

Archives

MASS. INST. TECH.

MAR 18 1976

LIBRARIES

~~75-010~~
ENERGY LABORATORY

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

TK1001

.M41

.E56

no. 75-

010

USE OF VIRTUAL MACHINES IN INFORMATION SYSTEMS

John J. Donovan

May 1975

Energy Lab in Association
with Sloan School

Report No. MIT-EL-75-010



Abstract

Acknowledgment

1. Introduction
2. Interfacing modeling facilities and data base facilities
3. Problems with interfacing
4. Description of Virtual Machine Concepts
5. Use of VM in Information Systems
 - 5.1 Communications between VM's
 - 5.2 Multiuser Coordination
 - 5.3 Multiple Modeling Interfaces
 - 5.4 Incompatible Data Management System
 - 5.5 A Practical Example
6. Performance
 - 6.1 Analysis as Separate Machines
 - 6.2 Analysis if all Machines are VM's
7. Techniques for Reducing Synchronization Overhead
8. Techniques for Reducing Effect of Virtual Machine on Response Time
9. Summary

Abstract

This paper presents a scheme using the virtual machine concept for creating:

- 1) An environment for increasing the effectiveness of researchers who must use analytical, modeling systems and have complex data management needs.
- 2) A mechanism for multi-user coordination of access and update to a central data base.
- 3) A mechanism for creating an environment where several different modeling facilities can access the same data base.
- 4) A mechanism for creating an environment where several different and potentially incompatible data management systems can all be accessed by the same user models or facilities.

The paper investigates and formalizes the performance implications of this scheme specifically directed at the question of response time degradation as a function of number of virtual machines, of locked time of the data base machine, and of query rate of the modeling machine.

1. Introduction

Many applications demand both a very good analytical and modeling capability, as well as a flexible data base management capability. They demand that the capability be on-line and interactive. These demands are particularly acute in information systems for assisting public policy decisions and in particular we have found in the area of energy [Donovan: 1975; MacAvoy: 1974]. Such systems have a spectrum of users ranging from the non-technical to the researcher to the computer professional. Each grouping demands a different level of detail in capabilities. Further such systems have:

- a need to build models quickly.
- a need to place complex protection rights on data.
- a need to validate data.
- a need to access data according to any number of criterion.
- a need for mechanisms for changing the system to meet new demands and different data series and needs.
- a need to handle all types of data.

Modeling systems like TROLL [TROLL: 1972], EPLAN [Schober: 1974], and TSP / [Hall: 1975] provide flexible analytical capabilities such as sophisticated statistical methods, arithmetic operation, plots, graphs, histograms and facilities for constructing and executing mathematical models. All of these have some short

comings but the most serious shortcoming is in their limited data management capabilities. There are very limited facilities for protecting data, storing different types of data, changing the structure of data or tables in the system, validating data, quering data by specifying different conditions. Some of these facilities are single user non interactive systems. None allow multiple users accessing the same data base.

Corresponding there exist data management systems like IMS, DBDG, ENQUIRE, TOTAL which provide some degree of data manipulation capabilities but are seriously lacking in analytical or modeling capabilities. They also lack the flexibility in use, access, and protection of data demanded by some applications [Jacoby; 1975]. They do however have considerably more data capability than the modeling systems previously mentioned. This lack of flexibility is a particularly damaging limitation in the context of the certain applications for several reasons:

1. Since unforeseen uses and needs for the data inevitably arise, the system must be flexible so that it can adapt to these changing needs. This is particularly true when providing information for policy decisions in so volatile an area as energy.
2. There are varying constraints imposed by changes in the quality, availability, and protection requirements of data. The system must be able to adjust to such moving constraints.
3. The system must be able to accommodate changing needs and constraints at reasonable expenditures of cost and effort. Computer systems of a decade or two ago could support most current applications, but in many cases, only at a high cost.

A flexible system makes it possible to easily experiment with many uses of the data at modest costs.

We have developed a very flexible data management system called TRANSAC [Donovan & Jacoby: 1974] that meets these criteria. The purpose of this paper is however not to promote any one modeling system or data management system but rather to present a scheme whereby the good features of any system can be best utilized.

2. Interfacing modeling facilities and data base facilities

Let us explain a scheme whereby we could interface a modeling system e.g. TROLL, to a data base system.

For conceptual purposes, let us just speak of two separate machines, one at Yale which is running TROLL under Yale's operating system and one at M.I.T. which is running the data base system under M.I.T.'s operating system.

The interface scheme would be whenever the Yale machine needs data, it would request a courier to run to M.I.T. and get the data out of the data management machine. The courier would then bring the data back for the modeling machine.

3. Problems with interfacing

Starting with the scheme of using two independent computer systems, let us evolve into a proposed viable scheme which we advocate.

1. Many modeling facilities are single user non-interactive batch oriented (e.g. TROLL is single user, IBM's TSP is batch oriented). A multiuser interactive facility is desirable.

- Solution: place each modeling facility on a separate machine.

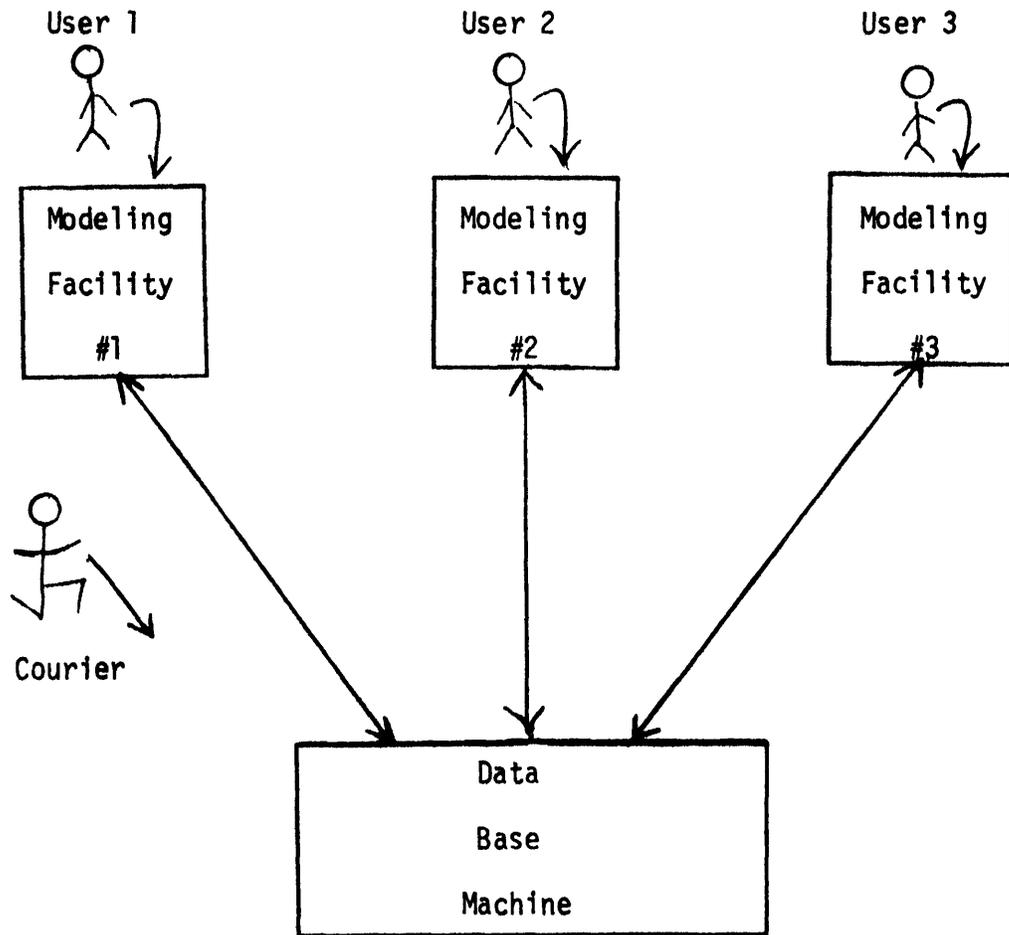


Figure 1
Multiple Users of the Same Data Base

2. We would like more than one user (modeler) to be able to access the data base at one time.
 - Solution: Allow many machines to communicate with the same data base machine as in Figure 1.

3. The solution to 2 creates the problem of coordination of updating the single data base.
 - Solution: Only one modeling system will be serviced by the data base machine at one time.

4. Not every user will want the same modeling facility; some will want TSP; others, TROLL, etc.
 - Solution: One solution is to require all users to convert and all existing models be redone in one modeling language. Another solution is to run a courier between machines that have different modeling capability on them and the single data base machine as in Figure 1.

5. Data series may already exist in several and incompatible data base management systems. How can a user access these data series.
 - Solution: Interface machines that have different data base systems as in Figure 2.

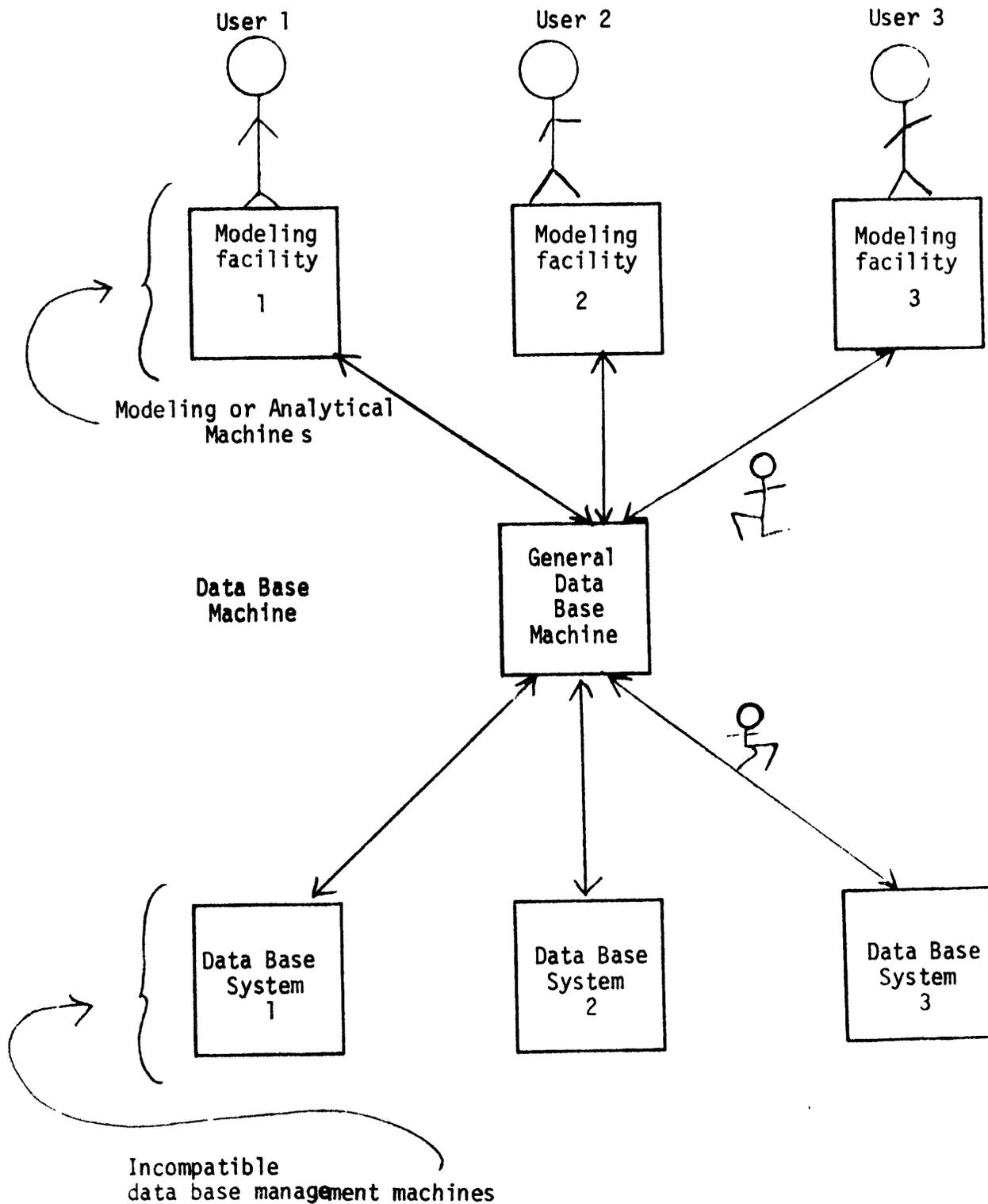


Figure 2

6. The cost of many separate machines is high.

Couriers between all these machines are slow and not practical.

- Solution: Have all these machines run on the same machine, that is, have one machine simulate several machines (virtual machines). On some of these virtual machines, run the modeling facilities; on others run the data base facilities; on one run the general data base facility. What about communication? This will be discussed in Section 5.

7. What about performance?

We discuss this in Section 6.

4. Description of Virtual Machine Concepts

A virtual machine may be defined as a replica of a real computer system simulated by a combination of a Virtual Machine Monitor (VMM) software program and appropriate hardware support. (See [Goldberg: 1973] for a more precise definition.) For example, the VM/370 (IBM 72) system enables a single IBM

System/370 to appear functionally as if it were multiple independent System/370's (i.e., multiple "virtual machines"). Thus, a VMM can make one computer system function as if it were multiple physically isolated systems.

A VMM accomplishes this feat by controlling the multiplexing of the physical hardware resources in a manner analogous to the way that the telephone company multiplexes communications enabling separate and, hopefully, isolated conversations over the same physical communications link.

By restricting itself to the task of multiplexing and allocating the physical hardware, the VMM presents an interface that appears identical to a "bare machine". In fact, it is usually desirable to load a user-oriented operating system into each virtual machine to provide the functions expected of modern operating systems, such as Job Control Language, command processors, data management services, and language processors. Thus, each virtual machine is controlled by a separate, and possibly different, operating system. The feasibility of this solution has been demonstrated on the VM/370 system and the earlier CP-67 and CP-40 systems

In addition to VM/370 and its predecessors, several other operational virtual machine systems have been developed, such as the DOS/VM of PRIME Computer, Inc. [PRIME: 1974], the virtual machine capability provided under the Michigan Terminal System (MTS) [Morrison: 1973], and a virtual machine system for a modified PDP-11/45 used by UCLA for data security studies [Popek & Kline: 1974].

The VM concept, once understood, is quite simple and logical. Unfortunately, it is sufficiently different from most conventional operating systems that many people have difficulty in understanding the concept. The papers [Buzen: 1973, Goldberg: 1973, Hogg: 1973, Madnick: 1969, Parmelee: 1972, and Madnick & Donovan: 1974] give additional insight.

At first the idea of replicating the bare machine interface may seem foolish since you end up back where you started. The key difference is VM/370 produces the effect of multiple bare machines. In this way each user appears to have his own 370 computer. Thus, each user can select the operating system (e.g. OS/360 DOS, etc.) of his choice to run on his "private" computer.

How does VM/370 produce this feat? How do the users of VM/370 communicate with it? Programs running under VM/370, usually operating systems physically execute in problem state but can behave as if they were in supervisor state. When they issue a privileged instruction, such as START I/O or SET STORAGE KEY, an interrupt occurs and control transfers to VM/370. The interrupt is handled in such a way that the program thinks that the privileged instruction was actually executed. Thus, these privileged instruction interrupts are the subtle interfaces between users and VM/370.

Additional advantages of VM are outlined in [Buzen: 1973] and [Madnick & Donovan: 1974, 1975].

5. Use of VM in Information Systems

As was discussed in Section 4, having multiple machines gives the effect of having multiuser modeling facilities which can access data stored in several different data bases. Proposed communication between all these was via courier. Another possibility and the scheme we advocate is to simulate several different machines on one machine using the VM concept. This section discusses the implications and mechanics of this possibility.

Combining the solutions of the previous section, we could, for example, create a configuration of VM's whose architecture could be depicted as in Figure 2, where each box denotes a virtual machine.

5.1 Communications between VM's

Configuring several VM's on one real machine as in Figure 2 allows several modeling systems to access data from a single data base management system. When a modeling facility issues a request for data, that request is output on a virtual card punch and sent to the data management machine's virtual card reader. The data management machine reads the request, selects the data, and transfers the data back to the modeling facility via the transfer of data from the data management virtual punch to the modeling facility's virtual reader.

Note that no (physical) cards are involved in this process. The "card files" which are punched and read, are in fact stored on (physical) disks for the transfer.

The amount of reprogramming and design involved in modifying the data base management system DBMS to accept requests and output data to its virtual card devices is relatively small, compared to the amount of work and/ complexity that would be involved in rewriting the modeling system to include a facility for data handling, for multiusers, for interactive editing, for synchronization of data base access and updating.

Since all modeling facilities have mechanisms to store data in files and facilities to operate on this data, the modification to a modeling system under the VM scheme consists of adding three commands:

- adding a command to convert the data outputted from the DBMS into the format that the modeling facility uses.

By adding two more commands, a modeling system which has very poor data management capabilities can appear to a user as if he had a very powerful facility for storing, quering, updating, and manipulating data.

- adding a command that has as possible arguments the commands of the data base system. The modeling system "passes" the command on to the data base machine via virtual cards.
- adding a command which prints data passed back to the modeling facility.

This scheme will also work with most data base systems, as most of them have (or it is easy to add) a mechanism for reading request in from files or cards and outputting results to cards or files.

5.2 Multuser Coordination

The basic problem with having multiple users of the same data base is how to prevent race conditions and uncertainties resulting from several users accessing and updating the same data base. A mechanism we advocate is to have the data base virtual machine only allow one user to access or update its VM at one time. Thus, whenever the data base virtual machine is processing a request, it queues all other requests. The queue is serviced on a FIFO basis.

The performance implications of this approach have not been experimentally tested. A mathematical analysis of the performance is presented in Section 6.

5.3 Multiple Modeling Interfaces

Adding the commands outlined in the previous section to other modeling facilities and running each of these different modeling facilities in a separate VM allows several different modeling facilities to communicate with

the same data base. Thus, incompatible systems, such as TROLL and EPLAN, can work from the same data base.

5.4 Incompatible Data Management System

Let us suppose that there is a need to create a DBMS that uses data from several data bases, each of which is on an incompatible data base system. We reject copying all data bases into one data base system because, for example, the existing DB systems may be specialized to keep the data up to date. Thus, how can we treat these four physically separate data bases as one logical unit?

A solution to this problem is also shown in Figure 2, where we could configure three virtual machines to allow the mutually incompatible data base management systems to run on the same physical computer. We then implement another VM to act as an interface, analyzing the data query and funneling it to the appropriate DBMS (via virtual card files). All of these mechanisms can be made invisible to the user, who can use the system as though he had all the data in one "virtual" data base.

Note the "user" in this sense can be a modeling facility or a person, i.e., a user here is anything that makes a data request.

5.5. A Practical Example

We have configured a cluster of VM as in Figure 2 to produce a total system for research in energy policy analysis. We call the system GMIS (General Management Information System). Figure 3 depicts the ultimate GMIS system [Donovan et al: 1975], where across the top several modeling or analytical systems are depicted as running on separate virtual machines. Note that each of these analytical systems may be running under a different

operating system, e.g., TSP running under MVT, TROLL running under CP/CMS, EPLAN running under VSZ. TRANSACT [Donovan and Jacoby: 1975] is a data base system based on the relational model of data [Codd: 1970] and uses some IBM software [Chamberlain: 1974]. TRANSACT is implemented in a hierarchical fashion [Dijkstra: 1968, Madnick: 1970, Donovan: 1972], and as such it is a very flexible and powerful data management system. Across the bottom of Figure 3 are depicted several data base systems, each of which may be incompatible and running under different operating systems.

Note that in this paper, independently of any one data base system, we are advocating the use of VM to produce an environment where multiple analytical machines can be used on the same facility and these analytical systems have access to data base systems.

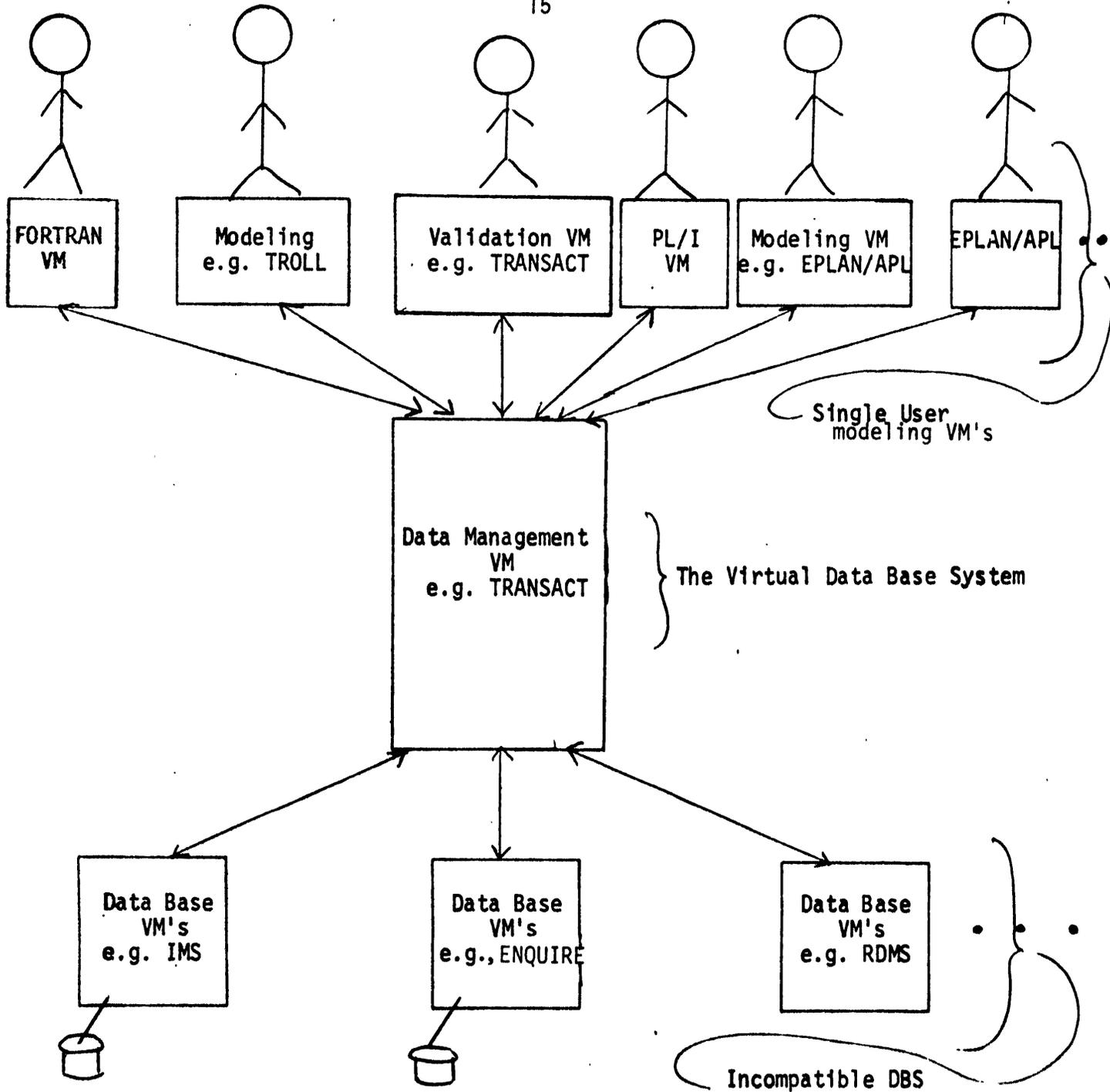


Figure 3

6. Performance

Not only does the VM approach solve all the problems of Section 3 but it also has the following cost benefits:

- no conversion cost in bringing up existing models as long as they run on an IBM machine (independent of language or operating system).
- no retraining cost involved as programmer's may use whatever system they are familiar with.
- little cost involved in implementing the simple interfaces.

What is the possible disadvantage - performance, which is reflected in additional overhead costs: For example, the following questions arise:

- How many users (modeling machines) can use the same data base machine? That is, what is the degradation of response time as a function of the number of modelers?
- What is the degradation cost due to the synchronization mechanism?
- What is the degradation cost due to VM?

We have separated the two performance costs:

- (1) due to lock out synchronization mechanisms and
- (2) due to VM overhead.

The approach to answering these questions we take here is an analytical one. We will first analyze the performance issues of lock out by configuring a system of separate real machines. We then analyze the cost of VM by configuring the separate real machines or as virtual machines on one real machine. Other approaches to gain other factors of performance in VM are discussed in [Hatfield: 1972, Goldberg: 1974].

7.1 Analysis as Separate Machines (performance degradation due to lock out)

Assuming a configuration as in figure 1, where several modeling facilities each running on a separate real machine are accessing and updating a data base which is managed by a data base management system running on its separate real machine. What is the degradation of performance with each additional user? What is it as a function of the length of time the DB machine takes to process a request?

An access or update to the DB machine may be initiated either by a query from a person which would be passed on by the modeling machine or by a model executing on the modeling machine.

In either case, the DB machine while processing a request locks out (queues) all other requests. Let us write a function that specifies total response time of a model.

$$T_{\text{total}} = T_{\text{overhead}} + T_{\text{model}} + T_{\text{request and wait}}$$

where

T_{total} = total response time of a task, (e.g., a model) that is, total time from the start of execution of a model to the answer.

T_{overhead} = amount of CPU time spent executing instructions in the operating system of the modeling machine.

T_{model} = amount of CPU time executing the instruction associated with the model.

$T_{\text{request and wait}}$ = time modeling machine waits for request to be processed plus time spent waiting for request to be serviced by the DB machine.

What one would want to know is what happens to T_{total} as a function of the number of users. That is, how many users can we tolerate on the system.

Assume that:

- (1) a configuration of separate real machines as in figure 1.
- (2) the time spent in executing the model in a modeling machine before issuing a request for data to the DB machine is negative exponentially distributed with mean $1/\lambda$
- (3) the time for the DB machine to serve a request is negative exponentially distributed with mean $1/\mu$
- (4) the order of service at the DB machine is FIFO
- (5) the number of modeling machines is m

We can formulate the problem as a machine-repairman model [Satty: 1961] as shown in Figure 4. The steady state equations are:

$$\left. \begin{aligned} m\lambda P_0 &= \mu P_1 \\ [(m-i)\lambda + \mu] P_i &= (m-i+1)\lambda P_{i-1} + \mu P_{i+1} \end{aligned} \right\} \text{ for } 0 < i < m$$

$$\mu P_m = \lambda P_{m-1}$$

Where P_i is the steady-state probability that there are i modeling machines waiting and being served.

The solution is:

$$P_i = P_0 \left(\frac{\lambda}{\mu} \right)^i \binom{m}{i} i!$$

$$= P_0 \left(\frac{\lambda}{\mu} \right)^i \frac{m!}{(m-i)!}$$

where $i = 1, \dots, m$

where,

$$P_0 = \left(\sum_{i=0}^m \frac{P_i}{P_0} \right)^{-1} =$$

$$= \left(\sum_{i=0}^m \left(\frac{\lambda}{\mu} \right)^i \binom{m}{i} i! \right)^{-1}$$

$$= \left(\sum_{i=0}^m \left(\frac{\lambda}{\mu} \right)^i \frac{m!}{(m-i)!} \right)^{-1}$$

The average response time for a request to DB machine as derived by [Little: 1961] is:

$$R = \sum_{i=1}^m \frac{i P_i}{\mu (1 - P_0)}$$

Figure 5 illustrates the wait and process time for a single request as a function of modeling machines. For example, with $\frac{\lambda}{\mu} = 1$, five users on the system degrades the response time of each user by a factor of four. With a $\frac{\lambda}{\mu}$ ratio of less than .1 there is almost no degradation of response until a large number of users are using the system

Note: In all the remaining graphs μ is set at a constant value of 1.0, and N (number of data requests) is a constant 10. The values of T_{overhead} is a constant equal to 1.0.

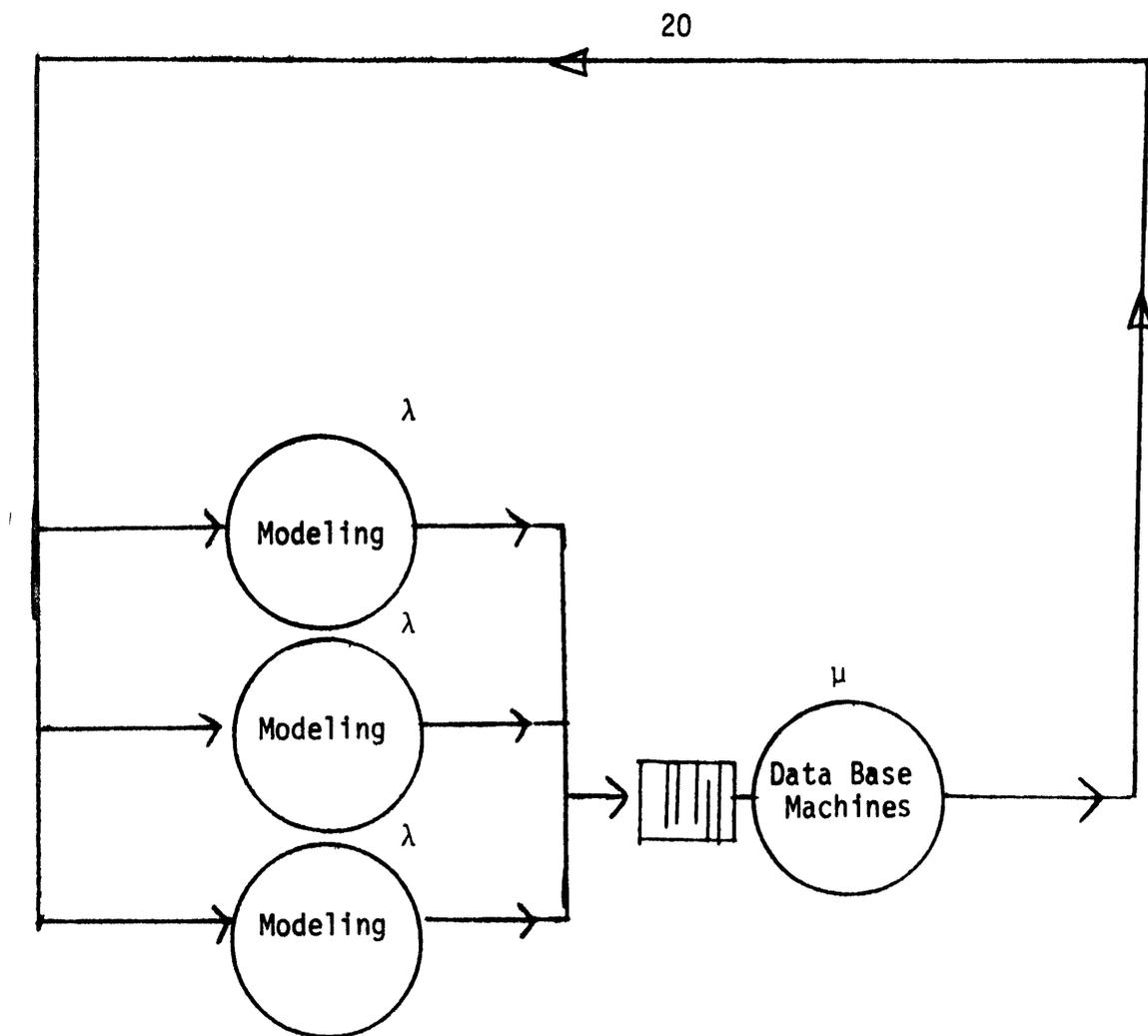


Figure 4
Model of Figure 1

Average
Response
Time for a
Single Data
Request
($\mu = 1$)

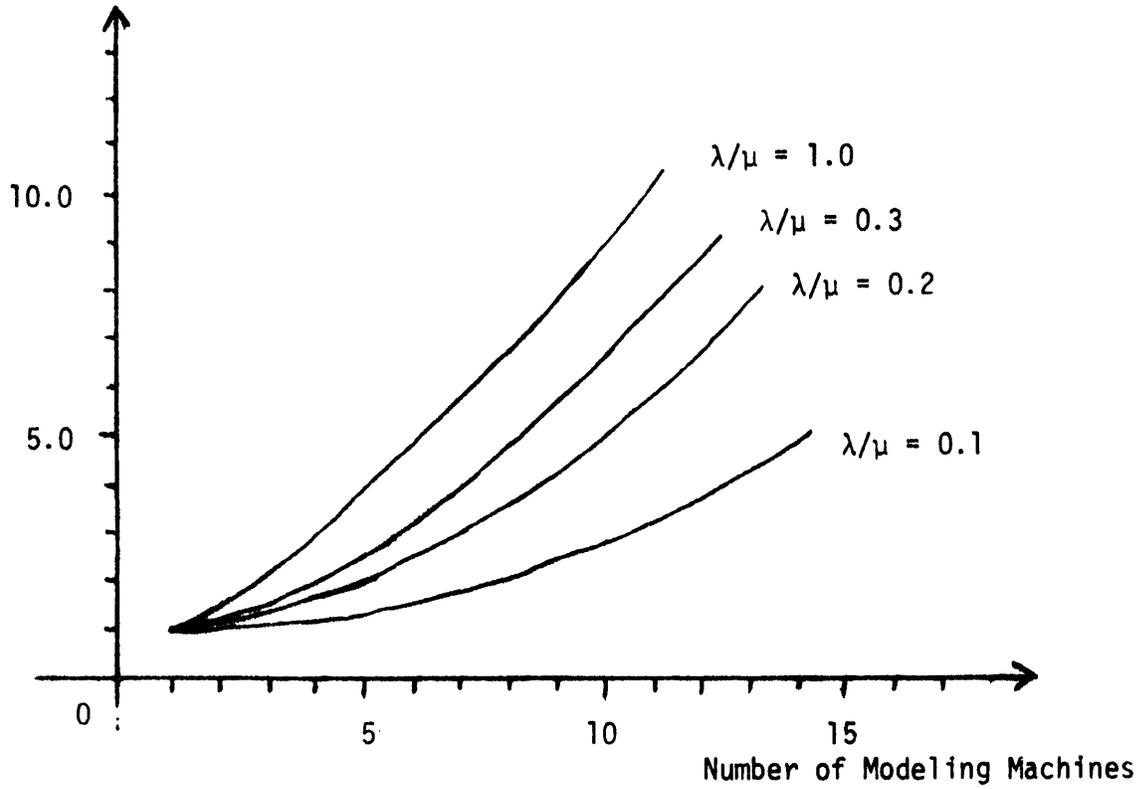


Figure 5

Response Time for a Single Request

Assume that the average number of data requests to DB machine in running a model in a modeling machine is N . The data base is locked only while a request is being processed. We are assuming there is no reason to lock the data base for the whole period while a model is running. The situation where a data base must be locked for the entire period of execution (e.g., a possible danger that other modeling machines will change sensitive data in between requests) requires another analysis.

The total time waiting for data from the Data Base machine is:

$$T_{\text{wait for data}} = N \cdot R$$

The average time spent in executing a model in a modeling machine is a constant:

$$T_{\text{model}} = N \cdot (1/\lambda)$$

The overhead of the operating system of one modeling machine is fixed and is equal to a constant T_{overhead} . The total time to execute a model in the modeling machine is:

$$T_{\text{total}} = T_{\text{overhead}} + T_{\text{model}} + T_{\text{wait-for-data}}$$

and is plotted in Figure 6.

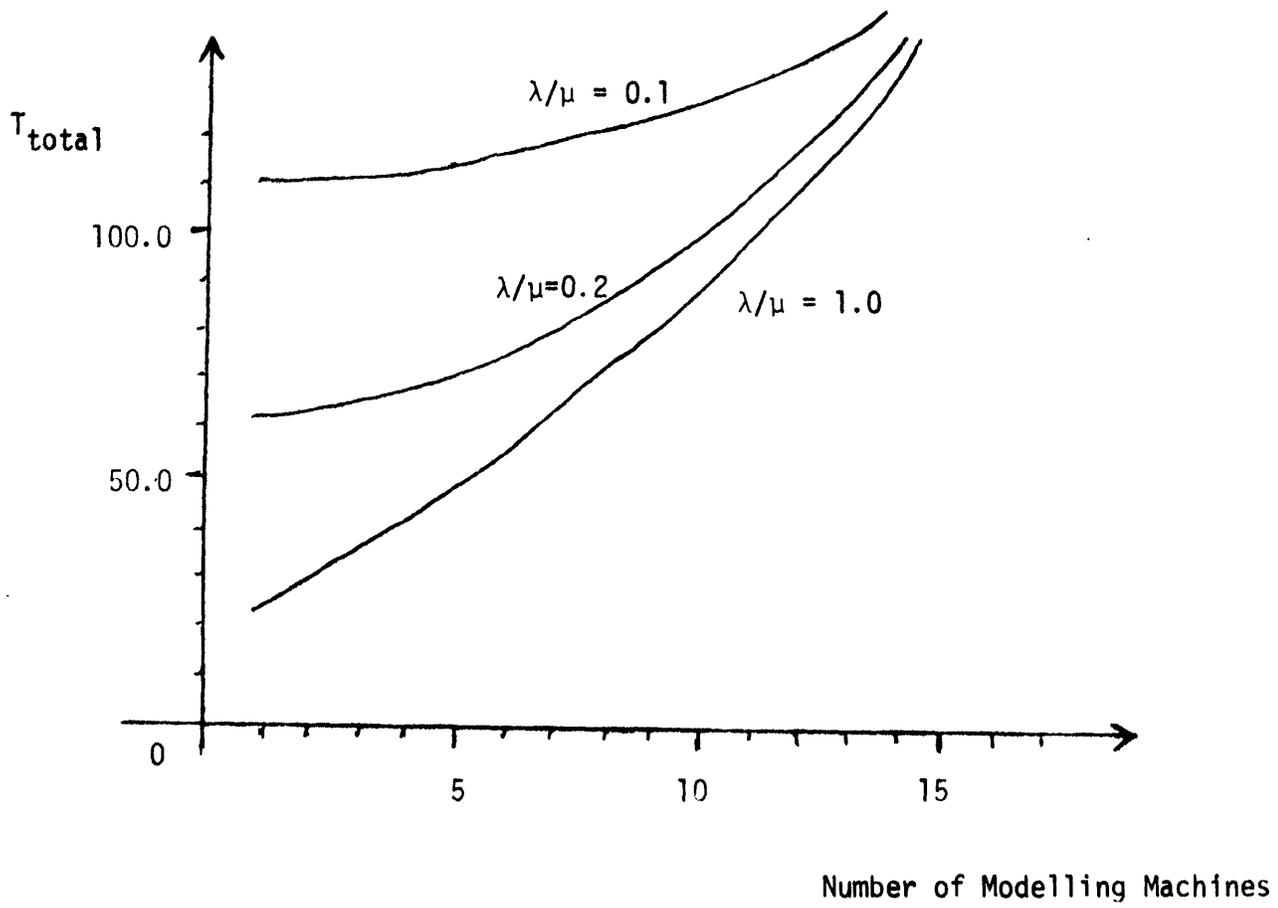


Figure 6
Total Response Time

6.2 Analysis if All Machines are VM's on One Real Machine

If all machines are run as virtual machines on one real machine, what is the additional degradation of response time?

In the VM configuration actually the real machine spends a small portion of its time on each VM. As the number of VM's increase, then each VM will get less of the real CPU's time thus further increasing the elapse time between the start of a model and the production of the answer.

The analysis is further complicated by the fact that as some VM's become locked then others get more of the real CPU's time, therefore, they generate requests faster. However, the DB VM gets more of the CPU's time thereby processing requests faster. For example, if there are ten virtual machines, each one receives one-tenth of the real CPU. However, if seven of the ten are in a locked state, then the remaining three receive one-third of the CPU. Thus, these three run (in real time) faster than they did when ten were running. The following is an analysis of VM's performance for the use outlined in this paper.

We have assumed that the virtual speeds of VM's are constant and equal. However when some VM's are blocked (i.e., waiting for data from the DB VM), the remaining VM's (including DB VM) are allocated a larger share of CPU processing power and become faster in real time. We assume that each unblocked VM receives the same amount of CPU processing power and at the initial state m machines are running (i.e., the data base machine is stopped if no modeling machines are making requests). ' λ ' is request rate of each modeling VM when there are m VM's running. ' μ ' is the service rate at which the data base virtual machine is running when there are $m-1$ modeling VM and one data base VM running. Thus, we may write the relations:

$$\mu_i = \frac{m}{m-i+1} \mu \quad (i = 1, 2, \dots, m)$$

$$\lambda_0 = \lambda$$

$$\lambda_i = \frac{m}{m-i+1} \lambda \quad (i = 1, 2, \dots, m)$$

where i ($i = 0, 1, \dots, m$) is the number of modeling VM's being blocked.
Using a birth/death process model [Drake: 1967], the state transition diagram is shown in Figure 7.

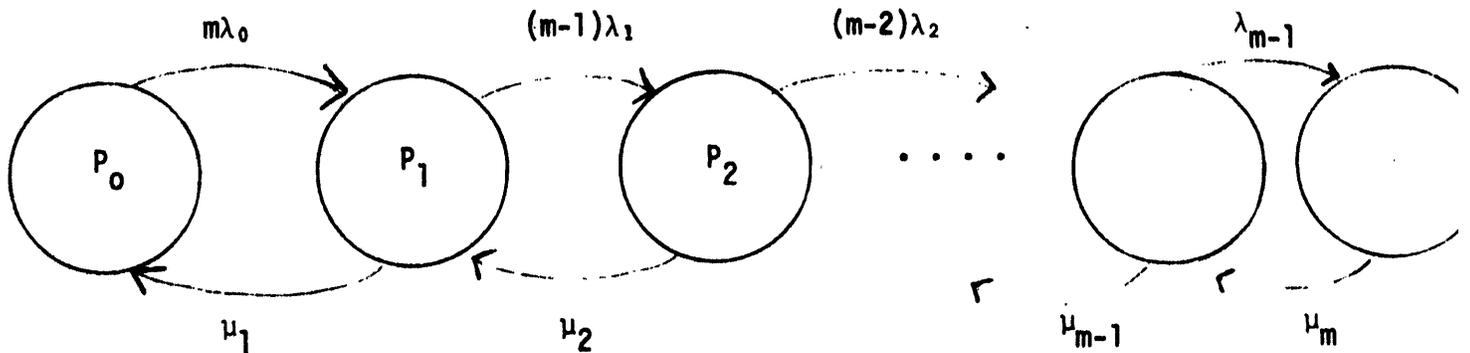


Figure 7

State Transition of Multi-VM Model

From this model, the steady state equations are [Drake 1967]

$$m \lambda_0 P_0 = \mu_1 P_1$$

$$[(m-i) \lambda_i + \mu_i] P_i = (m-i+1) \lambda_{i-1} P_{i-1} + \mu_{i+1} P_{i+1}$$

$$\mu_m P_m = \lambda_{m-1} P_{m-1}$$

$i < m$

The solution of the above set of equations is:

$$P_i = \left(\frac{\lambda}{\mu}\right) \frac{(m-1)!}{(m-i)!} (m-i+1) P_0 \quad (i = 1, 2, 3, \dots, m)$$

where

$$P_0 = \left[1 + \sum_{i=1}^m \left(\frac{\lambda}{\mu}\right)^i \frac{(m-1)!}{(m-i)!} (m-i+1) \right]^{-1}$$

The average response time for a request to the DB VM in this VM configuration is obtained by generalizing the analysis [Little: 1961] to this situation where there is queue dependency.

$$R' = \frac{\sum_{i=1}^m i P_i}{\sum_{i=1}^m \mu_i P_i}$$

Figure 8 illustrates the response time of a single request as a function of the number of modeling VM's.

Similar to equation of section 6.1,

$$T'_{\text{overhead}} = T_{\text{overhead}}$$

$$T'_{\text{wait-for-data}} = N.R.$$

T'_{model} is calculated similarly to the way T_{model} was calculated in

section 6.1. That is, $T'_{\text{model}} = N \cdot \frac{1}{\lambda}$. However, the λ 's vary.

Thus we take a weighted sum and get the following. (Note that if λ_i are constant, this reduces to the T_{model} of section 6.1.)

$$T'_{\text{model}} = N \cdot \left(\frac{\sum_{i=0}^{m-1} P_i \left(\frac{m-i}{m}\right)}{\sum_{i=0}^{m-1} P_i \left(\frac{m-i}{m}\right) \lambda_i} \right)$$

$$T'_{\text{total}} = T'_{\text{overhead}} + T'_{\text{model}} + T'_{\text{wait-for-data}}$$

Figure 9 illustrates the total time to execute a model as a function of the number of modeling VM's.

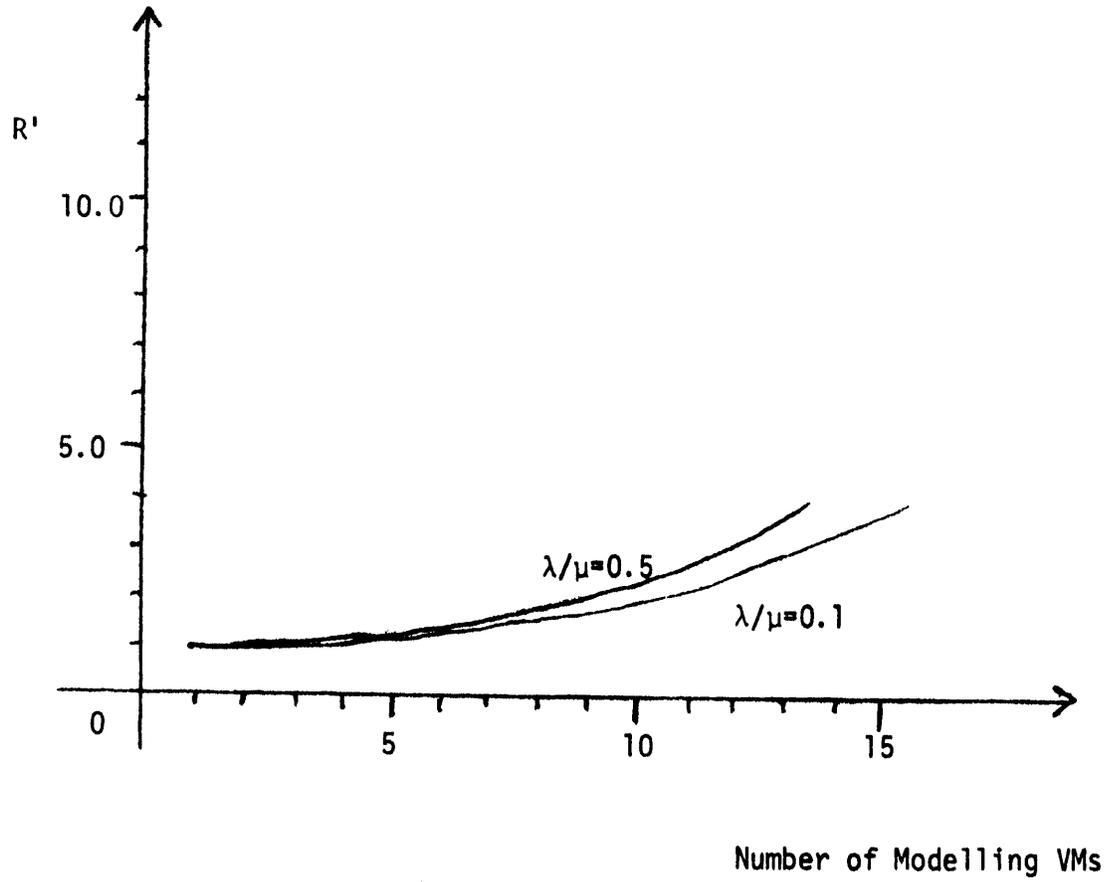


Figure 8

Response Time of a Single Request in a VM Configuration

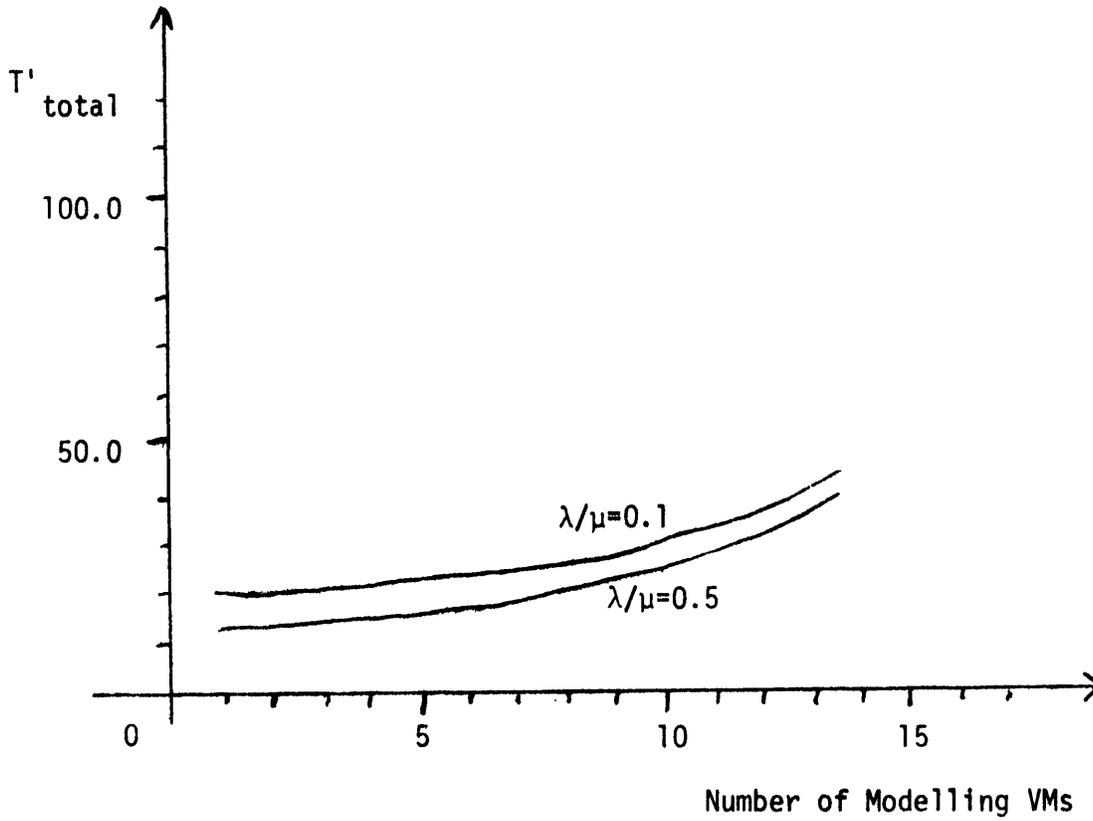


Figure 9

Total Elapsed Time in a VM Configuration

7. Techniques for Reducing Synchronization Overhead

The synchronization of the access and updates to the DB Virtual Machine is accomplished by what we call a spin lock. That is, if the modeling VM encounters a "locked" DB VM, then it must wait (in a queue); until the DB VM is unlocked, the modeling machine cannot do anything else.

The use of locks where the VM's must wait if encountering a lock, as we have seen, has an adverse effect upon system performance. Several techniques may be used to reduce this synchronization overhead, and the relative merits of each must be weighed.

One approach is to use a single lock (as we have done) to cover all shared in the single DB VM. data bases/ The alternative is to identify all separate data bases carefully and associate a separate lock with each.

There are many factors to be considered in choosing between a precise lock approach (i.e., a large number of separate locks) and an overall lock approach (i.e., one lock for all data bases). In the precise approach, considerable overhead is incurred in setting and resetting locks, even though the particular data base is not needed by any other VM. This multitude of locks also greatly complicates debugging.

In the overall lock approach (also called brute force), the lock may be on for long periods of time (up to 50 percent or more). This greatly increases the likelihood of software lock-out and the resulting slow response time.

8. Techniques for Reducing Effect of VM on Response Time

The basic reason for the degradation of performance due to VM is the fact that one real machine is being used to simulate several VM's. That is, one real CPU spends a little time on VM #1, then on VM #2, then on VM #3 and so forth. Thus, each VM only gets a fraction of real CPU time.

One method of increasing the amount of real CPU each VM gets is to increase the number of real CPU's. That is, use a multiprocessor configuration. Note all processors are executing instructions in the same memory.

The trade off is, the cost of the extra processors and their real effect. That is, each additional processor incurs some overhead and introduces a lower level set of locking problems. The lower level locking problem arises from having to lock "system" data bases whose access and updating must be synchronized (e.g., the system table which keeps track of what process the processor should be assigned to).

Treating each VM as corresponding to a separate process, we may perform a similar analysis [Madnick and Donovan: 1975] to determine the effectiveness of additional CPU's.

9. Summary

Running individual modeling facilities on separate machines all interfaced to a single database machine creates a total facility that is multiuser, interactive, suited to individual tastes and provides access to a single common data base. Simulating all these machines as virtual machines on one real machine provides a mechanism for fast and inexpensive communication between machines.

Multiple use of a single database creates the problem of synchronization of access and updates to that database. The spin lock provides a synchronization mechanism, however, at a performance cost in increased delays in response time. Figure 10 dotted curves give these times assuming separate real machines.

The performance implications of the use of VM can be seen in Figure 10, that is, the degradation because of VM becomes significant with large numbers of VM's.

Response time degradation due to a lock can be improved by partitioning the data base and using more than one lock. Degradation due to overhead associated with VM (one real processor simulating many) may be improved by adding more processors.

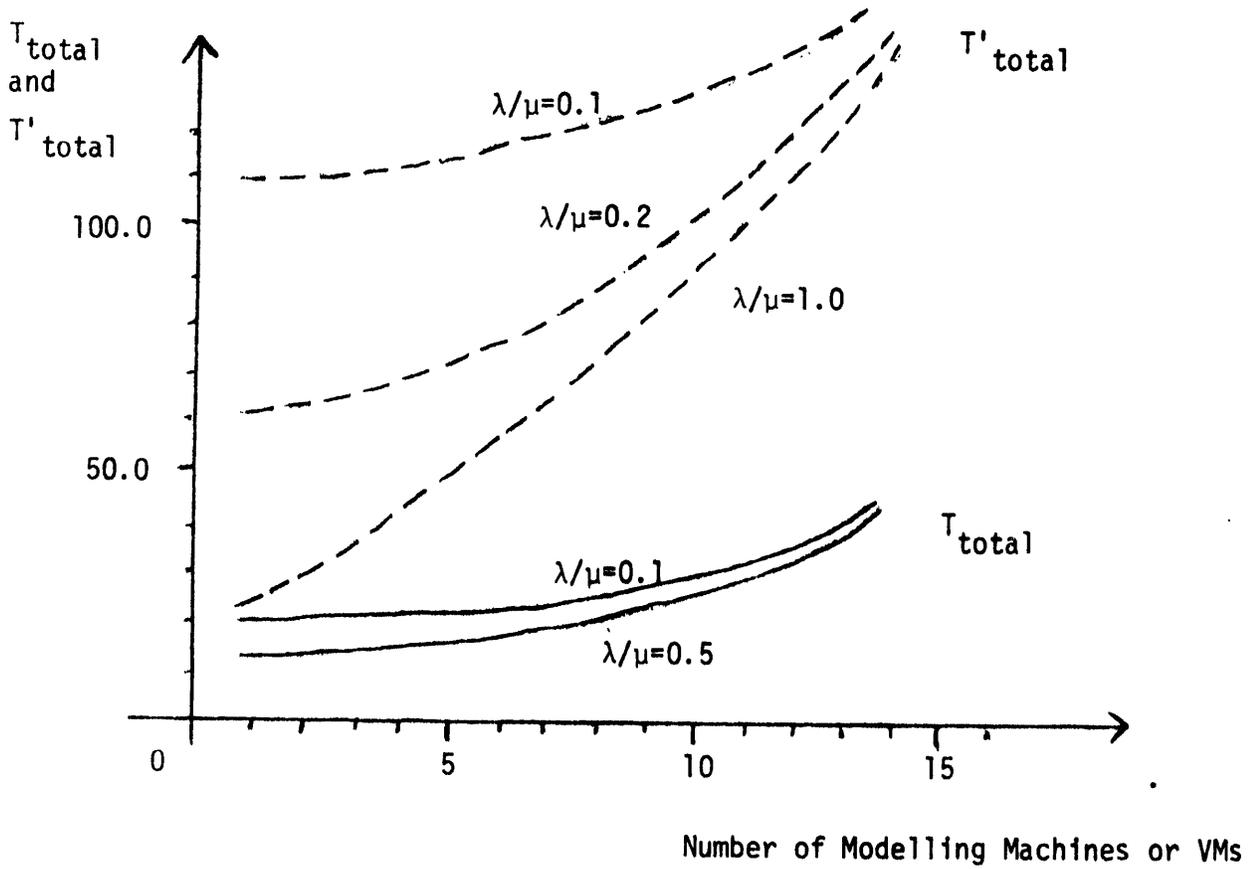


Figure 10

Comparison of Total Elapsed Times for a VM and a non-VM Configuration

Acknowledgment

Want to thank Professor Peter Chen for helping with the analytical construction and solutions to the performance equations and to Stu Madnick for helping in formulating these equations. We acknowledge the work of Marvin Essrig in applying this VM scheme in developing a system for leading Energy Indicators and in expanding the scheme for creating an environment where several different and potentially incompatible data management systems can all be accessed by the same models or facility.

We acknowledge the assistance of Drs. Stuart Greenberg and Ray Fessel of the IBM Cambridge Scientific Center for their assistance in implementing the scheme here. We acknowledge Louis Gutentag and the MIT students who worked with him for making the system operational.

REFERENCES

1. Buzen, J.P. and U. Gagliardi: "Evolution of VM Architecture", AFIPS Conference Proceedings, National Computer Conference 1975, Vol. 42, pp. 291-299.
2. Buzen, J. P., Peter P. Chen, and Robert P. Goldberg: "Virtual Machine Techniques for Improving System Reliability", Proceedings of the ACM Workshop on Virtual Computer Systems, (March 26-27, 1973).
3. Chamberlain, D. D., and R. F. Boyce: "SEQUEL: A Structured English Query Language," PROCEEDINGS 1974 ACM/SIGFIDET Workshop.
4. Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, No. 6, June 1970, pp. 377-387.
5. Codd, E. F.: "A Data Base Sublanguage Founded on the Relational Calculus," PROCEEDINGS 1971 ACM/SIGFIDET Workshop.
6. Coffman, E.G., Jr, and L.C. Varian; "Further Experimental Data on the Behavior of Programs in a Paging Environment," CACM, vol. 11, No. 7, pp. 471-474.
7. Denning, P.J.: "Virtual Memory," ACM COMPUTING SURVEYS, vol. 2, No. 3, pp. 153-190, September, 1970.
8. Dijkstra, E.: "T.H.E. Multiprogramming System," CACM, May 1968.
9. Donovan, John J.: SYSTEMS PROGRAMMING, McGraw-Hill, New York, 1972.
10. Donovan, John J., and Henry D. Jacoby: "A Hierarchical Approach to Information System Design," Report CISR-5, M.I.T. Sloan School Working Paper 762-75, January 1975.
11. Donovan, John J., Louis M. Gutentag, Stuart E. Madnick, and Grant N. Smith: "An Application of a Generalized Management Information System to Energy Policy and Decision Making: The User's View," Proceedings of Nation Computer Conference, Anaheim, CA, May 1975.
12. Donovan, John J., and Stuart E. Madnick: "Application and Analysis of the Virtual Machine Approach to Computer System Security and Reliability," IBM SYSTEMS JOURNAL, May 1975.
13. Drake, A. W.: FUNDAMENTALS OF APPLIED PROBABILITY THEORY, McGraw-Hill, New York, 1967.
14. Goldberg, R. P.: "Architecture of Virtual Machines," Proceedings AFIPS, National Computer Conference, vo. 42, pp. 309-318, 1973.
15. Goldberg, R. P.: "Survey of Virtual Machine Research," Computer, Volume 7, Number 6, (June 1974), pp. 34-45.
16. Hall, Robert: "TSP Manual," Harvard Technical Report No. 12, Harvard Institute of Economic Research, Cambridge, MA, April 1975.
17. Hatfield, D. J., "Experiments on Page Size Program Access Patterns, and Virtual Memory Performance," IBM Journal of Research and Development, vol. 16, no. 1, pp. 58-66, January 1972.

18. Hogg, J., and P. Madderom, "The Virtual Machine Facility -- How to Fake a 360," Internal Note, University of British Columbia and University of Michigan Computer Center, (1973).
19. IBM, "IBM Virtual Machine Facility/370: Introduction," Form Number GC20-1800, (July 1972).
20. Iverson, Kenneth E., A Programming Language, John Wiley and Sons, 1962.
21. Jacoby, Henry D.: "Final Report, FEA Leading Energy Indicator Project," Energy Lab Working Paper, M.I.T., April 1975.
22. Jacoby, Henry D., and John J. Donovan: "Final Report, New England Energy Management Information Systems," Energy Lab Working Paper, M.I.T., February 1975.
23. Little, John: "A Proof of the Queueing Formula: $L = \lambda\omega$," Operations Research 9, (1961), pp. 383-387.
24. MacAvoy, P., and R. Pendyke: "A Model for Natural Gas," Bell System Journal, 1975.
25. Madnick, S. E.: "Design Strategies for File Systems," M.I.T. Project MAC TR-78, October 1970.
26. Madnick, S. E.: "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach," Modern Data 2, 3 (March 1969), pp. 34-36.
27. Madnick, Stuart E., and John J. Donovan: Operating Systems, McGraw-Hill, New York, 1974.
28. Morrison, J. E.: "User Program Performance in Virtual Storage Systems," IBM Systems Journal, vo. 12, no. 3, pp. 216-237, 1973.
29. Parmelee, R. P., T. I. Peterson, C. C. Sullivan, and D. S. Hatfield: "Virtual Storage and Virtual Machine Concepts," IBM Systems Journal, vol. 11, no. 2, pp. 99-130, 1972.
30. Popek, G. J., and C. Kline: "Verifiable Secure Operating Systems Software," AFIPS Conference, 1974 NCC.
31. PRIME Computer, Inc., "DOS/VM Reference Manual," (1974).
32. Schober, F., EPLAN- An APL-based Language for Econometric Modeling and Forecasting, IBM Philadelphia Scientific Center, 1974.
33. Smith, Grant N., "Internal Intermediate Language, Version 2," M.I.T. Sloan School Report CISR-6, November 1974.
34. Satty, T. C.: Elements of Queueing Theory, McGraw-Hill, New York, 1961.
35. TROLL REFERENCE MANUAL, National Bureau of Economic Research, 1972.