The Information Management System, IMS, began in the mid-1960s as a batch-only data base system that was known then as Data Language/I (DL/I). IMS was introduced in 1969 as IMS/360, a program product for the System/360. As the System/360 evolved into System/370, including support for virtual storage, the operating system evolved into OS/VS1, OS/VS2, and then MVS. At the same time, IMS evolved to become IMS/VS. The Information Management System has continued to be adapted to new requirements, especially those of interactive, on-line operations that require data communications. Recent advances in the following categories of IMS/VS functions are discussed in this paper: Fast Path, Data Sharing, System Logging, Data Base Recovery Control, on-line changes in system environment, Intersystem Communications, MVS Common Services Area usage, and architectural restructuring.

IMS/VS: An evolving system

by J. P. Strickland, P. P. Uhrowczik, and V. L. Watts

The Information Management System, IMS, has undergone continuous evolutionary development since its initial release in 1969. In 1977, McGee[1] published a description of IMS/VS that included the function provided up to that date. That series of papers provides a good background for the reader to more fully appreciate the present paper, which is intended to describe some of the enhancements made to IMS/VS, with emphasis on its last two releases. To understand the evolution of IMS/VS it is first necessary to understand how system usage has been developing over the last few years. A glossary of terms is given in the Appendix.

One of the most important developments in the usage of IMS/VS has been the rapid increase in data-communication-based (on-line) applications. The number of terminals per IMS/VS complex has been increasing continuously. In the last five years the median has increased from 150 terminals to 600 terminals per IMS complex. While in 1977 approximately one percent of installations reported more than 1000 terminals, now 47 percent of customers are in this

category. The largest numbers of which the authors are aware are approximately 10 000 terminals on a single IMS/VS system and 18 000 in a network of IMS/VS systems. Transaction rates have undergone similar increases. Besides increasing end-user dependency on on-line communication services, these trends are changing the manner in which a Master Terminal Operator (MTO) must react to normal and abnormal situations and are requiring further decreases in outage times.

A second major trend is the increasing length of time that on-line IMS/VS is required to be operational without being shut down. (On-line IMS/VS has been designated IMS/VS—Data Communication or IMS/VS-DC.) Approximately twenty percent of on-line IMS/VS users consider themselves to be in a continuous-operation mode. This leaves them without adequate time to perform planned maintenance or system changes, such as adding a new transaction.

As a result of the above requirements, there has been a growing demand for IMS/VS to

- Support increased capacity (transaction rates, number of terminals, data base sizes, etc.)
- Simplify operation (Master Terminal Operator decisions, system restart, data base recovery, etc.)
- Support uninterrupted operation (minimize the need for planned shutdowns)

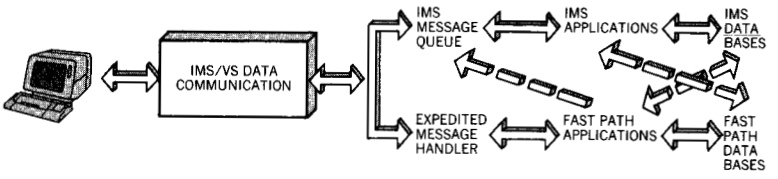
This paper describes some of the major functions that have been developed and announced in the last few years to satisfy these needs. The functions include the following:

- Fast Path enhancements
- Data Sharing
- System Logging enhancements
- Data Base Recovery Control
- On-line changes of system environment
- Intersystems Communications
- MVS Common Services Area (CSA) relief
- Architectural restructuring

Fast Path enhancements

When the Fast Path feature of IMS/VS was introduced in 1977, it was primarily intended to solve specific problems related to environments with large numbers of terminals entering simple transactions that access simple data bases processed later by batch programs. For example, banking is one of the industries with such characteristics. These requirements resulted in the design of a series of new facilities: (1) a Data Entry Data Base (DEDB) with a root segment and one

Figure 1 Single-mode and mixed-mode Fast Path processing



dependent segment to be used for journaling; (2) a single-segment Main Storage Data Base (MSDB); and (3) a new message-handling facility that does not use the normal IMS/VS message queuing.

In a DEDB, the dependent segment (called the sequential-dependent segment) was designed to allow user journaling of transaction data. These segments, while related to specific root segments, are also retrievable in a sequential manner in their order of insertion. They are inserted in a Last-In-First-Out (LIFO) manner from the corresponding root and stored in the physical order in which they are created. This optimizes the insertion of these segments as well as their retrieval for later batch processing in insert order. Since data bases in this type of environment tend to be large and the availability requirements very high, the concept of data base partitioning (Areas) was introduced to allow DEDB data to be stored by key range in a number of different data sets (Areas). This capability allows the data base size to be up to 960 gigabytes when stored in the maximum number of Areas (240). Different Areas may be stored on different device types to favor some key ranges. In addition, program scheduling has been disassociated from data base availability. Although one or more Areas may be unavailable due to reorganization or I/O error, application programs using this data base can still be scheduled. If the program does access an unavailable Area, an error status code is returned to the application program.

A new message-handling facility (called Expedited Message Handling) provides an alternate technique to the standard IMS/VS scheduling and queuing process. However, a Fast Path application program previously accessed only the Fast Path type of data bases. As a result, by 1977 there were two distinct types of transactions in IMS/VS, Full Function and Fast Path transactions. An IMS/VS Full Function transaction could only access Full Function data bases and a Fast Path transaction could only process Fast Path data bases. This was called single-mode processing; it is indicated by solid arrows in Figure 1.

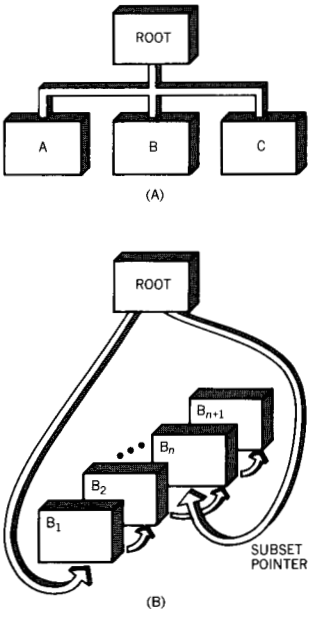
Following the introduction of these capabilities, it was determined that for Fast Path to be of more general use, (1) the separation of Full Function and Fast Path transactions was too restrictive; (2) the DEDB structure had to be extended. These needs have been addressed in Fast Path enhancements, first in 1978 and in IMS/VS 1.3.[2]

To make the Fast Path function more generally applicable, a Fast Path program can now access Full Function data bases, and Full Function programs can access Fast Path data bases. This is termed "mixed mode"; it is indicated by the combination of dashed and solid arrows in Figure 1.

The DEDB structure has been enhanced to include new segment types and hierarchical levels; it now allows up to 15 levels of hierarchy and 127 segment types. One segment may be designated as a sequential-dependent segment, with the remaining ones as direct-dependent segments. The direct-dependent segments have characteristics similar to that of the dependent segment in an IMS/VS Hierarchical Direct Access Method (HDAM) data base. Hence, a DEDB is now similar in function to a HDAM data base, with the exclusion of support for logical relationships and secondary indexing.

To help achieve a more uniform response time for transactions that are accessing DEDB data bases with long dependent segment chains, a new *subset pointer* has been introduced. With this facility a parent segment can point to more than one "first" segment in dependent segment type, and the application program interface (the DL/I Call interface) has been expanded to allow manipulation of each of the multiple pointers. The concept of Fast Path DEDB subset pointers is shown in Figure 2.

Figure 2 Fast Path Data Entry Base (DEDB) subset pointers showing (A) DEDB structure and (B) subset pointer in segment B

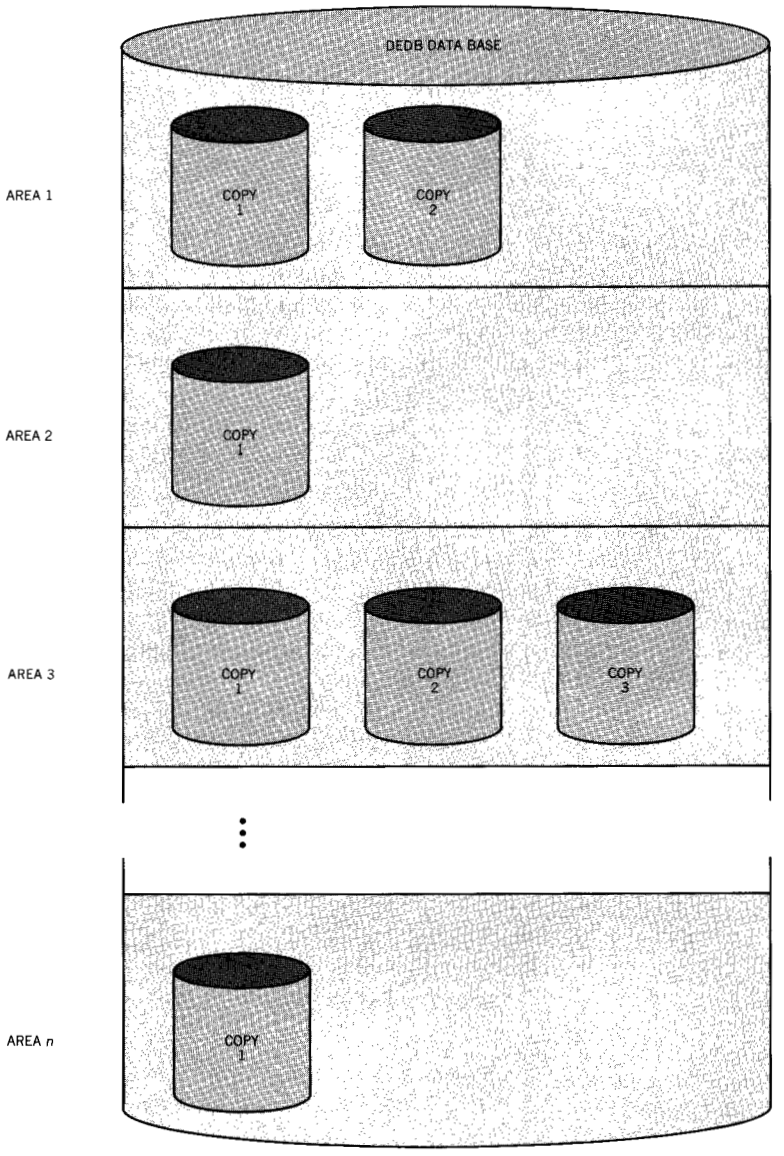


To satisfy increasing requirements for better system availability, we have relaxed the earlier requirement to recover a data base that is in error as soon as the error is detected. Two new capabilities have been added to the DEDB data base support: (1) *record deactivation* and (2) *data base replication.*

Record deactivation eliminates the need for immediate data base recovery in case of write errors. In the event of a write error, the entire Area is no longer deactivated; instead, only the VSAM Control Interval (CI) in error is made unavailable to the application program. Although the Area must be recovered eventually, the user may delay this action until a convenient time, since the information about which CIs have been deactivated is carried across system restarts. Application programs continue to be scheduled, and a status code is returned to the programs if the unavailable CI is accessed. A similar record deactivation capability has also been added to full function data bases. In this case, however, the information is not carried across system restarts, so that the user must still recover the data base before the next system restart.

Data base replication allows any Area of a DEDB to be written multiple times (i.e., multiple data sets for any Area). The system ensures that all the copies are maintained as exact duplicates. When reading, the system reads any of the available data sets. Data base replication is illustrated in Figure 3.

Figure 3 Fast Path data replication



The replication is started or reestablished by copying the Area(s) via an on-line utility without stopping the normal transaction processing against the DEDB. This technique can also be used to migrate one or more Areas from one device to another while the system is operational.

Frequently, the question arises as to how much is gained by using the Fast Path facility instead of using the Full Function IMS/VS facility.

If, for example, an application program is developed using Full Function (e.g., HDAM data bases and normal queued scheduling) and is contrasted with the same application developed using Fast Path (e.g., DEDB and Expedited Message Handling), the path length improvement for Fast Path is approximately two to four times. The application code path length is excluded from this comparison. These differences are achieved by providing a limited set of system capabilities to the application, thus reducing specific overhead activities.

Data sharing

A Data Sharing facility was introduced into IMS/VS in 1981 to satisfy three primary requirements: (1) Multiple IMS/VS-DC systems on multiple processors may share the same data in a manner that is transparent to application programs, resulting in an increase in total (on-line) IMS/VS-DC capacity. (2) Improved batch processing allows the concurrent processing of batch jobs using the same data bases as the on-line IMS/VS data bases, thus minimizing the effect of a "batch window," during which time the system is unavailable to on-line users. (3) The minimization of operational errors, thus preventing programs from erroneously accessing data bases that must be recovered, backed out, or backed up.

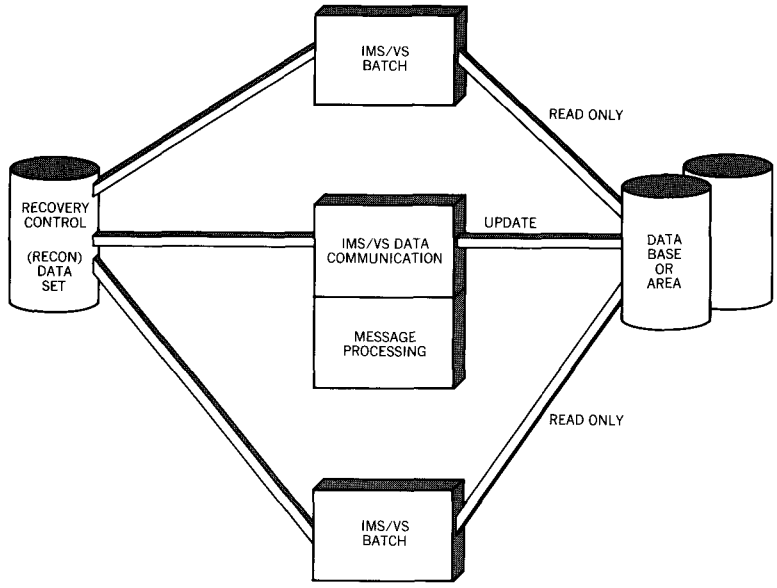
The IMS/VS Data Sharing facility controls the use of data bases across multiple on-line and/or batch IMS/VS systems. The data bases to be shared are registered and controlled with the Data Base Recovery Control (DBRC) facility of IMS/VS. Several levels of sharing are supported, varying from no sharing to multiple concurrent updaters.

Data Base/Area level and Block level are the two major types of data sharing provided. (Data Base/Area level sharing is shown in Figure 4.) In both cases, IMS/VS inhibits the use of a data base when that use could result in a processing integrity exposure or a data integrity exposure. A processing integrity exposure exists when one program reads uncommitted changes made by another program. A data integrity exposure exists when two or more programs are updating the same data without proper control over concurrent usage.

DL/I data base level and Fast Path Area level data sharing are controlled by DBRC on the basis of information recorded in its Recovery Control (RECON) data set. This level of sharing permits only one updater.

Block level data sharing, shown in Figure 5, is controlled by both the DBRC and the IMS/VS Resource Lock Manager (IRLM). This level of sharing permits multiple concurrent updaters. The IRLM provides global locking in an environment that consists of two MVS systems and any number of IMS/VS subsystems (on-line or batch).

Figure 4 Data Base and Area level sharing



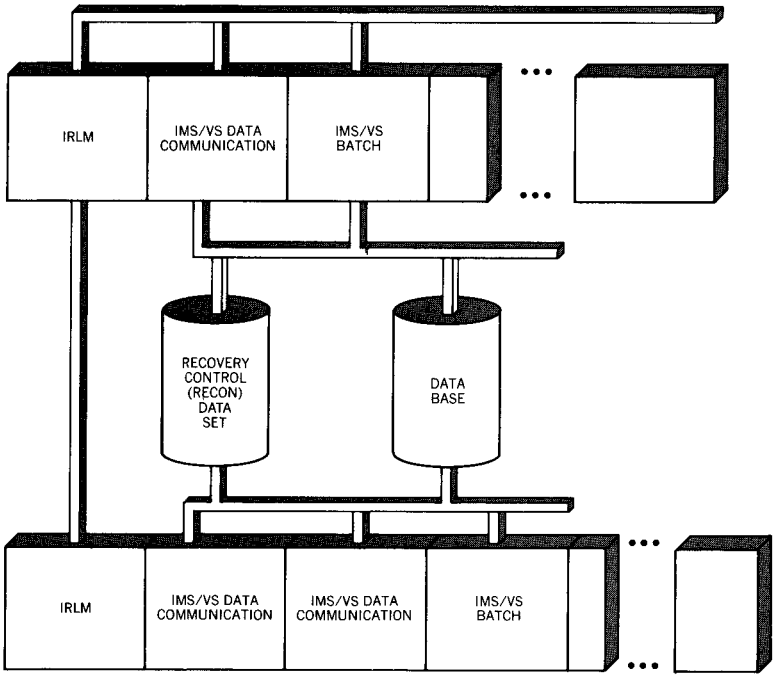
When IMS/VS data sharing was first introduced, two lock managers were used. One lock manager was program isolation, which has existed in IMS/VS for a long time. It controlled locking across multiple application programs associated with a single on-line IMS/VS control region. The other lock manager was the IRLM, which controlled locking across multiple IMS/VS subsystem images.

In IMS/VS 1.3, the IRLM has been expanded to perform program isolation locking as well as data sharing locking. This simplifies the locking protocols. A single IMS/VS now performs all its locking with either the Program Isolation Function or the IRLM. This eliminates duplicate locking and the possibility of false deadlocks.

The IRLM uses ACF/VTAM to perform intersystem locking notification. IMS/VS sharing minimizes intersystem communication with a two-level lock hierarchy, a resource name-hashing scheme, and a request-batching technique.

The two-level lock hierarchy involves obtaining a data set lock when a data set is opened. Later, as data within the data set are accessed, locks associated with the data items are obtained. The lower-level data item locks are related in a hierarchical manner to the data set locks. When a data set lock is held only by IMS/VSs executing within a single MVS, intersystem notification is not required before a lock is granted on a data item within the data set.

Figure 5 Block level data sharing



The resource hashing scheme involves associating a hash value with each resource name. The hash value is used to address a hash table, and an entry in the hash table shows the current intersystem *interest* in the corresponding group of resources. A system has interest when it holds locks or has waiting lock requests for one or more resources in the hash group. In cases where the hash table entry for a resource to be locked shows previously declared interest by this system and no interest by the other system, the lock can be granted without intersystem notification.

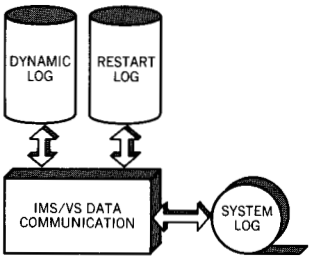
The IRLM uses a request-batching technique to avoid communications on individual lock requests. Requests that require intersystem notification are queued. The request queues are processed periodically (when a timer interval elapses), and required intersystem notifications are communicated at that time.

These techniques of minimizing intersystem communications permit a complete software-based implementation of the IMS/VS Data Sharing Facility; no specialized hardware is required.

System logging enhancement

With increasing transaction rates and need for easier and faster system restart, the original IMS/VS tape-oriented logging, illustrated

Figure 6 IMS/VS logging prior to IMS/VS 1.3



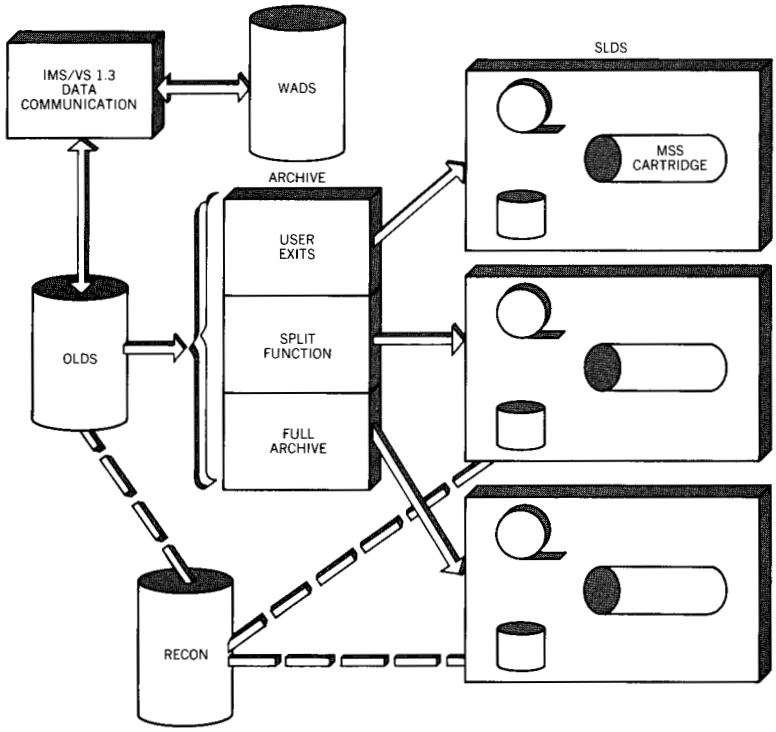
in Figure 6, showed some potential limitations. For example, due to writing short blocks in support of the log-write-ahead function of IMS/VS, the number of log tape volumes could become very large in a highly active system. Also, the amounts of time required for different system restarts were different—due to the need for tape mounting—and, after MVS or hardware failures, the need to close the IMS/VS log tape from a storage dump.

Although some of these problems could have been solved by improving the tape/disk logging, it was clear that only a fully disk-oriented logging method with space management, improved device integrity, and improved usability could resolve all of these conditions. The resulting design relies on DASD for all on-line logging.

In IMS/VS 1.3 a completely new logging technique is being introduced. With the new logging, shown in Figure 7, all log data are written to a series of DASD log data sets, called On-line Log Data Sets (OLDS), in a wraparound fashion. The original Dynamic Log and Restart Data Sets are no longer used for logging. Full blocking can be used for the OLDS through the use of a new buffering technique. The technique is known as the Write Ahead Data Set (WADS). In general, the WADS contains committed log records not yet written to the OLDS, thus eliminating the need for closing the log from a main storage dump before system restart in case of MVS, hardware, or power failures. Spare WADS data sets are supported which permit continuous operation after a write error. Once an OLDS buffer is full, it is written to the OLDS, at which time all the WADS space used to back up the unwritten buffer is available for reuse. All restarting can be done from the OLDS and the information in the WADS is automatically used to complete the OLDS. Dynamic backout (after an application failure) is also done from the OLDS, unless the needed data are contained in the OLDS buffers. When an OLDS data set becomes full, it is closed and the next available OLDS data set is used. If a write error occurs on an OLDS, the data set is closed and the next available OLDS is used. Dual logging can be used for both OLDS and WADS. When dual OLDSs are used, full or error conditions cause both data sets to be closed, and the operation continues on a new pair.

Since eventually all the OLDSs may become full, they must be archived on tape, disk, or Mass Storage System (MSS). This archived log, called the System Log Data Set (SLDS), is produced via an archive utility that can be started either via a command or automatically after a specified number of OLDSs become full. During archiving, the log data set can be further reblocked as well as split into different data sets. One such data set, the Recovery Log Data Set (RLDS), contains only the information necessary for the recovery of data bases. Any additional splitting of log records into separate output data sets may be selected by the user in the same archiving run.

Figure 7 IMS/VS 1.3 logging



Although log information is eventually archived (presumably to a medium slower than DASD), system restart should always be possible from the DASD OLDS. Even if the message queue needs rebuilding, one can ensure that the latest backup of the message queues is on the OLDS. This can be done via two facilities provided in IMS/VS 1.1.5: (1) Automated Operator Interface (AOI) and (2) on-line message queue dump. The AOI facility allows the interception of system messages (such as an OLDS switch) in a user exit. The exit can trigger the execution of a user transaction that issues an MTO command to dump the message queue without stopping the on-line IMS/VS system. Incidentally, since the AOI exit can also examine all input from the MTO terminal, the AOI can be used to develop commands tailored to the installation.

A key element in IMS/VS 1.3 logging is the following technique used for the log write-ahead of critical information. *Log write-ahead* is the term for the process of ensuring that recovery information has actually been written to a log data set (OLDS or WADS) before performing an operation that must be recoverable. Examples of IMS/VS operations that are recoverable are the following:

- Updating a data base
- IMS/VS 1.3 acknowledging that it has received an input message
- Acknowledging from an external destination that an output message has been received from IMS/VS 1.3
- Recognizing that an application program has reached a commit or synchronization point

When an application program declares it has reached a commit point and the commitment has occurred, all data base changes and message queue changes made by the application since the previous commit point now become available for access to other IMS/VS 1.3 applications. Conversely, changes made between commit points are visible only to the application that is making the changes. In the event the application or IMS/VS 1.3 terminates before the application reaches its next commit point, the changes are discarded (or backed out) by the system control function.

The IMS/VS DL/I data base buffer handlers, with their look-aside and deferred-write capabilities, have used log-write-ahead protocols for some time. The overall protocols are the following:

- When a DL/I module modifies the contents of a data base buffer, it first creates a data base change log record and submits it to the log manager. The log manager then places the log record in a buffer and returns a token that identifies the position of the log record in the buffer. This token is stored in the data base buffer control block.
- Later, when it is necessary to write the modified data base buffer to the data base, the buffer handler makes a *check-write* call to the log manager specifying the log token previously saved in the buffer control block. The log manager checks to see whether the requested log data have been written to the log data set. Logging activity that has occurred since the data base buffer was modified may have filled the log output buffer, resulting in writing to the log data set. Otherwise, the check-write request causes the log manager to write the log buffer.

The IMS/VS Fast Path data base manager achieves the log-write-ahead function via a different logging and data base updating technique. It saves all changes in storage until the application reaches a commit point. At that time, data base change log records are created and placed in the log buffers. The processing then waits until the log buffers have been written to the log data sets. A check-write request is not used to force the log write. A timer routine is used to force the log write if other activity does not fill the log buffer in a short period of time. Once the log buffers have been written, the Fast Path processing resumes and makes the changes to the data bases.

In IMS/VS 1.3 the log-write-ahead protocols have been extended to cover IMS/VS data communications message sending and receiving operations.

To maintain integrity and acceptable response times for both DL/I and Fast Path processing, it is often necessary to write a log buffer before it is filled. Previously, this resulted in writing truncated or short blocks in the log data set. In IMS/VS 1.3, the Write Ahead Data Set (WADS) has been added to avoid having to write short blocks to the log. Once data are written to either the WADS or the log data set, the log-write-ahead requirement is met and IMS/VS 1.3 may proceed to update its recoverable resources.

Check-write requests and a log-timer routine cause partially filled log buffers to be written to the WADS. The logging process continues to add data to the buffer until it is full and then writes the buffer to the log data set (OLDS). The same buffer may be written to the WADS multiple times before the OLDS buffer is finally full. This sounds like a simple process. However, the IMS/VS high data and processing integrity requirements prohibit an implementation that overwrites committed log data. Such an implementation runs the risk of losing data in the event of failure during the write operation.

Also, in a very-high-transaction-rate environment it may be necessary to write to the WADS frequently. A writing technique is used that minimizes DASD rotational delay. Relatively small, fixed-length records (approximately 2048 bytes) are recorded in the WADS. Each record contains a one-byte hardware key field. The key value is zero in all records. Records are written by a search-key-equal write data channel program sequence. All records on a track meet the search key criteria. Hence, a write operation begins after an average rotational delay of one-half a record rather than the conventional delay of one-half rotation.

The log buffer to the WADS write algorithm breaks a log buffer into pieces. The size of each piece is 2048 bytes. In the case in which a single 2048-byte piece of the buffer must be written multiple times, two tracks in the WADS are used with the write alternating between the two tracks. Once a complete (full) 2048-byte piece of the buffer has been written to the WADS, that piece is not written again. The WADS track containing the full 2048-byte piece cannot be reused until the entire log buffer has been filled and written to the log data set. In cases in which more than 2048 bytes of the buffer are to be written to the WADS, a single channel program containing multiple-search-key write data sequences is used to write the multiple pieces during the same rotation of the DASD.

Data base recovery control

A persistent requirement in IMS/VS has been the need for ensuring the integrity of data bases. In the past, the user was responsible for some aspects of this integrity. For example, when recovering a data base, the user was expected to ensure that all pertinent log information was input to the IMS/VS data base recovery process.

The logging of data base changes produces a series of log volumes in which several volumes may contain changes to a specific data base, say, data base A. Batch jobs produce a series of different log volumes, some of which contain changes to data base A. In the past, the user—not knowing which volumes contained the changes to A—would usually submit all the log volumes. Although many users mechanized this process, in many installations it remained a manual process. This process was, at times, error prone: for example, by omitting the logs from a batch run, or more frequently by selecting more log volumes than was necessary. The solution was to record all log volumes, along with an indication of which volumes contained changes to specific data bases. This capability was provided with the introduction of the Data Base Recovery Control (DBRC) facility.

The first release of DBRC provided for inventory control of all log volumes and the automatic selection of the proper log volumes (minimum set) for Log Change Accumulation or Data Base Recovery utilities. The information to support this function was kept in a Recovery Control (RECON) data set. The user designated the data base(s) that were to be included in this process by registering the data base name in RECON.

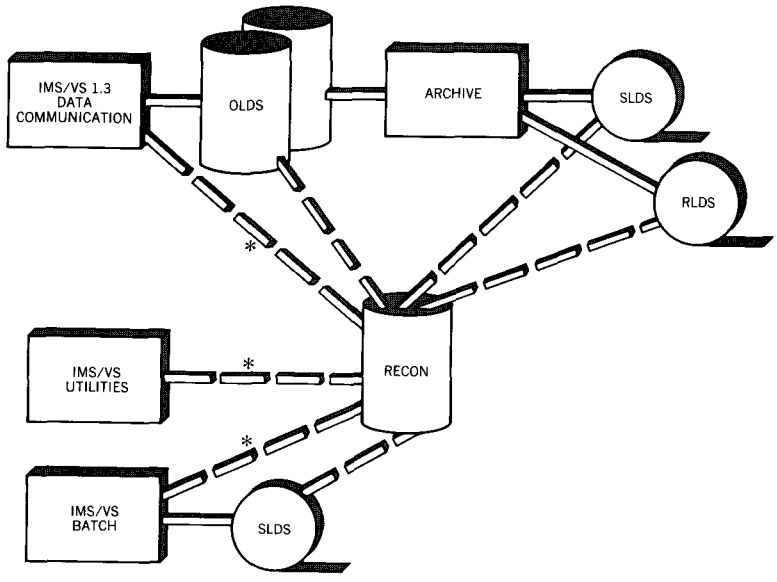
With the introduction of IMS/VS data sharing, IMS/VS also required a central recording concerning the status of each shared data base (e.g., for recovery, backout, etc.), as well as the allowed level of data sharing. Since the RECON data set already recorded information about data bases, it was a natural extension to the RECON structure to support the data sharing needs as well. This included not only additional information about data bases but also information about each executing on-line or batch IMS/VS subsystem.

Whenever an IMS/VS subsystem accesses a data base that can be shared, it must request authorization from DBRC to do so. DBRC determines, from the information kept in the RECON, the allowed level of sharing for the data base(s) and the executing subsystem status.

IMS/VS 1.3 logging incorporates the recording of additional data about IMS/VS logs: the status of OLDSs (in use, full, being archived, etc.) and the creation of the SLDSs. The place for storing data is the RECON. RECON, as used in IMS/VS 1.3, is shown in Figure 8. Note that the previously mentioned Recovery Log Data Set (RLDS) is also recorded in the RECON. When DBRC is used to control the recovery process of data bases, it selects RLDSs first as input to recovery, since they contain less log data than the SLDSs. If no RLDSs are available, the SLDSs are selected as a second choice.

The increased importance of the RECON data set has motivated the enhancement of its backup and recovery capabilities. In its original design, dual RECONs were utilized, so that the loss of one RECON

Figure 8 IMS/VS 1.3 usage of Recovery Control (RECON) data set



*AUTHORIZATION FUNCTION

would not allow the starting of another subsystem until duality was restored off line. A third RECON may now be allocated. If one of the active RECONs fails, duality is restored automatically by copying the remaining active data set into the spare without stopping IMS/VS 1.3.

In summary, IMS/VS 1.3 DBRC offers the following levels of control:

1. Inventory and control of on-line logs (archiving and reuse)
2. Inventory of batch logs
3. Data base recovery control
4. Subsystem authorization control

These capabilities are listed in the order in which one might choose to use them over time, and each increasing level of DBRC includes the previous controls.

Finally, the RECON is also used by Fast Path. In addition to the data sharing support of DEDBs, RECON is also used to keep information about the multiple data sets of a DEDB Area (data replication).

On-line change of system environment

The increasing trend toward continuous operation has highlighted the need to enhance two aspects of the early IMS/VS design. The addition of IMS/VS objects, such as transactions and programs,

required an IMS/VS system definition (SYSDEF) process. Also, a way was needed to decrease the amount of system outage required to introduce the new or changed definitions.

The IMS/VS 1.3-DC solution consists of (1) improving the granularity and speed of the SYSDEF process and (2) allowing the introduction of the changes into a running on-line IMS/VS system. The SYSDEF performance is improved by

- Reducing the number of SORTs by moving them from STAGE-2 to STAGE-1
- Improving the SORT algorithm
- Providing a preprocessor to STAGE-1 to check name uniqueness of the objects to be defined. In the past, this was done as part of STAGE-1. Every name was checked against every other name, resulting in an exponential processing time as a function of the number of objects being defined.

The preprocessor approach is to sort the names before determining uniqueness. This provides processing times linearly proportional to the number of objects. The preprocessor can be run by itself to get a source definition free of basic syntax errors and duplicate names before using the IMS/VS 1.3 STAGE-1 and STAGE-2. If the preprocessor is used, one may choose to bypass the name checking in STAGE-1. If the preprocessor is not used, name checking in STAGE-1 also follows a linear relationship to the number of objects. Because of this linear characteristic, time improvements are greater for a larger number of objects than for a smaller number.

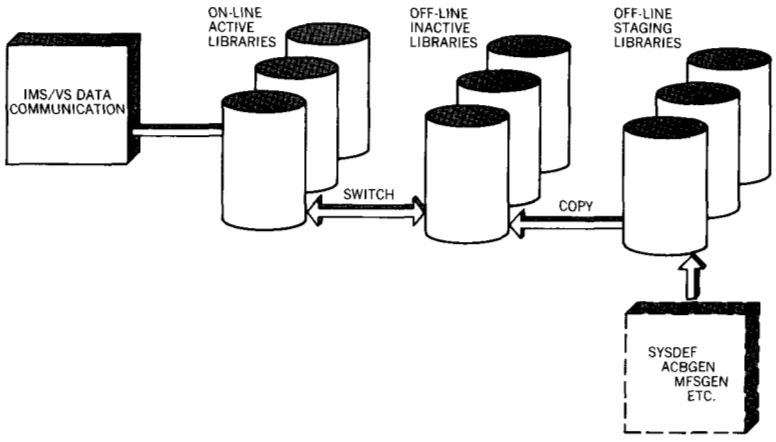
Early testing of the SYSDEF improvements indicates that running STAGE-1 (with name checking) plus STAGE-2 is roughly comparable in processing time to running the preprocessor plus STAGE-1 (without name checking) plus STAGE-2. Because of this, the value of the preprocessor is mainly in those cases where multiple STAGE-1 executions are performed to edit the SYSDEF source.

Although the improved performance of the IMS/VS system definition (SYSDEF) process tended to make frequent IMS/VS generations more acceptable, there still remained the problem of entering these changes into a running on-line IMS/VS system.

The entering of the newly defined objects into an on-line IMS/VS system is achieved by switching one or more libraries allocated to the on-line IMS/VS-DC (the *active libraries*) with the corresponding libraries containing the new definitions (the *inactive libraries*), as shown in Figure 9. The system objects that can be changed in this manner are the following:

- Data bases
- Application programs

Figure 9 Inserting an IMS/VS system definition (SYSDEF) change while the system is running

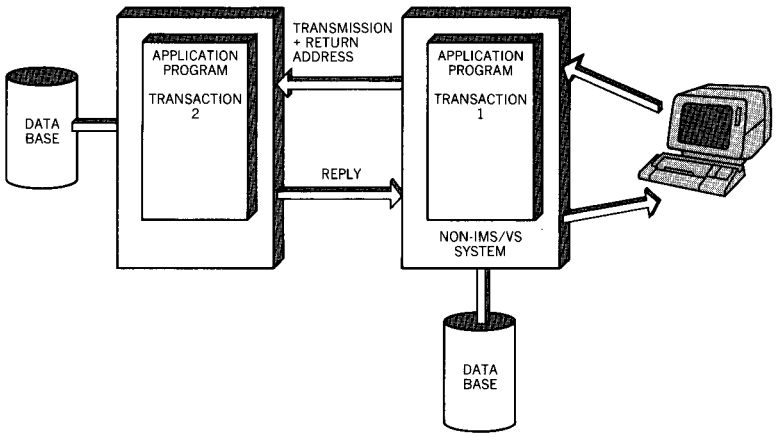


- Transactions
- Message Format Service (MFS) formats
- Security
- Fast Path routing codes

When a SYSDEF is performed, as illustrated in Figure 9, the results go to a set of staging libraries where the changes are held until all desired changes have been accumulated and verified. A utility program then determines which corresponding library is inactive (off line) and copies the contents of the staging libraries. When the switch from the active to the inactive libraries is desired, the MTO enters commands that indicate to IMS/VS the intent to switch a particular library or a set of libraries. The system determines which objects are being changed and which transactions are affected by the change. For example, a change to a data base definition affects all transactions that reference that data base. The system then enters a new quiesced state that allows the affected transactions on the queue to be processed but no new input for these transactions to be accepted. The activity for unaffected transactions continues normally. The determination of which transactions are affected is based on internal knowledge (control blocks) of the relationships among data bases, application programs, and transactions. Therefore, changes to any of these three objects result in a quiesced state for the affected transaction(s). Since no such internal relationships exist for MFS blocks and program switches, IMS/VS cannot determine the affected transactions, and the quiescing of the transactions in these cases is a user responsibility.

Once a switch is performed, internal control blocks in main storage are refreshed or updated, and the new definitions remain in use in

Figure 10 Intersystem communication



future IMS/VS restarts. The active libraries become inactive and can be used as backup. The fallback, if desired, is achieved by the MTO commands to switch back to a particular library or set of libraries, as just described.

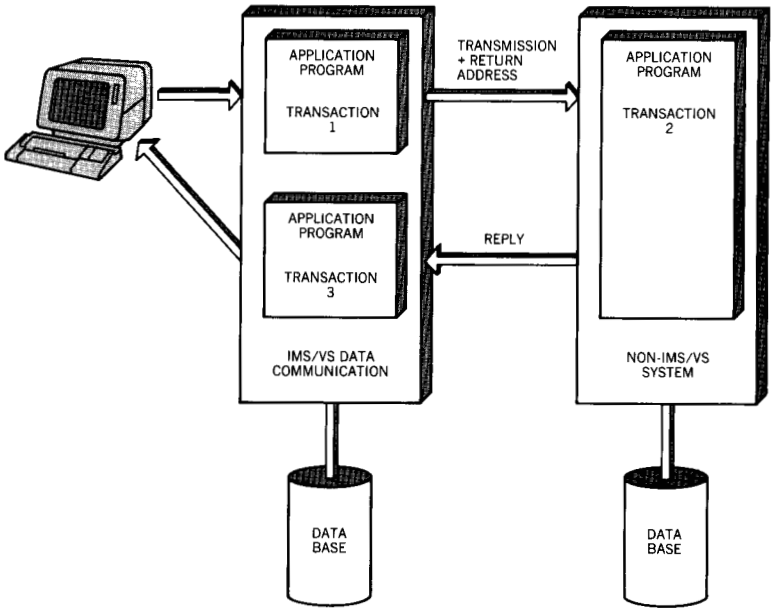
Intersystems communications

The introduction of the IMS/VS Multiple Systems Coupling (MSC) facility allowed for connecting a series of on-line IMS/VS systems using private IMS/VS-controlled protocols. There remained, however, the need for terminals on a non-IMS/VS system to use the applications and data being used by the on-line IMS/VS system. An example of such a system is CICS/VS. This capability is provided via the Intersystem Communication (ISC) facility of IMS/VS.

Prior to ISC, IMS/VS SNA support of connection protocols fell into two categories, host-to-device (e.g., LU1, LU2) and host-to-intelligent controller (e.g., LUP) Logical Units. These protocols allowed terminals and controllers to interact with IMS/VS without being aware of the IMS/VS transaction protocols. However, none of these protocols addressed the requirements of communication between transaction-based systems. This requires communications among the applications themselves. To address these requirements, a new communication protocol (LU6) was defined. This protocol allows applications in such transaction-oriented systems as IMS/VS and CICS/VS to establish communication and exchange information.

In its simplest form, the LU6 protocol allows terminals in one system to access transactions and data in the other and receive replies. In more advanced applications, it provides for the partitioning of a

Figure 11 Intersystem communication



single application into multiple transactions executing on two or more transaction-processing systems. With this capability, the IMS/VS application sees the connection as a *Logical Terminal* (LTERM). Input messages and output replies are processed through the Input/Output Program Control Block (IOPCB) just like normal terminal input. The IMS/VS system supports the additional transaction addressing necessary to route the reply back to an appropriate transaction and/or terminal in the other Intersystem Communication (ISC) system, as shown in Figure 10.

Program-initiated messages are processed through an alternate program control block. The IMS/VS transaction, through facilities provided through the Message Format Service of IMS/VS-DC, can specify a new transaction which will receive the reply from the other ISC system, as shown in Figure 11.

MVS Common Services Area (CSA) relief

An on-line IMS/VS system consists of multiple related MVS address spaces. Application programs reside in some of the address spaces and IMS/VS control functions reside in the other address spaces. The initial IMS/VS design for MVS reduced the cost of application address space access to the DL/I data bases by placing most of the DL/I control blocks, buffers, and code in the MVS Common Services Area (CSA).

In most cases, the access would be performed without address space switching. The access would be performed under the application program task structure, thus permitting parallel access from the multiple concurrently executing applications.

System demands on CSA have increased with time, and IMS/VS requires rather large contiguous pieces of CSA. This has created an environment where it has become quite challenging to operate an MVS in such a way as to permit the startup of a large on-line IMS/VS system without disrupting other work.

The last few releases prior to IMS/VS 1.3 have included new options that reduce the amount of CSA required. First, a Local Storage Option (LSO) permits many of the DL/I control blocks and much of the DL/I code to be placed in the IMS/VS control address space rather than in the MVS CSA storage. An MVS task exists in the control address space for each application address space. DL/I processing on behalf of a specific application address space is performed under its assigned control address space task. This permits IMS/VS to continue its parallel DL/I operations when operating under the local storage option. However, this technique does incur the overhead of cross address space task switching as control passes between the application and control address spaces for system services.

With the advent of MVS cross-memory functions, IMS/VS 1.2 offers the Cross Address Space Local Storage Option (XLSO), which utilizes these MVS functions. In this case, the DL/I blocks and code also reside in the IMS/VS control address space. The MVS cross-memory program call function is used to access the DL/I code and data resident in the IMS/VS control address space while executing under a task belonging to an IMS/VS application address space. Again, this maintains the parallel DL/I capability while eliminating most of the overhead of the cross address space task switching incurred by the LSO option.

In IMS/VS 1.3 the IRLM has been expanded to support an option that places most of its lock control structure and code in local storage in a separate address space. With this option, the cross address space program call function is used for transferring control from an IMS/VS to the IRLM. Use of the IRLM with this option has the effect of removing the program isolation lock tables from the IMS/VS control address space and placing them in the IRLM address space. This is true because when IRLM is used, the Program Isolation Function is not used. In this case, the design goes beyond solving the CSA problem and reduces the growth of virtual storage demand. Splitting the IMS/VS control function across multiple address spaces, as is the case when the lock tables move from control address space into IRLM address space, is one technique that is being used to reduce the demands placed on the control address space as individual user configurations continue to grow.

Architectural restructuring

In the past, IMS/VS provided parallelism in its control region by the use of two techniques. The first was a dispatching capability to manage IMS/VS-created subtasks with less overhead than the generalized MVS or VS/1 operating system dispatchers. The second technique was multiple Task Control Blocks (TCBs) to allow some work to be performed under other TCBs.

IMS/VS 1.3 provides improvements to both techniques. In prior releases of IMS/VS, initialization of IMS/VS had to be executed under a different environment than that provided for IMS/VS execution. The execution environment is now provided very early during initialization, allowing most of IMS/VS initialization to run under the normal IMS/VS control. This removes the distinction between initialization and execution, allowing functions that were formerly available only during startup to be invoked at any time. With this capability, control blocks become more dynamic. For example, the number of serviceable dependent address spaces is not fixed at startup time, but can be increased and decreased during normal operation up to a new limit of 255 for a single on-line IMS/VS system.

The second area of improvement is in the handling of the TCB structure. In addition to providing more parallelism, the new approach simplifies the adding of new TCBs if and when the need arises. The TCB structure is defined by a table. The addition of a definitional entry into the table is most of the work required to add a new TCB.

Concluding remarks

The evolution of IMS/VS described in this paper has been achieved while preserving the application program interface for both data base and terminal I/O requests. An application program written in the late 1960s for the then IMS/360 continues to run on the most recent IMS/VS release. This continuity has been the overriding objective throughout this evolution.

The new facilities of IMS/VS are mainly the result of the rapid increase in the use of on-line applications by its users and the growing need for continuous operations. These trends are expected to continue in the years ahead, and the new IMS/VS facilities provide a solid base for continuing evolution in that direction.

ACKNOWLEDGMENTS

The authors wish to recognize the IMS/VS development and test teams for the many years of dedication and personal effort in the evolution of IMS. Special thanks go to Dieter Gawlick and Don Lundberg for their invaluable assistance in preparing this paper.

Appendix: Glossary of terms

ACF/VTAM	Advanced Communications Function/Virtual Telecommunications Access Method
AOI	Automated Operator Interface
CI	Control Interval
CICS/VS	Customer Information Control System/Virtual Storage
CSA	Common Services Area
DASD	Direct Access Storage Device
DBRC	Data Base Recovery Control
DEDB	Data Entry Data Base
DL/I	Data Language/I
HDAM	Hierarchical Direct Access Method
IMS	Information Management System
IMS/VS	Information Management System/Virtual Storage
IMS/VS-DC	Information Management System/Virtual Storage—Data Communication
IOPCB	Input/Output Program Control Block
IRLM	IMS/VS Resource Lock Manager
ISC	Intersystem Communication facility
LIFO	Last In First Out
LSO	Logical Storage Option
LTERM	Logical Terminal
LU	Logical Unit
MFS	Message Format Service
MSC	Multiple Systems Coupling
MSDB	Main Storage Data Base
MTO	Master Terminal Operator
MVS	Multiple Virtual Storage
OLDS	On-Line Log Data Set
RECON	Recovery Control
RLDS	Recovery Log Data Set
SLDS	System Log Data Set
SYSDEF	IMS/VS System Definition
TCB	Task Control Block
VS1	Operating System/Virtual Storage 1
VSAM	Virtual Storage Access Method
WADS	Write Ahead Data Set
XLSO	Cross Address Space Local Storage Option

CITED REFERENCES

1. W. C. McGee, "The information management system IMS/VS," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).
2. *IMS/VS General Information Manual, IMS/VS 1.3*, GH20-1260-11, IBM Corporation; available through IBM branch offices.

The authors are located at the IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, CA 95150.

Reprint Order No. G321-5178.