XEROX

Competitive Report

A TIMESHARING COMPARISON - TSO vs. CP-V

The enclosed document is a detailed report on TSO, IBM's Time Sharing Option. The report examines TSO both from a system and a user's point of view. It traces TSO from the current non-virtual implementation to the far more sophisticated release due in March 1974.

The report shows how TSO will be improving from its current BTM-like architecture to a much more advanced system which resembles CP-V. Like the CP-V user, each on-line TSO user will finally have his own virtual space. The only difference is that while the virtual space for CP-V is limited by the size of the physical memory of the host computer, each virtual space on a TSO system may be as large as 16 million bytes. For the first time the on-line TSO user will be treated exactly the same as a batch job. For the first time he will have full access to all system resources such as tape drives, removable disk packs, and slow speed peripherals. In brief, it seems that IBM has finally "discovered" the CP-V approach to multi-use systems. As such we can expect IBM to be even more competitive in the multi-use marketplace in the near future.

Yet, there are many reasons why CP-V remains superior in the multi-use environment. These include:

- Xerox processors are generally more interactive.
- Response time with TSO is much slower than with CP-V.
- TSO requires far more memory than does CP-V.
- There is a high level of CPU interference between TSO and batch.
- TSO continues to be plagued with software bugs.
- TSO uses the extremely complex file structure of OS.
- TSO JCL is as complex as OS JCL.
- Many TSO processors have incompatible file formats.
- IBM's file preallocation can require excessive disk space when many users are on-line.

In summary, TSO has come a long way but it has not caught CP-V. TSO requires far more overhead - both in terms of CPU time and memory - and is a far more complex system to use. Because TSO is essentially identical to IBM batch processing, it has become a reasonable IBM software development tool, especially since the people who would be using the system for that type of application would be well versed in the complexities of IBM systems. Yet, for the casual timesharing user - for the majority of our prospects who want timesharing as an easy-to-use tool to solve his problems - the the complexity, the performance, and the cost of TSO are still prohibitive.

The previous statements highlight the theme of the report which examines both the strengths and weaknesses of TSO in much more detail. However, TSO, like all software packages, is a constantly changing system, so this report must eventually become obsolete. If you are engaged in a competitive opportunity and require updated information on TSO or if you encounter new information about the system, please feel free to call Phil Becker on extension 4687.

A TIMESHARING COMPARISON

IBM and XEROX

(TSO vs CP-V)

Phil Becker
Competitive Analysis
January 15, 1974
In Conjunction With
Sarah Jones
Los Angeles District

Table of Contents

Table of Contents (Continued)

INTRODUCTION

Although IBM currently offers a wide variety of 32-bit software and hardware combinations, any detailed analysis will show that the primary attribute of all of them is to function as a very competitive commercial batch processor. To satisfy the increasing demand for interactive timesharing capabilities, IBM has introduced a series of timesharing subsystems of varying sophistication. These have ranged from Conversational Remote Batch Entry, which is essentially a terminal job entry capability through ITF (Interactive Terminal Facility), which gives an interactive Basic and PL/1 capability to smaller systems, to TSO (Timesharing Option), which is a rather complete timesharing system for use on IBM's larger systems. With the continued enhancement of TSO, IBM is now becoming a formidable competitor to CP-V on its home ground. It becomes important to understand and appreciate the capabilities and weaknesses of this "new" competitor in order to sell successfully against it.

The TSO that is discussed here is currently available and runs under OS/MVT and release one of OS/VS2. Although TSO can be resident under the VS2 (virtual storage) operating system now used by some IBM customers, it does not take particular advantage of the virtual memory or paging techniques other than the fact that the region in which TSO resides becomes a virtual region instead of a real one. It is the same TSO which runs on OS/MVT--a non-virtual system.

A more sophisticated TSO scheduled for release in 2nd quarter 1974 will run under Release 2 of VS2. It will be a virtual timesharing option, and more similar in design philosophy to CP-V relative to utilization of memory. Little information is available on this yet unreleased product; however, because users have stated their reluctance and even refusal to go to the latter IBM operating system, we can assume that TSO as we see it today will continue to be the timesharing system most frequently proposed by IBM. What is known about TSO under VS2, Release 2 is documented later in this paper.

SECTION I: THE SYSTEM

TSO and CP-V: A Systems Comparison
TSO Operations
Summary

TSO AND CP-V: A SYSTEMS COMPARISON

Both OS/MVT and OS/VS2, Release 1 are region-oriented operating systems. What this means is that each active job is assigned to a fixed region in memory and remains there until the job is terminated. TSO, much like the BTM implementation, is activated in a region and continues to occupy that region until it is deactivated. Since MVT does not require fixed partition sizes, when TSO is turned off, the memory which it had been using is freed for other system functions. The operating system treats the TSO region exactly like any other batch region and as such, TSO vies for the system resources with the other partitions. Operator assigned priorities determine which region will get access to a resource whenever there is contention.

Use of Memory With TSO

TSO looks a lot like BTM in its handling of on-line users. A foreground partition is assigned for the users who share it in a round-robin fashion. Like BTM, each job occupies the partition for a time slice and then is swapped out. Unlike BTM, TSO allows multiple partitions, but each partition has its own set of users assigned to it, so that there is no load balancing between partitions. Figure 1 on the following page shows how memory is assigned for TSO. This can lead to the situation where one or two jobs share one partition while a dozen or so share a second. Since each partition shares system resources equally, the two jobs in the first partition will perform much better than will the dozen in the second one. Note that this characteristic can make for a very nice demonstration. A user can be shown a terminal which is assigned to an empty partition and receive excellent response even though there are many other users logged onto the system.

The number of foreground regions is a parameter which can be controlled by the operator when he initiates TSO and represents the maximum number of timesharing users which can be concurrently core resident since only one user per region is in core at any given time. The region has a predetermined size which does not vary with the size of the user and must exist in a contiguous area of memory.

In contrast, CP-V is an operating system which supports both batch and timesharing users with the same system routines; since once users are initiated into the system, there is essentially no difference between batch and timesharing. The resident portion of the CP-V monitor requires 48KW of memory and does not grow with additional users. All of core is mapped through the use of hardware implemented mapping registers and all users, processors, libraries, etc., need not be contiguous in memory, thus making the most efficient use of the memory resource. The mapping registers also allow service routines, language processors — all systems software — to be reentrant so that any number of resident users may have access to a routine although only one copy of it exists in core. A user only requires enough pages of core to contain his identification, his own code and whatever processor data is particular to him. CP-V also makes the distinction between data and pure procedure and does not swap out pure procedure unnecessarily.

To understand how memory is allocated for TSO, one should first understand how OS/MVT uses memory. Basically, memory is divided into three areas:

Low main storage — contains the operating system and system queues.

TSO ARCHITECTURE
NON-VIRTUAL SYSTEM

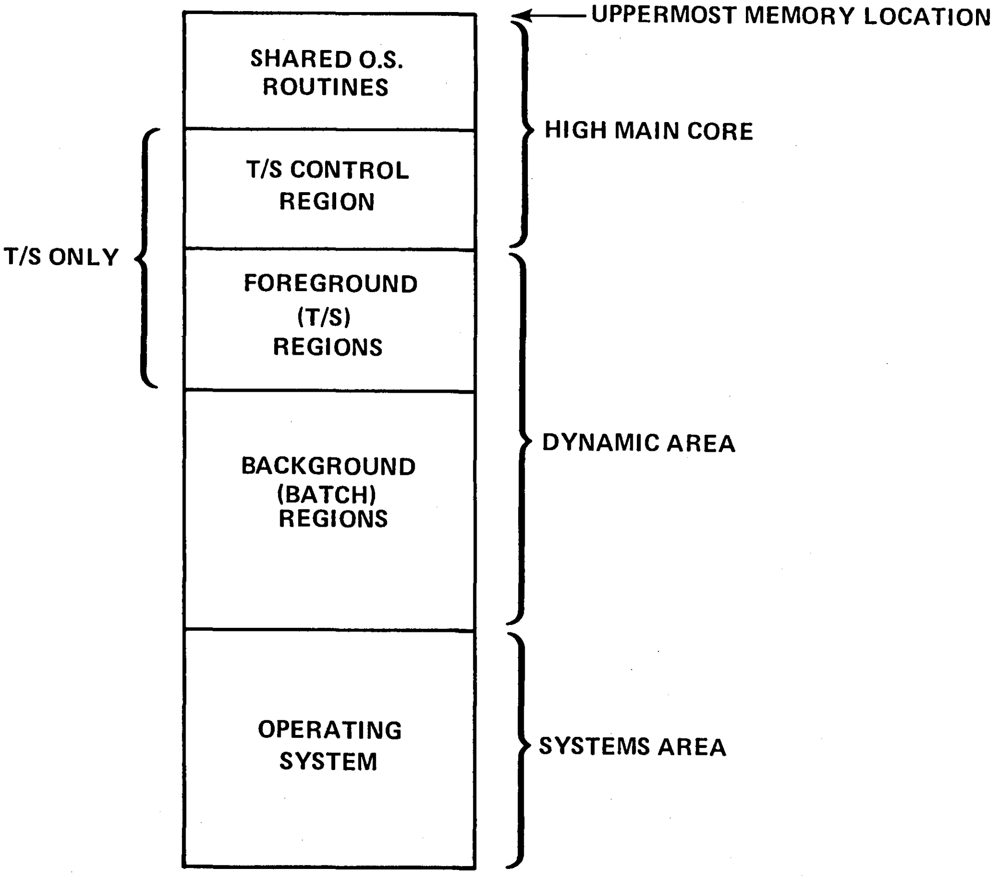

FIGURE 1

Use of Memory With TSO (Continued)

Dynamic Area — the middle portion of memory which contains the user jobs.

High main storage — contains the link pack area (system processors and I/O routines), the master scheduler, and the time-sharing control region when TSO is actuated.

When an operator brings up TSO by entering START TS on his console, the operating system obtains memory for the timesharing control region. The control region will then obtain one or more foregound user partitions, depending on how the system was generated. This memory layout is diagrammed in Figure 2.

As is indicated in the storage map, the timesharing control region is composed of a variety of routines, each of which has a specific function.

The Timesharing Driver

The Timesharing Driver isolates in one component the decision-making algorithms for the division of system resources among all the users of the system. By passing parameters to the Driver with the START command or from the system parameter library, the installation controls the various scheduling algorithms to gain the desired performance for its job mix. These "tuning" parameters and the algorithms are discussed in the last section of this chapter.

The Driver has a unique relationship to the other control routines. It is used as a service program by all the levels from the MVT supervisor down to the Terminal Monitor Program. The calling programs inform the Driver of events throughout the system — time slice end, user waiting for LOGON, job waiting for input, etc. From this stream of information, the Driver maintains a current picture of the system load and activity. Based on this picture, the Driver orders actions such as swapping, changes in priority, and assignment of a user to a particular region.

The Driver component itself is completely insulated from the rest of the system by the Timesharing Interface Program, which accepts all calls to the Driver, then passes them through a standard interface to the Driver itself. The Driver returns parameters to the Interface Program that request various actions by the other control routines. Thus, an installation can modify or replace the Driver — effectively, provide its own system scheduler — without modifying the system implementation programs. The operator uses the START command to specify which Driver — the standard one or an installation-written one — is to be used.

The capability to change the timesharing driver can be either a liability or an asset, depending on the situation. On the plus side, it offers extreme flexibility to tailor the system to a given T/S load. This is especially valuable when the load is considerably different from the "typical" environment for which TSO was designed. The interface is clean and very well documented and the switch has actually been implemented by several large installations which are well staffed by sophisticated system analysts.

TSO: TYPICAL MAIN STORAGE MAP

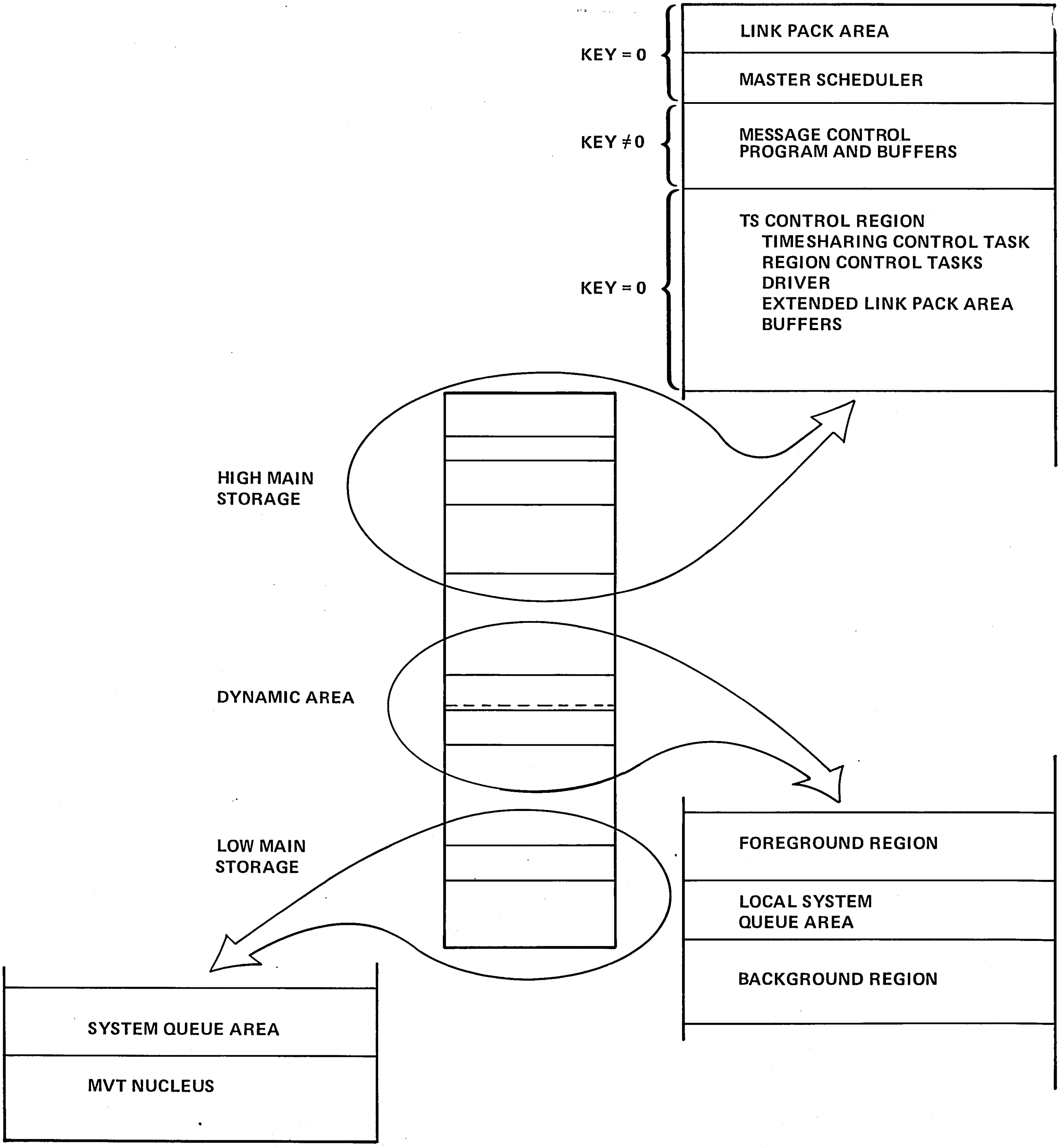

FIGURE 2

The Timesharing Driver (Continued)

However well documented the interface may be, it is still a very complex operation. The analyst who elects to write his own driver must fully understand the subtleties of T/S scheduling; he must be aware of what effect his changes will have on the rest of the system. Even a slight error can severely impact performance. IBM will be the first to warn the installation that a new driver should only be installed by a very experienced team of programmers.

Because the design of the scheduler is so closely tied to the overall performance of the system, the designers of CP-V have elected not to permit its modification. The feeling is that considerable effort has been devoted to optimizing the scheduler for CP-V's architecture and any major changes would have to damage the system.

The Timesharing Control Task

The Timesharing Control task, as shown in Figure 4 handles all functions affecting the entire timesharing portion of the system. This includes responding to the START, MODIFY, and STOP operator commands, and handling the swapping of foreground jobs into and out of main storage.

When the operator enters the START command for TSO, the initialization module of the Timesharing Control is given control. The initialization module calculates the size of the Timesharing Control region that will be needed and obtains it from the main storage management routines of MVT. In this region, the Timesharing Control task builds the control blocks and buffers the system will need, and invokes a Region Control task for each foreground region.

The installation may override the calculated TSC region size by specifying the size it wants in a parameter list or on the START command. This may be necessary if an installation written Driver has greater main storage requirements than the Driver supplied with TSO.

While the timesharing system is operating, the major function of the Timesharing Control task is the swapping of foreground jobs into and out of main storage. Swapping is handled at this level so it can be optimized on a system-wide basis when multiple foreground regions are active. A swap out is scheduled whether a channel is free or not. This means that on a multi-partition system there could be significant swapping interference.

The Timesharing Control task maintains an input queue and an output queue for swap requests (one of each set if parallel swapping is being used). Seek time is minimized by attempting to swap jobs out to the direct access area from which the last job swapped in, or if this is not possible, by using the free space closest to the current arm position.

In determining what portion of a foreground region to swap out, the Timesharing Control task uses a map of the foreground job created by the Region Control task. Each entry in the map identifies the starting address and length of a section of the region that the job is using. The number of entries in this map is the same for every job and is specified by the installation in the system parameter library. If there are too few entries, inactive main storage must be included (and swapped). A large number of entries cuts down on the amount of inactive storage that has to be swapped, but adds to processing overhead.

The Timesharing Control Task (Continued)

When the operator enters a STOP command to shut down the timesharing operation, the Timesharing Control task initiates a logoff for each active user. When all users are disconnected, the Timesharing Control task ensures that all the system resources that had been assigned to it are returned; the Timesharing Control task then terminates, returning its main storage region to the system.

If any users cannot be logged off, the Timesharing Control task cannot terminate. The operator is given the facility to "force" TSO to terminate even if it appears that normal STOP processing cannot be completed.

The Region Control Task

A major function of the Region Control Task is quiescinq and restoring foreground job activity before and after swapping. Conceptually, there is one Region Control task for each active foreground region, invoked by the Timesharing Control task, although only one copy exists in the TSC region.

Before a foreground job can be swapped out of main storage, any activity associated with it must be brouqht to an orderly halt, or set up to be handled by some supervisor routine that will be remaining in main storage. This includes removing control blocks associated with the job from system queues, or flagging them as inactive.

Quiescing of I/O activity is initiated by the Region Control task (at the request of the Driver), which issues the Purge Supervisor Call for each task associated with the foreground job. The Purge routine removes I/O requests from the I/O Supervisor's queues of pending requests if they have not yet been initiated. If a request has been started, that is, if data transfer is already taking place, it is allowed to complete before the job is marked ready for swapping. The control blocks associated with unstarted requests are stored in the foreground region where they will be swapped out of main storage along with the job.

I/O requests that address the terminal are an exception to the quiescinq procedure because of their long completion time. These requests are handled through the TSO interface with the Telecommunications Access Method routines which process all terminal I/O requests. The requests are buffered in supervisor main storage, not in the foreground region. Data can be written or read to these buffers whether the job is present in its main storage region or not.

Many control blocks, like the I/O requests mentioned above, reside in the foreground region. For background jobs, these control blocks would be created and maintained in the System Queue Area, a section of main storage set aside for this purpose during nucleus initialization. Foreground regions, however, each contain a Local System Queue Area to hold control blocks. As part of quiescing, the Region Control task removes pointers to these control blocks from system queues. The blocks can then be swapped out of main storage along with the foreground job. The only control blocks for foreground jobs that are assigned in the System Queue Area (and remain in main storage) are requests for timer interruptions, operator replies, and assignment of resources through ENQ.

The Region Control Task (Continued)

When a job is swapped into main storage by the Timesharing Control task, the Region Control Task receives control to restore the I/O requests it intercepted at swap out time, and to return the control blocks associated with the job to the appropriate system queues.

The Region Control Task is the only reentrant TSO system processor and resides in the TSC region. Note – no such module exists in CP-V because the system does not remove a user queued for I/O. The overhead is eliminated by placing him in a special state queue and allowing another user to execute until the I/O is completed.

Other Control Programs

The remaining TSO control programs are in the foreground region. The most important are LOGON/LOGOFF which is invoked by the RCT when a user wants to log on or off of the system and the Terminal Monitor Program (TMP) which takes care of user functions subsequent to logon – i.e., connects the user to processors he wishes to use, translates his commands for that purpose, and handles abnormal termination of command processors or problem programs. Neither LOGON nor TMP are shared. A copy of LOGON/LOGOFF must be maintained on secondary storage for each region and the TMP must always be resident with the user no matter what function he is executing! This means additional swap time to bring in extra copies of the processor, and also means that extra memory is required to maintain the redundant copies in a multi-partition system.

Control Logic-On Overview

An overview of the control logic which connects these various TSO modules is shown in Figure 3, titled TSO – a system overview.

TSO – A SYSTEM OVERVIEW

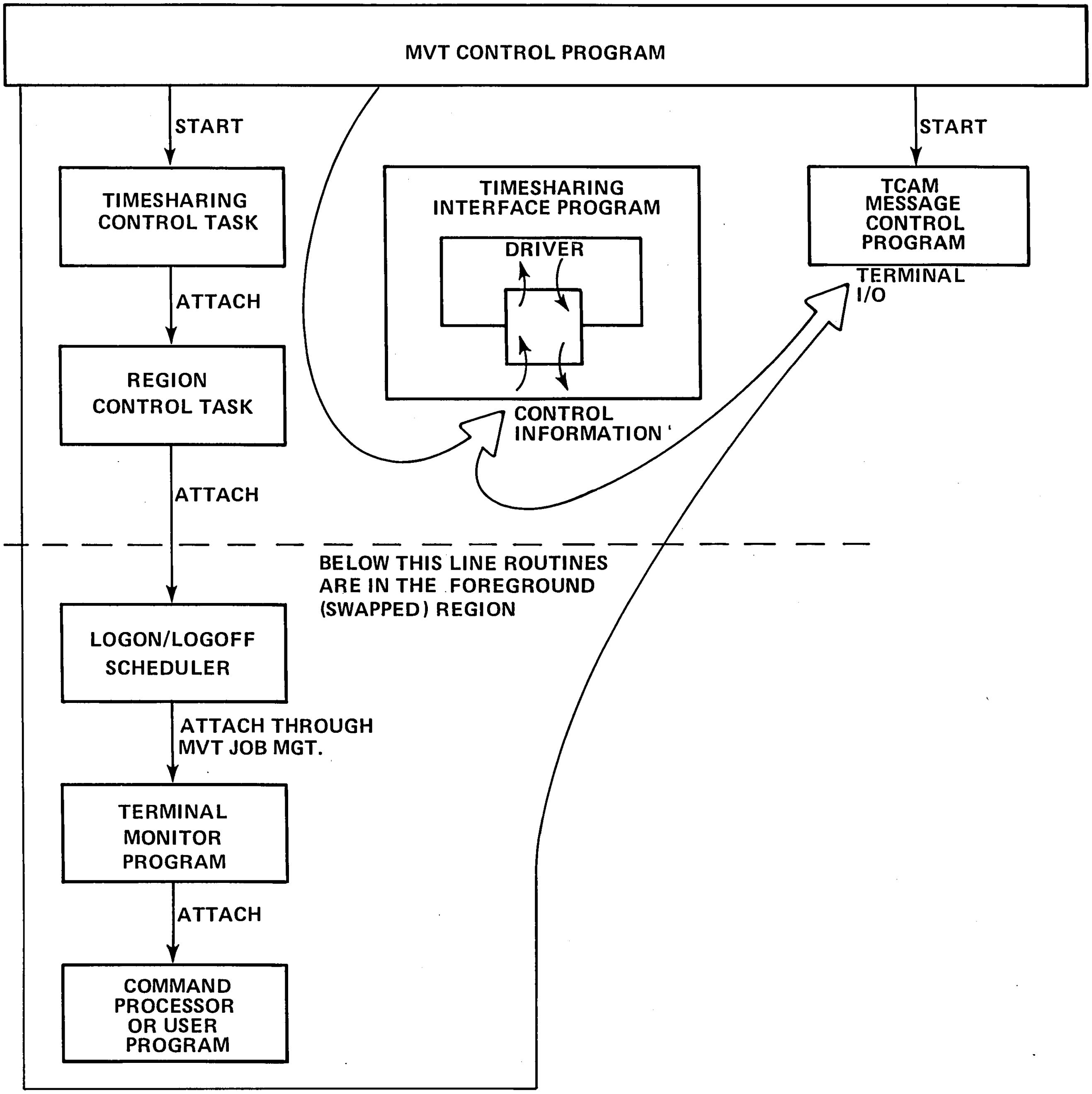
MVT CONTROL PROGRAM
START
START
TIMESHARING CONTROL TASK
TIMESHARING INTERFACE PROGRAM
DRIVER
TCAM MESSAGE CONTROL PROGRAM
TERMINAL I/O
ATTACH
REGION CONTROL TASK
CONTROL INFORMATION
ATTACH
BELOW THIS LINE ROUTINES ARE IN THE FOREGROUND (SWAPPED) REGION
LOGON/LOGOFF SCHEDULER
ATTACH THROUGH MVT JOB MGT.
TERMINAL MONITOR PROGRAM
ATTACH
COMMAND PROCESSOR OR USER PROGRAM

FIGURE 3

TSO OPERATIONS

LOGON

When a new user is first recognized by TSO he is potentially assigned to a region and the LOGON scheduled associated with that region begins to check his authorization and set up the predetermined Data Control Block (DCB) assignments for him. This may include some dummy DCB's, but since the number of DCB's required varies widely, dependent upon what the user does during his terminal session, this individual user default (as opposed to system wide in CP-V) is an inflexible restraint which may force the user to interrupt his session of some point then log off and then back on under a different set of logon JCL (if he is so authorized) in order to gain enough DCB's to do his tasks in a multi-step session.

If LOGON discovers that the user is too big for the region he has been assigned to, it reports to the region control program which calls the timesharing control task and he is reassigned to a bigger region.

The Terminal Monitor Program (TMP) is usually the processor invoked by LOGON. TMP is the equivalent to TEL in CP-V. TMP is not reentrant and a copy of TMP will be swapped with each user throughout his session. TMP must remain with the user to handle attention characters and program and processor errors and termination.

During the terminal session under TSO, a user is swapped in and out of core and is allotted execution time in this manner.

Allocating CPU Time

The Timesharing Driver interfaces between MVT and the various timesharing modules. It is an event recorder and is front-ended by an interface program (TSIP) which acts as an interface between the Driver and the rest of the system. In order to go further we must define some terms.

CYCLE – The time required for all users assigned to a particular region to get an execution time slice. This happens one user at a time on a round-robin basis.

MAJOR TIME SLICE – The length of time each user in a region will be resident in memory.

$$\text{MTS} = \frac{\text{CYCLE} - \text{overhead}}{\#\ \text{Users}}$$

MINOR TIME SLICE – The available execution time for an individual user.

The job of the Timesharing Driver is to parcel out the execution time not allocated to batch jobs (which are not swapped) among the jobs in various regions as they are swapped in and out. TSD uses one of several algorithms, taking cognizance of installation parameters. Although a region can have more than one queue for the jobs assigned to it, the parameters used for queue assignment are size and interactiveness, the latter being measured by historical data on that job's past executions. The collection and maintenance of these statistics constitute overhead which does not exist in CP-V. Under TSO if too many users are assigned to a queue the major time slice gets so small that the system spends all of its time swapping and no useful work is accomplished. To make sure this doesn't happen, the installation must specify a minimum major time slice for each queue in each region, an additional complexity for the system manager. The system manager is allowed to specify some nine parameters which influence the calculation of the major time slice – e.g.,

maximum number of users logged on
number of foreground regions
number of queues per region
queue weighting (preference for jobs in one queue over those in another).

With CP-V, the system manager controls global features rather than attempting to run the system by controlling detailed regional parameters. He can vary such things as number of on-line users, number of simultaneous batch users (and their sizes, etc.), amount of resources to be allotted to batch and on-line users (memory, time, drives, etc.), terminal responsiveness desired, etc., and he may change all these dynamically.

The minor time slice is the result of dividing the available execution time among the regions of main storage containing a ready foreground or background job. This calculation is made each time a major slice expires and one or more swaps occur. Remember, swapping and scheduling are both done by region. The installation can calculate this execution time slice in one of three ways –

1. Simple dispatching – the job at the top of the ready list (the one most recently swapped in) gets all the time available until the next scheduled swap. i.e.

$$MS = AT$$

(where MS is the minor time slice and AT is the available execution time)

2. Even dispatching – when there is more than one foreground region the available execution time can be divided evenly among foreground regions containing a job ready to run. i.e.

$$MS = \frac{AT}{N}$$

(where N is the number of foreground regions containing jobs ready to run)

3. Weighted dispatching – for this option the system keeps running statistics on the amount of time each user spends waiting – usually because of pending I/O requests. Then jobs which historically have given up large parts of their minor time slices are given less CPU time (than those which have not), i.e., the minor time slice is computed in the following algorithm:

$$MS = \frac{\text{this job's EWT\%}}{\text{sum of EWT\%}} \times (AT)$$

(where EWT% is the Estimated Wait Time percentage and AT is the available execution time for this minor time slice.)

Analyzing the Allocation Schemes

The multiplicity of variables which can be set by the systems manager does not seem to be an advantage in any way. Rather it seems to complicate that person's job unnecessarily and increase system overhead because of the backlog of statistics which must be kept and massaged on a regular basis. Even if everyone gets his part right and if this scheme works, the system will wind up being "tuned" on a regional level and through an averaging process to an operating environment which existed fifteen minutes ago!

The major negative design flaw of TSO is that it works on the basis of regional activity rather than on the basis of individual users. Once a user has been assigned to a region (during the LOGON process) he stays there for his entire session, regardless of what he is doing or what his memory requirements are. The assignment to region is balanced to some extent either by an attempt to keep the number of users in each region equal or through a calculation yielding average amount of activity per region. But when a bad mixture occurs there is nothing to be done and users in one region may get poor service while those in another get good service. In CP-V there are no multi-regional queue considerations. CP-V knows through one set of queues in which entries are made by logical and physical interrupts just what each user did last, and is responsive to each user as well as on the basis of overall resources. When all the system resources (and that includes memory) are in one pool, and resources are allocated as required with no fixed partitions or regional considerations, those resources can obviously be managed more efficiently.

CP-V is an operating system which was designed to handle both batch and timesharing users in a stand-alone environment, not as a subset of some other existing operating system with built-in overheads and design constraints. From a design standpoint there are similarities between CP-V and TSO – in that both systems' terminal users need not be in core while doing terminal I/O, and probably won't be. In both, users are swapped in their entirety not partially. Both have a user interface processor, a swapper, a scheduler, and special handler for terminal I/O – as opposed to other types of I/O. However, there are many features incorporated into CP-V which show that it was designed specifically for timesharing and multi-use.

In CP-V the Scheduler is the heart of the operating system. It controls approximately 30 state queues which contain at all times one entry for each user in the system. The user moves from state queue to state queue based on events which happen as a direct result of user action or some system occurrence.

Users are scheduled for execution and swapping based on the state queue they are in. The order in which the queues are searched gives priority to users who are interactive and need quick response. A similar algorithm is used to search the queues in a different order for users who are not using, or likely to use, the CPU in the near future. These users are candidates for being swapped out. Samples of state queues are:

The queue of all terminal users who have hit the break key
the queue of all users whose terminals are inputting
the queue of users having I/O in progress
The queue of compute-bound users

Since all users and processors run mapped and most system processors including the Terminal Executive Language (TEL) are reentrant or shared, a user may be swapped out but a system processor never need be. If the processor pages are needed they are overwritten, and should the processor be required again it is copied in from secondary storage and associated with the user through his map (in his Job Information Table, JIT). Users and processors can always be swapped into core without an outswap if enough free pages exist even though those pages are not contiguous. This is of course transparent of the user. In CP-V there need not be unused core because a user is assigned to a partition larger than he requires. There never need be two copies of EDIT, or TEL, or FORTRAN resident and this code is not taking up swapper storage space.

SUMMARY

- TSO is an option under MVT requiring well over 150KB additional storage. Batch and on-line users are treated differently.

- CP-V is an integrated timesharing and batch system where the on-line user receives the full capabilities afforded the batch user.

- TSO allocates memory on a contiguous region basis, with memory (which is the critical resource in a timesharing system) wasted. The number of regions is fixed and hence, the maximum number of users is also fixed. Memory tied up by a user does not vary with his changing requirements as he goes from step to step. It must be sufficient to hold his largest step.

- CP-V has no restructive region concept. Users employ only as much memory as is required. There is no maximum to the number of users that may be resident at a given time. The important difference is that CP-V has a much better memory utilization; thus more users can be in memory at any given time; thus, a better opportunity of having someone in memory who can utilize the CPU.

- TSO has introduced all sorts of system parameters in an attempt to control which users go into a particular region, priorities and number of queues per region, size of major and minor time slices. It does running calculations to keep track of activity in each region, interactiveness of users, amount of minor time slice not used by a user etc., in an historical manner. All of this is because the designers were aware of the challenge of being responsive to terminals while making constant progress on all users. The complexity of this approach is such that it is improbable that the system is ever adjusted properly for the current load at any given instant. There must be considerable overhead in keeping the running statistics and calculating "who gets how much and when". Once again, there problem is fixed regions, with users assigned to a particular region for an entire session.

- CP-V features a superior scheduler. Due to the use of state queues, it always knows what is happening right now and it can pick the appropriate user or users to swap in or out at any time quickly. CP-V knows, for example when a user has 10 seconds worth of terminal output and can get him out of memory immediately rather than waiting for the end of his minor time slice as TSO must.

- TSO has no concept of shared processors, pure procedure, etc. The entire user and whatever processors he is using swap out and in.

- CP-V has shared processors, libraries, debuggers, etc., which are shared by multiple users thereby conserving memory. Share processors are never swapped while in use out conserving swaptime. CP-V also recognizes when it is not necessary to swap out a users pure procedure.

SECTION II: THE COSTS OF TSO

Buying a TSO System
Upgrading to TSO
Summary of Costs

BUYING A SYSTEM

There are two ways for a potential user to acquire a TSO capability – either acquire a new 370 or upgrade the existing system if one already exists in house. In the former case a user would have to acquire at least a 512KB 370/145. A minimal system to support sixteen lines and very little batch should cost $1,146,330 or lease for $25,254 per month. A more realistic system which supports thirty-two lines and two or three batch partitions would be a 1.5MB 370/158 which would lease for $57,966 per month or sell for $2,645,100. The complete configuration for each case is shown on the next two pages.

Xerox and IBM: Competitive Configurations

As a basis for comparison with the two IBM configurations, consider the two Sigma CP-V configurations on the page following the two IBM configurations. The first is a minimal system designed to support sixteen timesharing lines and some batch work. It should perform at least as well as the 512KB 370/145 in a concurrent batch and timesharing environment. Likewise, the 128KW Sigma 9 system shown in the second Sigma 9 configuration should match the performance of the 370/158 in a multi-use environment when all thirty-two timesharing lines are active.

IBM SYSTEM 370
MINIMAL TSO CONFIGURATION
(512KB 370/145 Running OS/MVT)

Qty	Device	Monthly Lease	Purchase
1	3145-12 CPU with 512KB Memory	$16,605	$ 797,000
1	3047 Power Unit	350	16,800
1	3333 Disk Controller with 2-100MB Spindles	1,627	65,000
1	3210-1 Operator's Console (15 Chars/Sec)	175	5,600
1	7844 Console Printer Adapter	189	9,070
1	4660 Integrated Storage Control for 333	2,025	81,000
1	3420-5 Tape Drives (1600 bpi; 125 ips)	475	18,170
1	6631 Single Density Feature for Tape Drive	85	3,260
1	3803 Tape Controllers	675	25,820
1	6631 Single Density Feature for Tape Controller	85	3,260
1	2501, 500 CPM Card Reader with Controller	510	25,460
1	1403-2, 600 LPM Printer	750	28,030
1	2821-2 Printer Control Unit	600	23,040
1	3704-A1 Communications Controller	646	26,000
1	Type 1 Line Interface Bases	24	960
8	Type 1A Line Sets (Each Handles 2 Lines) at 35/1450	280	11,600
1	Type 1 Line Scanner (Program Controller Character Assembly)	24	960
1	Communications Channel Adapter	129	5,300
	Total cost	$25,254	$1,146,330

IBM SYSTEM 370
TYPICAL 32 LINE TSO SYSTEM
(1.5MB 370/158 Running OS/MVT)

Qty	Device	Monthly Lease	Purchase
1	3158-JI 1.5MB 370/168 CPU	$38,500	$1,845,000
1	4650 Integrated Storage Control for 3333	2,200	106,800
1	3333 Disk Controller with 2-100MB Disk Drives	1,627	65,000
1	3330-1 100MB Additional Disk	770	31 000
1	2305-2 11.2MB Swapping Drum	3,900	155,810
1	2835-2 Drum Controller	2,500	99,880
6	3420-5 1600BPI; 125IPS Tape Drives	2,850	109,020
6	3550 Dual Density.Features for Tape Drivers	660	25,260
1	3803 Tape Controller	675	25,820
1	3550 Dual Density Feature for the Controller	110	4,210
1	2540 Card Reader-Punch (1000/300 CPM)	710	32,930
1	2821-1 Unit Record Controllers	970	37,180
1	1403-N1, 1100 LPM Printer	875	33,970
1	3704-A2 Communications Controller	764	31,000
2	Type 1, Line Interface Bases	48	1,920
16	Type 1A Line Sets	560	23,200
1	Type 2 Line Scanner	118	4,800
1	Communications Channel Adapter	129	5,300
	Total cost	$57,966	$2,645,100

XEROX SIGMA SYSTEM
Minimal CP-V Configuration
(Supports 16 lines and some batch)

Qty	Device	Monthly Lease	Purchase
1	64KW Sigma 9, Mod 2	$ 7,800	$275,000
1	7012 Keyboard and controller	150	6,000
1	7122 400-CPM card reader	400	12,000
1	7270 Controller and (2) 49MB disk drives	1,600	65,000
1	7315 Controller and 800 bpi tape drive	600	16,000
1	7440 600 LPM printer	875	35,000
1	7630 Comm. controller and 8 lines	350	14,000
1	7631 8 line expansion	145	5,800
	Total cost	$11,920	$428,000

XEROX SIGMA SYSTEM
Typical CP-V Configuration
(Supports 32 lines and batch)

Qty.	Device	Monthly Lease	Purchase
1	8610E 128KW Sigma 9	$16,500	$525,000
1	7012 Keyboard/printer & controller	150	6,000
1	7270 Controller and (2) 49MB disk drives	1,600	65,000
4	7271 Add-on 49MB disk drives	2,200	90,000
1	7231 Extended performance RAD controller	350	14,000
1	7232 6.2MB extended performance RAD	1,250	50,000
1	7330 1600 BPI tape controller	710	28,400
6	7332 1600 BPI tape drives	2,610	111,000
1	7441 1000 LPM printer	1 150	46,000
1	7630 Comm. controller + 8 lines	350	14,000
3	7631 8 line expansion	435	17,400
1	7160 300CPM card punch	800	32,000
	Total cost	$28,105	$998,800

UPGRADING TO TSO

If a batch shop wishes to upgrade to TSO, it is going to encounter some significant hardware and software costs. While these costs can vary widely from installation to installation, consider a typical example in an attempt to quantify the expense. Since the same level of peripherals and memory are needed for TSO, no matter what IBM system is really expanded, these costs are an accurate estimate for any TSO upgrade. Assume a 370/158 with one megabyte of core running OS/MVT, release 21.6. We shall add the capability to support thirty-two timesharing terminals, each using a 20K word program. Extra resources will be added as needed to keep the impact on the current system to a minimum.

Adding More Memory

Since TSO requires its own region (or regions depending on the number of users), more memory must be added to the system if we are not going to sacrifice memory which is currently devoted to batch processing. TSO requires incremental storage of the following magnitude for a variety of modules, not including the users area.

Module	Additional Bytes
Addition to nucleus	4,000
Master Scheduler	4,000
Link Pack (SVC's)	16,000
Message Control Program (TCAM)	52,000
Timesharing Control Region	100,000
Foreground LSQA and Control Blocks	12,000
Total to Support One Region	188,000 Bytes

Although the above numbers are the ones published by IBM, actual user interviews have shown memory requirements range from 140KB to 300KB purely for the TSO system modules needed to support a single TSO region. Each additional region requires about 15KB for system tables and scheduler enhancements. For this example we will use the IBM figures: 188KB plus 15KB per additional region. Since IBM strongly discourages more than twenty active terminals per region (fifteen or sixteen is a better upper limit) the proposed system will require two TSO regions.

It is interesting to note that in almost any situation there is no way to avoid adding this memory. Of all the features which are listed above, only TCAM has any reasonable chance to already be resident on the system. This would happen if the installation were already using some sort of on-line system such as transaction processing or ATS.

Adding More Memory (Continued)

The incremental memory requirements for the TSO system are:

TSO support for first region	188,000 Bytes
First Region	80,000 Bytes
Additional TSO system requirements for second region	15,000 Bytes
Second Region	80,000 Bytes
Total Additional Memory Requirements for 32K User TSO	363,000 Bytes

On an IBM 370/158, the next memory increment is 1.5MB. This results in an increase in the monthly rental for CPU and memory from $35,900 to $38,500 giving a jump of $2,600 per month. The purchase price goes up to $1,845,000 from $1,730,100 for an increase of $114,900.

Adding a Swapper

Upgrading to the TSO option will also require the addition of a swapping device. Either a 3330 type disk system or a 2305 fixed head drum will work. The former would be chosen when cost is a controlling factor, but this will sacrifice timesharing performance. The most likely choice will be the fixed head device. The cost and performance of both devices are summarized below:

Device	Capacity	Average Access Time	Transfer Rate	Monthly Rent (Device)	Monthly Rent (Controller)	Purchase (Device)	Purchase (Controller)
2305-2	11.2MB	5MS	1.5MB/sec.	$3,900	$2,500	$155,810	$99,880
3333	200MB	30MS	806KB/sec.	1 627	included	65,000	included

Adding Communications Gear

Normally, a pure batch system will include no communications equipment, so it too must be added. The least-cost way to add a flexible system to support thirty-two lines would be to use a 3704 Model A1 communications subsystem. The rest of a 3704 and related equipment is shown on the following page.

Item	Monthly Lease	Purchase
3704-A1 Communications Controller	646	26,000
Type 1 Channel Adapter	129	5,300
Type 2 Communications Scanner	118	4,800
Type 1 Line Interface Bases (one per 8 line sets) (2)	48	1 920
Type 1A Line Sets (one per 2 lines) (16)	560	23,200
Total Communications Costs For 32 Lines	1,501	61 220

If an installation wishes to increase to more than 32 lines, the 3704 must be upgraded to a much more expensive 3705. Also, note that these costs for communications gear are only approximate; the actual list of available features and options for the 3704 and 3705 is quite forbidding and costs can vary greatly depending on line type and speed and the type of terminal to be attached.

Adding TSO Software

Since IBM is almost completely unbundled, all non-systems software has a monthly charge which must be a serious consideration to potential users. Although each installation has its own software needs, a TSO installation which takes full advantage of all the compilers that are supported on TSO would require the items shown below:

System	Monthly Cost	
TSO Assembler prompter	$ 30	
TSO Fortran IV prompter	30	
TSO Cobol prompter	30	
TSO ITF BASIC	120	
TSO ITF PL/1 (CALC is included)	60	($120 if BASIC is not already on system)
TSO Data Utilities (copy, list, format, Merge)	145	
TESTCOB Cobol Interactive Debug	220	
TESTFORT Fortran Interactive Debug	150	
*Code and Go Fortran	275	
*PL/1 Optimizing Compiler with TSO prompter	185	
*PL/1C Checkout Compiler with TSO prompter	340	
PL/1 Transient Library	40	
Total TSO Software Cost	$1,625	Per Month

*indicates that this software may already be installed on a batch system

Adding TSO Software (Continued)

Combining all of the above figures, the cost to add a thirty two user TSO system to an IBM 370/158 comes out to:

Item	Purchase	Monthly Rental
512KB Memory	114,900	2,600
2305-2 Fixed Head Device	155,810	3,900
1835-2 Controller	99,880	2,500
3704 Communications Controller System	61,320	1,501
TSO Software	---	1,625
Total Cost to Add TSO	$431,910 Plus $1,625 Per Month for Software	$12,126 Per Month

Even if an installation wished to add only a minimal timesharing capability (as in the 16 line example used previously) many of the costs would not vary much. The installation might be able to avoid additional direct access devices but the memory increment would still be the same. The communications needs would include:

Item	Monthly Lease	Purchase
3704-A1 Communications Controller	$ 646	$26,000
Type 1 line interface base	24	960
Type 1A line Sets (8)	280	11,600
Type 1 Line Scanner	24	960
Communications Channel Adapter	129	5,300
Total Cost for 16 Lines	$1,103	$44,820

Combining all costs will result in the following total expense to add a minimal sixteen line capability to an existing 370/158:

Item	Monthly Lease	Purchase
512KB Memory	$2,600	$114,900
3704 Communications System	1 103	44,820
TSO Software	1,625	–
Total Cost to add a minimal TSO	$5,328	$159,720

Adding TSO Software (Continued)

Note that this cost of adding TSO to an existing system does not take into account the possible degradation of batch processing which can occur when a timesharing load is added to an already busy CPU. "Fortunately" for IBM, their CPU's are frequently idle so the system may be able to support significant additional work as long as the critical resources (memory and peripherals) are available. Since we had added both resources it is possible that the batch performance may not degrade noticeably. However, if the CPU is already "well" utilized (60% – 80%), we can expect the batch capability of the system to degrade when TSO is brought up. Thus, a difficult-to-measure, but very real cost in degraded performance can result from adding TSO to an existing system. However, the cost depends both on the current job mix and the timesharing load to be added. Since there is no fair way to make a general statement about the magnitude of the cost, it has not been added into the above numbers, but all potential TSO installations should be aware of it.

SUMMARY OF COSTS

A summary of relative costs emphasizes how Sigma and CP-V compare with IBM, when buying or leasing a timesharing capability.

	Acquire a new System 370		Upgrade an existing 370/158		Acquire a Sigma system	
	Lease	Buy	Lease	Buy	Lease	Buy
Minimal System	$25,254	$1,146,330	$ 5,328	$159,720	$11,920	$428,000
Larger System	$57,966	$2,645,100	$12,126	$431,910	$28,105	$998,800

SECTION III: USER'S OVERVIEW

Hardware Requirements
Using TSO

HARDWARE REQUIREMENTS

TSO Configuration

To run TSO, a user must first have access to the proper sized computer. The minimum TSO configuration consists of a System/360 Model 50 or larger, or a System 370, Model 145 or larger. At least 512K bytes of memory is needed to provide concurrent batch and timesharing operation. A 360/50 with 384KB will support a limited set of TSO functions but will not support batch and timesharing operations at the same time. Any one of a series of direct access devices (2301, 2303 or 2305 drum or 2314, 2319, or 3330 disks) may be used for a swapping device. Additional direct access space is normally required for library storage, with the size requirement as a function of the given installation.

Terminal Support

The following terminals, or their equivalents, are supported by the two systems:

TSO	CP-V
IBM 1050 Data Transmission System	Xerox Model 7015 Keyboard/Printer
IBM 2741 Communications Terminal	Teletype Models 33, 35, 37, and 38
Teletype Models 33 and 35	IBM 2741 Terminals
2260 Display Station	Tektronix Models 4010 and 4013
2265 Display Station	Datapoint 3300

USING TSO

The Control Commands

To begin operation with TSO a user must first logon in a process similar to the logon for CP-V. The user specifies a user ID, optional password, a file containing the JCL to control the logon procedure, and a specification of user space for the session. The latter two parameters have system assigned defaults. A successful logon connects the user to the Command Language Processor, which is quite similar to TEL in CP-V. The user may then invoke any of a series of processors to build files, manipulate them, and use them. A list of acceptable commands and a short description of each, for each operating system, follows:

COMMANDS arranged in Xerox (CP-V) alphabetical sequence

CP-V Command	TSO Equivalent	DESCRIPTION
BACKUP	NDE*	Saves the specified file on a system tape. In case of a crash in which files are lost, files on the tape will be restored.
BATCH	SUBMIT	Enters the specified file(s) in the batch job stream.
BUILD	EDIT	Accepts a new file from the terminal.
BYE	LOGOFF	Disconnects the terminal from the system and provides an accounting summary.
CANCEL	CANCEL	Cancels a previously submitted batch job.
COMMENT	NDE*	Directs error commentary to the specified device, or counteracts the preceding DONT COMMENT command.
CONTINUE	(CR)	Continues processing from the point of interruption.
COPY	COPY	Copies a file or device input to the specified file or device.
DELETE	DELETE	Deletes the specified files.
DELTA	TEST	Calls the on line debugging subsystem.
DISPLAY	NDE*	Lists the current values of various system parameters.
DONT COMMENT	NDE*	Stops error commentary output.
DONT LIST	NDE*	Stops listing output.
DONT OUTPUT	NDE*	Stops object output.
EDIT fid	EDIT	Calls EDIT to modify a file.
END	END	Terminates the current job step.
FORT4	FORT	Compiles a FORTRAN IV source program.

*No direct equivalent

CP-V Command	TSO Equivalent	DESCRIPTION
GET fid	NDE*	Restores the previously saved core image.
GO	(CR)	Continues processing from the point of interruption.
JOB jid	STATUS	Requests the status of remotely entered jobs.
LIST LIST (A)	LISTDS LISTCAT	Lists file names and optionally, attributes from the account dictionary, tape, or disk pack.
LINK	LINK	Forms the load modules as specified.
LIST	NDE*	Directs the listing output to the specified device, or counteracts the preceding DONT LIST command.
load module name	CALL	Initiates execution of a load module.
MESSAGE text	SEND	Sends the specified message to the operator.
META	ASM	Assembles the specified source program.
OFF	LOGOFF	Disconnects the terminal from the system and provides an accounting summary.
OUTPUT	NDE*	Directs object output to the specified device, or counteracts the previous DONT OUTPUT command.
PASSWORD xxxx	Not a user privilege	Assigns a new logon password for the user.
PLATEN	PROFILE	Sets the value of the terminal platen width and page length.
PRINT	OUTPUT	Sends print output to the line printer and punch output to the punch.
PROCEED	(CR)	Continues processing from the point of interruption
QUIT	(ATTN)	Terminates the current job step.

*No direct equivalent

CP-V Command	TSO Equivalent	DESCRIPTION
RESET	NDE*	Resets all DCB's back to their system default values.
RESTORE fid	NDE*	Restores the previously saved core image.
RUN	CALL	Loads the specified module and starts execution.
SAVE	NDE*	Saves the current core image on the designated file.
SET	ALLOCATE and ATTRIB	Assigns file or device to a DCB or sets DCB parameter.
START	LINK & CALL	Loads a load module into core and starts execution of the program.
STATUS	TIME	Displays the current accounting values.
STOP	END	Terminates the current job step.
Subsystem Calls		
APL	Not available under TSO	
BASIC	BASIC	
COBOL	COBOL	
DELTA	TEST	
EDIT	EDIT	
FLAG	GOFORT	
FORT4	FORT	
META	ASM	
PCL	COPY	
TEXT	Not available under TSO	
TABS	PROFILE	Sets the simulated tab stops at the terminal.
TERMINAL type	PROFILE	Sets the terminal type for proper I/O translate
TERMINAL STATUS	PROFILE	Lists the terminal type and the current values parameters associated with its operation.
U	TEST	Associates on-line debugger.

*No direct equivalent

COMMANDS arranged in IBM (TSO alphabetical sequence)

TSO Command	CP-V Equivalent	FUNCTION
ACCOUNT	SUPER	Add, modify or delete a users authorization.
ALLOCATE	SET	Define files that user plans to execute.
*ASM	META	Invoke the assembler.
ATTRIB	not required	Build list of attributes for data sets.
*CALC	BASIC	Execute statements in desk calculator mode.
CALL	(load module name)	Start execution of a load module.
CANCEL	CANCEL	Cancel a previously submitted batch job.
*COBOL	COBOL	Invoke the Cobol compiler.
*CONVERT	not needed	Change file formats so that files may be passed between processors.
*COPY	PCL	Invoke the processor which copies a file or input data to a specified file or device.
DELETE	DELETE	Delete the specified file(s).
EDIT	EDIT	Build or edit a file.
EXEC	none	Execute a cataloged JCL procedure.
*FORMAT	TEXT	Format a page of output as determined by a series of control words.
*FORT	FORT4	Invoke the Fortran compiler.
FREE	not required	De-allocate data sets.
HELP	none	Obtain information about function and syntax of TSO commands.
LINK	LINK	Build load modules.
*LIST	COPY	Prints all or selected records from a sequential file.

*Means that this capability is available only as an extra cost IBM Program Product. All TSO commands may be entered in "free format" – there are no required starting columns. If a command has a required subcommand which is not entered, TSO will request it.

TSO Command	CP-V Equivalent	FUNCTION
LISTALC	none needed	List all allocated data sets.
LISTBC	COPY MESSAGES	List all messages sent to user.
LISTCAT	LIST	List all cataloged data sets.
LISTDS	LIST(A)	List attributes of all data sets.
LOADGO	compile, run	Compile and execute a program.
LOGOFF	OFF, BYE	Logoff.
LOGON	LOGON	Sign on to system.
MERGE	MERGE (under EDIT)	Merge files.
OPERATOR	none	To define a terminal as an operator's console.
OUTPUT	COPY file name	Direct output of a batch job to a terminal.
*PL1	none	PL/1 compile.
*PL1C	none	Invoke the PL/1 checkout compiler.
PROFILE	several (TERMINAL, PLATEN, etc.)	Establish terminal defaults.
PROTECT	special command not required	Establish passwords and access restrictions.
RENAME	COPY A over B	Change the name of a file.
RUN	FLAG (for Fortran)	Compile load and go.
SEND	MESSAGE	Send a message to the operator or another user.
STATUS	JOB	Discover status of batch jobs.
SUBMIT	BATCH	Submit jobs to the batch stream.
TERMINAL	several (TERMINAL, PLATEN, etc.)	Define operating characteristics of the user's terminal.

*Means that this capability is available only as an extra cost IBM Program Product. All TSO commands may be entered in "free format" – there are no required starting columns. If a command has a required subcommand which is not entered, TSO will request it.

Command	CP-V Equivalent	FUNCTION
TEST	DELTA	Interactive debugging routines.
TIME	STATUS	Display connect and execution times.
PROC	none	Starts a JCL procedure file.
WHEN	none	In JCL procedure allows conditional jobstep execution.
END	none	Terminates a JCL procedure file.

TSO File Conventions

TSO file names are composed of three segments, separated by periods. The first segment is the users ID which was entered at LOGON, the second is any set of one to eight alphanumeric characters, and the last is one of eighteen descriptive qualifiers which tells what sort of processor will eventually use the data file. For example an ASM qualifier would mean that the data is to be used for assembler input, while CLIST would mean that the file is a command list of JCL statements. The descriptive qualifier is necessary because the TSO language and service processors are not compatible in terms of input format. The qualifier permits the system to select automatically the proper file format for the appropriate processor. IBM BASIC requires that each record contain a statement number in the first five locations, but COBOL will not accept such a format. Most processors will accept only upper case data, but certain TEXT processors as well as many user written programs may deal with upper or lower case. Different processors expect different input record lengths. To accommodate this selection of file formats, TSO uses the qualifiers as a guide to construct the file in the proper format. The qualifier also allows the file to be used quite readily. The user says "RUN filename" and the operating system automatically selects the proper combination of compiler and loader.

Although the filename consists of three parts it usually only need be referenced by the middle name. TSO will automatically prepend the account number specified at logon. There can be some confusion if two files have been built with the same middle name but different qualifiers. For example if a user had built both PROGRAM ASM and PROGRAM FORT and then gave the command "RUN PROGRAM" it is impossible to tell which would run. Likewise "DELETE PROGRAM" causes both files to be deleted.

Some other awkward problems arise because of this naming convention. If the user wants to use the file for something other than that for which it was originally built, he must change the name. Since the operating system makes its own decision on when to convert lower case input to upper case and what length and format of record to work with, the user must be quite careful in his dealings with this file system.

TSO files maintain compatibility with OS/MVT batch file formats. This is both a benefit, because of the great flexibility offered the user, and a problem, because of the resulting JCL required to control this flexibility. The complexity is apparent in the way a given file is assigned to a job. In

CP-V the user has a DCB in his program which is referred to by a single SET statement at the users terminal. The SET statement points the DCB to the appropriate file or device. To accomplish the same step with TSO requires two commands. The first is ATTRIB which defines a list of attributes to be later associated with a file. Entries can include blocksize, record size, number of buffers, length key length for keyed files, input or output file, expiration date, error processing and record format. A second statement, ALLOC, ties the file and allocation list to a particular file and also specifies the device on which that file is located and its eventual disposition. While most of these options have system defaults, the defaults can vary with different circumstances so the user must be aware of how and under what conditions the file will be used. It has become very difficult to use TSO without fully understanding the complex IBM/OS file structure.

Because TSO uses the OS file structures, it must share the strengths and weaknesses of those structures. A most significant consideration is the preallocation of direct access space on IBM systems. OS requires a user to specify the size of his file before it is OPENed. At OPEN time the operating system will then allocate that space as a contiguous block on the direct access device. The major advantage is that there is no chaining between logically sequential granules, as there is in CP-V files, so large data transfers or a series of sequential reads or writes will be very fast on IBM gear. Since the records are contiguous there is little excess head movement or rotational delay between successive accesses.

As nice as this file structure is, there are some significant drawbacks. First, once a file space is allocated, none of it is available to another user, no matter how much is actually used by the allocating program.

Second, the user must be pretty sure of how big his file is going to be before he starts. If he guesses too high, he will waste disk space by reserving unneeded space. If he guesses too low, his program will abort unless he also specified how to handle the overflow. In the latter case the system will allocate up to twenty-four additional file segments – one at a time – to the program as needed. The length of these overflow segments must be specified by the user. These segments will not normally be contiguous but will be chained as they are in CP-V, thereby eliminating any performance advantage attributed to a contiguous data file.

The third problem arises when the user starts to release old files and build new ones of different sizes. Since the OS file structure requires that the initially-requested space be allocated contiguously, the disk equivalent of memory fragmentation will occur. To alleviate this problem the disk pack must frequently be reorganized (the files moved around so that no gaps are left between them), a time-consuming process which ties up both a disk and tape drive as well as some CPU time.

The primary argument for using such a file structure and living with the limitations is increased performance: since the file is contiguous, additional seeks are not required to get to subsequent records. This is fine in a batch system where the relatively low number of active jobs will just about guarantee that a given user will not have to share a drive with another active user, and the read/write arm on the drive will not be moved to a second file by another concurrently active job between accesses by the first user. However, in a timesharing environment where there are many more active users than there are direct access devices, each device will usually be shared by several active users. In this environment the chance is small that the arm will not be repositioned by another user between successives reads or writes by the first user. Thus, most reads or writes will require a complete seek routine, obviating the primary advantage of the IBM file structure.

One significant advantage of IBM's technique is that since they are not limited to granule-sized records on the direct access device, they can transfer up to 8,000 words on a single physical read or write. CP-V requires sixteen physical I/O's to accomplish the same transfer. Because of this, programs with large block or record sizes will tend to run faster on IBM equipment.

TSO & CP-V System Processors

Both CP-V and TSO make available to the user a fairly complete line of software. Both offer an interactive, interpretive BASIC which, while not fully compatible with each other, are both quite complete. TSO supports ITF: PL/1 which is an interactive implementation of PL/1 that has been done as an interpretive processor. It is a subset of the full IBM PL/1 compiler but lacks some of the more sophisticated PL/1 capabilities, especially in the I/O area. However, it does include a CALC subsystem that allows the user's terminal to act like a hand calculator. CALC essentially causes each PL/1 statement to be interpreted and executed as soon as it is entered. A significant problem with both ITF:BASIC and ITF:PL/1 is that their data files must be processed by the CONVERT subsystem of TSO before they can be used with other OS-based processors. ITF:PL/1 source statements also must be run through CONVERT before the regular PL/1 compilers will accept them.

CP-V and TSO both feature interactive debugging capabilities for FORTRAN and COBOL. COBOL-EOO under CP-V contains a flexible series of commands that allow the user to trace, interrupt, or breakpoint process, then display values and alter them if necessary. IBM offers a similar capability under TSO with the TESTCOB command. The Xerox FORTRAN Debug Package and the IBM TESTFORT capability give similar power to on-line FORTRAN users.

The above processors are the only interactive compilers/application programs available under TSO. The rest of the systems are purely batch-oriented, although they have been interfaced with TSO. Due to the complex JCL required to set up and initialize the various temporary and permanent files under OS, a series of interfaces, called prompters, (even though the only prompting they perform is to ask the user for file descriptors) have been developed to perform this overhead for the more common systems. Due to the far more direct operation of the CP-V compilers, such initiators are not required and the processors may be called directly. The following prompters are currently available under TSO:

TSO & CP-V System Processors (Continued)

TSO prompter	CP-V equivalent	Function
ASM	META	Initiates the assembler
COBOL	COBOL	Initiates the ANS COBOL compiler
FORT	FORT4	Initiates the FORTRAN IV (G level) compiler
GOFORT	FLAG	Initiates the FORTRAN code and go processor
PL1	none	Initiates the PL 1 optimizing compiler
PL1C	none	Initiates the PL 1 checkout compiler

It is worth remembering that as with the ITF files, GOFORT source files must be run through CONVERT before they can be compiled under the FORTRAN IV compiler. Also, even though the above-mentioned compilers are actuated from a terminal, they retain their batch characteristics, including the requirement for fixed-format source input. Finally, none of the above compilers are reentrant.

TSO lacks several of the interactive processors that are currently available under CP-V. APL and TEXT both run in the same interactive environment that supports the other on-line processors on Xerox equipment, but the IBM equivalents (APL and ATS) each require their own regions. The additional memory requirements to support these additional regions are significant. An ATS system which can support ten terminals will require a 45KB partition. An APL system to support two concurrent workspaces will require a 170KB region. IBM has not yet committed itself to supporting either of the above processors under their new virtual operating systems, although we expect that such support will be automatic.

Other Xerox processors have no interactive equivalent under TSO, although there are usually similar IBM batch processors which can be run on-line:

CP-V Processor	Similar IBM batch processor	Comments
GPDS	GPSS/360, GPSS-V	GPDS and GPSS-V both complete supersets of GPSS/360
CIRC	ECAP-II	No AC analysis under ECAP
SL/1	CSMP III	SL/1 is the only superset of the industry std: C3SL
TOM	RPG-II	RPG-II is more powerful but less convenient to use

In summary both IBM and Xerox offer a wide selection of processors. While IBM offers a much wider selection of batch compilers and applications programs, Xerox offers the superior choice of interactive processors which enable the on-line user to take full advantage of his real-time connection to the system. Also, IBM has some strange compatibility, problems which are not present with CP-V. In some cases files are not directly sharable between different versions of the same processor (ex. ITF: PL1 and full PL 1 or GOFORT and FORTRAN IV), in some other cases interactive processors not run under TSO but require their own private regions. Both IBM's lack of coordination between processors and the lessor number of interactive processors seem to reflect a lack of full consideration for the multi-use concept, a crucial selling point for potential buyers in this marketplace.

TSO and CP-V Editors

One of the most frequently used facilities of TSO is EDIT. Like the CP-V version, TSO's EDIT is used both to build and to manipulate files. Whereas CP-V EDIT always builds the same file format – keyed sequential with the key being the line number – the TSO version operates in either of two fashions depending on whether line numbers are legal for the given file qualifier. If line numbers are not valid for the file type all line references are made relative to a current line pointer, as in "current line minus two". This is known as "context mode". With context mode, the current line pointer can be moved relative to the current location or to the beginning or end of a document, but editing without a line number has some obvious problems. For example, instead of just inserting a line 2.5 between existing lines 2 and 3, a TSO user operating in context mode must first set the current line pointer to the line corresponding to line 1 which matches a special character string and then use the "insert after" subroutine. If the user is not extremely careful in his specification of the contents which identify line 2, he can wind up pointed at another line and update the wrong section of his file.

TSO's EDIT has some very nice features not available with CP-V. the SCAN command involves the appropriate syntax scanner as indicated by the file name qualifier. PL/1, BASIC and FORTRAN scanners are currently available; curiously, there is no COBOL scanner. The RUN command will compile load and execute the source file which was being built or modified by EDIT.

The FORMAT command invokes a Program Product which can be used to format text output, and be compared to a subset of Xerox Text. Commands permit FORMAT to print headings on each page, center text on a page, left and right justify margins, indent paragraphs and preselect page size. However TSO/EDIT should not be equated to Xerox TEXT, as the latter contains much more sophisticated editing and output capabilities and is comparable to ATS, a product not available under TSO.

Even considering the above features, the CP-V version of EDIT would seem superior. TSO/EDIT lacks most of the intra-record capabilities of CP-V/EDIT. The former has only a simple string-for-string substitution capability; the latter features a variety of intra-record operations including insert commentary, delete string, insert string, replace string, delete record containing string, overwrite string, and shift string. TSO/EDIT contains no capability to move lines within the file while CP-V offers both "move and keep" and "move and delete" factors.

TSO/EDIT's use of blanks and TAB is also weak. All tabs are converted to an equivalent number of blanks so subsequent listings cannot take advantage of tabbing on the output device. There is no equivalent of CP-V blank suppression which saves file storage space.

TSO and CP-V Editors (Continued)

A summary of all TSO/EDIT commands and their CP-V/EDIT equivalent with their function is shown as follows:

EDIT/TSO Command	EDIT/CP-V Equivalent	FUNCTION
BOTTOM	not needed	Used in context editing to set point or to last line of file.
CHANGE	E, F, O, P, S	Modify a sequence of characters in one or more lines.
DELETE	DE	Delete one or more records.
DOWN	not needed	Used in context editing to point to subsequent lines.
END	END	Terminate EDIT processing.
FIND	FT, FD, FS	Locate a character string.
*FORMAT	none	Formats and lists data.
INPUT	BUILD	Prepares for data input.
INSERT	IN, IS	Inserts records.
insert/replace/delete	IN, IS, DE	Alternate method to insert, replace, or delete a line.
LIST	TS, TY	List one or more lines of data.
*MERGE	MERGE	Combine all or parts of files.
PROFILE	**none	Define characteristics of terminal.
RENUM	*none	Renumber a file.
RUN	none	Compile, load, and execute a file.
SAVE	not required	Save a data set permanently
SCAN	none	Request syntax checking on input.

*Means that this capability is available only as an extra priced program porduct

**Individual, i.e., PLATEN

TSO and CP-V Editors (Continued)

EDIT/TSO Command	EDIT/CP-V Equivalent	FUNCTION
TABSET	TA	Set tabs.
TOP	not required	Used in context editing to set point or to the first record in the file.
UP	not required	Used in context editing to point to previous lines.
VERIFY	TY	Type any line which has been modified.

TSO/EDIT contains many capabilities similar to those found in CP-V/EDIT. The former also contains some capabilities not found in the latter, although some features such as context editing are there only because TSO must be able to build files that are not numbered, something not necessary with CP-V.

TSO File and EDIT Summary:

- Files are compatible with batch files.
- Processors require different input file formats.
- DCB assignment is not simplified, nor flexible as it is in CP-V.
- Context mode of TSO Editor is not as easy to use as the Keyed method always available in CP-V.
- TSO SCAN syntax scanner is convenient and has no EDIT counterpart in CP-V.
- Intra-record capabilities are weak compared to CP-V/EDIT, and in TSO whole lines cannot be moved.
- CP-V/EDIT contains a more complete and flexible set of editing commands.

TSO & CP-V Interactive Debuggers

Both CP-V and TSO contain powerful interactive debugging tools which are especially helpful in testing and correcting assembly language programs. In TSO the debugging program is called TEST; in CP-V it is DELTA. Both TEST and DELTA contain many of the same features. With either system the user can involve the debugger at any time and then go into his program to set breakpoints, display and change specific locations in memory, and step through programs to trace their execution. The two systems have similar addressing capabilities: symbolic, relative, or absolute hexadecimal addressing may be accepted as input and displayed at the terminal.

Each debugger has some capabilities not found in the other. TEST permits multilevel indirect addressing in a single command, a very nice feature when the user is trying to find his way through a series of address tables. TEST will recognize zoned or packed decimal data formats. TEST has a COPY command which permits the user to copy strings of data from one place to another within the program.

The LIST DCB command will print the contents of a specified DCB in a special, easy to interpret format. Similar commands format and print data extent blocks, task control blocks, and the user storage map. However, these functions can essentially be duplicated by the "address,address/" in DELTA command which displays the contents of successive memory locations and these locations can be DCB's etc. TEST also allows the user to list his Program Status Word directly, while the DELTA user must use separate commands to display the instruction counter, condition codes and floating controls in order to get similar data.

A complete list of the commands available under TEST the equivalent DELTA command and its function follows:

TSO TEST Command	CP-V DELTA Equivalent	FUNCTION
VALUE ASSIGNMENTS (=)	location/	Modifies values in main storage.
AT	location; B	Set breakpoints.
CALL	location; G or "S program U"	Initialize and start a program.
COPY	none	Move data string.
DELETE	DELETE in EDIT	Delete a load module.
DROP	symbol; K	Remove symbols from a symbol table.
END	esc Y	Terminates debugging.
EQUATE	symbol ! value<symbol>	Add a symbol to a symbol table.
FREEMAIN	none	Release a specified portion of main storage.
GET MAIN	address ~~/~~ \	Get a specified portion of main storage.
GO	;P	Restarts an interrupted program.
HELP	none	List and explain the various subcommands of TEST.
LIST	address;address/	Display a portion of main storage.
LISTDCB	none	List the contents of a data control block.
LISTDEB	none	List the contents of a data extent block.
LISTMAP	none	Displays a map of storage.
LISTPSW	none	Displays the current program sta.
LIST TCB	none	Lists the contents of a task control block.
LOAD	START	Brings a program into memory.
OFF	;B	Removes breakpoints.
QUALIFY	;R	Establishes a base for relative addresses.
RUN	esc Y GO	Terminates debugging and completes execution of the program.
WHERE	symbol =	Displays ~~address~~ value of a symbol.

DELTA both includes several functions not available in TEST and also allows much more flexibility in many seemingly equivalent functions. DELTA can be set to display output of memory in terms of machine statements, hexadecimal or decimal numbers, EBCDIC characters, or floating point numbers. TEST output cannot be set to such a variety of formats but will chose a format selected by IBM as most appropriate. TEST has no equivalent of many other DELTA features including the evaluation (=) command, the tracing or tracing breakpoint capability (;Y), memory searches (;W, ;N, ;M, and ;L), memory initialization (;Z), and the printer output (;O and ;J) directives.

DELTA seems to be better engineered than TEST for ease of use. Almost all DELTA directives are single letters or special characters, while the TEST commands are often whole words plus additional operands. This can result in much faster, easier user interaction with DELTA. For example, a common operation with this sort of debugger is to look at a memory cell to learn what instruction it contains, and then look at the contents of the data cell which the instruction is using. With DELTA this is straight forward and requires two commands: "address/" (DELTA responds with the contents of that address) and then either a single "TAB" or another "I" to display the contents of the effective data field.

With TEST, the user must first "LIST address" which displays the contents of the instruction at that address. The user must then calculate the effective absolute address of the data by listing the active base register and adding its contents to the hexadecimal value of the instruction operand field. He then "LISTs" the contents of resulting address. This second step of figuring in a base register may be avoided only if the user knows the symbolic representation of the operand. He then can use the WHERE directive to translate the symbol to an effective absolute address and avoid the awkward base register calculation, but he still must also use both list commands.

TEST breakpoint capability also is much weaker than in DELTA. All TEST can do is set instruction breakpoints. Yet a very frequent debugging problem to be solved is "how did the contents of a memory location get changed?". With DELTA the user simply sets a data breakpoint at memory location in question and then runs the program. DELTA will halt processing when the memory cell is about to be modified and point to the guilty instruction! There is no reasonable way to do this with TEST.

Most of the other DELTA commands are equally more powerful than their TEST counterpart. In summary then, both TEST and DELTA are similar tools designed to do similar tasks. Each contains features not available on the other, but overall, DELTA appears to be the more powerful yet easier to use assembler debugging tool.

Debugger Summary

- DELTA displays memory in a variety of formats – TEST does not.
- DELTA commands are abbreviated – fast and easy to use.
- TEST permits multi-level indirect addressing in a single command.
- TEST allows copying of data strings form one place in the program to another.
- TEST can only get instruction breakpoints – with DELTA we can halt on memory access.

SECTION IV: TSO AND VIRTUAL STORAGE

TSO under OS/VS2, Release 1
TSO under OS/VS2, Release 2

TSO UNDER OS/VS2, RELEASE 1

New Features

The introduction of OS/VS2, the virtual storage implementation of OS/MVT, has a significant impact on TSO. Release one of OS/VS2, which has been in the field for several months now, is almost identical in format to OS/MVT, except that the total size of effective memory is a single vertical space of 16MB. This very large pseudomemory is "squeezed" into a much smaller real memory by using a demand paging algorithm. As is the case with OS/MVT, the effective memory is sectioned into specific regions for foreground (timesharing) and background work. There are also low and high portions of memory assigned to the non-pageable monitor and to pageable portion respectively.

This implementation alleviates one of the critical problems associated with the original TSO – the large memory requirement to support even a few on-line users. With the advent of virtual memory, the TSO requirement is for a large virtual (as opposed to real) memory. The demand paging algorithm will bring the required portions of the TSO routines into real memory only when they are needed, which should not be often when there are only a few on-line users. As the number of active users grows, there will be an increasing demand for the TSO service routines. Real memory must increase in order to keep a higher percentage of the routines in memory at one time or the system will enter a non-productive thrashing condition in which it spends most of the time swapping the required routines into memory and little time doing useful work. An overview of how both virtual and real storage is shown in Figure 4.

Performance Questions

Much has been said about the problems associated with the performance of virtual memory. The problem is that when the ratio of virtual to real memory exceeds 1 1/2 to 1 in a batch environment, thrashing begins and performance falls apart. The ratio for a system supporting TSO is not so clear cut; it depends on the number of on-line users (as is described above) and how active they are. As their number and activity increase, the requirement for real memory will increase even though the size of the virtual system remains constant. Selection of the proper memory size is a most critical factor for both TSO and batch performance in a virtual environment, because there is not the smooth degradation in service when the overload level is reached as there is in CP-V. Instead, when a virtual system reaches a thrashing state, there is a sudden, dramatic and nearly complete disintegration of performance for batch and on-line users alike.

This means that TSO users have a unique problem. Memory size is a critical factor – something which should be used to tune the system if that were possible, but obviously it is not. The system is essentially pre-tuned to a given environment by the selection of memory size and if the anticipated number of users or even their anticipated activity level proves incorrect, TSO performance will be miserable.

A classic example has been in a New York city bank which underestimated the amount of time its TSO users would spend executing large programs (as compared to building and debugging them). The on-line users are experiencing a one to three minute response time during prime shift, even

TSO ARCHITECTURE
VIRTUAL SYSTEM UNDER OS/VS2 – RELEASE 1

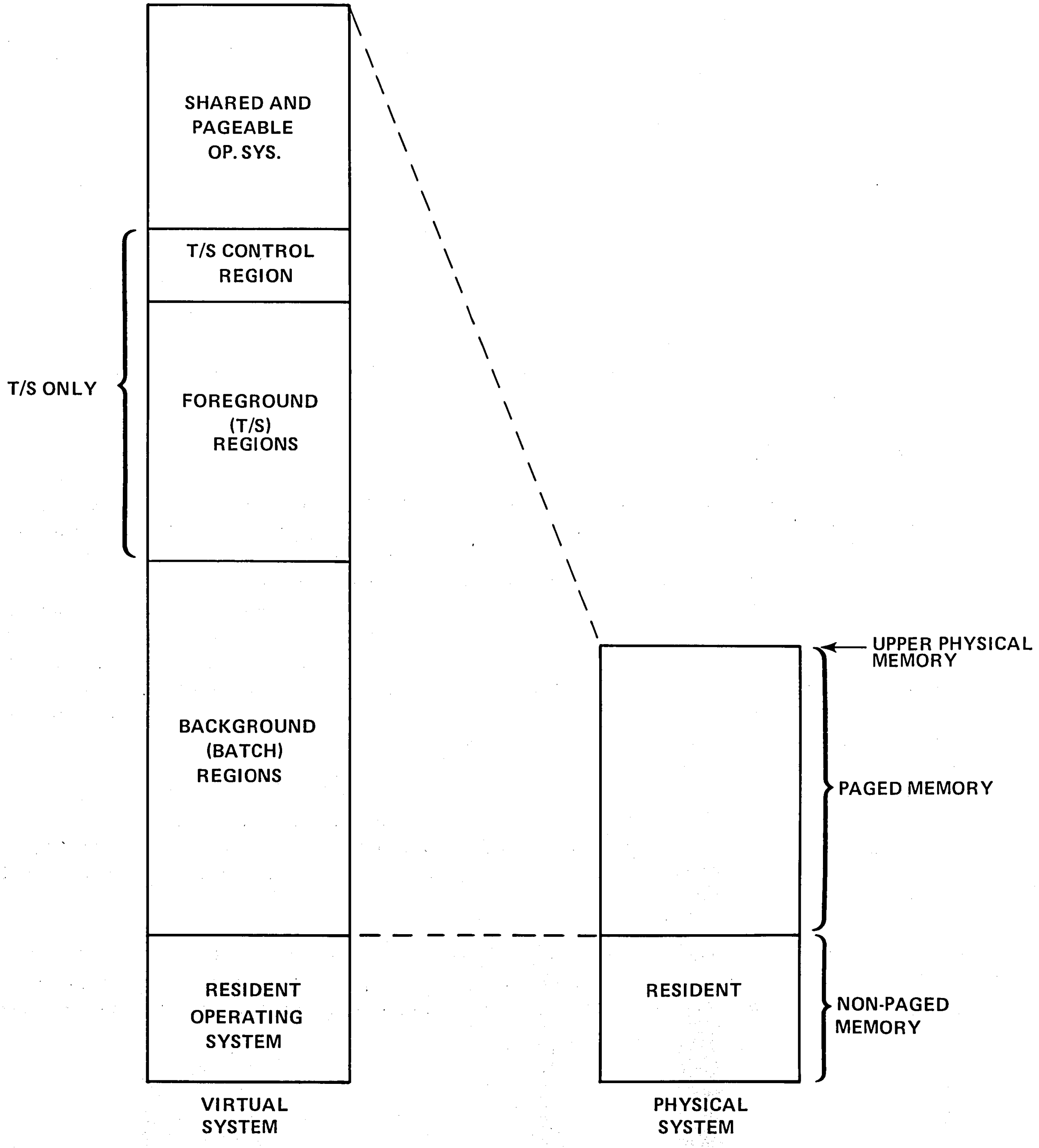

FIGURE 4

Performance Questions (Continued)

though the number of active users is controlled by the number of available lines. Because of the complexity of the process and the number of I/O calls and system subroutines which must be paged in, a simple LOGON requires five minutes. While this may be an extreme case, it does illustrate the pitfalls of incorrectly sizing a TSO system.

IBM was not unaware of the severe impact of thrashing on its virtual systems. To try to control thrashing they build into the operating system a paging monitor to keep track of the paging rates. Whenever this rate exceeds a certain level the monitor starts suspending jobs, lowest priority first, until the paging rate is reduced to a more reasonable level. When the crisis has passed the jobs are automatically resumed. Normally batch jobs are suspended first, but it can happen that thrashing can occur when only timesharing is active. This means that an active on-line user can suddenly find himself suspended in the middle of processing for periods of time ranging from a few seconds to several minutes. While this procedure may control the performance of the overall system, it obviously will result in some very irate users.

Summary

In summary OS/VS2 – Release 1 can help in the solution of TSO's requirement for a lot of memory. However, performance becomes suspect with thrashing being the culprit. It is hard to forecast exactly when thrashing will occur, but when it does performance disintegrates. To control thrashing OS suspends jobs, a viable approach for batch but not for timesharing. Such disintegration does not occur with CP-V; instead an increasing overload condition results in a more predictable and far more gentile degradation in performance. Again, we come back to the fact that IBM has designed a very good batch system and treats TSO as an add-on. This results in some strange performance problems which can seriously impede the on-line user.

TSO UNDER OS/VS2, RELEASE 2

While Release 1 of OS/VS2 is merely a direct conversion of OS/MVT to virtual memory, Release 2, which is due in March 1974, is a major modification to the basic architecture of the operating system. Since IBM has let it be known that Release 2 has about 70% new code (some cynics who are familiar with the way operating systems grow wonder if that just means 70% additional code), it appears that Release 1 is only a stop gap measure until the "real" VS2 is available.

More New Features

In any case Release 2 contains major enhancements including:

- Each user has his own virtual region.
- TSO resources is an integral portion of the system.
- A sophisticated new System Management Facility schedules all system resources.
- Multiprocessing for 370/158 and 370/168 installations.

More New Features (Continued)

The first two enhancements shown above seem to reflect a very important shift in the traditional IBM approach to large scale on-line processing. For the first time they have accepted the timesharing user as equivalent to a batch user and they have given each user, batch or timesharing, access to all system resources. Gone is the concept of treating TSO as an add on, gone is the memory-region concept; instead IBM has suddenly embraced the basic design goals of CP-V. This of course is going to make IBM a much tougher competitor in the traditional Xerox CP-V marketplace.

Probably the most significant enhancement to Release 2 is giving to each batch or timesharing user (and several of the monitor service routines) its own private address space. The approach is essentially the same as in CP-V where each user appears to have the whole system to himself. One significant improvement over the current Xerox implementation is that with VS2 each user is not restricted to physical memory size but instead has its own 16MB virtual memory.

Each user has complete access to all system resources. After LOGON a TSO user is treated by the operating system exactly as if it is another batch job. Because of this the TSO driver and several related system features can be eliminated; all job scheduling is controlled through the redesigned System Resources Manager. This System Resources Manager is quite complex, very powerful and may be a big user of CPU time; it is the subject of the next part of this report. In any event, since all jobs look the same to the operating system an on-line user can for the first time change his allocated number of active DCB's, can ask for tapes or disk packs to be mounted, and can access the slow speed peripherals. Output can be sent to other remote printers or timesharing terminals. As is the case with CP-V the actual complement of devices which are available to a given TSO user depend on his particular authorization which has been set by the management of the computer center.

Use of Memory

Figure 5 on the next page shows the architecture of the Release 2 memory requirements. Note that even with a virtual system the requirement for real memory with Release 2 is significant. The resident, non-pageable portion of the monitor by itself requires 512KB. This space is used only by the operating system and can never be freed for other users. Compare that to the fact that CP-V uses only that much memory for a complete system.

On top of that there is a 5 1/2 MB virtual memory reserved for pageable operating system functions. While this is pageable material and does not need to reside in memory except when referenced, it is obvious that there could be a significant demand on real memory due to this portion of the operating system. The only thing that saves TSO from even more serious problems is that the upper portion is all sharable code so each user does not have to have his own copy. Nevertheless when you consider that this discussion has not even considered the space requirements for the users' programs and we are already allocating 512KB of real memory for the resident operating system and another 5 1/2 MB of virtual memory for the pageable portion, we are discussing a very large memory requirement.

TSO ARCHITECTURE
VIRTUAL SYSTEM UNDER OS/VS2 – RELEASE 2

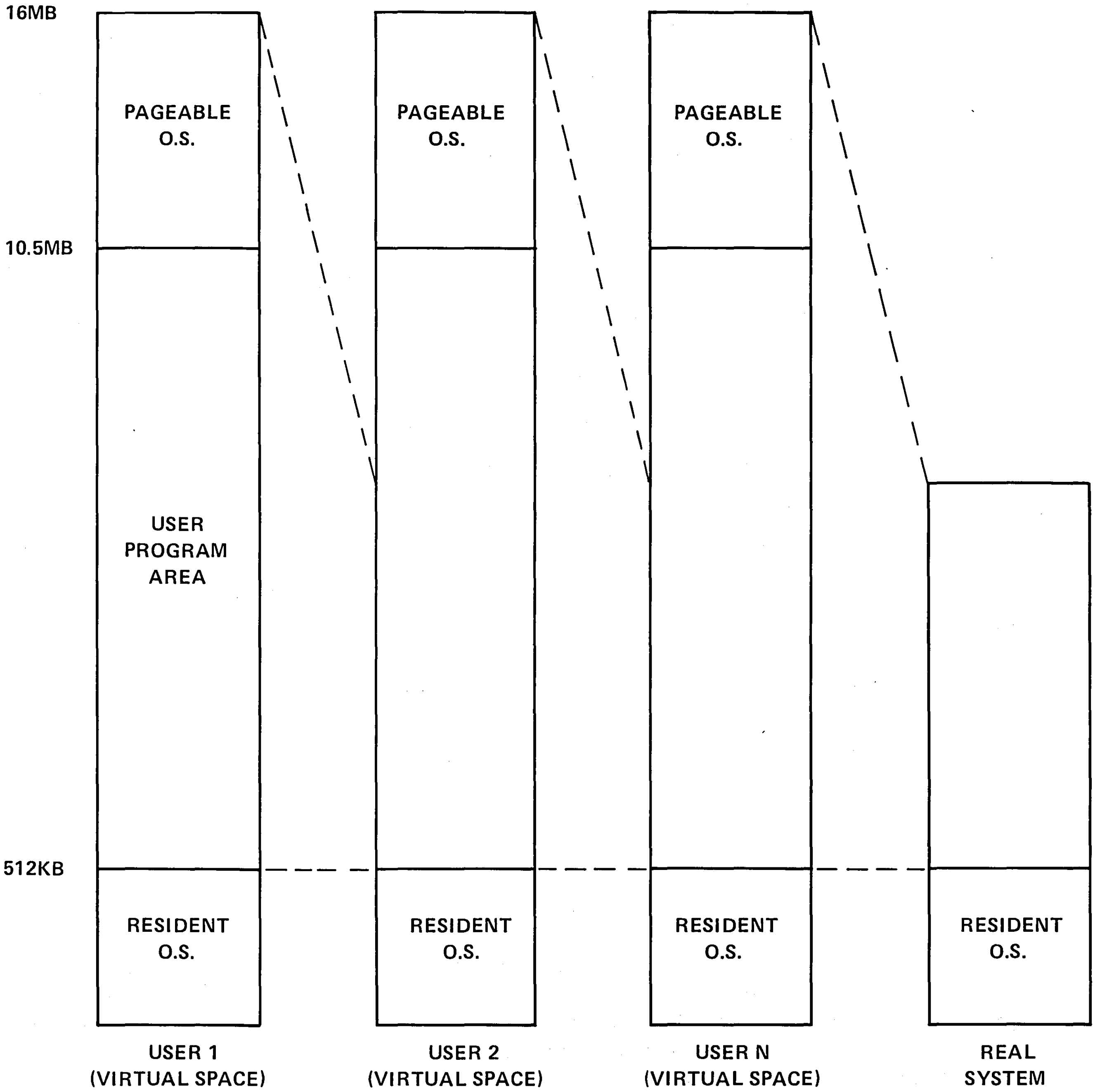

FIGURE 5

Use of Memory (Continued)

In keeping with IBM's new-found concept, demand paging, the concept of swapping a complete user has been eliminated. The demand paging algorithm is responsible for keeping the proper pages, and hence the active user, in memory. This raises the ugly spectre of performance of Release two. As mentioned earlier problems with virtual memory performance have already appeared whenever the virtual is real ratio for a system with a single address space exceeds 1 1/2 to 1. Now we are contemplating a system which permits not one but many (would you believe up to 1536?) address spaces, each as large as the single address space allowed on the previous systems.

More Performance Questions

Curiously, IBM has refused to tell even its preferred customers what to expect in the way of performance from Release 2. This may stem from the bad publicity that resulted when their original virtual memory systems failed to live up to the promises given in the new product announcements.

There is another performance question which arises from IBM's decision to page out instead of swap timesharing jobs. Demand paging and its "Least-Recently-Used" (LRU) algorithm work on the strategy of keeping the most recently used portions of a program or programs in memory. This is based on the assumption that most jobs will tend to continue to use the portion of the program which they just finished using. To use IBM's terminology most programs have a well-defined section which is by far the most frequently used portion of the program. This is called the "working set" of pages, and is usually much smaller than the overall program. As long as the sum of the working sets of all the active programs does not exceed the size of available memory, all of the working sets may reside in memory at one time. In this case there will be little requirement for other pages and the system will not approach a thrashing condition. However, as more programs become active the total working set of the system will increase accordingly. (Note that if the timesharing users are swapped, they are effectively inactive when not in memory and do not affect the working set. However, timesharing users which are not swapped out do add to the working set.) Hence as more users log on or as more batch jobs are initiated the requirement for real memory increases accordingly. As was mentioned earlier, a virtual system is extremely sensitive to memory, so we have that same problem: if the system is not configured for precisely the actual load, the overall performance of the system will disintegrate.

TSO under Release 2 still contains many other problems which have been associated with it since it was first released. It remains cumbersome to use due to the complex I/O structure and the complicated JCL. There is no indication that the file and data incompatibilities have been resolved. Most important there is every indication that system overhead will increase as it always seems to when new features are added.

Summary

In summary then TSO under OS/VS2, Release 2 is a significant architectural improvement over earlier versions. It is like going from BTM to CP—V. But problems still remain and performance is a major question mark. Until IBM can demonstrate that TSO performance under VS2, Release 2 far exceeds both that of earlier releases and all expectations of future ones CP-V remains the superior timesharing system, although the gap has certainly narrowed a great deal.

CONCLUSIONS

A customer will probably make a decision for TSO under the following conditions:

1) He is already a user of large IBM machines.

2) His programmers are familiar with IBM job control language.

3) His primary interactive use will be to develop, checkout and maintain programs to be run in the batch mode on his IBM equipment.

4) He doesn't need to support more than 32 terminals.

5) He does not consider an additional monthly charge of $12,000 for this option to be outside his budget.

When a customer wants a complete timesharing system which offers a full multi-use capability with upward expansion to 128 on-line users, even though points 1) and 2) above are true, CP-V is a logical choice.

APPENDIX: RESOURCE MANAGEMENT IN VS2

The System Resources Manager
Resource-Use Routines

THE SYSTEM RESOURCES MANAGER

A major feature included in Release 2 of OS/VS2 is a new System Resources Manager (SRM) whose function is to solve the problem of an equitable distribution of system resources. While SRM is not intrinsic to the operation of TSO, it is closely associated so it will be discussed in detail here. The SRM provides facilities to:

- Predict and control the response or execution time of any particular job in relation to the system workload.

- Handle various users differently, at certain times allowing some jobs to be favored over others.

- Assure an acceptable level of performance to important jobs.

The objective of the system resources manager is to keep in real storage those address spaces that both best use the system resources and meet installation-specified performance objectives, at any instant of time and under any workload in the system. To meet its objective, the system resources manager has two major functions:

- Managing the workload according to installation-specified performance objectives.
- Managing the use of system resources.

Managing the Workload

In VS2 Release 2 an installation can specify, in measurable terms, the performance that any member of any subset of its users is to receive, under any system workload conditions and during any period in the life of a job. The system resources manager is responsible for tracking and controlling the rate at which resources are provided to users in order to meet the installation's requirements.

The installation sets up an installation performance specification (IPS). Optionally, the system can define more than one IPS, although only one can be used at a time. In the IPS, the installation defines:

- Performance groups — subsets of users that should be managed in distinguishable ways.

- Performance objectives — distinct rates, called service rates, at which CPU, I/O, and real storage resources are provided to users in a performance group at a certain workload level in the system.

Service rate is the number of service units per second a user should receive; a service unit is a measure of processing resources. The system resources manager monitors the rate at which service is supplied to a user in order to ensure that the installation performance specification is met.

The average user is not concerned with the activity of the system resources manager or with the IPS. To take advantage of the system resources manager, he simply identifies the performance group in which he is to be included, as prescribed by the installation.

The following paragraphs describe in greater detail the concepts of service, performance groups, and performance objectives.

Service — The Measure of Performance: Service units are used to measure the amount of processing resources provided to each address space. They are computed as a combination of the three basic processing resources:

- CPU execution, where one unit is the execution of 10,000 instructions.
- I/O measure; i.e., the SMF I/O event count unit.
- Real storage occupancy; i.e., one frame occupied for some multiple of CPU execution units.

The installation supplies coefficients which give a weighting to the relative criticality of each factor. The coefficients are then multiplied by their factors, as illustrated by A, B, and C in the following formula:

$$\text{service units} = A(\text{CPU}) + B(\text{I/O}) + C(\text{frames})$$

When an installation specifies performance objectives, it specifies one or more service rates, which mean how many service units per second a user should receive. The installation is not specifying any particular amount of the individual resources that a user is to receive; it is assumed that different users will use CPU, I/O, and real storage resources in different proportions. However, by supplying coefficients to be multiplied by each resource, the installation can adjust the relative importance of CPU, I/O, or real storage resources within the service definition. For example, if real storage is a critical resource, the installation can increase the value of C in the service formula. Therefore, a job that uses a great deal of real storage will accumulate more service units. Likewise, if real storage is not a critical resource, C can be assigned the value 0, and a user will not accumulate service units at all by using real storage — the real storage factor will drop out of the service definition. Defaults will be supplied for each of the coefficients.

Once the installation has determined the service definition (supplied the coefficients or accepted the defaults), the installation should normally be interested only in the number of service units and the service rates each user receives, not in particular amounts of CPU, I/O, or real storage used. System management facilities (SMF) will record service unit measurements. From SMF reports, the installation can interpret service unit measurements in terms of response or turnaround time.

Performance Groups: The purpose of performance groups is to group user transactions that the installation considers to have similar performance requirements. Basically, a user transaction in a batch environment is a job or job step; in a timesharing environment, a single user interaction. The installation can define as many as 255 performance groups, each identified by a distinct performance group number.

Each performance group can further be divided into as many as eight periods. By dividing a performance group into periods, an installation can associate different performance objectives with different periods in the life of a transaction. The duration of a period can be specified either as a number of real-time seconds or as a number of accumulated service units. For example, one performance group might be defined for short compile-load-go jobs that should have a very rapid turnaround time. The installation divides the performance group into two periods. The first period is associated with a high service rate and lasts until N number of service units are accumulated. Where N has previously been determined to be sufficient service units to complete a short compile-load-go job. If a job in the group is not complete after N service units are accumulated, it enters the second period, which can specify a lower service rate for the duration of the job.

Performance Objectives: A performance objective states service rates, how many service units per second an associated transaction should receive under different system workload conditions. The installation can define as many as 64 performance objectives, each identified by a distinct number from 1 to 64. Note that these numbers are only arbitrary labels and have no intrinsic meaning.

The motive for specifying different performance objectives is to give certain users better service at the expense of other users. However, the relative importance of this motive depends on the size of the system workload (the degree to which the demand for resources exceeds the supply). If the system workload is very light so that the demand for resources does not exceed the supply, all users can be assured satisfactory service rates. However, if the system workload is heavy, the installation will want to assure an acceptable service rate to high priority users and lower the service rate for low priority users. Under very heavy workload conditions, the installation might completely sacrifice the service rate of low priority users by assigning a service rate of 0 in order to continue to provide acceptable service to the high priority users. By defining workload levels, which essentially means the sum of all current demands upon the system, the installation can express the varying service relationships between groups of users at different system workload levels.

The installation can define as many as 32 workload levels, identified by integers from 1 to 128. The numbers on installation chooses to identify the workload levels it defines are arbitrary. A higher workload level number, however, must always indicate a higher system workload and all workload levels defined must have a corresponding service rate included in each performance objective.

For example, an installation defines four performance objectives, numbered 3, 6, 9, and 12. When the system workload is light, each performance objective should assure a satisfactory service rate to the users associated with it; the installation assigns the number 10 to this workload level and assigns corresponding service rates to each performance objective, as illustrated in Figure 6 titled "Performance Objectives" on the following page.

Under very heavy workload conditions, the installation wants to completely sacrifice the service rates for objectives 3 and 6, lower the service rate for objective 12, but still assure acceptable service to users associated with objective 9. This workload level is assigned the number 40; the corresponding service rates associated with each objective are illustrated in Figure 7. The two intermediate workload levels shown in Figure 7 are established to reflect the changing relationships between the performance objectives as the workload level increases from 10 to 40.

The system resources manager tracks both the service rates to maintain the relationships between the performance objectives (and therefore between the users associated with the objectives) that the installation has defined.

PERFORMANCE OBJECTIVE NUMBER	SERVICE RATE FOR WORKLOAD LEVEL 10	SERVICE RATE FOR WORKLOAD LEVEL 20	SERVICE RATE FOR WORKLOAD LEVEL 30	SERVICE RATE FOR WORKLOAD LEVEL 40
3	40	20	10	0
6	30	15	0	0
9	50	45	40	30
12	70	50	35	15

Figure 6. Performance Objectives

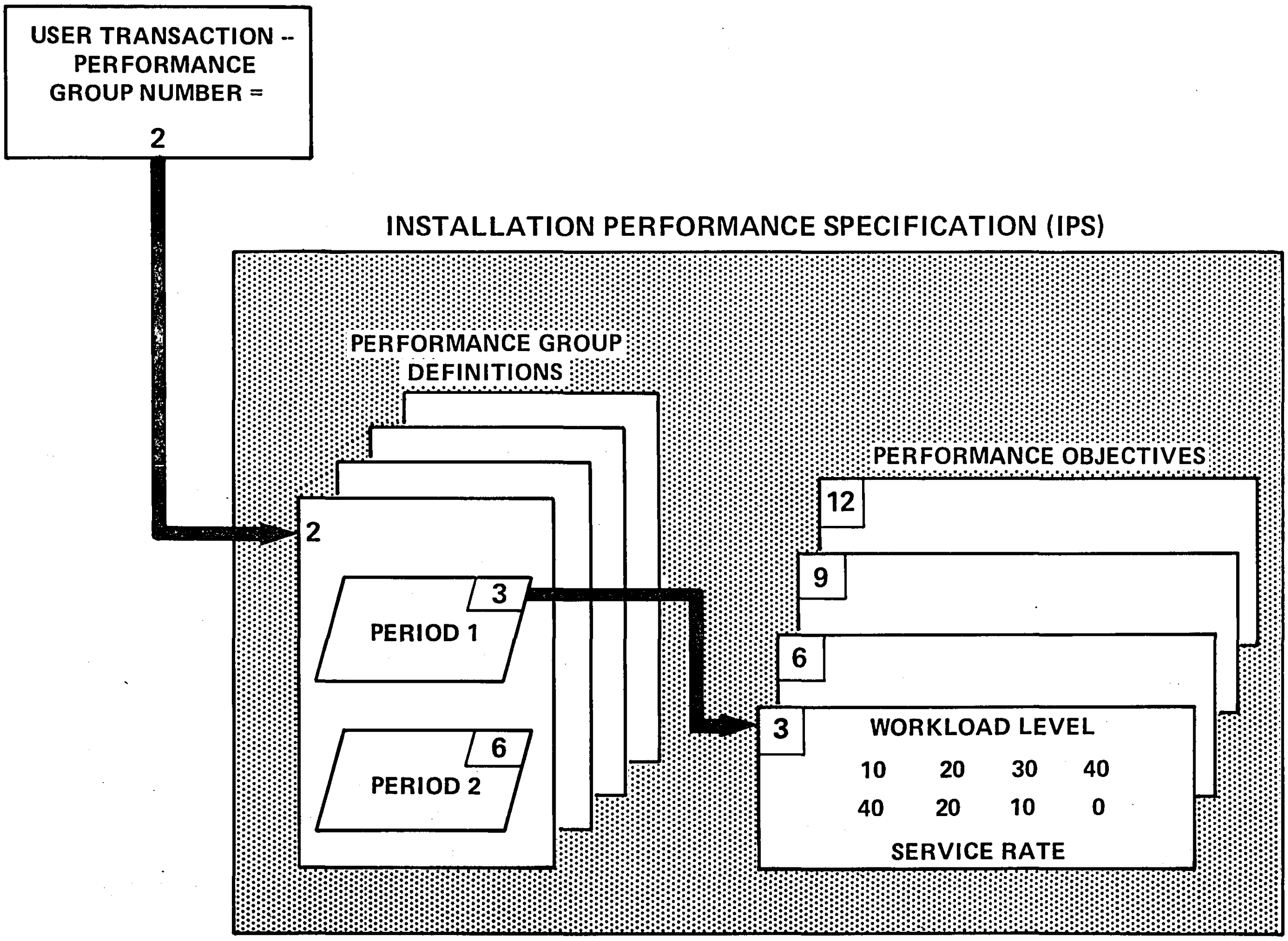
USER TRANSACTION -- PERFORMANCE GROUP NUMBER =
2
INSTALLATION PERFORMANCE SPECIFICATION (IPS)
PERFORMANCE GROUP DEFINITIONS
2
3
PERIOD 1
6
PERIOD 2
PERFORMANCE OBJECTIVES
12
9
6
3
WORKLOAD LEVEL
10 20 30 40
40 20 10 0
SERVICE RATE

Figure 7. Associating a User With a Performance Objective

Associating a Transaction with a Performance

Objective: User transactions are associated with performance objectives by means of performance groups and periods within each performance group: each period of a performance group definition includes a performance objective number.

For example, performance group 2 includes timesharing student users; it is divided into two periods. The first period lasts until 200 service units are accumulated and is associated with performance objective 3 in the illustration titled "Associating a User with a Performance Objective". At workload level 10, the user transaction will receive 40 service units per second. As the system workload level increases the service rate for the job drops in accordance with the performance objective. In this case, we can see that at the heaviest workload level – level number 40 – the job will not be allocated any system services.

Since the performance group specified for this job has two periods associated with it, when the job enters the second period it becomes associated with performance objective six. Normally the new performance objective will allocate a different set of service rates for the various workload levels. In this way the system can automatically associate either more or less system service with the job after it has been running for a while.

This is obviously a very sophisticated technique for allocating service to jobs at a variety of levels. It essentially tunes the system by degrading or suspending low-priority jobs by following the complex descriptions supplied by the installation management. However, a bit problem is that management must devote a lot of time to define and construct the data for the SRM. The installation management must analyze its own job mix, define a series of appropriate Performance Objectives, combine them in appropriate fashion in a list of Performance Group definitions and see that every batch and timesharing job selects or is assigned to the proper Performance Group. These are all critical decisions because what the installation is doing is essentially programming the scheduler. Errors here just about guarantee substandard system performance. In effect IBM has given the user a very sophisticated tool, if the installation management is not as sophisticated or does not understand the demands on his system, he can destroy throughput of a very expensive machine.

IBM has not yet indicated the overhead required to support such a complex system, yet it must be considerable. SRM requires that the operating system monitor the service rates allocated to every active job and make sure that the requested performance objectives are being met. The frequency of observation is yet another installation parameter, but IBM recommends sampling at 250 millisecond intervals – four times per second.

Thus we have a familiar scenario for IBM customers: a very powerful system but one which requires considerable sophistication to use and which can use considerable service time in an effort to save additional time.

RESOURCE-USE ROUTINES

In addition to the above-mentioned workload management routines, VS2, Release 2 features a variety of resource-use routines. The routines consist of a set of algorithms that provide recommendation values for managing system resources on an absolute basis without regard to the installation's job performance specifications. These algorithms include:

I/O Load Balancing

This algorithm attempts to maintain a dispatchable job mix that has a balanced use of logical channels. The algorithm monitors the I/O load and produces recommendation values that indicate when swapping is necessary to correct a detected I/O subsystem imbalance. The I/O load balancing algorithm can be used only when SMF data set activity recording is being performed.

CPU Load Balancing

This algorithm produces recommendation values for maintaining a job mix in which jobs that make heavy use of the CPU are dispatched at reasonable intervals while keeping CPU utilization at a high level.

Main Storage Occupancy

This algorithm determines the amount of swapping necessary to maintain real storage occupancy within certain bounds. It also determines whether or not the bounds are adequate, and adjusts them if they are not.

Page Replacement

This algorithm determines candidates for page stealing. Page stealing takes place when a certain page within a user's private address space has gone unreferenced by that address space for a certain amount of execution time, and when pages within the pageable link pack area have gone unreferenced for a certain amount of time. It results in the page frames associated with that page being freed for other users.

Device Allocation

This algorithm determines the unit to be used for data sets for which an allocation choice must be made from a list of more than one device candidate. The goal is to allocate from lightly-utilized channels, and to collect allocations for the same job on the same logical channel without collecting them on the same device, so that suspending that job will have a predictable effect on the system I/O load.

Automatic Priority Group (APG)

This algorithm reorders dispatchable address spaces within the automatic priority group based on the degree to which they are I/O bound; within the APG, I/O-bound jobs are given higher dispatching priorities than CPU-bound jobs.

ENQ/DEQ Algorithm

This algorithm determines when a batch or terminal job is in control of a system resource and is delaying other jobs that are enqueued on the same resource. When such a determination is made, the job that is enqueued upon the system resource is made non-swappable for an installation-specified time interval.

System Activity Measurement Facility (MF/1)

The system activity measurement facility (MF/1) is new in VS2, Release 2. It is the UTSPM in that installations may use it to monitor selected areas of system activity and obtain feedback in the form of trace records and/or formatted reports. Measurements may be gathered independently on:

- CPU activity
- Channel activity and channel-CPU overlap activity
- I/O device activity and contention
  - Unit record devices
  - Graphics devices
  - Direct access storage devices
  - Communication equipment
  - Magnetic tape devices
  - Character reader equipment
- Paging activity
- Workload activity

MF/1 activity is divided into three categories:

- The System Activity Measurement Gathering routines obtain the measurements of system activity requested by the installation.
- They consist of distinct sets of measurement gathering routines associated with each class of measurement data. Only those sets associated with the measurement data requested by the system operator are loaded and active during MF/1 reporting intervals.
- The System Activity Report Generation routines produce formatted reports of desired system measurements.
- They generate formatted summary reports of information requested by the system operator, collected by the system activity report operator, and routed to it by the measurement facility control. Printing of these reports may be done as they are generated, or may be deferred until MF/1 termination.
- The Measurement Facility Control routines control the collection, tracing and reporting of system activity. They control MG/1 operation. They cause the loading of the measurement gathering routines that will be needed to obtain the desired information, pass control to these routines at the desired intervals, route collected information to SARG as required, and terminate MF/1 execution at the end of its specified duration.