table C.28.

Table C.27 The values of x_1 and their coded values.

x_1	2	4	6	8	12	16	18
coded x_1	-1.2221	-0.8930	-0.5640	-0.2350	0.4230	1.0810	1.4100

Table C.28 Results of the calculations for the complete model (J8 group).

parameter	estimate	T for H_0 : parameter = 0		
intercept	5.82112169	19.55097049		
x_1	3.89757013	19.21358152		
x_1^2	-0.09125201	-0.32740211		
x_2	0.85088211	2.02076621		
$x_1 x_2$	1.46525693	5.10755644		
$x_1^2 x_2$	0.57401782	1.45629534		
DF	$T_{0.025}$	SSE	F value	R^2
8	2.306	1.76849365	250.689	0.99365807

Dropping two of the three parameters which have the T test value below $T_{0.025}$, namely, b_2 and b_5 from (C.1), the prediction equation to be fitted next is (C.15). The F test has the null hypothesis:

$$H_0: b_2 = b_5 = 0$$

and the alternative hypothesis is

H_a : at least one of them differs from 0.

$$E(y) = b_0 + b_1x_1 + b_3x_2 + b_4x_1x_2 \quad (C.15)$$

The results of the calculations are shown in Table C.29.

The F statistics, based on 6-4=2 and 14-6=8 degrees of freedom is

$$F = \frac{(2.45507812 - 1.76849365) / (6 - 4)}{1.76849365 / (14 - 6)} = 1.553$$

The null hypothesis is accepted as the computed F value is below $F_{0.05} = 3.11$.

To confirm that there is difference between the performance of the FVMs and the IVMS, a F test is done with the null hypothesis:

$$H_0: b_3 = b_4 = 0$$

against the alternative hypothesis

H_a : at least one of them differs from 0.

The data is used to fit the model:

$$E(y) = b_0 + b_1x_1 \quad (C.16)$$

The results of the calculations are shown in Table C.30.

Table C.29 Results of the calculations for (C.15) concerning J8 group.

parameter	estimate	T for H_0 : parameter = 0		
intercept	5.74291229	30.66539292		
x_1	3.87608600	19.16162617		
x_2	1.34288049	5.07035906		
x_1x_2	1.60040045	5.59438829		
DF	$T_{0.025}$	SSE	F value	R^2
10	2.228	2.45507812	375.279	0.99119592

Table C.30 Results of the calculations for (C.16) concerning J8 group.

parameter		estimate	T for H_0 : parameter = 0	
intercept		6.41435242	20.49841712	
x_1		4.67628622	13.83538293	
DF	$T_{0.025}$	SSE	F value	R^2
12	2.179	16.45031738	191.418	0.94100809

The F statistics, based on $4-2=2$ and $14-4=10$ degrees of freedom is

$$F = \frac{(16.45031738 - 2.45507812) / (4 - 2)}{2.45507812 / (14 - 4)} = 28.503$$

The null hypothesis is rejected because the computed F value exceeds $F_{0.05} = 4.10$. Therefore (C.15) is accepted as the prediction equation for the J8 group of jobs. After substituting the uncoded x_1 and x_2 into it the prediction equations for the FVMS and the IVMS are:

$$\text{FVMS:} \quad E(y) = -0.26914016 + 0.63763999x$$

$$\text{IVMS:} \quad E(y) = -1.40858113 + 0.90091572x$$

References

- [1] R. J. Creasy The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development* September 1981 Vol 25 No. 5.
- [2] R. A. Meyer and L. H. Seawright A Virtual Machine Time-Sharing System. *IBM System Journal* 1970 Vol 9 No 3.
- [3] R. P. Parmelee, T. I. Peterson, C. C. Tillman and D. J. Hatfield Virtual Storage and Virtual Machine Concepts. *IBM System Journal* 1972 Vol 11.
- [4] R. P Goldberg Survey of Virtual Machine Research. *Computer* June 1974.
- [5] *IBM System Journal.* 1979 Vol 18 No. 1.
- [6] Y. Bard Performance Analysis of Virtual Memory Time-sharing Systems. *IBM System Journal* 1975 Vol 14 No 4.
- [7] P. H. Gum System/370 Extended Architecture: Facilities of Virtual Machines. *IBM Journal of Research and Development* November 1983 Vol 27 No. 6.
- [8] W. J. Doherty and R. P. Kelisky Managing VM/CMS Systems for User Effectiveness. *IBM System Journal.* 1979 Vol 18 No. 1.
- [9] S. M. Cox The Implementation and Effectiveness of a Function Processor SubSystem. PhD dissertation. Imperial College, London. 1988.
- [10] D. R. Cheriton The V Distributed System. *Communications of the ACM* March 1988 Vol 31 No 3.
- [11] D. R. Cheriton and W. Zwaenepoel The Distributed V Kernel and its Performance for Diskless Workstations. *ACM SIGOPS Operating System Review, Symposium* 1983.
- [12] J. K. Ousterhout Medusa A Distributed Operating System. UMI Research Press. 1981.
- [13] R. P. Bird A Compiler Server Node in a Local Area Network. *Proceedings of International Computing Symposium*, B G Teubner. 1983.
- [14] M.V. Wilkes and R.M. Needham The Cambridge Model Distributed System. *Operating Systems Review* January 1980 Vol 14 No. 1.
- [15] IBM IBM Virtual Machine/System Product: System Programmer's Guide.
- [16] IBM IBM Virtual Machine/System Product: System Logic and Problem

Determination Guide Volume 1 - CP.

- [17] IBM IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 2 - CMS.
- [18] I. Stinson Virtual Machines as a Vehicle for System Service. Imperial College, London.
- [19] A. S. Tanenbaum and R. V. Renesse Distributed Operating Systems. *ACM Computing Surveys.* December 1985 Vol 17 No. 4.
- [20] V. B. Gyls and J. A. Edwards Optimal Partitioning of Workload for Distributed Systems. *IEEE Compcon* 1976 Fall.
- [21] J. D. Bagley, E. R. Floto, S. C. Hsieh, V. Watson Sharing Data and Services in a Virtual Machine System. *SIGOPS Operating Systems Review*, 1975 Vol 9 No. 5.
- [22] I. Mitrani Modelling of Computers and Communication Systems. Cambridge University Press. 1987.
- [23] M. J. Marsan, G. Conte and G. Balbo A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, May 1984 Vol 2 No. 2.
- [24] A. K. Jones, R. J. Chansler Jr., I. Durham, K. Schwans and S. R. Vegdahl StartOS. A Multiprocessor Operating System for the Support of Task Forces. *Proceedings of the Seventh Symposium on Operating System Principles* December 1979.
- [25] K. A. Lantz, W. I. Nowicki and M. M. Theimer An Empirical Study of Distributed Application Performance. *IEEE Transactions on Software Engineering.* October 1985 Vol SE-11 No. 10.
- [26] D. Gelernter and A. J. Bernstein Distributed Communication via Global Buffer. *Symposium on Principles of Distributed Computing* Ottawa, Canada. August 1982.
- [27] W. D. Sincoskie and D. J. Farber The Series/1 Distributed Operating System: Description and Comments. *Compcon* 1980 Fall.
- [28] G. Pugh Inter Program Communication Facility. Concepts and Facilities. 1983 Imperial College, London.
- [29] G. Pugh Source listing of IPCF. Department of Computing, Imperial College London.
- [30] IBM Source listings of IUCV, DMKIUA, DMKIUC, DMKIUE and DMKIUG. These are the service modules of IUCV.
- [31] IBM Source listing of DMKVMC. This is the primary VMCF support module.

- [32] R. M. Jensen A Formal Approach for Communication between Logically Isolated Virtual Machines. *IBM System Journal*, 1979 Vol 18 No. 1.
- [33] G. R. Andrews and F. B. Schneider Concepts and Notations for Concurrent Programming. *Computing Surveys*, March 1983 Vol 15, No. 1.
- [34] G. R. Andrews, D. P. Dobkin and P. J. Downey Distributed Allocation with Pools of Servers. *ACM Symposium on Principles of Distributed Computing.* August 1982. Ottawa, Canada
- [35] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin and G. Thiel LOCUS: A Network transparent. High Reliability Distributed System. *Proceedings of 8th Symposium on Operating System Principles.* December 1981.
- [36] M. Satyanarayanan Multi-Processors A Comparative Study Chapter 3. Prentice-Hall. 1980.
- [37] J. W. Atwood Concurrency in Operating Systems. *IEEE Computer*, 1976 Vol 9 No. 10.
- [38] J. F. Shoch and J. A. Hupp Measured Performance of an Ethernet Local Network. *Communications of the ACM* December 1980. Vol 23, No. 12.
- [39] H. Kobayashi Modeling and Analysis: An Introduction to System Performance Evaluation Methodology Chapter 4. Addison-Wesley, 1978.
- [40] W. Mendenhall and T. Sincich Statistics for the Engineering and Computer Sciences. Collier Macmillan Publishers, 1988.
- [41] Y. Parker Multi-microprocessor Systems. Academic Press, 1983.
- [42] A. G. Anderson and E. D. Jensen Computer Interconnection Structures: Taxonomy, Characteristics and Examples. *ACM Computing Surveys*, December 1975 Vol 7, No 4.
- [43] INMOS Transputer Reference Manual. Prentice Hall, 1988.
- [44] R. M. Metcalfe and D. R. Boggs Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of ACM.* July 1976. Vol 19 No.7.
- [45] A. P. Black Supporting Distributed Applications: Experience with Eden. *Proceedings of 10th Symposium on Operating System Principles.* December 1985.
- [46] S. C. Schaffner and M. Borker Segue: Support for Distributed Graphical Interfaces *IEEE Computer* December 1988

- [47] Y. Wang and R. J. T. Morris Load Sharing in Distributed Systems. *IEEE Transactions on Computers* March 1985 Vol C-34 No. 3.
- [48] K. Li IVY: A Shared Virtual Memory System for Parallel Computing. *IEEE Proceedings of the 1988 International Conference on Parallel Processing.*
- [49] R. M. Keller, G. Lindstrom and S. Patil A Loosely-Coupled Applicative Multi-Processing System. *National Computer Conference* 1979.