Data Base System Evaluation

Harry L. Hill, IBM

The evaluation of data base systems embraces four very significant fields, the first being the design of resource management necessary to build into the product necessary performance attributes to make that product or system an attractive saleable item. The second part is the prediction of performance for a given configuration and workload. The third is the ability to measure the performance and confirm or deny the expectation obtained from the predictive process; and finally the ability to tune the system to accommodate changes made either in the configuration that exists or the user workload that is currently presented to the system.

To cover these four elements of data base evaluation, I have chosen to describe within this paper these topics:

1. Concepts of system performance
2. Performance and the development process
3. Predicting and measuring system performance
4. System performance tuning

1. CONCEPTS OF SYSTEMS PERFORMANCE

Let us look at some of the basic concepts behind system performance. The key question is one of systems performance sensitivity - the problem is always to find what is in the critical path. Fig. 1 describes clearly the approach that is taken, given that one can identify the bottleneck in the system; the key question is that if I remove that bottleneck, at what point and under what conditions do I hit the next one (because there is always a next one).

When we talk about the goodness of performance, i.e. how well a system performs, it is necessary to establish measures of goodness. We talk about performance in the following ways, as shown in fig. 2 - in terms of throughput, jobs per unit time, system data rate, number of accesses per second to a storage device, etc. There are perhaps more sophisticated and better ways of describing performance. For example, throughput per rental, dollars per second per access to a storage device, cost per job, cost per transaction. These latter measures of performance tend to be more revealing of the 'value to the user' as we sometimes call it, i.e. the cost performance trade-off.

It should be observed, as in fig. 3, that there are some very significant trends in performance evaluation. In the early days when we described performance in terms of component or device productivity, you will recall the measures of CPU goodness were in terms of add time, subtract time, multiply time, etc. We have emerged from that somewhat primitive measure of performance and today we talk about performance in terms of systems productivity, where the system is the sum of the hardware, the software and the workload effects. Tomorrow I am confident that we will be talking about systems performance not so much in terms of just the system but in terms of the user relationship to that system. I call that 'people productivity', where people's productivity is geared to maximise the objectives of a given enterprise or business. The computing system is then but one key element in meeting a business objective. This is particularly important for live terminal systems where the business of a company may be totaliy dependent on the availability and usability of the total system.

System performance is really best described in terms of the management of time spent waiting for systems resources. Fig. 4 describes a representation of systems resources because that is what performance is all about, the management of resources within a system allocated to a given profile of work. Every single system that has been constructed to date behaves in this way. The element of work is offered to the central processing unit or work engine and that work is executed by merging data with a program to a point where more data or programs are required. At that point in time the processing ceases and a request is queued in front of a storage device (i.e. a resource) in order to obtain additional data or programs to continue or complete the processing. When that work is completed, the processing engine proceeds on to another task. What we have is a serial processing engine operating on elements of work who's data and programs are queued

in parallel against system resources. By placing a 'meter' in the line between the storage and the queue for processing one can get a measure in terms of transactions per second or system data rate.

Fig. 5 shows a plot of system performance against the number of tasks, that is, the depth or level of multiprocessing and the consequence on the system of these tasks executing work. Notice that as you increase the number of tasks, the system performance increases to the point where a bottleneck is reached and I have chosen in this case to show the channel at the first bottleneck. If I were to add channels to the system I would relieve that bottleneck within the system and I would hit the next one which I have, in this case shown to be storage devices. So performances progress through 'ceilings' or bottlenecks.

Work that is presented to a computing system does not represent a constant load on all resources. In fig. 6 I have shown diagramatically a time varying workload effect on the system where the height of each pedestal represents 100% utilisation of that resource - notice that I am showing only 3 resources, a channel, a CPU and a drive device. The point is that not all of the time is any one resource the bottleneck, but the bottleneck changes from rsource to resource depending upon the demand of the time varying workload placed against it. When that resource is 100% utilised, it clearly forms a black mark on top of the pedestal, so by removing that bottleneck, that is, by putting a more powerful CPU in or a larger number of channels, this serves to improve the overall system performance. Clearly we are seeking an economic design where the number of black marks on top of the pedestal is reasonably balanced, that is, resources are not wasted. Fig. 7 depicts a system transaction rate versus a time varying workload, and a similar argument applies.

All transaction-based systems tend to behave in a similar way and fig. 8 shows a three-dimensional plot of response times versus real storage versus transaction traffic rate. Notice that as the real storage available for processing is decreased, the response time increases. Similarly, as the transaction traffic rate increases, the response time increases and all systems tend to behave this way. It should be realised that in virtual operating systems the decrease of storage causes an increase in paging rate. Under these conditions the CPU utilization generally decreases and the system gradually becomes I/O bound.

2. PERFORMANCE AND THE DEVELOPMENT PROCESS

As data base systems have grown and become sophisticated, it is necessary to achieve not only good performance, but predictable performance. This has to be built into the development process of the product. I should like to take as an example the development of storage which is a key resource in any data base system. Fig. 9 shows a typical development process which, in the early days of the computer industry, started off with the research and development of what I would describe as the basic parameters of the storage device. These parameters were offered to engineering groups who designed them into products and we developed on that basis the well-known disk drive. The drives were offered to the CPUs and were integrated with software systems which in turn were offered to industries to configure and use on behalf of that industry, and those industries designed those systems together with their applications to generate useful data processing facilities. The point is that in the early days we started off with the basic technology and we did what is described as a 'bottom-up' design - that is how the technology of the industry grew up. If we look today at the basic relationship of the direct access storage device (fig. 10) you will see that only certain combinations of those basic parameters are of interest to the systems designer, such as data rate and access times - areal density is frankly not very significant to the system designer. Similarly as block size decreases data rate becomes less important than access time. The consequence of this 'bottom-up' development process has been that we have decreased in a rather dramatic way the effective cost to the user of storage.

The decrease in storage cost as seen by the user is shown in fig. 11, i.e. the relationship between dollars per megabyte per month for a variety of products versus the year of announcement. In fig. 12 you will also notice the access rate characteristics where the accesses per dollar and the accesses per second are shown for the same range of products. If we are to look now at fig. 13 we will see that the storage technology spans a range of access times, storage capacities and cost per bit. This figure is interesting - observe the gap in the continum of storage devices. This gap occupies the same time domain as task switching in several of the medium and high speed processors. The technology for storage and data base systems is rich - rich in function and rich in performance and in cost choices. There is in fact sufficient technology to reverse the process and instead of doing a 'bottom-up' design, to take the requirements of modern applications and do a 'top-down' design (again see fig. 9), that is, to define the systems and the applications that are required in a business or enterprise and to map them into the technology.

3. PREDICTING AND MEASURING SYSTEM PERFORMANCE

The timely development of performance tools forms an essential part of developing a computing system. It has two major characteristics. One, it is important to be able to predict the performance of a complex data base/data communications system prior to either the hardware or the software being in existence and two, it is important that having predicted it and built it, it is important to be able to measure it and validate the prediction. The learning process is being able to describe differences.

The essential objective in developing performance tools is to be able to establish a discipline both for developers and subsequently for users of avoiding surprises in performance, since late discoveries are hard to correct. Fig. 14 describes this objective and describes the methods that are generally used to achieve them, that is, to develop models, to validate those models, to be able to track the instruction path length within a system and, as knowledge is gained, to be able to document that experience and construct a vocabulary that communicates both the predictive and the measurement processes. Fig. 15 shows the process. There are really two types of predictive capabilities, one is analytic and the other is simulative. In the measurement area there are two types of facilities required to produce the data necessary for measurement; one is hardware and the other is software monitors.

Measurement is both time consuming and expensive, therefore there has been significant emphasis and progress placed upon the development of models in order to determine the performance of a system, while measurement techniques are increasingly used to validate these models so that performance information and guidelines can be generated spanning a range of applications, configurations and workload demands. It should be recognised, however, that multiple sub-systems operating within one operating system are often hard to handle by conventional analytic means, and one is forced to consider hybrids of analytic and simulative techniques. It is most important that the developer or user of a model has clearly in his mind the question he wants the model to answer. Rarely is a general purpose model sensitive to questions that were not known at the time the model was developed.

It is perhaps useful to examine a data base/data communication system from a performance standpoint, and for this I have chosen IMS/VS and have constructed a flow chart for the main processing blocks of that system. Fig. 16 shows the flow of such a transaction;

notice that it divides itself into three major parts. The communication part where message switching and message queues are handled; the processing of that message against program and data and the multiple calls to that data base for that particular transaction; the completion of that transaction and the generation of the output message in the message queue, and the handling of that message through a terminal access method to a terminal. That is, if you like, the life of a transaction; it is born at the terminal where it enters the system and it dies at the terminal when the transaction is completed. If we were to place 'meters' in the lines joining those function to queues and libraries, etc., we could in fact measure the activity that is going on with the system. As we pass multiple messages into such a system, we see that the problem of performance resolves down to the allocation of resources, CPUs, channels, programs and data to handle the requirements of each different transaction. The job, then, is to define algorithms for using resources and for waiting for resources. These algorithms start with what priorities are associated with each transaction type and must include recovery strategies in the event that a resource, a data path or a queue discipline fails. Availability and performance are becoming increasingly dependent upon recovery schemes designed into the product.

There are really only two ways of improving the performance of a data base/data communication system. One is to shorten the transaction path length and the other is to provide either faster or parallel processing resources. It is thus often desirable to be able to calculate the number of instructions executed on behalf of an IMS transaction. Fig. 17 shows a typical appraoch to such a problem, where T is the total instructions executed for the IMS transaction, K1 through K5 are coefficients representing various IMS and VS releases; Q,U,N and C represent major parameters of most importance and significance in terms of overall systems performance.

Now if we were to take these transactions and were to apply values to those parameters, it is conceivable that one could divide the instruction processing capability of the machine by the path length of the transaction and come up with a theoretical maximum number of transactions per second that that resource could process, given that the processing unit was in fact the major bottleneck in the system. This has been done in fig. 18 and shows the difference in transactions per second processed for an 85% utilised 158 and 168. It should be clear that these are not measured values, they are predicted values, and are shown merely to demonstrate the sensitivity of system performance to changes in the key parameter values that affect it.

Fig. 18 is, then, designed to show the sensitivity of a system to changes in the major parameters that affect the system performance. Again this is not a measured environment this is a predicted environment and it is probably not possible to accurately reproduce this in a measurement environment without rigorously defining several other important system and user dependent factors. It does, however, also show on the same theoretical basis the difference in path length between an MVS system and an MVT system. Traditionally, it is thought that the systems that have higher sophistication have longer path lengths and whereas in general this is true, it is clear that in the MVS system, as the data base call structure becomes more complex, the difference in path length diminishes significantly in favour of MVS.

Independent of the investment made in developing and using models of the system, it is essential to measure the real thing as rapidly as possible. One method used in IBM is shown in fig. 19, where a simulated network is represented in both hardware and software and a data base is constructed to represent the application and system data bases. The simulated network is programmed to generate scripts at a given interval and with a given think time, or range of think times, such that the system under test appears to be loaded with transactions as though they were coming from real terminals. By the applications of suitable hardware probes and suitable software probes, we are able to measure the utilisation of resources occurring within the system under a variety of transaction rates, types and call structures. A typical measurement is shown in fig. 20, in this case an IMS/VS 1.0.1 system running under VS2 release 2. Notice the linear CPU utilisation as transaction rate goes up on this 158 CPU with 2400 Baud lines and 4800 Baud lines.

The measurement in question is designed to explore the sensitivity of line speeds to system performance. Note that in the 2400 Baud lines case, with ten lines, the line utilisation became a significant bottleneck in the system and this is evidenced by the response times starting to rise rather rapidly, whereas at 4800 Baud line speed, the response time is well contained.

System performance can be viewed in two ways and fig. 21 shows that we are either using a resource or we are waiting for it. Let us now take the flow chart (fig. 16) that we developed to show the life of an IMS transaction. Let us look at that flow chart with respect to the time we spend waiting for a resource, that is, waiting for a line, waiting

for buffers, waiting for a processing region, waiting for an application program to be brought in, waiting for I/O, that is, storage accesses to bring data or programs into the system, waiting for lines to handle the output message and waiting for services to transmit that message to the terminal. Let us also look at the amount of time using the resources. Fig. 22 shows, and it is drawn to scale, where if this were 8 inches long, the response time from beginning to end would be 1 second, making 3 loops around the DL1 call. It is also clear, as we approach a 100% utilised system, the units of processing occupy a smaller and smaller portion of the total response time. This chart shows the waiting time and processing time for only one transaction within a 75% loaded system.

4. SYSTEM PERFORMANCE TUNING

The goodness of performance then, of a data base/data communication system is balancing or tuning two things. It is balancing the supply of resources with the demand on them, because we are either waiting for that supply or we are using that supply. Fig. 23 shows this balancing scheme. If we have a high supply with respect to the demand, then we are wasting resources. If we have a high demand with respect to the supply of resources we are going to suffer poor response times. In general, performance is a user option since it requires the additon of resources and these generally cost money; but not always is that the case. In some cases, it is necessary and possible that the resources be tuned to meet the demand of the workload. Performance tuning is concerned primarily with the elements shown in fig. 24, being data base profiles, transaction profiles, profiles of the IMS system, of the processing requirements of the region, of the hardware and software configuration, of the overall teleprocessing configuration, and importantly, the use of tools to measure these resources.

Fig. 25 shows the primary factors affecting the performance and the design of the system. The number of transactions per second is typically in the range of 1 to 50, although within the next five years I am confident that you will see that range grow towards 200 transactions per second. In terms of EXCPs per call, we are looking today in the range 0.1 to 5 per data base call. In terms of calls per transaction, we typically find anywhere from 5 to 50 calls with several transaction types exceeding 50 and reaching close to 100 calls per transaction, so the data base designer is faced with designing a system of resources which can efficiently and economically accommodate the range of performance critical factors.

The tuning of data base systems is clearly a complex matter involving firstly an awareness of utilisation of resources, and secondly the understanding and knowledge about the sensitivity of changing the resource allocation to achieve an overall system performance level. The objective then is shown in fig. 26 - either minimise the transaction path length and/or invoke parallelism of key resources. The method recommended is firstly to quantify the profiles of the transaction and of the system; understand the behaviour of the system in response to changes in the workload; use software monitors to quantify that behaviour and resort to hardware monitors which do not interfere with the processing characteristics of the system; to define experiments to uncover and order the bottleneck; and to make changes, one at at a time, to the system and measure the effects. Only by measurements do we really get smart.

Performance tuning can be an iterative process because what one is trying to do is to optimise the utilisation of resources and match them against the workload. Frequently that workload is changing and one's job is not done until one has resolved the differences between what one expects, that is the expectation of performance, and what one has actually got. If there is significant differences between those two elements, then clearly there must be an explanation which always seems to lie in better understanding of what the system is doing. I mentioned the complexity of tuning a data base/data communication system. It is certainly not true that every one behaves differently. There are some typical causes of bottlenecks which are frequently uncovered and those really fall into three categories, as shown in fig. 27 - resources of a teleprocessing network - balancing of those resources and the selection of buffer sizes and message format buffers; the region resources, that is the amount of program loading that is done; the structure and the size of application programs; the structure and the size of the data base; the use of extended function within that data base structure; and lastly, the CPU resources, where its use is determined largely by the amount of system and user I/O and the use of bufferpool services.

Finally, I should like to discuss trends within data base/data communication system performance. Those trends really fall into three broad areas - trends in prediction, trends in measurements and trends in tuning. I think that over the next five years we are going to see generalised use of analytic tools for dedicated systems and some guidelines based on analytic tools for mixed systems. We are going to see the specific use of simulation and hybrid tools for mixed or complex systems. We are also going to see the availability of tools at an early point in the design of systems to help users choose

amongst different configurations which have different price performance characteristics.

In terms of measurement trends, we are going to see integrated software performance monitors, because basically performance is a user option and it is proper that the user understands what the system is doing and what choices he has to change it. Where a software monitor impacts the basic behaviour of the system, we are going to see integrated hardware built into the product to facilitate measurement and so be able to monitor the performance with little or zero overhead. We are going to see selective performance report generation, and we are going to see dynamic performance information and monitoring of key resources, so that information can be made available to a user to permit him to manage his system in line with some overall strategic direction that has known cost performance trade-offs.

Lastly, in performance tuning, I believe that we are going to see a family of tools available for the design of major components. That is, the design of TP networks, of data bases, of multiprocessing systems to permit the designer at an early stage to become familiar with the behaviour of those elements of the system that are likely to be a system bottleneck. We are going to see system-managed performance generation reports, and tuning controls that are made available on an open loop basis. It is conceivable that in the next five to ten years many of the tuning controls can be architected into a closed loop system so that the system is able to tune itself, and at this point I refer to tuning of the system in terms of allocating resources in accordance with a predetermined set of performance strategies. Some of these can be determined by the manufacturer and some will be determined and selected by the end user.

This concludes my presentation on the Evaluation of Data Base Systems.

Concepts of System Performance Sensitivity

The Problem: Find What's in the Critical Path, i.e., What's the Bottleneck

And . . . What's the Payoff When I Remove That Bottleneck and Hit the Next One.

Fig. 1 Because . . . There Always is a Next One

Performance Measures of Goodness

How Can We Talk About Performance?

Thruput (Jobs/Unit Time)

System Data Rate

Accesses/Sec

Terminals Supported

Terminal Response Time

Or Perhaps:

Thruput/Rental

$/Sec/Access

Cost/Job

Fig. 2 Cost/Transaction

Trends in Performance Evaluation

Notice the Trend from:
Component or Device Productivity

To

System Productivity

(System = Hardware + Software + Workload)

To

People Productivity

Fig. 3 (People Productivity = Maximized Enterprise Objectives)

A Representation of System Resources

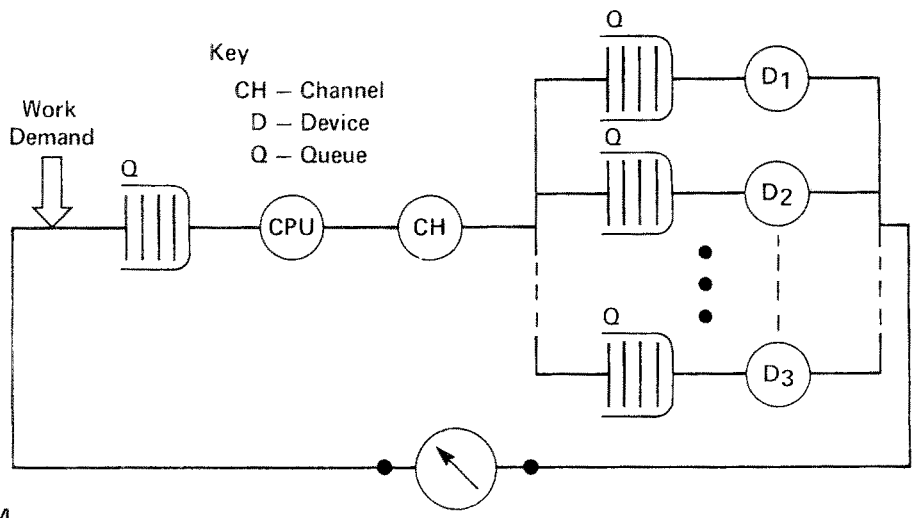


Fig. 4

A Way to Think About Bottlenecks

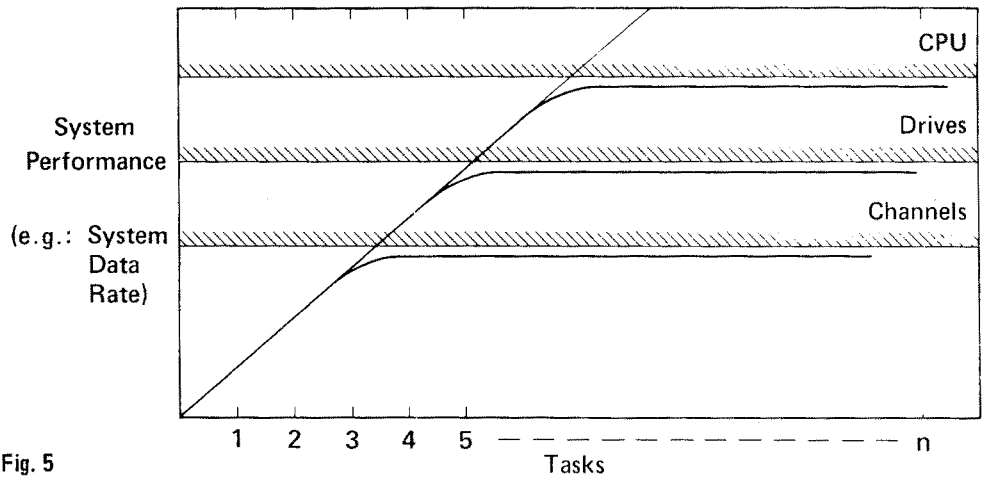


Fig. 5

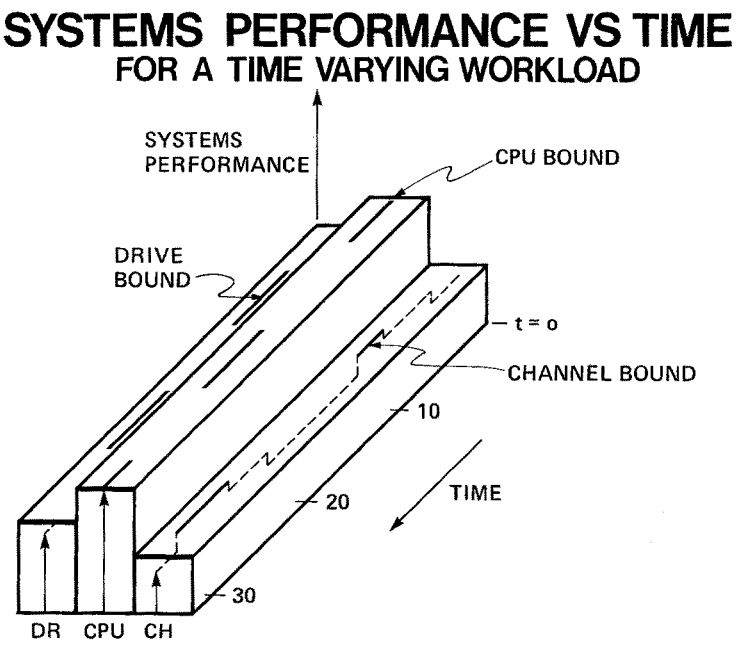


Fig. 6

TRANSACTION RATE VS TIME
FOR A TIME VARYING DBDC WORKLOAD

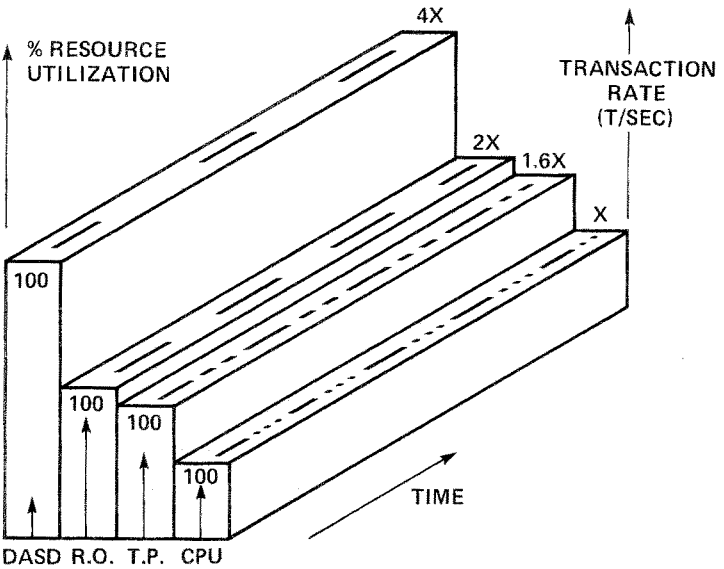


Fig. 7

DBDC PERFORMANCE RELATIONSHIPS

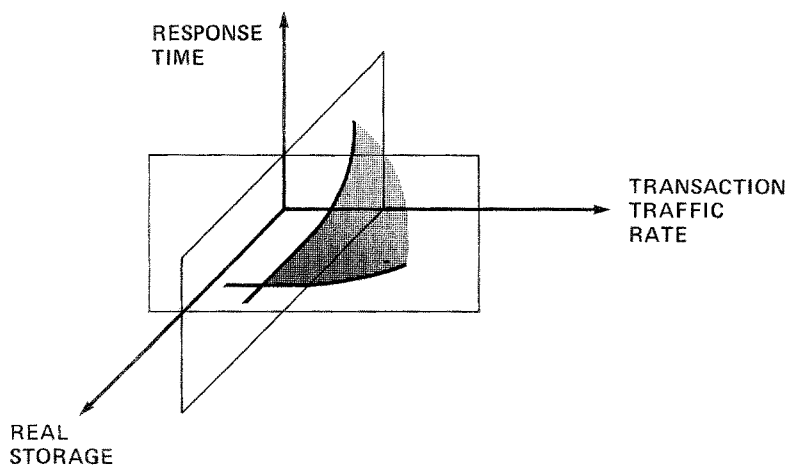


Fig. 8

The Development Process

A View of the Development Process

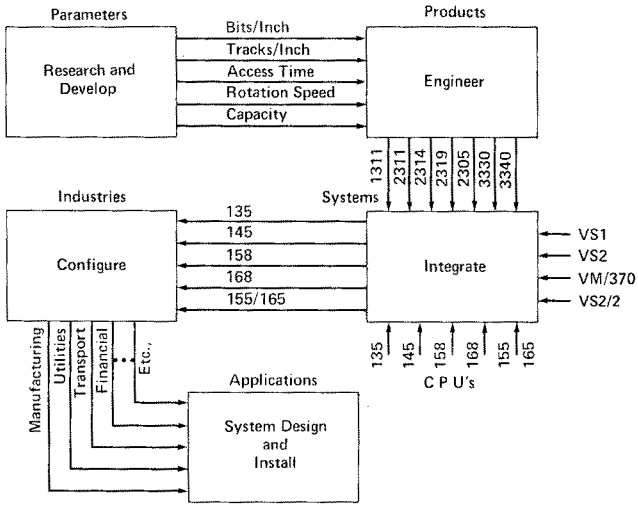


Fig. 9

DASD Parameter Relationships

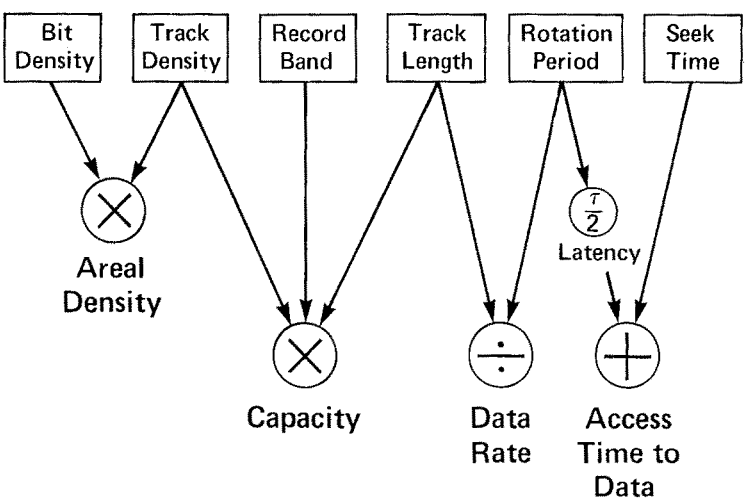


Fig. 10

The Cost of Attached Storage

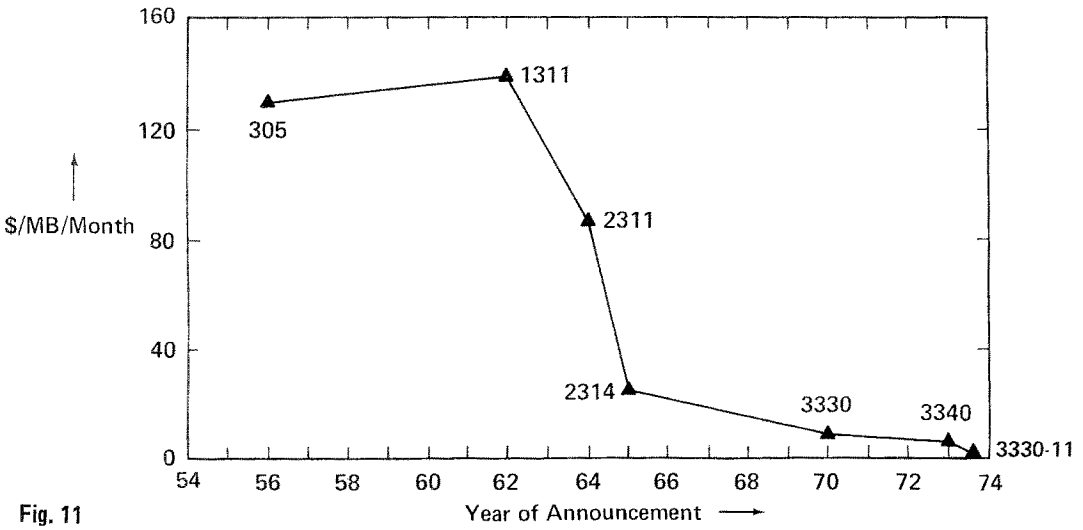


Fig. 11

Access Rate Characteristics

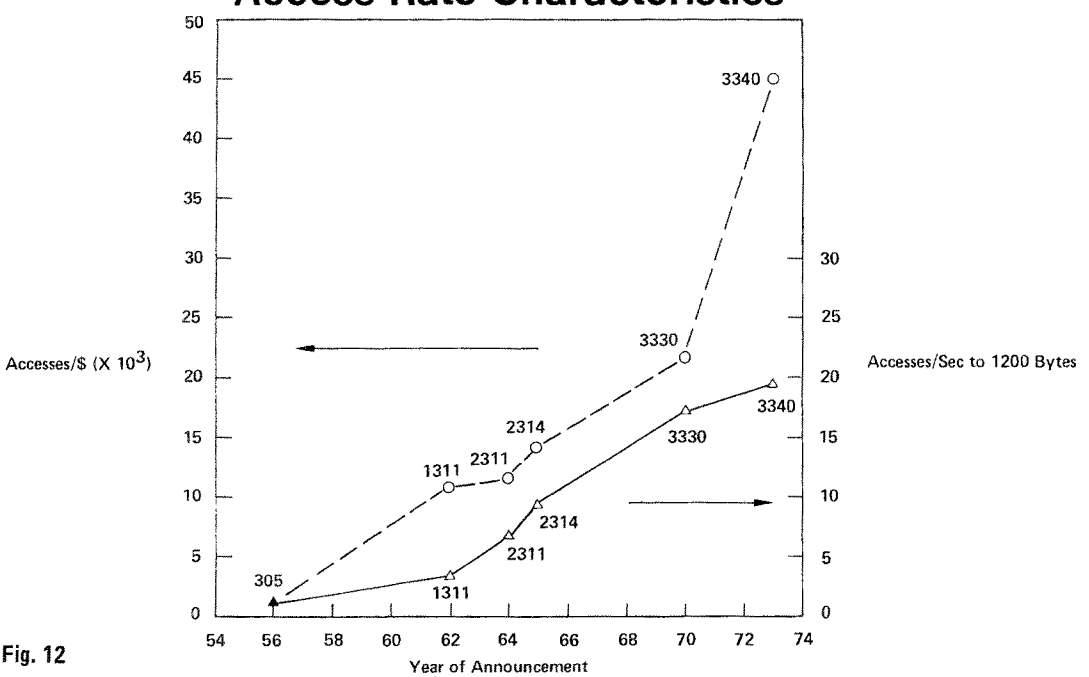


Fig. 12

Present Storage Technologies

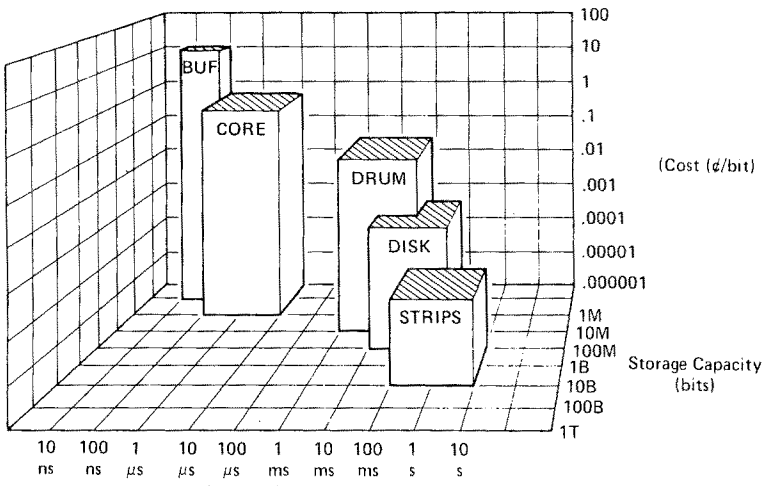


Fig. 13

OBJECTIVES AND METHODS

Objective

- DON'T CREATE SURPRISES IN PERFORMANCE – LATE DISCOVERIES ARE HARD TO CORRECT

Method

- DESIGN TOOLS (MODELS) TO ASK/ANSWER QUESTIONS IN A DISCIPLINED WAY
- DO IT EARLY TO INFLUENCE DESIGNERS
- SPECIFY AND TRACK PATH LENGTHS
- VALIDATE MODELS AND MEASURE TO GET SMART
- WHEN YOU'RE SMART – DOCUMENT IT

Fig. 14

PERFORMANCE TOOL DEVELOPMENT

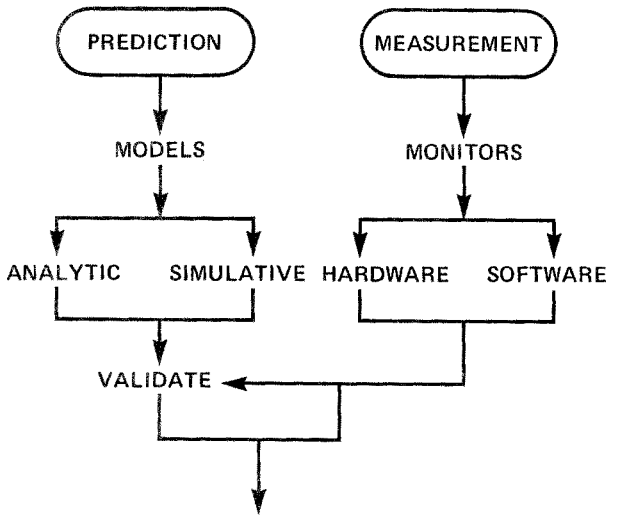


Fig. 15

MAIN PROCESSING BLOCKS OF A TRANSACTION IMS/VS

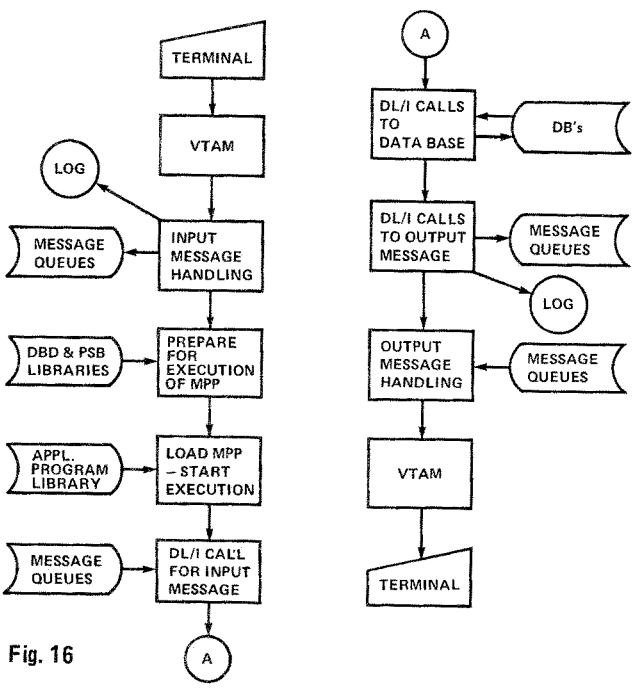


Fig. 16

IMS PATH LENGTH ANALYSIS

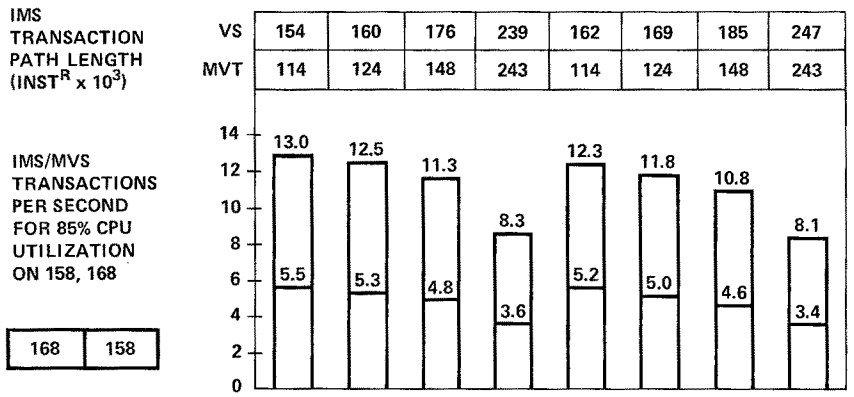
HOW MANY INSTRUCTIONS ARE EXECUTED
ON BEHALF OF AN IMS TRANSACTION?

$$T = (K_1 + K_1^1) + (K_2 \times Q) + (K_{3X} U) + N[K_4 + (C \times K_5)]$$

K_1 ---- K_5 ARE COEFFICIENTS REPRESENTING VARIOUS IMS AND VS RELEASES.
Q = FRACTION OF INQUIRY TRANSACTIONS
U = FRACTION OF UPDATE TRANSACTIONS
N = NUMBER OF DATA BASE CALLS/TRANSACTION
C = NUMBER OF DATA BASE IOS/CALL
T = TOTAL INSTRUCTIONS EXECUTED FOR ONE IMS TRANSACTION

Fig. 17

IMS PATH LENGTH ANALYSIS



VS	154	160	176	239	162	169	185	247
MVT	114	124	148	243	114	124	148	243
CALLS/TRANSACTION	3	5	10	30	3	5	10	30
IO'S/CALL	3.3	2.0	1.0	0.3	3.3	2.0	1.0	0.3
O/O INQUIRY	0.5	0.5	0.5	0.5	0	0	0	0
O/O UPDATE	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0

Fig. 18

PERFORMANCE MEASUREMENT ENVIRONMENT

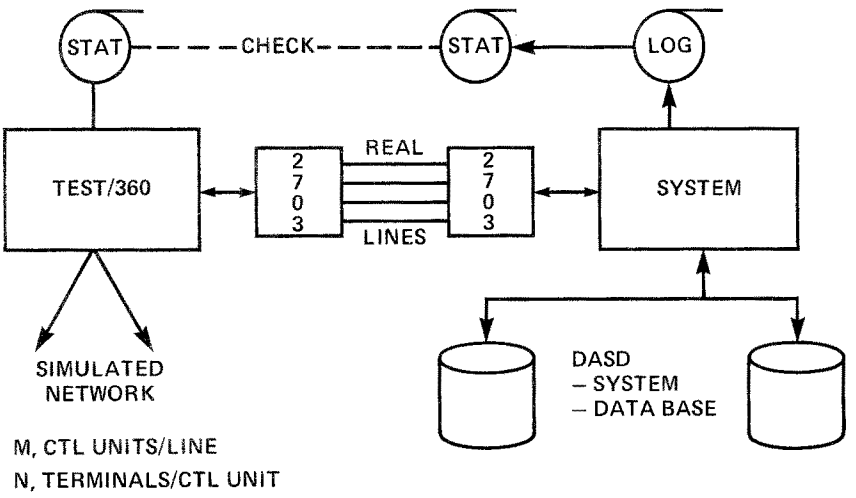


Fig. 19

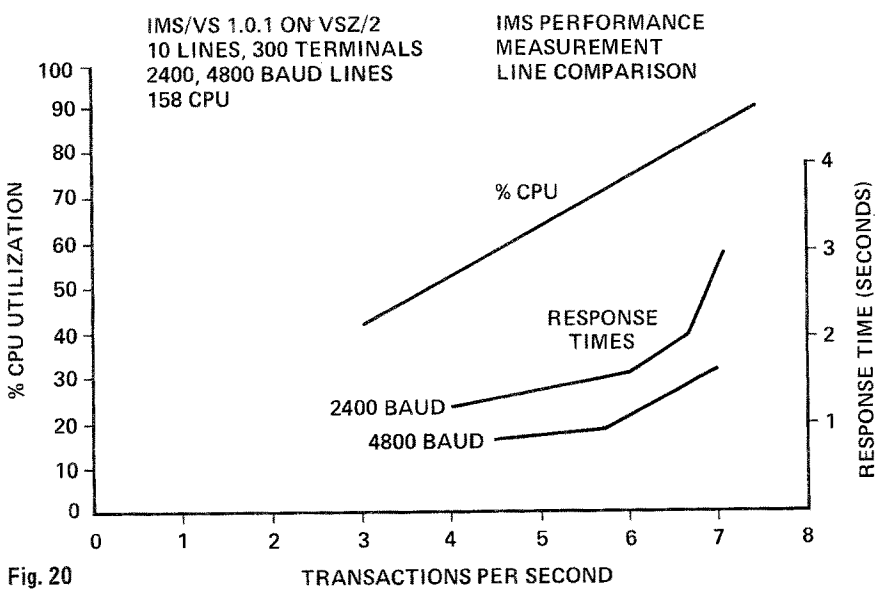


Fig. 20

WHAT IS PERFORMANCE

A SYSTEM OF
RESOURCES

(CPU, CHANNEL, DASD, TP, STORAGE, PROGRAM, QUEUE, LOCKS, . .)

USE OF
RESOURCES

(UTILIZATION)

WAITING FOR
RESOURCES

(WAIT/RESPONSE TIME)

DEFINE WHAT YOU MEAN BY PERFORMANCE

Fig. 21

TIMING AN IMS TRANSACTION

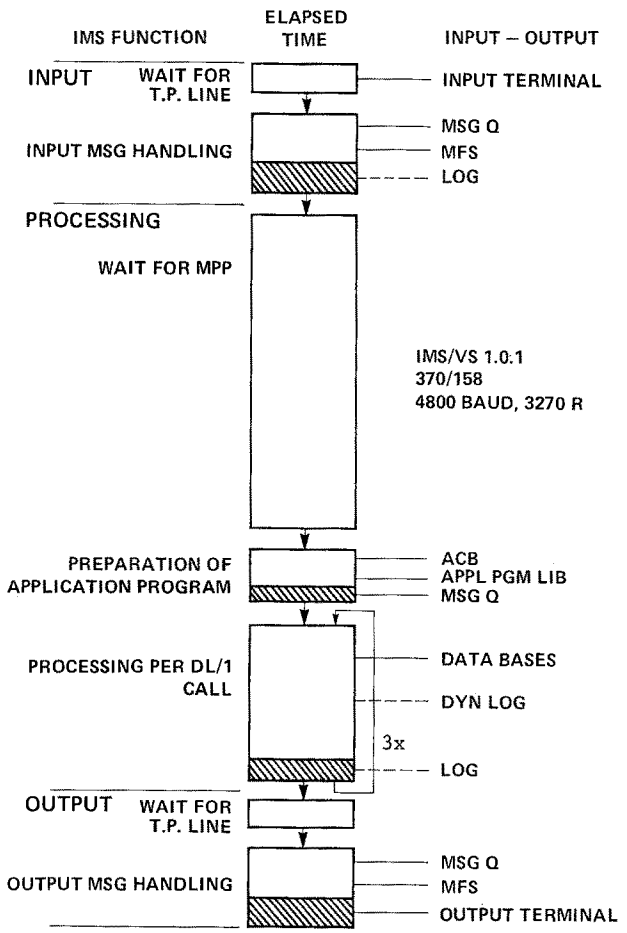
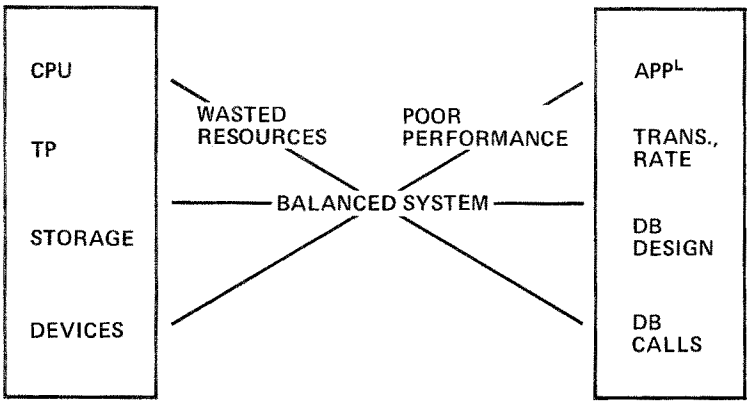


Fig. 22

DBDC PERFORMANCE TUNING

Supply ⟷ Resource ⟷ Demand



TUNING ⟶ BALANCE RESOURCE SUPPLY AND DEMAND

Fig. 23

DBDC PERFORMANCE TUNING

Primarily concerned with:

- DATABASE PROFILES
- TRANSACTIONS PROFILES
- IMS PROFILES
- MPP PROCESSING REQUIREMENTS
- HARDWARE CONFIGURATION
- OPERATING SYSTEM PROFILE
- TELEPROCESSING CONFIGURATION
- OTHER

and the use of tools to measure critical parameters

Fig. 24

PRIMARY FACTORS AFFECTING PERFORMANCE/DESIGN

PARAMETER	TYPICAL VALUES
– # TRANSACTIONS	1 – 50
– # EXCPS/CALL	0.1 – 5
– # CALLS/TRANS	5.0 – 50

Fig. 25

A DBDC TUNING APPROACH

Objective

- MINIMIZE THE TRANSACTION PATH LENGTH.
- INVOKE PARALLELISM OF KEY RESOURCES.

Method

- QUANTIFY PROFILES – TRANSACTIONS, SYSTEM CONFIGURATION AND PERFORMANCE GOODNESS.
- UNDERSTAND SYSTEM BEHAVIOR IN RESPONSE TO WORKLOAD.
- USE SOFTWARE MONITORS TO QUANTIFY BEHAVIOR (Δ TIME), MAYBE – HARDWARE MONITORS AND DETAILED TRACE.
- DEFINE EXPERIMENTS TO UNCOVER AND ORDER BOTTLENECKS.
- FORM IMPROVEMENT HYPOTHESIS, MAKE CHANGE, MEASURE EFFECT.
- DOCUMENT EXPERIMENT AND RESULTS. GET SMART.

Result

- OPTIMUM UTILIZATION OF SYSTEM RESOURCES TO MATCH WORKLOAD.
- RESOLVE DIFFERENCE BETWEEN EXPECTED AND ACTUAL PERFORMANCE.

Fig. 26

TYPICAL CAUSES OF DBDC RESOURCE BOTTLENECKS

TP RESOURCES	• BALANCING NETWORK LOADING • SIZE OF TP BUFFERS • SIZE OF MESSAGE FORMAT BUFFERS
REGION RESOURCES	• AMOUNT OF PROGRAM LOADING • STRUCTURE AND SIZE OF APPLICATION PROGRAMS • DATA BASE STRUCTURE AND # CALLS • USE OF EXTENDED IMS FUNCTIONS • AMOUNT OF I/O
CPU RESOURCES	• AMOUNT OF SYSTEM AND USER I/O • USE OF BUFFER POOL SERVICES

Fig. 27