Many existing monitors that are intended to assist in system tuning are based on the utilization approach which focuses on the active time of the system resources and activities and their users. This paper presents an alternative approach that is based primarily on the analysis of the contention in the system. The focus here is on the queuing delay time of the users and their activities when accessing the system resources.

Utilization and contention are two different ways of looking at the system. The two approaches complement each other, yet each may serve a different purpose or address different performance objectives. A prototype monitor was implemented on MVS (Multiple Virtual Storage) to produce the information necessary to continue investigations in contention analysis.

System contention analysis— An alternate approach to system tuning

by A. Yuval

Measuring the contention in a computer system, as part of a monitoring process carried out for the sake of system tuning, is not new. References 1 through 5 are only a few of such past uses. This is true not only for resources where utilization statistics do not apply, such as logical resources,[6] but also for physical resources, such as the CPU and I/O devices, where *contention along with utilization* is given.[7] Yet, these monitors are basically utilization-oriented,[8] focusing primarily on the active time of the system resources and their users. Some even determine the bottleneck resources based on the utilization statistics only.[1,9,10]

Contention analysis certainly deserves its own primary place in both the measurement and the evaluation phases of the Computer Measurement and Evaluation (CME) process. A contention-oriented analysis focuses on the *queuing delay time* of the users and their activities when accessing the system resources. The focal point is the scanning of users who are held (delayed) in their execution and the determination of the reason for the delay.

The definition of a user depends on the definition of the system monitored. The system can be the entire computing facility, a particular operating system, the supervisor of that operating system, or even a specific multitasked address space. A resource is defined as any service entity on which a user can be queued. This definition may *add* new logical resources that do not show up in utilization analysis (e.g., the page-in system service or the MVS, i.e., Multiple Virtual Storage, domains[11]). At the same time it may *exclude* many little-utilized or "private" resources on which contention does not occur.

The queuing (wait) time Q and the service (active) time S, when added together, account for the entire transaction delay time D.

$d = q + s$ for a specific service request

$D = Q + S$ for an entire transaction (task, job, etc.)

The first objective of contention analysis in looking at Q is, therefore, to *complement* the utilization type of analysis that measures S. Since detailed utilization information is readily available in many systems, through their accounting programs, a contention analysis is indeed the "missing brick." Such is the case in MVS, for instance, where the accounting system[12] gives detailed utilization information for the user, and the Resource Management Facility (RMF) gives the information on a system-wide basis.

Contention may also have a justification by itself, especially in highly multiprogrammed time-sharing systems. In such systems, Q is known to be very high compared with S (a high expansion factor; see Reference 13). Furthermore, S is much more difficult to change because it is both user-program- and device-dependent. The system programmer's main task in such systems is to minimize users' collisions and to maximize the chances for users' programs to get the resources they need as quickly as possible. Reducing Q can be a goal in itself in such systems.

Any implementation of the contention approach in an actual monitor should take into account that it is during periods of system saturation that we are primarily interested in contention delays. The monitor itself should therefore be as efficient as possible using minimum system resources and "locking" the system for the shortest period possible. Yet, it should give us enough information from which meaningful CME results can be calculated. Such an implementation seemed therefore quite important at the early stages of our study.

Prototype monitor and data produced

construction of monitor

It is quite desirable to let a monitor have two modes of operation: a low-overhead default mode and an extensive investigation

mode. The default mode should provide sufficient data on which to base a sound contention analysis. The extensive mode should be used to aim at special (or weird) periods (or phenomena) for which the regular default-mode reports are not enough. It was therefore decided to first implement the default mode and see how far it would take us. (In MVS the General Trace Facility, GTF,[14] can always be used for extensive analysis.)

In order to achieve the objective of low overhead, it was decided to implement the prototype based on state-sampling techniques[15,16] rather than the potentially more expensive intercept or event-driven techniques. (GTF uses intercept techniques.) In taking a sample, the default mode monitor should concentrate on the contention points in the system, try to collect information as detailed as possible with regard to the points, and *not* try to measure other "interesting" terms.

The data to be sampled is found in the operating system control blocks. The main function of the operating system is to satisfy users' requests for service from the resources. Any inability to immediately fulfill such requests is reflected in the system control blocks. By taking full advantage of these characteristics, the prototype monitor can be expected to collect detailed information with reasonably low overhead. This should be true for any "system" that manages users' requests for resources and that keeps track of the status of these requests.

In taking a sample, the monitor should first differentiate between users who are voluntarily idle (e.g., a user in "think time") and those who wish to use the system. (Throughout this paper the terms *nondemanding* and *demanding* are used to describe these two states of users.) The monitor then determines whether any demanding user is *waiting* because the resource it needs is not available. If any is found, it will produce one or more "contention records" whose exact format is described below. If no such users are found, it will produce a single record that says "no contention found."

The prototype monitor was found to be indeed very efficient with low overhead. The CPU time consumed and the required memory size were extremely small. Moreover, the monitor executes as a regular nonprivileged program and is therefore fully pageable and interruptible. The external storage required to record the data is also quite acceptable. Appendix A describes the prototype monitor in much greater detail, particularly its more interesting features and characteristics.

We now show how contention analysis within overall system analysis can be done using the data produced.

Figure 1 Contention record format

ID	resource status	queue length	holding user information	waiting users information

data produced: contention records

Although the basic approach is to look for *users* who are waiting, the contention output-records are summarized by *resource*. A separate contention record is produced for each resource on which at least one waiting user was found. Thus, each contention record contains the name of a *contended-for* resource along with the users who are *waiting* for it and the user who is *holding* it. It is sometimes desirable to show precisely what program and module are accessing what part of the resource. Such a breakdown is generally referred to as an *activity*. Figure 1 shows the general format of the output records.

The ID (identification) field denotes the resource class (or system component) to which this resource belongs, namely, I/O device, CPU, etc. The "resource status" field varies from one resource class to another. For example, I/O device type records contain the device address, the unit type, and the volume ID. For disk units, the status field also contains the cylinder and track addresses where the "holding user" was operating. The information given about the users (both the holding and the waiting) shows the user name, the user type (batch, time-sharing, etc.) and limited information on the activity involved.

The contention records generated in a single sample are preceded by a time-stamped control record which also contains some other statistics.

Relation to classical queuing theory

Throughout this paper, we try to conform to the standard queuing notation as described by Allen.[17] For those terms not mentioned there, we try to use similar notation:

R is the number of samples in the measurement period.
T is the period of measurement.
I is the number of end users in the system.
J is the number of resources (servers).
i is the index of the users in the system.
j is the index of the resources in the system.
λ is the service request arrival rate.
λT is the total number of requests for a resource.
q is the expected time a service request will wait to be served.

s is the expected service time.
d is the expected total delay time.

Let us also define Q, S, and D as follows:

$$Q = \lambda Tq \qquad S = \lambda Ts \qquad D = \lambda Td \tag{1}$$

The basic Little[18] relationship

$$d = q + s \text{ for each resource} \tag{2}$$

when multiplied by λT will give us

$$D = Q + S \tag{3}$$

We refer to Q, S, and D as "aggregates" to differentiate them from q, s, and d. Q, S, and D can be further defined by using indexing on i and j:

- Q_{ij} is the aggregate queuing wait time for the ith user on the jth resource.
- Q_i is the aggregate queuing wait time for the ith user across all resources (i.e., that portion of the session, or job-duration time, in which the user was waiting to get access to the system resources).
- Q_j is the aggregate queuing wait time caused by the jth resource.

In the same way we get the parallels for S and D.

We can also index q, s, and d to show all the possible indexing and summary relationships. Equation 2 can be indexed as follows:

$$d_{ij} = q_{ij} + s_{ij} \tag{4}$$

This more detailed indexing is neither easy to obtain nor does it appear in the literature.

terms relating to resources

In the queuing theory literature, there is normally no distinction between users, so the common equation used is

$$d_j = q_j + s_j \tag{5}$$

Multiplying Equation 5 by λT gives us

$$D_j = Q_j + S_j \tag{6}$$

If we multiply Equation 5 by λ only, we get the equation

$$\lambda d_j = \lambda q_j + \lambda s_j \tag{7}$$

However, this may be cast in the more familiar form

$$L = L_q + \rho \tag{8}$$

where L is the mean number of users in the server, L_q is the mean number of users waiting before the server, and ρ is the server utilization.

In exactly the same way that Equation 5 is used to compare different servers on the "micro" level regardless of their different behavior, Equation 6 can be used to compare them on the "macro" level. D_j shows the total delay time, caused by the jth server, on the entire workload.

terms relating to users

Equation 4 can be very useful from the user's point of view, but, as earlier stated, it is quite difficult to obtain. However, information in the form

$$D_{ij} = Q_{ij} + S_{ij} \tag{9}$$

can also show how much time the users spend waiting for and using each of the resources.

The equation

$$D_i = Q_i + S_i \tag{10}$$

shows the overall elapsed time, contention time, and utilization time for the ith user over *all* resources.

contention (*H*) term

The time during which a resource is in contention, namely, the time when the number of users waiting (L_q) before the resource is nonzero, is of special interest in contention analysis. Let us add the following notation and definitions:

$P_w j$ is the probability that the jth resource is in contention; i.e., L_q is not zero.

H_{ij} is the aggregate contention time on the jth resource while the ith user was using it (i.e., "caused" by the ith user).

H_i is the aggregate contention time caused by the ith user across all resources.

H_j is the aggregate contention time for the jth resource.

The equation

$$H_j = P_w j T \tag{11}$$

shows the relationship between H and P_w.

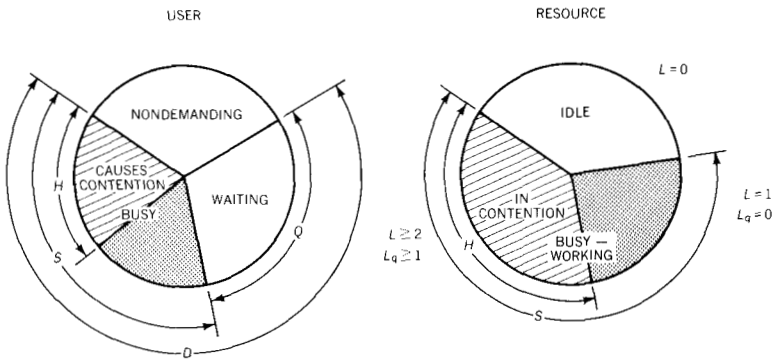
The diagrams in Figure 2 show the overall possible states for both a user and a resource in the above terms.

Measured terms

Three terms are directly measured by the prototype sampling routine:

$L_q j$ is the average number of users waiting before the jth resource.

Figure 2 User and resource possible states



R_{ij}^{Q} is the number of samples in which the ith user was found to be waiting for the jth resource.

R_{ij}^{H} is the number of samples in which the ith user was "holding" the jth resource, i.e., causing it to be in contention.

The probable value of Q_{ij} is obtained from R_{ij}^{Q} by

$$Q_{ij} = (R_{ij}^{Q}/R)T \tag{12}$$

In a similar way H_{ij} is obtained by

$$H_{ij} = (R_{ij}^{H}/R)T \tag{13}$$

terms not directly measured

Two terms are not directly measured by the prototype monitor: λ and S. Other information which is "missing" is the ability to go down the user control blocks and identify the exact *activities* involved.

These three elements were deliberately left out in the default mode. They all are good candidates for the extensive mode. The impact of not having this information on both the evaluation and the tuning action phases is discussed below. In practice, knowledge from other sources is used to supplement any missing information.

The analyzed (evaluated) data

The matrix in Figure 3 shows all the measured contention terms in summary as well as in detail for each user and resource in the system. The matrix contains the main data required for contention analysis. One can quickly determine which resources are creating contention (high Q_js on the rightmost column), which users cause this contention (high H_{ij}s across the jth resource row), and which users suffer from it (high Q_{ij}s on that row). From the user point of view, each column shows the "contention pro-

Figure 3 The users-resources contention matrix

RESOURCES	USERS: A	USERS: B	USERS: C	TOTAL
1	Q_{A1} H_{A1}	Q_{B1} 0	Q_{C1} 0	Q_1 H_1
2	0 0	0 H_{B2}	Q_{C2} 0	Q_2 H_2
TOTAL	Q_A H_A	Q_B H_B	Q_C 0	

file'' for a specific user, namely that part of the contention in the system which affects that user. Each user can thus determine the resources and the other users that are the cause for the delays in his or her workload.

Figure 3 shows the contention matrix at one point of time (summarized across some time interval). Remembering that each sample taken is time-stamped, one can have this matrix summarized for any desired time interval within the sampling interval and get a time-series analysis for each of the elements in the matrix. (This type of analysis was done in the prototype.)

Appendix B gives a summary of the main reports produced by the prototype monitor along with the practical tuning results that users of the prototype have experienced.

An extension of the contention matrix (where Q and H terms are shown) into an overall delay matrix (where Q, S, and D are shown) requires the knowledge of S or the ability to develop S from H. This information can be obtained in any one of the following ways:

1. By using queuing theory relationships such as the following:

   $$\rho^2 j = P_w j \tag{14}$$

   for the M/M/1 queuing system.[19] To get from $P_w j$ to H_j we use Equation 11, to get from ρ to S we use the rightmost part of Equations 6, 7, and 8. Thus:

   $$S = \sqrt{HT} \tag{15}$$

2. By using the relation which is always true:

   $$P_w j \leq \rho \leq 1 \tag{16}$$

   and henceforth also

   $$H \leq S \leq T \tag{17}$$

The higher the value of H, the more accurate the approximation of S. Also, in a comparable study of two resources, one may be able to determine whose D is bigger without knowing its exact value. This is again when H is big and also when Q for one resource is far greater than Q for the other.

3. By invoking the extensive mode of the monitoring.
4. By using information about S already available from existing sources such as the accounting system in Reference 12.

From evaluated terms to tuning actions

The effectiveness of the evaluation phase is determined by its ability to immediately lead to the required tuning actions. Some tuning actions may be impractical or expensive to carry out, but then at least management knows that alternative actions (e.g., administrative, capacity planning, etc.) should be pursued. This section will briefly show how contention analysis quickly leads to the appropriate tuning actions.

classification of tuning actions

Both user-program and system-wide tuning can be done by either speeding up the "biggest" activities or by executing as many activities as possible in parallel. Figure 4 summarizes the tuning actions for a system-wide, multiprogramming case. (Appendix C shows an example of MVS structured along the lines of Figure 4.)

significance of occurrences

It is quite useful to set some thresholds for the values of the terms in the contention matrix (Figure 3). Any Q or H value that goes above its threshold will be called *significant*. By carefully analyzing the information in the contention matrix, one can quickly get to the class of tuning action required.

quick way from reports to actions

For system-wide tuning the decision table shown in Figure 5 can be used. A similar table can be built for the user-program tuning actions. It is interesting to note that many actions which at first glance seem to be applicable only for system-wide tuning are quite applicable for user tuning too. Such are actions 2.1 and 2.2.[20] Action 2.3 is indeed system-only.

For both user and system tuning, the carrying out of actions 1.1 and 1.2 may sometimes require the extensive mode that shows the exact activities (e.g., operating systems modules) involved. Actually, in many cases this was found to be unnecessary.

When action 2.2 is considered for functionally equivalent resources that have a different s_j (namely, one resource is much faster than the other), Q_j or even D_j are not enough, and one must know the individual d_j too. It could be that

$$D_1 > D_2$$

Figure 4 System tuning actions in multiprogramming

1. Activities enhancement.
   1.1 Reprogramming of code.
   1.2 Resource restructure.
2. Parallel processing enhancement.
   2.1 Increase the MPL (Multiprogramming Level)
   2.2 Allocation algorithm improvement: Spread work between functionally equivalent resources.
   2.3 Dispatching algorithm improvement: Ensure high priority to the least demanding user.

Figure 5 System-wide tuning action decision table

What do we see in the matrix?		*What action to take.*
Significant H_{ij} in a row	⟶	1.1, 2.2, 2.3
Significant Q_j and no significant Q_{ij} on row	⟶	1.2, 2.2
No significant Q_js	⟶	2.1

and yet

$d_1 < d_2$ (and $s_1 < s_2$)

due to

$\lambda_1 >> \lambda_2$

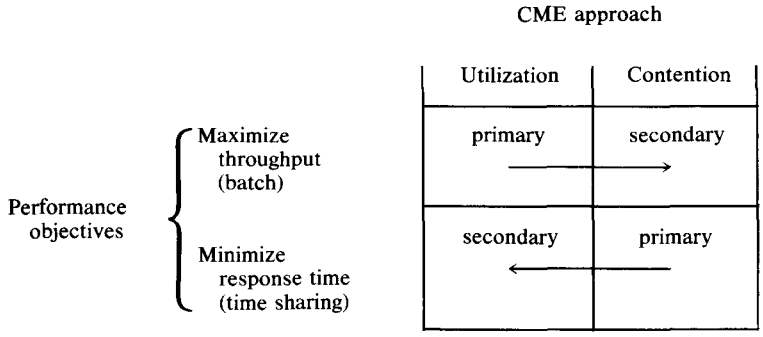
The shift of work from resource 1 to resource 2 could be a mistake. The extensive mode is again required to measure both λ and S so that d_1 can be compared with d_2. See Reference 21 for further discussion of this point.

Summary

In theory as well as in practice, *contention analysis* emerges as a new, interesting approach to both system and user tuning. The prototype was found to be low in cost and very simple both in its implementation and in its use. It provides affluent information that directly relates the users and resources associated at each contention point. The ability to provide contention information for each individual user in the system was found to be extremely valuable.

There are three practical reasons for the performing of contention analysis in one's system:

Figure 6 CME approach for performance objectives



1. Contention information nicely complements utilization information which, in many installations, already exists.
2. Contention analysis may be a target in itself, especially in highly multiprogrammed time-sharing systems. Contention analysis misses only those cases where a user manages to use a resource without causing delays to other users. Such cases are indeed less important in those systems.
3. For certain cases, the queuing time (Q) and the time in contention (H) do give a good approximation of the overall delay time (D). Contention analysis covers, in these cases, the entire "picture," and complementing it with utilization data may be unnecessary.

Basically, both contention and utilization information are needed in order to get the "whole picture." Yet, two different approaches are conceivable: one that looks primarily at utilization and, when required, looks at contention, and the other one that looks first at contention and, when required, at utilization too. Keeping in mind the *performance objectives* (which sometimes are simply forgotten), we can see in Figure 6 the relationship between utilization and contention.

ACKNOWLEDGMENTS

The author warmly thanks Dr. P. Capek for his constant encouragement and guidance. Appreciation also goes to Messrs. M. Kienzle, G. McQuilken, and M. Rimon, with whom I had fruitful discussions. The author is especially indebted to Mr. M. E. Drummond, Jr. without whose help this paper would have never been written.

In the writing of the prototype monitor, the author received valuable help from the MVS development people at IBM Poughkeepsie. Mr. M. Kienzle wrote most of the prototype report programs, and Mr. A. Birman helped in the understanding of the MVS control blocks. The author wishes also to thank some of the early and

Figure 7 System components (record types) and their resources

Record ID	*System component (record type)*	*Resources*
1	CPU	CPU (one resource in MP)
2	ENQ	All symbolic resources
3	Channel	All logical channels
4	Control unit	All I/O control units
5	I/O device	All I/O devices
6	Lock manager	CMS, ENQ/CMS locks
7	SRM	Domain, swap-in
8	RSM	Page-in, deferred queue
9	ASM	IOEs delayed

ENQ—The enqueue macro used to synchronize references to logical resources
CMS, ENQ/CMS—The two global spin locks
SRM—System Resources Manager
RSM—Real Storage Manager
ASM—Auxiliary Storage Manager
IOE—ASM I/O elements

enthusiastic prototype users for relating back the experience they gathered. This work was done at the IBM Thomas J. Watson Research Center while the author was on a sabbatical leave from the Weizmann Institute of Science, Rehovot, Israel.

Appendix A: The monitor construction and characteristics

The aim of this appendix is to demonstrate the *universality* of the contention analysis concept by showing how a prototype monitor for the MVS environment was built and operated.

The prototype is a normal, nonprivileged program which uses the timer to pause between consecutive samples. The only recommendation is to make it nonswappable and to give it a high dispatching priority. It can operate in three modes: as a batch job, as a started task, and through TSO (Time Sharing Option, the time-sharing subsystem).

The monitor looks at the system from the MVS supervisor (task management) point of view.[22] The *users* are therefore both address spaces and their tasks. The *resources* are derived from the main system components that comprise the MVS supervisor. Figure 7 summarizes these system components and their resources. (For a detailed explanation of the terms, see References 11 and 22.)

The monitor executes completely as a nonprivileged user program. It does not require supervisor state, protection key 0, or fixed pages, nor does it ever hold any locks or execute in a disabled state. The system control blocks may change while a

sample is being taken. The monitor solves this problem by continuously performing logical checks on both the system control blocks and the data produced. Very rarely a program-check interrupt can also occur. The monitor handles that by means of the STAE macro. For both types of fault, logical and program-check, the monitor simply drops the sample and continues without delay to perform another one.

The observed statistics of one faulty sample per 400 to 1000 good samples (see below) suggests that other monitors should stay away from "locking" techniques. It also demonstrates the feasibility of outboard monitors, which cannot easily synchronize themselves with the system. Fligliuzzi[23] shows a different interesting technique for the implementation of a nonlocking monitor.

Other interesting features and statistics of the prototype are as follows (Most of the statistical results are from runs made on the MVS machine at the IBM Thomas J. Watson Research Center, normally in the afternoon when the system is most loaded. Some statistics were cross-checked in some other IBM installations.):

1. The monitor is written in the Assembler language.
2. Program size is 12K including buffers.
3. The sampling cycle time ranges from 0.1 to 9.9 seconds.
4. One sample typically takes between 1.5 and 4.5 milliseconds of CPU time. Using a sampling cycle of one second and including the time to write to the external file, we anticipate an overhead of 0.5 percent on the CPU.
5. With an output block of one third of an IBM 3330 storage device track, a sampling cycle of one second and an average of six records per sample, eight cylinders of a 3330 per hour are required for output.
6. The highest fault rate found, i.e., aborted samples because of either logical or program-check errors, was one every 400 valid samples. One error in 1000 valid samples was the average in peak-time runs.

Appendix B: Prototype reports and related experience

The monitor post-processor produces four reports:

1. The General Contention Report.
2. The Time-Series Histogram.
3. The User's Wait Profile Report.
4. The Disk Seek Analysis Report.

The General Contention Report is arranged in a hierarchical manner and is further divided into the subreports that are described below.

Figure 8 Report 1.1, Overall System Contention

14:45 07/NOV/78—16:00 07/NOV/78 SAMPLES 4500

	Waiting	*Demanding*	*Working*	*Percent wait/demand*
Address spaces	5.8	26.7	20.9	22.0
Tasks	6.5			

System component (1)	*Count* (2)	*Average no. per sample* (3)	*Av. tasks waiting* (4)	*Waiting* (5)
CPU	3510	0.78	3.39	2.63
ENQ	3531	0.78	1.21	0.95
Channel	744	0.16	1.27	0.21
C-Unit	89	0.02	1.05	0.02
I/O devices	3693	0.82	1.62	1.33
CMS	75	0.02	2.51	0.04
SRM	1620	0.36	1.17	0.41
RSM	2432	0.54	1.66	0.89
ASM	41	0.01	1.84	0.02

Report 1.1, Overall System Contention, in Figure 8, shows the contention on the main system components and a summary of the waiting, demanding, and working users. The top line (which appears in all the reports) shows the time and date when the sampling started and ended and the number of samples taken.

The next part of the report shows that for the period of observation there were on the average 26.7 users (address spaces) who wanted to use the system (demanding); 5.8 of them were delayed because of some contention. Thus 22 percent of the demand is not fulfilled because of contention. The 5.8 address spaces waiting correspond to 6.5 tasks waiting (which is the sum of Column 6). The demanding minus the waiting spaces are referred to as the working address spaces and are equal here to 20.9.

The main part of the report consists of the following:

- Column 1 is the resource (as defined in Figure 7).
- Column 2 is the number of contention records pertaining to this system component.
- Column 3 is the result of dividing Column 2 by the total samples. It shows the frequency of appearance of contention on that component.
- Column 4 is the average number of users waiting at the time of contention.
- Column 5 is the average number of users waiting at the entire period of measurement. It corresponds to the multiplication of 3 by 4.

Figure 9 Report 1.2 for the CPU resource

Task holding (6)	*Type* (7)	*Count* (2)	*Percent of resource* (8)	*Cum.* (percent) (9)	*Av. tasks waiting* (4)	*Wait ST* (percent) (10)	*Wait TSO* (percent) (11)	*Wait batch* (percent) (12)
JES3	S	814	18	18	5.01	23	42	35
TCAM	S	489	11	29	4.23	9	51	40
STEP6	B	426	10	39	1.72	0	2	98
SUPPORT	T	283	6	45	3.39	6	46	48

Figure 10 Time-Series Histogram

No.	*CPU*	*ENQ*	*Channel*	*I/O*	*RSM*	*SRM*	*Wait*	*Dem.*	*Work*	*%W/D*
1	TTBBB		22	222	LL		12	25	13	48
2	STTBBB			2	LLC		10	24	14	42
3	TTBB	B	72	722	LC	4	13	27	14	48
4	TB	BB	722	67222		45	14	27	13	52
5	B	TBB	2	722		445	11	26	15	42
6		TTBB		2		4455	9	25	16	36
7	BB	B		72	LL	45	9	26	17	33
8	BB	B		2	L	45	7	25	18	28

The nonblank characters in the histogram are defined as follows:
For CPU and ENQ: S—started task, T—TSO, B—batch job.
For Channel and I/O device: the channel ID.
For RSM: L—local page fault, C—common area page fault.
For SRM: The domain ID.

Report 1.2, User Holding within Resources, is produced for each contended-for resource. It shows the distribution of the holding (contention-causing) users within those resources. The example in Figure 9 shows Report 1.2 for the CPU resource.

New column headings are introduced. Column 6 is the holding task. Column 7 denotes the type of holding task: started task (S), TSO (T), and batch job (B). Column 8 shows the percentage of the count of this holding user from the total counts of this resource. Column 9 is the accumulation of Column 8. Columns 10, 11, and 12 show the distribution (in percentage) of the waiting users among the three workload groups: started tasks (ST), TSO, and batch. For each line (user) these three columns should sum up to 100 percent.

The Time-Series Histogram, Report 2, in Figure 10 is divided into two parts. The left side is a histogram which shows the contention by system component on a time-series basis. Each line corresponds to a sample. In each column, which corresponds to one system component, any nonblank character denotes one user waiting.

Figure 11 Report for the master scheduler

Resource name (13)	*Count* (2)	*Percent of total* (8)	*Cum.* (percent) (9)	*Holding ST* (percent) (10)	*Holding TSO* (percent) (11)	*Holding batch* (percent) (12)
Device 240	691	67	67	12	52	36
Page-in	71	7	74	—	—	—
Channel 2	69	7	81	—	—	—
ENQ SYSIKJBC	56	5	86	0	100	0
Total	1021	100				

Figure 12 Disk Seek Analysis Report

Cylinder address	*Total references*	*User*	*Refer-ences*	*User*	*Refer-ences*
0	12	JES3	9	VTAM	3
11	239	*MASTER	239		
66	23	JOB5	8	CAPEK21	6
112	25	JOB2	18	ASHERASM	7
113	64	JOB3	64		
114	76	JES3	76		
225	181	*MASTER	181		
337	62	JOB3	62		
556	17	JOB1	17		
677	244	JES3	244		
799	14	JOB2	14		

The right side shows, for each sample, the number of waiting (*Wait*), demanding (*Dem.*), and working (*Work*) users. It also shows the percentage of the waiting from the demanding (*%W/D*).

Report 3, a user's contention (wait) profile, is built for each user (address space) in the system. In it one can see the resources for which this user was waiting. Figure 11 gives an example of the report produced for the master scheduler address space. This report is the "reverse" of Report 1.2. Columns 10, 11, and 12 show the distribution of the holding user among the three workload groups.

Figure 12 shows that part of Report 4, the Disk Seek Analysis, where the "contention-causing" cylinder addresses, in ascending cylinder number, are shown along with the users who were "holding" them. There is another part of this report, not shown here, that provides contention information from the user's point of view. For each holding user, all disk addresses, where he or she was operating, are shown.

experience highlights

Some of the interesting observations made by the users of the prototype follow:

1. The average numbers of demanding, waiting, and working users (Report 1.1) have simple straightforward meaning. A 50 percent figure in the waiting/demanding statistics simply means that half of the workload demand is not executed. A comparison between the overall statistics in Report 1.1 and the "second-by-second" fluctuation in Report 2 is very instructive. Some installations found, for instance, that the distribution of contention among the system components was the same for periods of high and low contention.

   Yet, the meaning of contention may vary from one system to another and from one resource to another. Contention on domains, for instance, does not have the same meaning as contention on the CPU. Within domains, contention in the heavy batch domain has a different meaning than contention in the domain that serves the short TSO transactions. It takes time to become familiar with the measured system and to know how to evaluate the contention shown on its different components.

2. In some installations some comparisons were made between utilization and contention for I/O devices and channels. In all cases, nine out of the ten most busy resources also appeared on the ten-most-contended-for list.

3. The cross analysis of Report 1.2 and the User's Wait Profile Report (which together show the users/resource matrix in Figure 3) was found to be the biggest success of the prototype. Especially in I/O analysis, the user names were found to be quite sufficient to identify the cause of contention. Occasionally, one may need to look at the disk-seek address report. More detailed activity information was felt to be unnecessary.

   For other resources, mainly the CPU, Report 1.2 only stressed the need to look even deeper into the system and to locate the exact *activity* that causes the contention.

   For shared I/O devices, the users can see the instances where the device is held by the other machine. This is marked as "**SHARED" in Column 6 of Report 1.2. But they cannot tell who is causing that on the other machine. A parallel operation of the prototype on both machines can be a partial solution.

4. The division of the system workload into the three "natural" groups of TSO, batch, and started tasks was found to be very useful. In the CPU part of Report 1.2, for instance, one can sometimes observe a batch job that causes a lot of contention. By looking, however, at the workload groups that are waiting,

one can determine whether only other batch programs are being delayed by that job, or TSO and/or started tasks, too.

5. From the left side of Report 2, a high contention on domains was sometimes observed at a time when there was almost no contention on the real resources. This may be due to a too-restrictive definition of the maximum multiprogramming level.

   From the right side of this report, one can identify the periods when any increment in the demanding column immediately results in an increment in the waiting ones. The value in the "working" column, in these periods, shows the maximum multiprogramming level the system can handle (at such periods).

6. Report 3 can be used for partial system tuning. There are many cases where a "user" in MVS is a big subsystem, such as VSPC (Virtual Storage Personal Computing), IMS (Information Management System), or CICS (Customer Information Control System), which has its own end users. A "private" tuning of such systems can be done by the people in charge, without waiting for overall system tuning. For such systems, the analysis of the internal contention among their own end users may be the target of a "private" contention analyzer. Such an approach could be implemented inside those systems to supplement the information from a system-wide contention analyzer.

7. A comparison study between Report 4 and full seek analysis reports[24] found that, for the public active disks, the two reports gave similar results. If this is generally true, then again the H term serves as a good term for S at a much (much!) lower cost.

8. Users reported on many problems that they were able to solve "on the fly" by using the interactive (TSO) mode of operation of the prototype monitor. A real-time observation of abnormal figures in the contention matrix (Figure 3) can immediately point to the exact location of the "congestion," its cause, and its effect. If the operator has to take unpopular actions, at least he should move in the right direction.

9. The contention "language" was found to be comprehensible and meaningful to *managers* and to the computer users. A high number of users being delayed on a resource is a simple statistic that draws management attention. A decrease in this number clearly shows that an improvement was made. Talking to users in terms of delays that their jobs either cause or suffer from is talking to them in a language they seem to like and

understand. Telling a user that his job causes delays to other users is much more effective than to tell him that his job consumes a lot of resources.

Appendix C: Classification of MVS tuning actions

The purpose of this appendix is to validate Figure 4 for the MVS case. We assume that the reader is familiar with MVS basic terms.

The following is the grouping of MVS tuning actions as taken from References 25 and 26.

- *1.1 Reprogramming of code:* Use optimizing compiler, write critical subroutines in assembler, use more buffers for I/O, increase the block size of the data set, use Virtual I/O and disk allocation in cylinders not in tracks, care for boundary alignment, care for locality of reference.
- *1.2 Resource restructure:* Cluster system libraries and VTOC placement, reorder I/O devices on the channel, increase the block size of system libraries, reorganize the placement of data sets within a disk pack.
- *2.2 Allocation algorithm:* Balance data sets between packs and packs between channels, make system packs "non-storage," move members to FLPA, increase the BLDL list, channel rotate.
- *2.3 Dispatching algorithm:* Change the APG definitions, use priority instead of FIFO for system packs.
- *1.2 and 2.2 together:* Spread the catalog (CVOL), separate swap data set from page data sets, add page data sets.

Action 2.1 does not appear because in MVS the System Resources Manager[11] is supposed to handle this function by dynamically raising or lowering the MPL. Yet, Item 5 of the experience highlights in Appendix B indicates that Action 2.1 may still be required in certain cases.

CITED REFERENCES

1. Y. Bard, "Performance analysis of virtual memory time-sharing systems," *IBM Systems Journal* **14,** No. 4, 366-384 (1975).
2. B. J. DiMarsico, "UCBMON," *Selected Papers from the SHARE CME Project*, 85-96 (April 1975-September 1976), SHARE Inc. Basic Systems Division, 111 East Wacker Drive, Chicago, IL 60601. See also: L. Riss, "UCBMON Revisited," *ibid.*, pp. 113-117.
3. J. Kessler, "SLACMON enhancements for VS," *Selected Papers from the SHARE CME Project*, 102-112 (April 1975-September 1976), SHARE Inc. Basic Systems Division, 111 East Wacker Drive, Chicago, IL 60601.
4. J. Michlin, "The use of special performance software monitors in a batch/TSO environment," *Selected Papers from the SHARE CME Project*, 88-97 (December 1973-March 1975), SHARE Inc. Basic Systems Division, 111 East Wacker Drive, Chicago, IL 60601.

5. L. S. Wright and W. A. Burnette, "An approach to evaluate time sharing systems: MH-TSS, a case study," *Performance Evaluation Review (PER), ACM Sigmetrics* **5,** No. 1 (January 1976).
6. *IBM Program Product, OS/VS2 MVS Resource Management Facility (RMF) Version 2, Reference and User's Guide*, SC28-0922-1, IBM Corporation, pp. 5-23, 5-24; available through the local IBM branch office.
7. *SVS Performance Tool, Installed User Program*, Program Number 5796-PGN, SH20-1838-2, IBM Corporation; available through the local IBM branch office.
8. *IBM Systems Journal* **8,** No. 4 (1969). The entire issue is dedicated to system performance evaluation. See in particular the papers by A. J. Bonner and M. E. Drummond, Jr. Note also the extremely interesting work by C. E. Skinner and J. R. Asher on storage contention. It seems as if once again the hardware was much in advance of the software.
9. T. Beretvas, "Performance tuning in OS/VS2 MVS," *IBM Systems Journal* **17,** No. 3, 290-313 (1978).
10. *VS1 Performance Tool, Installed User Program*, Program Number 5796-PGL, SH20-1837-1, IBM Corporation; available through the local IBM branch office.
11. *OS/VS2 System Programming Library: Initialization and Tuning Guide, Part 3: The System Resources Manager*, GC28-0755-0, Sections 1-2, IBM Corporation; available through the local IBM branch office.
12. *OS/VS2 MVS SPL, System Management Facilities (SMF)*, GC28-0706, IBM Corporation; available through the local IBM branch office.
13. W. J. Doherty and R. P. Kelisky, "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18,** No. 1, 143-163 (1979).
14. *OS/VS2 MVS Service Aids Logic, Part 1: General Trace Facility*, SY28-0643, IBM Corporation; available through the local IBM branch office.
15. M. E. Drummond, Jr., *Evaluation and Measurement Techniques for Digital Computer Systems*, Chapter 7, Prentice-Hall Inc., Englewood Cliffs, NJ (1973).
16. W. H. Tetzlaff, "State sampling of interactive VM/370 users," *IBM Systems Journal* **18,** No. 1, 164-180 (1979).
17. A. O. Allen, "Elements of queuing theory for system design," *IBM Systems Journal* **14,** No. 2, 161-187 (1975).
18. J. D. C. Little, "A proof for the queuing formula $L = \lambda W$," *Operations Research* **9,** 383-387 (1961).
19. A. O. Allen, *op. cit.*, p. 168.
20. D. Towsley et al., "Models for parallel processing within programs: Application to CPU:I/O and I/O:I/O overlap," *Communications of the ACM* **21,** No. 10, 821-830 (October 1978).
21. J. Buzen, "Analysis of system bottlenecks using a queuing network model," *ACM-SIGOPS Workshop on System Performance Evaluation*, 82-103, Cambridge, MA (April 1971).
22. *OS/VS2 System Logic Library: Volume 1, Introduction*, SY28-0713-1, 1-13 and 1-16, IBM Corporation; available through the local IBM branch office.
23. M. E. Fligliuzzi, "DEVMON—Special purpose monitor to measure TSO/batch contention for direct access storage," *Selected Papers from the SHARE CME Project*, 88-97 (December 1973-March 1975), SHARE Inc. Basic Systems Division, 111 East Wacker Drive, Chicago, IL 60601.
24. *MVS Seek Analysis Program*, SH20-1814, IBM Corporation; available through the local IBM branch office.
25. *OS/VS2 MVS Performance Notebook*, GC28-0886, IBM Corporation; available through the local IBM branch office.
26. *MVS Measurement and Tuning*, IBM Education Course H3770, IBM Corporation; available through the local IBM branch office.

GENERAL REFERENCES

A. O. Allen, "Elements of queuing theory for system design," *IBM Systems Journal* **14,** No. 2, 161-187 (1975).

M. E. Drummond, Jr., *Evaluation and Measurement Techniques for Digital Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973).

L. Svobodova, *Computer Performance Measurement and Evaluation Methods: Analysis and Applications*, American Elsevier Publishing Company, Inc., New York (1976).

The author is located at Mehish Computer Services, Ltd., 15 Lincoln Street, Tel-Aviv, Israel.

Reprint Order No. G321-5123.