The largest IBM programming effort since the introduction of OS/360, the virtual storage operating system OS/VS2-2 has been designed to integrate efficient support for interactive, data base, and data communications applications.

Discussed from the point of view of its designers are tradeoffs, options, and objectives of the architectural features related to system parallelism, main storage exploitation, system resource allocation, and system recovery.

Functional structure of IBM virtual storage operating systems Part II: OS/VS2-2 concepts and philosophies

by A. L. Scherr

This paper discusses Release 2 of the IBM System/370 virtual storage operating system (OS/VS2-2) from the viewpoint of its designers, who have provided the primary control programming support for the large-scale models of the System/370 product line—Models 145, 155II, 158, 165II, 168, and the shared main storage multiprocessing versions of System/370, Models 158 and 168. The features of the operating system are described in the context of goals that the designers have attempted to meet. Since both OS/VS2 Release 1 and Release 2 are built on the base of the System/360 Operating System for Multiprogramming with a Variable number of Tasks (OS/360 MVT), differences in design and philosophy between the two systems are compared throughout this paper. It is assumed that the reader has a basic knowledge of OS/360 MVT,[1,2] and has been exposed to the overall concepts of virutal storage,[3] System/370, and OS/VS.[4-6]

technological perspective

The operating system OS/VS2-2 represents such a thorough revision of OS/360 MVT that, in many respects, it is a new system. The need for a new control program base can best be understood by considering several key points regarding OS/360 MVT.

OS/VS2-2 is based on OS/360 MVT, which, in turn, is based on OS/360 Release 1. The latter was designed to support useful work in a machine having 32K bytes or more of main storage.

OS/VS2-2 is aimed at supporting systems with substantially larger main storage, and requires at least 768K bytes. OS/360 MVT operates in an environment with at least 256K bytes of main storage. Thus the minimum OS/VS2-2 system is a factor of 3 larger than OS/360 MVT and a factor 24 larger than OS/360 Release 1.

Although OS/360 MVT is designed basically for batch processing operations, it also supports activities such as interactive, data base, and data communications applications. Most of the techniques for this type of support were developed after the initial system design.

As a consequence of these storage and application limitations, OS/VS2-2 has been designed and oriented toward larger main storage environments and the needs of more advanced application areas and operational environments. These objectives have been accomplished in essentially three major evolutionary steps.

The Time Sharing Option (TSO) was available in 1971.[7,8] An extension of OS/360 MVT, TSO is an integral part of OS/VS2-2. TSO is designed to provide general-purpose time-sharing support of OS/VS2-2-OS/360 compatible programs. Available in 1972 was OS/VS2 Release 1, which supports OS/360 MVT and TSO functions in a virtual storage environment for System/370 machines with Dynamic Address Translation (DAT). Finally, OS/VS2-2, previously outlined, is due to be available in 1974. Each of these steps represents a major revision to the base upon which they were built. Taken together, they have evolved OS/360 MVT into a fundamentally new system.

operating system objectives

The overall design objectives for the new operating system—OS/VS2-2—can be differentiated into the following categories:

- Performance
- Reliability and availability
- Compatibility
- Operating environment
- New functions

The OS/VS2-2 design objectives in the area of performance can be expressed in terms of responsiveness and parallelism. In an interactive environment, these goals require minimization of the effect of the execution performance of one application on another and the minimization of software-imposed bottlenecks. Another goal has been to automate as much of the performance-related installation activity as possible. That is, we have sought to make the system more self-regulating and self-tuning, to provide for the dynamically changing allocation of resources to applications as their usage changes, and to give to the installation more control and feedback in the area of performance.

It has been recognized that reliability and availability characteristics of our systems have prevented many significant applications from being implemented. Because many proposed applications place the business of its users under the control of the computer system, uncontrolled system failures could cause intolerable losses.

Thus the reliability levels of our systems have tended to make certain user applications infeasible. The objective of OS/VS2-2 is to make a quantum jump forward by providing significantly greater levels of reliability and availability.

Compatibility for users of previous systems—at the object and load module level for code, at the Job Control Language (JCL) level for job streams, and at the command language level for interactive users and operators—are fundamental requirements. Investments of users of System/360 and OS/360 in programming and education on these systems must be protected. Facilities have been added to the new operating system to provide ways for existing functions to continue to be used.

The operating environment of the new operating system should depend less on operations personnel for decision making. Situations where system performance is adversely affected by the speed with which the operator performs his role should be eliminated. In its broader scope, OS/VS2-2 must provide support for multiple systems that are connected either by shared main storage or by shared direct-access storage devices or telecommunications lines. A complex of systems might include connections of both types. Consolidation of the control and operational interfaces of such a complex in a consistent way is a significant requirement.

Requirements for new functions in OS/VS2-2 derive from the objectives previously discussed as well as from the overall objective of providing enhanced support for interactive, data-base, and data-communication applications. An important new item in this category is the Virtual Storage Access Method (VSAM), which provides improved data base support.

In connection with new system functions, another objective for OS/VS2-2 is to provide support for multiple applications running concurrently in a single system complex. Such a complex may include multiple CPUs connected as described previously. There are three key requirements for successful multiple concurrent application executions. Error isolation is required so that failures in one application do not affect the successful operation of other applications or the system itself. Security is also necessary in order that data or functions intended for one application can be protected against access by other programs in the system.

Dynamic resource allocation should make it possible for application programs in the system to be given resources commensurate with their current usage. As this usage shifts, the system must be capable of reapplying its resources appropriately.

The design objectives just discussed have consistently influenced the following topics related to OS/VS2-2, which are the main topics of this paper:

- Parallelism
- Main storage exploitation
- Workload management
- System resource allocation
- Recovery

Parallelism

A number of components of OS/VS2-2 have been redesigned so as to reduce or eliminate performance bottlenecks that were created in OS/360 MVT and were carried over into OS/VS2 Release 1. The resulting effect on the performance of the system is that a given program should experience substantially less delay due to interference from other programs. One effect of these redesigned components is that queues created by the system for various resources and services should be shorter and cause less delay to the programs being served. Compared and described now are changes in the following areas relating to parallelism:

- Job queue
- Data set and device allocation
- Data set catalog
- TSO region
- Control program synchronization in multiprocessing
- CPU dispatching

job queue

In an OS/360 MVT or OS/VS2 Release 1 system, the SYS1.SYSJOBQE data set is used to store information that describes the queue of jobs to be executed by the system. Also controlled by the same data set are the queue of output from completed jobs to be printed, punched, etc. by the system, and status information relating to—among other things—the data sets and I/O devices being used by jobs in execution.

All of this data can be divided into two categories. One category relates to interregion information that is used for communications among scheduling components in different regions of the system. The other case concerns intraregion information used as a work area for the process of scheduling a particular job and for communication between the scheduler and data management (e.g.,

OPEN and CLOSE). The primary reasons for the presence of this information in SYS1.SYSJOBQE rather than main storage is to help reduce the main storage requirements of the system and to provide audit trails on a direct-access storage device for use in recovery after a system failure.

If either the Houston Automatic Spooling Priority system (HASP) or the Asymmetric Multiprocessing system (ASP)[10] is used to perform I/O spooling, a secondary job queue is introduced into the system that contains data similar to the interregion information in SYS1.SYSJOBQE.

In most OS/360 MVT and OS/VS2-1 installations, the SYS1.SYSJOBQE data set is a major bottleneck primarily because of the high usage of the intraregion information. As a result, this data set is often placed on a high-speed device and/or an otherwise low-usage I/O channel.

In OS/VS2-2, the SYS1.SYSJOBQE data set and the bottleneck that it creates are eliminated. This is accomplished by moving the intraregion information associated with each job to a separate area of virtual storage called the Scheduler Work Area (SWA). Each job has its own SWA, and each can be accessed in parallel via the paging mechanism. The interregion information is consolidated into the job queue that is maintained by the OS/VS2-2 counterparts of the HASP and ASP JES2, and JES3 (Job Entry Subsystems 2 and 3).

Access to this job queue does not interact with SWA access. Because SWA does not automatically provide audit trail information for recovery, a special journal is created for this funciton and is stored by JES2 or JES3 in its direct-access spool space.

data set and device allocation

The process used to allocate devices, space on devices, and data sets is heavily serialized in OS/360 MVT and in OS/VS2-1. Serialization occurs not only with allocation for batch jobs, but also for TSO dynamic data set allocation, and for the deallocation process that occurs during end-of-job processing. In most cases, only one execution of any one of these processes can occur at a time. It is possible that tape or direct-access volume mounting may also be a part of this serialized processing. If a particular allocation request in OS/360 MVT or in OS/VS2-1 requires a device that is busy or off-line, all allocations are halted until the request is satisfied. The result of these serializations is that the process has been a serious bottleneck, and has caused excessive response times for allocation requests. In certain cases, serialization has seriously limited the overall performance of the system. The reason for the serialization of the OS/360 MVT allocation appears to be a result of a design philosophy that expected the worst-case situation to apply at each step.

In OS/VS2-2, the allocation mechanism has been redesigned to provide its services with as much parallelism as possible. Allocations for data sets on permanently mounted direct-access devices are not serialized except for the few instructions that are used to update certain control blocks in main storage. (This point is discussed later in this paper in connection with locking). Allocation requests of this type are the most common ones, and the program structure of allocation has been oriented to minimize the processing time for these requests.

Serialization of volume mounting has been eliminated except for the case where the system has exhausted the space on permanently mounted volumes for temporary data-set storage. In this case, all requests for temporary data-set space wait for the completion of the volume mounting.

Serialization for device allocation occurs at the device-type level. That is, a request for a direct-access drive can be processed in parallel with a tape drive request. Moreover, the servicing of requests for two different types of direct-access drives (e.g., the IBM 3330 and the IBM 2314) can be executed simultaneously. The installation can define groups of devices within a type for parallel processing, such as the 3330 drives on a particular channel. Device allocation occurs in a predetermined sequence that is alterable by the installation on the basis of one device type at a time. In this way, deadlocks among simultaneously executing allocations are prevented.

The net result of the redesign is that the allocation process ceases to be bottleneck, and response-oriented requests can be serviced without the delays previously introduced by the queuing of other requests.

data set catalog

The OS/360 data set catalog design also dates back to original release of OS/360 in 1966. This design was oriented toward applications that involved a relatively small number of entries (e.g., several hundred). This catalog becomes slow and difficult to manage as the number of entries and the update activity increase. In many contemporary installations, the number of catalog data sets has grown into the thousands. As a result, the catalog can become a serious performance bottleneck.

In OS/VS2-2, the data set catalog is implemented using an indexed VSAM data set.[9] In contrast to the OS/360 catalog, the indexed VSAM structure is related to the size and structure of the storage devices and the catalog itself rather than merely to the data set names.

TSO region

In the OS/360 MVT version of TSO, time-sharing user jobs share regions of main storage. Several such regions can be created, but

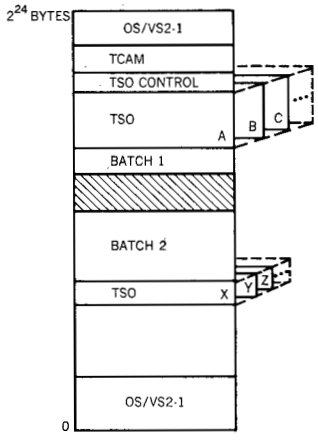
after a user is assigned to a particular region, his job can be executed only from that region of main storage. Hence, TSO users in the system are divided into groups with one group per region. Since only one time-sharing user job can be executing in a region at a time, the number of time-sharing regions determines the level of multiprogramming for time sharing. Also, the situation arises in which one region is relatively lightly loaded compared to the other(s), with no effective corrective action possible.

In OS/VS2-1, the situation is fundamentally the same with the only real difference being that the time-sharing user regions are allocated from a 16-megabyte virtual address space rather than real main storage. As a result, TSO jobs use only that main storage that is required for their active pages rather than, as in OS/360 MVT, using a region the size of which is determined by the size of the largest job to run in that region.

Figure 1 shows a typical storage map of OS/VS2-1 with TSO. Two time-sharing user regions are shown each with a group of users (A,B,C . . . and X,Y,Z . . .). Their placement in the map is a function of when they were created by the operator and what else was in the map at the time they were created. The TSO control region, created when the operator starts TSO, contains extensions to the control program that support unique TSO functions. The TCAM (Telecommunications Access Method) region contains the primary buffers for the TSO terminals and the programs that control them.

In OS/VS2-2, the concept of a time-sharing region shared by TSO user jobs has been eliminated. Referring to Figure 2, conceptually what has been done is to transform the entire system into a single TSO region that extends from the top of the lower system area to the bottom of the upper system area. All jobs—including TSO users and operator-started jobs—that previously ran in unique regions now share the same address space. Multiprogramming, however, can now occur at any level and in any combination. Thus, for TSO, the level of multiprogramming can be adjusted by the system according to the TSO load. TSO programs can now be executed in any combination, and the TSO regions and their potential for imbalance have been eliminated.

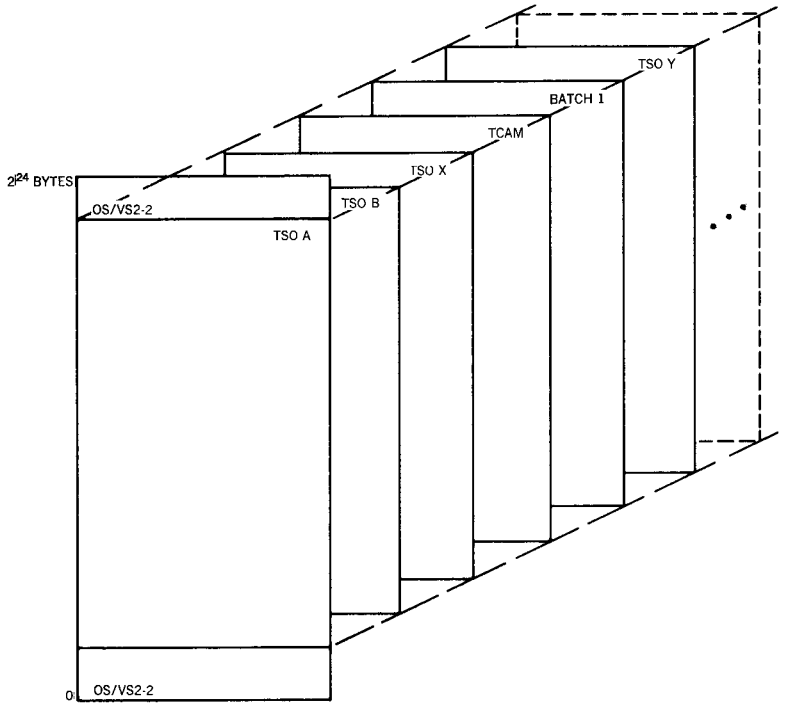
Figure 1 OS/VS2-1 virtual storage map



In OS/VS2-2, the specialized TSO functions that were previously performed in the TSO control region have now been integrated into the main line of the control program. The control region is thus eliminated and all of the following functions previously handled by this special region have been integrated into the main line control program:

- *Swapping*—now used to control the level of multiprogram-

Figure 2 OS/VS2-2 storage map



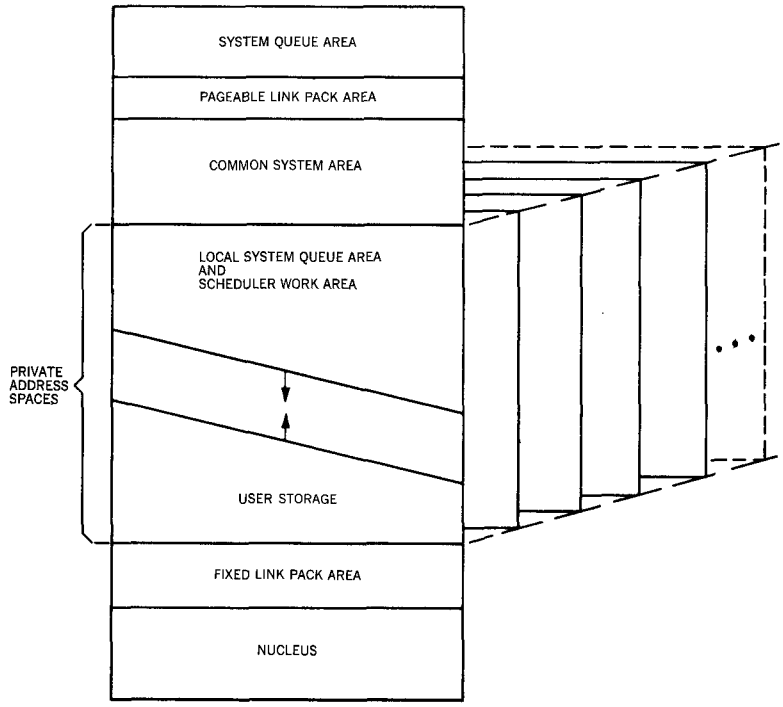
ming; also replaces the OS/360 MVT rollout/rollin function.

- *TSO driver*—replaced by a generalized program to control the entire system in the System Resources Manager.
- *TSO link pack area*—moved to the System Link Pack Area.
- *Secondary terminal buffers*—moved to TCAM.

In the OS/360 MVT version of TSO, the concept of a Local Supervisor Queue Area (LSQA) was introduced. The LSQA, adjacent to the user's region, is used to hold all job-related control blocks that were formerly held in a system-wide pool (called the System Queue Area—SQA). In OS/360 MVT, only TSO jobs use the LSQA so that these control blocks can be easily located and swapped. In OS/VS2 all jobs use LSQA, for the same reason.

Figure 3 shows the storage layout map of a user region in OS/VS2-2 as well as the contents of the system areas in greater detail. The nucleus contains the interruption handlers, the nonpageable supervisor (including the dispatcher, IOS, and the paging mechanism itself), and nonpageable control blocks. The fixed Link Pack Area (LPA) contains sharable re-entrant programs, which are made permanently resident by installation option. A private address space is allocated to each job analogously to a region in OS/VS2-1 or OS/360 MVT. User storage is allocated

Figure 3 Detailed OS/VS2-2 storage map



from the bottom of the space; the System Queue Area (Local) and System Work Area (SWA) spaces are allocated from the top. The Common System Area (CSA) is used as a communication area between private address spaces. This area is required because two private address spaces cannot be directly addressed simultaneously. The CSA is used by such components as TCAM, JES3, and the supervisor for this purpose. The pageable LPA is used for sharable, re-entrant programs that are paged. The SQA is used for nonpageable system control blocks that relate to information for multiple jobs or private memories of information that cannot be in the LSQA. This information is required even if the private address space is swapped out.

multiprocessing locks

In a shared main storage multiprocessing system, the fundamental performance objective is to make full use of the CPUs available to the system. One of the primary requirements for meeting this objective is to structure the control program so that it, too, can be executed on both CPUs simultaneously. The reason the control program might be unable to be run in parallel is that certain control information might be accessed and updated simultaneously, causing an error. Thus, such accessing must be synchronized.

In OS/360 MVT, synchronization is accomplished by disabling the system for interruptions. Prior to the use of information whose access must be synchronized, an interruption-disabling instruction is executed. After the information has been used, the system is again enabled for interruptions. In the OS/360 MVT multiprocessing support, inter-CPU synchronization is accomplished by having a single lock (or interlock) that prevents both CPUs from being disabled at the same time. The second CPU spins in a loop waiting for the first CPU to be enabled. If the proportion of time during which the system is disabled is relatively low, this approach does not lead to significant interference between the two CPUs. For example, if one CPU is disabled twenty percent of the time, the interference is approximately four percent (0.2×0.2). On the other hand, in a communication or data-base environment where there are many interruptions to handle and heavy use of control program services, the interference can be significant. For example, if the proportion of disabled time on one CPU is fifty percent, the interference between the two CPUs is approximately twenty-five percent (0.5×0.5).

In OS/VS2-2, multiple locks—instead of a single lock—are used, and they are defined for specific groups of control blocks and functions. Prior to the use or updating of control information, the specific lock (or in some cases locks) is obtained. The system does not have to be disabled for interruptions while a program holds a lock. Disabling occurs when the program is not prepared to handle interruptions. Either or both of the CPUs may be disabled at a given time.

The following are some of the specific types of locks:

- *Local lock*—used to synchronize services within a private address space. There is one local lock per private address space.
- *Dispatcher lock*—used by the dispatcher and others when referring to CPU work queues.
- *Common services lock*—used to synchronize supervisor services that involve shared address space.
- *Input/output (I/O) supervisor locks*—one per type of control block and logical I/O channel, which are defined to allow for parallel interruption processing.
- *Paging locks*—one for real main storage and one for auxiliary storage.

Other locks are defined for services such as VTAM and JES3, and in some cases, two or more locks are required for a particular operation. The latter locks must always be obtained in a specific order to prevent the possibility of the deadlock situation arising when two programs simultaneously request the same locks in a

different order. A system-wide order for obtaining locks is enforced by the central lock management service routine that is entered to obtain and release all locks.

The net result of this design is to substantially reduce the interference between multiprocessing CPUs. Elements of the control program that hold different locks, regardless of whether they are disabled, can execute in parallel. The relatively large number of locks provides that the utilization of any one lock is small, and thus interference for a given lock is minimal.

Main storage exploitation

Since the introduction of OS/360, the relative costs of main storage, auxiliary storage, and central processing units have shifted significantly. In OS/VS2-2, the goal is to take advantage of the relatively lower cost of main storage to provide improved performance characteristics.

In OS/360 and in OS/VS2-1, main storage is used strictly to hold programs or—more precisely—to hold the images of programs. The addition of main storage to a system has been motivated primarily by the need for more programs or larger programs in main storage. Ordinarily, additional capacity is used to enable new and larger applications to be run or to tune the operation of existing programs in a system, typically by removing overlay structures. The only other productive use of additional capacity is to increase the level of multiprogramming. However, when CPU and/or channel usage approaches one-hundred percent, there can be no further performance gain from increasing the level of multiprogramming.

In the past, increased performance in this situation has been achieved either by higher speed CPUs and I/O devices or by redesigning applications and/or system programs to use main storage to reduce CPU and I/O activity. Such redesign can take many forms, a simple example of which is using a directly addressed main storage work area rather than putting information on a disk or a drum. In this way, not only is I/O activity eliminated, but so is the attendant CPU activity necessary to schedule I/O, to translate channel programs, to handle interruptions, to support task switching generated by programs waiting for I/O to complete, and so forth.

Certainly, in many environments, there are data sets whose usage is high enough to warrant making at least a part of it resident on a higher speed device or even, perhaps, in main storage. In fact, there are environments where some blocks of data receive higher usage than some of the pages of a program. Ideally,

such data should receive preference for main storage occupancy. In OS/VS2-2, the design moves in the direction of allowing data to be placed in the storage hierarchy dynamically, according to its usage. The concept is to allow more data—more information—to be resident in main storage. Thus, given a fixed amount of main storage, there is a better choice of what to put there. More importantly, given a larger main storage, a greater quantity of useful information can be stored there. Automatic means of accomplishing this type of efficiency are provided in OS/VS2-2, which uses the paging mechanism to handle both data sets and program images.

Data sets in the paging hierarchy are called Virtual Input/Output (VIO) in OS/VS2-2. VIO allows certain types of conventional data sets to reside in main storage or in the paging data sets according to their usage, and is provided transparently to the application programs in the system. Thus, given sufficient frequency of use of a portion of the data set and sufficient main storage, data set pages may be resident in main storage when they are called for by the program. In this case, I/O activity is eliminated along with the associated CPU time. In the event that the required data pages are not in main storage, the access is performed by the paging mechanism and is scheduled along with other paging I/O requests. Auxiliary storage access for VIO is optimized on a systemwide basis.

Two other advantages for VIO arise from the fact that space for VIO data sets is allocated as needed, one page at a time. As a result, space is not wasted as it typically has been with conventional data sets for which relatively large blocks of space must be preallocated. Moreover, the relatively lengthy processing involved in allocating space for conventional data sets is avoided.

In OS/VS2-2 VIO is provided for the handling of temporary data sets (i.e., data sets created and destroyed during the execution of a job). The Sequential, Partitioned, and Direct Access Methods (BSAM, QSAM, BPAM, and BDAM) support VIO as does EXCP, the latter being an interface at the channel program level. Neither Job Control Language (JCL) nor the application program need not be modified to take advantage of VIO.

Initially, the choice to implement VIO for temporary data only was made because the design is simplified because the rest of the pages in the paging hierarchy are also temporary in nature. Perhaps even more important is that VIO aids temporary data set processing significantly, and avoids a bottleneck in the space allocation for temporary data sets. Moreover, this is a category of data with predictably frequent use, since every temporary data set is accessed at least twice per job—once for output and once for input.

The use of virtual storage to hold what previously had to be accessed as a conventional data set is another main storage usage that is similar in concept to VIO. However, this technique requires modification of the design of the program. Using a virtual storage work area rather than a work file, as is the case with the Scheduler Work Area (SWA) previously described, is even more efficient than VIO for data not passed from one job step to another because the data can be directly addressed with conventional CPU instructions without the overhead of the I/O access method interfaces.

Elimination of overlay structures by having the entire program in virtual storage allows for the possibility that the required program segment can be resident in main storage when needed. Again, if it is not in main storage, I/O is performed by the paging mechanism with the same advantages as described previously. Moreover, since overlay management usually requires logic in the application program to decide where to go next, where control came from, etc., elimination of this programming results in a saving of CPU cycles.

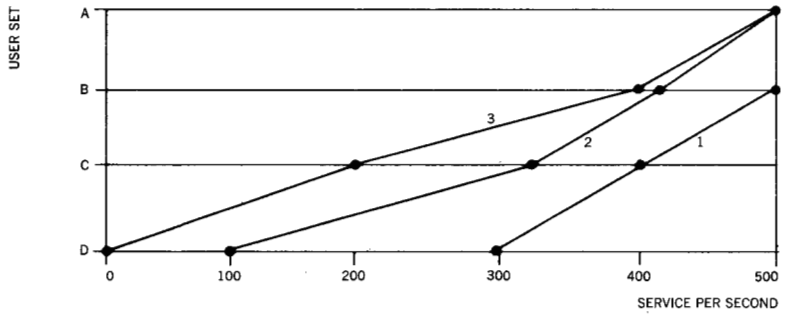
The general technique of expanding the candidates for main storage residency beyond program images only is one that should see more and more usage as main storage sizes increase. As was previously stated, when the level of multiprogramming causes saturation, the only productive use of main storage is to shorten the processing time to access data and to simplify program structures.

Workload management

The keystone of the resource management algorithms in OS/VS 2-2 is the concept of workload management. In previous operating systems, parameters that affect performance of resource allocation have generally been aimed at controlling specific internal algorithms with very little direct relationship to the desired external effects. Thus in OS/360 MVT, for example, setting the dispatching priority of a job has a direct and predictable effect, internal to the system. The job receives all the CPU time it can use from the excess time of jobs with higher priority. Any excess CPU time from a given job is available for jobs of lower priority in the system. However, the rate at which a particular job executes as a function of its priority is, in general, impossible to predict, unless the characteristices of other jobs in the system are also known.

Another example is that of setting the time slice size (or time-slicing algorithm parameters) for time sharing jobs. Internally, the effect is predictable. Jobs have to be swapped in and execut-

Figure 4 Example service rates



ed for no longer than the specified slice size. However, the resulting response time seen by terminal users cannot be directly derived. Moreover, when improved responsiveness is required – whether to increase or decrease the time slice size (or even which parameter to change) is difficult to determine.

OS/VS2-2 emphasizes the provision of controls that relate more directly to the external performance characteristics of the operating system. The primary parameters are expressed as rates at which jobs of various categories are executed. These rates are called *service rates* in OS/VS2-2 and they allow control over responsiveness and turnaround time for particular jobs or groups of jobs. To be more precise, the service rate for a particular job is a linear combination of its CPU time, the number of channel programs started (EXCPS) for user I/O, and the product of the number of assigned pages of main storage and the number of seconds of CPU time received while they are assigned.

Each of the three factors is multiplied by an installation-specified constant and summed as follows:

Service rate = A (CPU sec) + B (EXCPS) + C(CPU page-sec)

Another dimension added to provide for dynamic control is the ability to specify different service rates for various categories of jobs as the load on the system changes.

Figure 4 illustrates the use of the service-rate concept. The horizontal axis represents the rate of service, and the vertical axis shows four categories of user jobs, A, B, C, and D. Curve 1 shows that user sets A and B, which are assumed to be executing interactive jobs, are given the highest service rate. User set C, for jobs with given turnaround-time requirements, is given a lower service rate. User set D has other batch jobs, and has the lowest service rate. Each job in the respective user sets is serv-

iced at the rate specified, presumably set by the installation to provide acceptable performance for each other job categories.

OS/VS2-2 produces a report that shows the turnaround time or the time-sharing response time for each set together with the actual service rate achieved during the reporting period. In this way, service rate and responsiveness can be related, and adjustments can be made on the various service rates. The report also shows the number of jobs (or time-sharing transactions) started and finished during the measurement period.

As was stated previously, the system attempts to give each job in execution the service rate specified for its category. As jobs are added to the system, it may no longer be possible to give each of them the required service rate. In this case, other service-rate curves (2 and 3) may be drawn to indicate the contingency action taken by the operating system. Curves 2 and 3 show that no degradation is to occur for jobs in category A. There is a slight degradation called for in categories B and C, and set D is degraded the most. The system attempts to operate as far to the right as possible and interpolates between curves when necessary. Extrapolation occurs if the operating point falls below curve 3 (or above curve 1). The system report indicates the average curve number used during the reporting period. For example, a load of 2.3 implies that jobs have, on the average, a service of 0.3 of the distance from curve 2 to curve 3.

System resource management

Given the workload objectives of an installation as specified to the workload manager, the objective for the resource management algorithms is to achieve maximum utilization of the hardware configuration. All algorithms, including workload management and the necessary data gathering facilities, have been centralized into the System Resources Manager (SRM). The rationale behind this centralization is simply that the interaction between the various algorithms used to manage the system is high enough to warrant coordination of their actions. Moreover, having all of these routines in a single component allows improvements and modifications to be made more easily. Philosophically, the SRM is an extension of the TSO driver concept so as to cover the entire system.

The functions performed by the SRM include the following:

- *I/O load balancing*—choosing devices for data-set allocation so as to provide balanced use of I/O channels.
- *Heuristic dispatching*—for jobs in specified sets, giving higher

priority to CPU use for jobs that are I/O bound and lower priority to those that are CPU-bound, so as to maximize resource utilization and throughput.

- *Paging control*—controlling paging activity and the number of unused blocks of main storage.
- *Time sharing*—swapping, time slicing, and controling the level of multiprogramming.

The last two items are closely related in that the amount of paging activity in the system is controlled in two ways. Individual jobs in main storage are periodically examined and unused pages in main storage are claimed, copied out to auxiliary storage, if necessary, and the main storage space is placed on the available list. This examination occurs at a frequency that is determined individually for each job, and depends on the amount of time the job has executed since the last examination. The intent of this algorithm is to assure that each job, when it is in main storage, is allotted sufficient main storage to avoid excessive paging. Normally, this algorithm is not sensitive to the overall load on the system, and, therefore, the amount of main storage allotted to a job should be relatively independent of the other jobs in the system.

The second aspect of the paging control algorithm monitors the number of free main storage pages. When the number of pages falls below a given threshold, the algorithm causes a job to be removed from main storage. A job is added in main storage whenever another threshold is exceeded. Adding and removing jobs is done by an operation called "swapping" or "block paging," wherein all job pages are transmitted to generally contiguous sectors in direct-access storage. Selection of a job for swapping is usually based on the service rate criteria described earlier in this paper. The job that is selected for removal is the one that is farthest ahead of the service rate targeted for its user set.

Swapping out also occurs whenever a job enters a so-called "long wait" situation. Typically, this is a time-sharing job waiting for input from the corresponding terminal user. Analogous situations are recognized for other types of jobs with the same result.

The overall effect of the SRM (coupled with the workload management facilities and the elimination of system bottlenecks) is that the performance of the system is easier to tune. Tuning is essentially the manipulation of bottlenecks in the system so that their usage is balanced. Because of the significantly fewer bottlenecks and because of the dynamic resource balancing done by the system, the effort required on the part of the installation to tune the system is significantly reduced. One effect is that the number of static performance parameters that need to be speci-

fied is reduced as is their criticality. Another result should be a reduction in the need for dynamic intervention by the operations staff in correcting resources imbalances and in making tradeoffs between resource utilization and workload scheduling requirements.

Recovery

A major portion of the OS/VS2-2 design is aimed at significantly improving the reliability and availability of the system. *Reliability* is a measure of the frequency of occurrence of software errors; *availability* is a measure of the effect of these errors. Perfect reliability of the system is achieved by having an error-free implementation of a correct design. Many techniques exist for design verification and testing, but, in the final analysis, reliability is ultimately established by testing the resulting programs.

Although advances have been made in recent years in the areas of design and testing techniques, error-free systems of the size and complexity of OS/VS2-2 are still a long way off. As an example of a key problem in achieving high reliability, consider a software system whose minimum mean time to failure (MTF) goal is thirty days. Assume that an MTF of five days has been achieved, and that there are n failure-causing errors remaining in the system. If the software is being tested simultaneously on m hardware systems, and if each error has equal frequency of occurrence, and if each error is corrected immediately after it occurs, it takes approximately 15 (n/m) days to reach a thirty-day reliability level. Assuming one-thousand errors (i.e., one error per thousand lines of code in a million-line system) and ten test systems, the required elapsed testing time is fifteen hundred days or about four years running seven days a week. This approach to error correction is clearly ineffective. Error detection can be accelerated by careful selection of testing environments with emphasis on maximal load and stress situations, but clearly another approach is also required.

It should be noted in passing that, although the testing and debugging assumptions used in this example are quite optimistic, systems with a mean time to failure of thirty days or more are expected to be needed by the end of this decade, if not today.

To achieve high availability in OS/VS2-2, the decision was made to complement the testing efforts by including programs whose purpose it is to recover from the errors that do occur. These programs are an integral part of OS/VS2-2 and are specialized recovery routines, each associated with a particular control program function. Thus, the dispatcher has a recovery routine designed for it, as do the interruption handlers and other system

components. Whenever an error occurs (usually signaled by a program check interruption), control goes to the recovery routine associated with the main-line code having the error. To facilitate recovery, the main-line program creates easily accessible status information that defines its state as it executes.

The first objective of a recovery routine is to eliminate the effect of an error. If, for example, a control block is incorrectly chained, the recovery routine makes use of redundancy in the control block structure to correct the situation. In some cases, resources are lost temporarily as a result of an error. For example, an area of virtual storage might become unusable. This situation would be corrected by deleting the address space that contains the area when it is no longer needed. In cases where transparent recovery is not possible, the goal is to limit the effect of the error as much as possible. In this situation, error indications may be propagated upward to the caller of the function that detects the error. Frequently, the caller is also part of the control program with an associated recovery routine that attempts to correct the situation, possibly by a retry of the operation. Error indications may ultimately be passed to the application program, which may itself handle the error through a recovery exit.

The overall effect of these recovery facilities is to increase the availability of the system by reducing the impact of errors. Problems that formerly caused system failure should have a significantly smaller effect, and allow the system to continue unaffected work successfully. These same recovery routines are also used to handle control program loops and the effects of hardware errors. When there is a permanent hardware failure in one of the CPUs of a shared main storage multiprocessing system, the second CPU is used to execute the recovery routine associated with the program running on the first CPU at the time of the failure.

Overall, the philosophy is to attack the availability problem from two complementary directions: to reduce the number of software errors through rigorous testing of running systems, and to reduce the effect of the remaining errors by providing for recovery from them. An interesting footnote to this design is that now a system failure can usually be considered to be the result of two program errors: the first, in the program that started the problem; the second, in the recovery routine that could not protect the system.

Concluding remarks

In designing a system like OS/VS2-2, which is aimed at meeting a broad variety of objectives in a multitude of complex operational

environments, the tradeoffs and decisions are extremely difficult to decide. The intent of this paper has been to give insight into the thinking behind the major new elements of the OS/VS2-2 design.

OS/VS2-2 is the largest IBM programming effort since the introduction of OS/360 in 1966. OS/VS2-2 integrates many items into the overall system that previously were special-purpose options, such as TSO, ASP, HASP, and shared main-storage multiprocessing. The major new features that have been incorporated into OS/VS2-2 are recovery facilities, VSAM, virtual I/O, the System Resources Manager, the workload manager, and enhanced virtual storage support.

The metamorphosis of OS/360 into OS/VS2-2 represents a significant shift in emphasis toward the requirements of response-oriented, interactive, data-base, data-communications applications. OS/VS2-2 is intended to be the base for the development of these types of applications in the 1970s.

ACKNOWLEDGMENT

The OS/VS2-2 design is the accomplishment of a large number of IBM programmers whose contribution to this paper the author gratefully acknowledges.

CITED REFERENCES

1. *IBM System/360 Operating System, MVT Guide (OS Release 21), IBM Systems Reference Library*, Form No. GC28-6720, IBM Corporation, Data Processing Division, White Plains, New York 10604.
2. "The functional structure of OS/360," *IBM Systems Journal* **5**, 1 (1966).
3. *Introduction to Virtual Storage in System/370, Student Text*, Form No. GR20-4260, IBM Corporation, Data Processing Division, White Plains, New York 10604.
4. *IBM System/370, OS/VS2 Planning and Use Guide*, Form No. GC28-0600, IBM Corporation, Data Processing Division, White Plains, New York 10604.
5. *IBM System/370, Introduction to OS/VS2 Release 2*, Form No. GC28-0661, IBM Corporation, Data Processing Division, White Plains, New York 10604
6. *OS/VS2 Planning Guide for Release 2*, Form No. GC28-0667, IBM Corporation, Data Processing Division, White Plains, New York 10604.
7. A. L. Scherr and D. C. Larkin, "Time-sharing for OS," *AFIPS Conference Proceedings, Fall Joint Computer conference* **37**, 113–117, AFIPS Press, Montvale, New Jersey (1970).
8. *IBM System/360 Operating System Time Sharing Option Guide, OS Release 21.7, IBM Systems Reference Library*, Form No. GC28-6698, IBM Corporation, Data Processing Division, White Plains, New York 10604.
9. *OS/VS Virtual Storage Access Method* (VASM) Planning Guide, Form No. GC26-3799, IBM Corporation, Data Processing Division, White Plains, New York 10604.
10. *IBM System/360 and System/370, ASP Version 3, Asymmetric Multiprocessing System, General Information Manual*, Form No. GH20-1173, IBM Corporation, Data Processing Division, White Plains, New York 10604.