Transaction processing and a further discussion of communication facilities are presented. Also discussed are system operations, including startup and shutdown, restart, and system monitoring. Other parts in this series present IMS/VS objectives and architecture, data base facilities, batch processing, and data communication.

The information management system IMS/VS Part V: Transaction processing facilities

by W. C. McGee

The transaction processing facilities of IMS/VS permit the user to provide for the arbitrary processing of messages entered from terminals. This processing is accomplished by user-written application programs that are entered into the system at system definition, and are scheduled for execution as required to process incoming messages. Typical uses for the transaction processing facilities are:

- Data entry: Keyed input records are checked by the program for format and data errors, possibly referring to tables stored in data bases. Accepted records are written to a sequential file for later processing. Records in error are returned to the terminal operator for correction.
- Data base inquiry and update: An input query message contains one or more entity identifiers. The application program retrieves data about these entities from one or more data bases, and incorporates those data into a message that is sent back to the terminal. Additionally or alternatively, the program updates the data bases, using values in the input message.
- Data output: An application program is started by the system operator to prepare and transmit operating reports to printers at various plant locations.

The IMS/VS transaction processing facilities also provide for the communication of messages between application programs. Messages generated by one program may be placed in an IMS/VS queue, from which they may be subsequently retrieved and processed by a second program. For example, terminal-entered data that have been validated by an on-line application program may be queued for later processing by a batch application program.

This part describes the transaction processing facilities of IMS/VS. Transactions are described first. This is followed by a discussion of transaction processing programs, including their structure, the system services available to them, and their definition to IMS/VS. The scheduling of transaction processing programs is then described. The part concludes with a discussion of on-line executions and system monitoring facilities.

Transactions

A *transaction* is a message that is to be processed by a user-written application program. IMS/VS provides two general categories of transactions: *Data Communication (DC) transactions*, which are processed by facilities in the DC feature; and *Fast Path (FP) transactions*, which are processed under the FP feature. DC transactions may be generated by terminals and application programs, and may consist of one or more segments. FP transactions may originate only from terminals, and may consist of a single segment only. Transactions of both kinds may be processed concurrently in the same on-line execution.

Transaction types may be defined by the user. Attributes of a DC transaction type include the following:

- Transaction type code, a one to eight character code.
- Number of segments in the transaction, single or multiple.
- Editing routine, i.e., the name of a user-written edit routine to be invoked after the transaction type has been determined.

Transaction types are defined at system definition with the TRANSACT statement. For example,

```
TRANSACT CODE=AR01, MSGTYPE=(SNGLSEG . . .)
```

defines the Automobile Club of Michigan transaction type AR01 that is used to retrieve and display accounts receivable data for a specified automobile policy.

Transactions that originate from a terminal carry a transaction type code in the initial positions of the first segment. Transactions that originate from a program may or may not be so identified. If they are, receiving programs can be written to be independent of whether the transaction originates from a terminal or from a program.

For DC transactions, a separate *transaction queue* is maintained by the system for each transaction type. When a transaction of a given type is received, it is placed into the transaction queue for the transaction type. For purposes of scheduling application

programs to process transactions, transaction types may be grouped into *transaction classes*. Up to 255 such classes may be defined, and a given class may contain any number of transaction types. Classes and their membership are defined implicitly through the TRANSACT statement. Continuing the previous example, the statement

```
TRANSACT CODE=AR01, MSGTYPE=(SNGLSEG, ,1)
```

assigns transaction type AR01 to class 1.

For FP transactions, a separate transaction queue is maintained for each *fast path load balancing group* that is active in the execution. Each load balancing group is associated with one or more executions of a single application program, and its queue holds transactions that are destined for those executions. FP transactions do not use the concept of transaction classes.

Transaction processing programs

User-written application programs that process transactions are called *transaction processing programs*. The DC feature provides two types of transaction processing programs: *message processing programs* and *batch message processing programs*, each type being limited to the processing of DC transactions only. In the FP feature, the analogous program types are the *message-driven program* and the *non-message-driven program*, respectively. The former processes FP transactions only. The latter does not process transactions at all, but is treated here because of its close analogy to the batch message processing program of the DC feature. Each of the four program types runs in a specific dependent region type that is named after the corresponding program type. For example, a message processing program runs in a message processing region.

Transaction processing programs may be written in the same languages as batch processing programs, and they have the same structure and operating characteristics, as discussed in Part III. One difference is that the Program Communication Blocks (PCBs) that are associated with a transaction processing program—instead of residing in the same region as the application program—are loaded into the control region, when the program is first scheduled. The control program maintains a pool of main storage for PCBs, and retains as many PCBs as possible in main storage during system operation, to minimize access to secondary storage.

Data services for transaction processing programs

The data services that are available to DC transaction processing programs include those services that are available to batch processing programs. In addition, the DC feature provides a *program isolation* facility that permits multiple transaction processing programs to access the same data bases concurrently without interfering with one another and without compromising the integrity of the data. Data bases to be accessed by transaction processing programs are called *on-line data bases*, and must be declared at system definition as well as being defined through the DBDGEN utility program.

Program isolation provides three levels of access to data resources: read-only (level 1), single-update (level 2), and exclusive (level 3). A given resource (e.g., a segment of a physical data base) can be held at any moment by any number of read-only users; or by one single-update user and any number of read-only users; or by one exclusive user. When a program requests a resource, its intended access level is added to the largest of the access levels of the programs that currently hold the resource, and if the sum is less than 4, the program is granted access to the resource (i.e., it becomes one of the holders). If not, the program waits. When a program releases a resource, the waiting programs (if any) are scanned to determine whether any such program can be granted access.

Data base calls from application programs result in implicit requests for data resources. Examples of resources that may be requested are physical segments and data set records. Resources are acquired at the appropriate level and are released in such a way that (a) The integrity of the data is preserved; (b) A program's view of its data bases remains consistent; and (c) The time that a resource is held is as short as possible, consistent with (a) and (b). Consider the following examples:

- When a PCB current position pointer is positioned in a data base record, the root segment of that record is held at the single-update level. This insures that only one program is working in a data base record at a time.
- A segment that is returned to a program with a GHU, GHN, or GHNP call is held at the single-update level, so that no other program can acquire it through such a call. If the segment is replaced with a REPL call, the access level is raised to exclusive to prevent all levels of access from another program until the holding program reaches a synchpoint. (A *synchpoint* is a point from which an application program may be restarted. The concept is discussed more fully later in this part.)

The implicit protection that is provided by program isolation is adequate for most data sharing situations. For the occasional situation where protection is not adequate, IMS/VS permits application programs to explicitly acquire and release data resources. Segments may be acquired during program operation by including the 'Q' command code in retrieval calls, and may be released explicitly with a dequeue (DEQ) call. Additionally, a data base, or one or more segment types within a data base, may be acquired for exclusive use by a program when it is scheduled by defining the program to have the exclusive processing option with respect to the data base or segment types in question.

The data services available to FP transaction processing programs reflect the Fast Path objective of high transaction throughput rates for limited-function transactions, and accordingly are somewhat different from the data services available to batch and DC transaction processing programs. The Application Programmer (AP) data structure class for FP programs is the same as for other program types, but FP data bases are derived in all cases directly from Data Administrator (DA) physical data bases. Physical data bases have two implementations: the Data Entry Data Base (DEDB), whose records are limited to a root segment type and a single dependent segment type; and the Main Storage Data Base (MSDB), whose records consist of root segments only, and have imbedded fields or terminal names as keys.

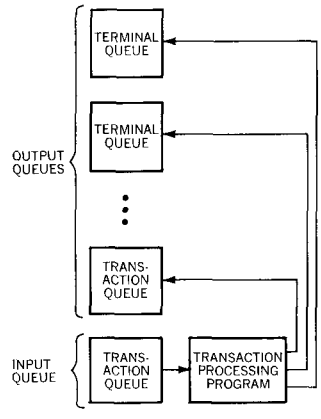
The manipulation of Fast Path AP data structures is accomplished through call statements of the type used with other AP data structures, but with certain limitations that are imposed in the interest of performance. In a DEDB, root segments can be accessed in the manner of root segments in other data base implementations, but operations on dependent segments are limited to insert and sequential retrieval, reflecting the DEDB goal of efficient data collection. In an MSDB, the operations permitted on the root segment depend on the MSDB type that has been declared by the user. In non-terminal-related MSDBs, which may have terminal names or imbedded fields as keys, operations are limited to retrieval and to a new Field (FLD) function. The latter permits one or more fields in a segment to be updated directly from application program work areas with a single call, in contrast to the usual procedure of first retrieving the segment to be modified with a get hold call, modifying it in a program work area, and then returning it to the data base with a replace call. In terminal-related MSDBs, which may have terminal names only for keys, root segments can be accessed in the manner of other implementations, but updating operations on a segment (replace, insert, delete) are permitted only when the program is processing a transaction from the logical terminal that "owns" the segment.

Much of the difference between FP and other IMS/VS data services stems from the special provisions made in Fast Path for reducing contention among concurrently running programs for shared data resources. By reducing contention, Fast Path reduces the delays incurred by a transaction in waiting for resources that are held by another transaction. In DEDBs and non-terminal-related MSDBs, contention is reduced by holding the data base changes requested by a program in main storage, and applying them to the data base only when the program reaches the next synchpoint. In terminal-related MSDBs, contention is reduced by permitting data to be updated only by transactions that are entered from the terminal that owns the data, and by making sure that no more than one transaction from a given terminal is in process at any time.

Communication services for transaction processing programs

IMS/VS provides facilities that enable a DC transaction processing program to communicate with transaction queues and terminal queues, hence to communicate with terminals and other programs. A transaction processing program can remove transactions from a single transaction queue that has been designated as the input queue, and can place messages in any number of terminal and transaction queues, or output queues, as shown in Figure 1. Messages that are placed in terminal queues are sent to the associated terminals, and messages that are placed in transaction queues are processed by the associated transaction processing programs.

Figure 1 Input and output queues for a transaction processing program



The communication services available to FP transaction processing programs are similar to those provided for DC programs, with the major difference that—in processing a given transaction—an FP program may place a message in a single output queue only, namely, the terminal queue for the terminal that entered the transaction. In the discussion that follows, DC communication services are described first, and the differences in FP communication services are then noted.

data communication

In DC transaction processing, the input queue for a message processing program is the transaction queue for the transaction type that caused the program to be scheduled. For a batch message program, the input queue is selected by the operator when he schedules the program.

Communication with queues is controlled through a second type of PCB, the TP PCB. A TP PCB in general holds the name of a queue (i.e., a logical terminal name or a transaction type code) and the results of operations that are carried out under its con-

trol. One PCB, the I/O PCB, is associated with the input queue and is generated automatically by the system. The remaining PCBs, *alternate PCBs*, are declared by the user as part of program definition. An alternate PCB may be permanently associated with a particular output queue by using a PCB statement of the following form:

PCB TYPE=TP, { LTERM=logical-terminal-name / NAME=transaction-type-code }

An alternate PCB may also be defined in such a way that its association with a queue is achieved during program execution as follows:

```
PCB TYPE=TP, MODIFY=YES
```

The latter facility is useful when a program must send output messages to a large number of destinations, and must avoid defining and dedicating storage to a separate PCB for each destination.

A transaction processing program invokes communication services through a call statement of the following form (in PL/I programs):

```
CALL PLITDLI (parmcount, function, pcbptr, workarea . . .)
```

where

- *function* designates a character string variable holding the name of the function to be performed.
- *pcbptr* designates a pointer variable that points to a TP PCB.
- *workarea* designates an area in the program where message segments are deposited and picked up by the system.

The remaining parameters depend on the particular function to be performed.

To obtain a transaction from its input queue, a program issues one or more call statements, one statement for each segment of the transaction. The first segment is obtained by specifying the Get Unique (GU) function. The system places the segment in the specified work area in the following format:

```
L L Z Z C C C . . . C T T T . . . T
```

(where L designates length, Z is reserved, C designates transaction type, and T is text) and returns the following information to the I/O PCB:

- Name of the logical terminal from which the transaction originated (or blank if the transaction came from a program).
- Sequence number for the transaction.
- Date and time the transaction was received.
- Status code, indicating the outcome of the call.

If the input queue is empty when the GU call is issued, the system returns a "queue empty" status code.

Succeeding segments of a transaction are obtained by specifying the Get Next (GN) function. Each segment has the following format:

L L Z Z T T T . . . T

where L designates length, Z is reserved, and T is text. When no more transactions remain, an "end of transaction" code is returned.

To place a message in an output queue, the program builds each segment of the message in a work area in the following format:

L L Z Z T T T . . . T

and issues a call that specifies the Insert (ISRT) function and a TP PCB that holds the name of the queue in which the message is to be placed. The call may also specify a Message Output Descriptor (MOD) if Message Format Service (MFS) editing is to be done on the message before it is transmitted to a terminal. The last segment of an output message is indicated to the system by a GU call to the I/O PCB, which implicitly delimits all previously issued ISRT call sequences; or by a Purge (PURG) call, which explicitly delimits the previously-issued ISRT call sequence against a specified PCB. When the system has received the last segment of a message, it places the message in a temporary destination queue. At the program's next synchpoint, the message is transferred to the final destination queue.

A program may send a response to the terminal that originated a transaction simply by specifying the I/O PCB in its ISRT calls, because the system places the name of the originating terminal in the I/O PCB, when the program retrieves the transaction. To set the destination of a PCB that is defined as modifiable, the program issues a Change (CHNG) call that specifies a PCB and the name of the destination to which the PCB is to be set.

In using IMS/VS communication services, transaction processing programs must be aware of certain attributes that may be assigned to transaction types at system definition. These attributes are the *response mode* and the *conversation mode*, and apply only to message processing programs.

A response mode may be specified in the definition of transaction types that originate from terminals. A response mode of RESPONSE specifies that the program must transmit a response message to the originating terminal before another transaction is accepted from that terminal. A response mode of NONRESPONSE specifies that such a response is not required. This facility may be used to keep the terminal operator in step with the application program.

A conversation mode may similarly be specified for transactions that originate from terminals. A transaction type is defined to be *conversational* if a scratch pad area definition is included in the definition of the transaction type. For example

```
TRANSACT CODE=ALPH . . . SPA=(156, CORE)
```

defines the Automobile Club of Michigan transaction type ALPH to be conversational, with a main storage scratchpad area of 156 bytes. If this definition is not included, the transaction type is *nonconversational.*

In processing a conversational transaction, a program may engage in a two-way exchange of messages with the originating terminal before the transaction is considered to be completely processed. A scratchpad area is provided in which the program may save intermediate results, while waiting for terminal responses. Thus the program need not occupy a message processing region during these intervals.

Conversational transaction processing is initiated by the terminal operator by entering a transaction in the normal way. When the system receives the transaction, instead of placing it in the appropriate transaction queue, it constructs a segmented message the first segment of which is a scratchpad that is associated with the originating terminal, and succeeding segments are those of the original transaction. This substitute message is then placed in the transaction queue.

When the associated transaction processing program is scheduled and issues a GU call to the I/O PCB, it receives in its work area the scratchpad segment that has the following format:

```
L L X X X X C C C . . . C X X X . . . X
```

where L designates length, X is reserved, C is the transaction code, and X is an initially blank work area. Ensuing GN calls against the I/O PCB retrieve the first segment (without a transaction type code) and succeeding segments of the original transaction.

To send a conversational response to the originating terminal, the program issues ISRT calls against the I/O PCB, as in nonconversational processing. One and only one response message must be sent to enable the terminal to continue the conversation.

To save the scratchpad between exchanges, an ISRT call against the I/O PCB is again used. The scratchpad segment is distinguished from output message segments by coding in the segment header. The system saves the scratchpad in main storage or direct access storage, as specified in the transaction definition. The scatchpad need not be saved on every exchange. If it is not saved, the original, or the most recently saved version, is returned on the next GU call.

When the terminal has received a program response, the operator can enter the terminal's response. This response does not represent a new transaction and therefore does not contain a transaction type code. On receiving the terminal's response, the system retrieves the scratchpad, and uses it to build and enqueue a multisegmented message, as with the original input. The program is rescheduled through the normal scheduling facilities, and at that time it may issue GU and GN calls to retrieve the scratchpad and terminal input segments.

Conversational processing can continue in this manner for any number of exchanges. The conversation is normally terminated by the program by blanking out the transaction type code in the scratchpad segment before saving it with an ISRT call. When the system receives such a scratchpad, it removes the terminal from conversational mode and discards the scratchpad. The next input from the terminal is treated as a new transaction.

Alternatively, a conversation may be terminated by the terminal operator's entering an EXIT command. The terminal then reverts to normal transaction mode. During a conversation, the terminal operator may temporarily leave and later re-enter the conversational mode by entering the HOLD and RELEASE CONVERSATION commands, respectively.

fast path

The communication services provided to FP transaction processing programs are similar to those just described for DC transaction processing programs. The principal differences are as follows:

- The input queue for an FP transaction processing program is the transaction queue for the associated load balancing group. The queue may hold transactions of more than one type.
- The (single) output queue is the terminal queue for the terminal that entered the transaction.

- All messages consist of a single segment only.
- All transaction types must have a response mode of RESPONSE and must be nonconversational.

Definition of transaction processing programs

A transaction processing program must be defined through the use of the PSBGEN utility program, in the same manner as a batch processing program. Program attributes that may be declared include program name, data bases to be accessed, type of access, and destinations to which messages are to be sent. The resulting Program Specification Block (PSB) is placed in the IMS/VS program specification block library.

A transaction processing program is also declared at system definition with the APPLCTN statement, as in the following example:

$$\text{APPLCTN PSB=psbname, PGMTYPE=}\left\{\begin{matrix}\text{TP}\\ \text{BATCH}\end{matrix}\right\}$$

One or more transaction types may be associated with the program by following the APPLCTN statement with one or more TRANSACT statements. The following statements:

```
APPLCTN PSB=MVP402, PGMTYPE=TP . . .

TRANSACT CODE=AR01 . . .

TRANSACT CODE=COMP . . .

TRANSACT CODE=CKDG . . .
```

associate transaction types AR01, COMP, and CKDG with the PSB and program named MVP402. A transaction type may be associated with one and only one application program.

For a message processing program, the defined association between the program and one or more transaction types forms the basis for scheduling the program. Specifically, the receipt of a transaction of a given type causes the corresponding application program to be scheduled.

As with batch processing programs, transaction processing programs must be stored in the IMS/VS application program library.

Transaction processing program scheduling

The facilities for scheduling transaction processing programs in the DC feature are designed for transactions having a wide range

of arrival rates and processing requirements, and are accordingly quite general. The corresponding facilities in the FP feature, on the other hand, are tailored to uniformly high transaction rates and relatively low processing requirements. In this section are described the two sets of facilities, starting with those in the DC feature.

data communication

The two types of DC transaction processing programs, message processing programs and batch message processing programs, are distinguished primarily by the manner in which they are scheduled. Batch message processing programs are scheduled manually by the system operator, whereas message processing programs are scheduled automatically by the system whenever a message processing region is available.

In scheduling a transaction processing program, the following three functions are performed:

- A program is selected.
- The schedulability of the program is determined. A program is schedulable if it has not been disabled and if all resources required, as reflected in its PSB, are available.
- The program and its associated control blocks are loaded, and control is given to the program.

In the case of batch message processing programs, the selection of a program is accomplished by the operator, when the batch message region job is started. Job control parameters that are specified by the operator include program name, the PSB name, and the transaction type code for the transaction queue that is to be the program's input queue. (The latter is optional, i.e., a batch message processing program need not access a queue.) The PSB that is associated with the program is retrieved in order to determine whether the data bases required are available. If so, the program is loaded and control is given to it. Otherwise, a message is sent to the master terminal and the job is terminated.

In the case of message processing programs, the selection of a program is accomplished with the aid of the association established between programs and transaction types in program definition, and the assignment of priorities to transaction types in system definition and message processing region startup. The job request that starts a message processing region specifies one or more classes of transaction types to be processed by that region. The order in which classes are specified determines the priority of the respective classes of transaction types. Within a given class, the priority of a transaction type is determined by priority attributes declared for the transaction type at system definition, and by the size of its associated transaction queue.

Program selection then consists of selecting the program with the highest-priority transaction type for which transactions are waiting.

Priority attributes to be declared for a transaction type include a normal priority, a limit priority, and a limit count. Initially, the transaction type has the normal priority. When the size of the associated transaction queue exceeds the limit count, the priority is changed to the limit priority and remains there until the queue is emptied, at which time it reverts to the normal priority. Limit priority is usually set higher than normal priority, so that large queues receive preferential treatment.

When a message processing program has been scheduled, it is not normally schedulable again until it has terminated. To expedite the processing of heavily used transaction types, provision is made for declaring a program to be *parallel schedulable*, and for declaring a *parallel processing limit* for any transaction type associated with the program. Such a program may be scheduled concurrently into two or more message processing regions. When an associated transaction type is considered for scheduling, it is scheduled only if the number of transactions of that type that are waiting exceeds the product of the number of regions currently processing the transaction type and the value of the parallel processing limit (PARLIM) parameter declared for the transaction type. In queuing system terms, PARLIM represents the largest value that the average number of waiting customers per server can attain before another server is added. The ability to automatically adjust the number of regions processing a given transaction type to the current activity of that transaction type is called *load balancing*.

A program completes by issuing a return call, and the region then becomes available for rescheduling. A program normally issues a return call when there are no more messages in its input queue. Under certain circumstances it may be undesirable for the program to relinquish the region. Although this can always be achieved by the program's withholding its return call, a less costly approach is to declare the transaction type to have the Wait For Input (WFI) attribute. When the associated queue is accessed and is empty, the system interlocks the program until a transaction arrives, but does not schedule another program into the region in the interim.

The scheduling of message processing programs may be controlled during system operation by master terminal commands that reassign transaction classes to regions, reassign transaction types to classes, and change the priorities that are assigned to transaction types. Commands are also available to enable and disable three types of resources that are central to program

scheduling: transactions, programs, and data bases. Disabled transactions are rejected, disabled programs are not scheduled, and disabled data bases render unschedulable any programs that reference them.

Once scheduled, a program runs to completion unless it is abnormally terminated. To minimize the impact of such terminations on system operation, provision is made for dividing the execution period into intervals bounded by synchpoints. A synchpoint is a point from which a program can be restarted in the event of an abnormal program end. A synchpoint occurs when a GU call is issued to the I/O PCB (on the assumption that the call starts the major loop in the program), and when the program issues a Checkpoint (CHKP) call. A batch message processing program may issue checkpoint calls if it does not access an input queue. Between synchpoints, the system saves the message taken from the input queue by the program, holds output messages in a temporary destination queue, and records all data base changes made by the program in a dynamic log. When a synchpoint is reached, the saved input message is discarded, the output messages are placed in their final destination queues, and dynamic log entries for the interval are erased, thereby committing the output messages and the data base changes.

A transaction processing program may be abnormally ended (abended) for any of the following three reasons:

- It attempts to execute an instruction or invoke a system service for which there is no useful outcome.
- It is terminated by IMS/VS to break a data resource deadlock.
- It terminates itself with the Rollback (ROLL) call.

For all abends, the system backs out—with the aid of the dynamic log—all data base changes made by the program since its most recent synchpoint. The system cancels output messages generated by the program since the last synchpoint by discarding the temporary destination queue, and places the message removed by the program from the input queue back in the input queue. For deadlock and rollback abends, the system automatically restarts the application program. For other abends, the system disables the abending program so that it cannot be rescheduled, and notifies the system operator, giving the program name, transaction code, input terminal name, and the first segment of the current input message. The operator can determine the cause of the abend, correct it, and re-enable the program with the /START command. Message processing programs restart automatically; batch message processing programs must be resubmitted.

fast path

In the Fast Path feature, both types of transaction processing programs—the message-driven and the non-message-driven—are scheduled by the system operator, in a manner similar to the scheduling of batch message processing programs in the DC feature. A non-message-driven program normally terminates itself with a return statement, whereas a message-driven program is designed to run continuously and be terminated by the system operator. Message-driven programs operate in the wait for input mode, i.e., they are interlocked when they attempt to retrieve from any empty input queue.

Multiple executions of a message-driven program may be scheduled concurrently into different regions. All executions are associated with a single load balancing group, and retrieve transactions in First In First Out (FIFO) sequence from the single transaction queue that is associated with the load balancing group. The load balancing group facility serves the same purpose as the parallel scheduling facility of the DC feature. In the load balancing group facility, the number of program executions is determined by the system operator, whereas in the parallel scheduling facility this number is determined by the system on the basis of transaction load.

A message-driven program creates a synchpoint when it issues a GU call to the I/O PCB, as does a non-message-driven program by issuing a new Synchpoint (SYNC) call. At a synchpoint, all data base changes requested by the program since the previous synchpoint are carried out, all logging for the interval is done, and the output message is placed in the appropriate terminal queue. Synchpoint processing is serialized, i.e., done for one program at a time, so that there is no possibility of two or more programs reaching deadlock over shared resources. The dynamic log is not used by Fast Path. If a program abends, data base backout is not required since data base changes have not yet been made. Instead, the operator can correct the cause of the abend, if necessary, and simply restart the program.

On-line execution operation and monitoring

An on-line execution of IMS/VS is initiated by a system console command that starts a control region and gives control to the IMS/VS control program. After initialization, the control program requests a restart command from the master terminal. When starting the system for the first time (*cold start*), the operator replies with the command /NRESTART CHECKPOINT 0. The control program responds by enabling all IMS/VS resources except lines (or logical units) and dependent regions. Communication lines are enabled by master terminal command, as described in Part IV. Dependent regions are started through operating system facilities as described in Part I.

Table 1 System shutdown options

	FREEZE	DUMPQ	PURGE
Message processing regions stopped	At program completion	At program completion	When transaction queues are empty
Batch message processing regions stopped	At checkpoint or program completion	At checkpoint or program completion	At program completion
Line input stopped	At message completion	At message completion	At message completion
Line output stopped	At message completion	At message completion	When terminal queues are empty
Transaction and message queues	Retained in queue data sets	Dumped to system log	Emptied normally

The normal shutdown of an on-line execution is accomplished with the following command:

/CHECKPOINT { FREEZE | DUMPQ | PURGE }

This command causes the various resources of the system to be disabled in an orderly manner, and a *system checkpoint* to be taken. A system checkpoint consists of flushing data base buffers and message queue buffers to direct access storage, and recording key system control blocks on the system log. The checkpoint is identified on the log and in a message to the operator by an identifier that consists of the date and the time of the checkpoint.

Three variations of shutdown are provided for control over the method of stopping programs and lines, and to control the method of disposing of queues. The effects of these variations are shown in Table 1. The DUMPQ option is useful in freeing the queue data sets, so that they can be reallocated on the next start-up. The PURGE option attempts to empty all the queues, so that no outstanding work remains. Messages that cannot be processed or transmitted are retained in the system log.

To restart (*warm start*) the system from the state attained at the previous FREEZE shutdown, the operator enters the command /NRESTART. The system searches the system log for the check-

point entry from the previous shutdown, restores the control blocks saved at that time, and waits for the operator to enable lines and regions, as in a cold start. If the previous shutdown had been a DUMPQ or PURGE, the operator must specify in the NRESTART command the identification of the shutdown checkpoint. After locating this checkpoint, the system rebuilds the message queues from entries in the log, and restores control blocks.

An on-line execution can be ended abnormally for a variety of reasons, including the following:

- The control region abends.
- The control region does not respond to terminals and must be terminated by operating system command.
- The operating system, power, or hardware fails.

IMS/VS provides facilities that permit the system to be restarted from abnormal ends without disruption to the terminal user, other than waiting for restart to take place.

The basic approach to emergency restart is to record the state of the system periodically in the form of system checkpoints and message queue dumps, and to log system activity that occurs between such recordings. When an outage occurs, the system is restarted by taking the most recently recorded state and using the log to update it to the point of the outage.

System checkpoints are taken at normal system shutdown, system restart, and at intervals throughout system operation. The checkpoint interval is determined by a user-specified number of log records to be written between checkpoints. Checkpoints may also be taken during operation by issuing the master terminal command /CHECKPOINT. Message queue dumps are taken at normal system shutdowns, using the DUMPQ or PURGE options.

Following an outage in which main storage or queue data sets are lost, the operator first determines that the system log has been properly terminated, using the Log Tape Termination utility if necessary. The operator then restarts the operating system and the IMS/VS control region. If only main storage has been lost, the operator enters the command /ERESTART. The system responds by selecting an appropriate checkpoint on the system log, and processing the log forward from that point to restore the message queues to their state at the time of the outage. For each DC transaction processing program that is active at the time of the outage, the system uses the dynamic log to reset system state to the program's most recent synchpoint, just as though the program had abnormally terminated. When restart processing is complete, the operator may start message processing regions

and communication facilities in the normal manner, and the scheduling facilities automatically restart any message processing programs that had been in progress at shutdown. Interrupted batch message processing programs and FP transaction processing programs must be restarted manually. If a batch message processing program uses the program checkpoint/restart facility, the system indicates to the operator the proper program checkpoint from which the program should be restarted. Otherwise the program must be started from the beginning.

If the message queues or scratchpad data sets are damaged in the outage, the operator specifies in the ERESTART command the identification of the checkpoint taken on a previous DUMPQ or PURGE shutdown. The system locates the specified checkpoint on the log, rebuilds the message queues and scratchpad data sets from the information that had been dumped to tape at the checkpoint, and then processes the log forward to restore the queues to their state at the time of the outage. Transaction processing programs are then backed up to their most recent synchpoints and restarted.

The routine operation of an on-line execution is facilitated through the use of the /DISPLAY command, which may be used to display at the master terminal two general kinds of information: system resource attributes and system resource utilization. Examples of system resource attributes that may be displayed are as follows:

- Assignment of logical terminals to physical terminals.
- Status of lines and terminals.
- Current priority of transaction types.

Examples of the utilization information are the following:

- Current size of queues.
- Number of conversational transactions in progress.
- Current utilization of main storage buffers.

The master terminal operator uses the display command to obtain information about the system that helps him select the best setting of parameters under his control. For example, by observing the lengths of transaction queues, the operator is able to adjust the number of message processing regions, the region-class assignments, or transaction type priorities to keep queue lengths short and uniform.

The system log contains a great deal of information that can be used to obtain better performance from the system, through the adjustment of execution-time or system-definition parameters. IMS/VS provides three utility programs for extracting information of this kind from the system log for on-line system executions.

The Log Transaction Analysis utility program produces a listing of transactions processed by the system in a specified time interval, thereby giving a detailed history of each transaction. The report is optionally recorded on secondary storage so that it may be sorted into various sequences and otherwise manipulated by user programs. The utility also optionally produces an extract of the system log for more convenient study of intervals of interest.

The Statistical Analysis utility program gives summaries of the activity in various system resources during a specified time interval, including line and terminal activity (e.g., messages sent and received and message sizes), transaction type activity (e.g., transactions processed and average response time), and program activity (e.g., transactions processed, calls issued, and CPU time expended). These reports may be used, for example, to schedule the use of terminals so as to achieve better line utilization and response times.

A Program Isolation Trace Report utility program lists instances of program waiting caused by the program isolation feature, along with waiting times. This information is useful in eliminating bottlenecks that have been created by heavily-used data resources.

Additional information on system operation can be obtained by including in the defined system the IMS/VS Monitor Facility, which may be enabled on any batch or on-line execution. In an on-line execution the facility may be started and stopped from the system console. The Monitor Facility produces an independent log of system activity that can be processed by an appropriate utility program. Logs created in on-line executions are processed by the DC Monitor Report Print program, which prepares reports that contain the following kinds of information:

- Buffer pool utilization, which is useful in determining the adequacy of assigned pool sizes.
- Region timing information, from which region utilization can be derived.
- Program activity, such as number of calls of various types, number and duration of waits for input and output, and CPU time.
- Transaction queuing activity, which is useful in setting transaction type priorities.
- Special events, such as failure to schedule application programs because of processing intent conflict or lack of buffer space.

Logs created by batch executions are processed by the DB Monitor Report Print program for reporting buffer pool utilization and program activity.

Summary and concluding remarks

Two general categories of transaction processing facilities are provided in IMS/VS: the DC transaction processing facilities that had their origin in early IMS systems, and are intended for a broad range of transaction processing applications; and the FP transaction processing facilities, a more recent addition, that are intended for applications with high transaction volumes and limited processing. DC transactions are queued by type, and are removed from the queues by user-written programs that are scheduled by the operator or by the IMS/VS control program, on the basis of transaction priorities and resource availability. FP transactions are queued by load balancing group, and are processed by operator-initiated executions of user-written programs. Both types of transaction processing may coexist in the same on-line execution.

The transaction processing facilities of IMS/VS contain a number of significant contributions to the technology of generalized data base management systems. These contributions occur in the areas of fixed-server transaction processing, the synchpoint concept, and data sharing.

Many early teleprocessing monitors (and many today) employ the variable-server approach to transaction processing. When a transaction arrives, the monitor creates a server, i.e., some internal entity such as a job or a task, to process the transaction. When the processing is completed, the server is destroyed. In the fixed-server approach, on the other hand, a fixed number of servers is established at system startup, and the monitor assigns transactions to them as they are received. In IMS/VS, the servers are the program controllers running in the dependent regions. In the DC feature, program controllers are capable of loading different programs to customize their response. With the Fast Path feature, application programs are continuously resident, and they in effect become the servers. The variable-server approach is best suited to fluctuating workloads of many different transaction types, since it avoids tying up system resources during low-demand periods. The fixed-server approach, on the other hand, is best suited to steady workloads of a few transaction types, since it avoids much of the allocation and deallocation activity on each transaction that occurs in the variable-server approach. IMS was among the first systems to recognize the latter requirement, and continues to be oriented toward this kind of workload.

The concept of dividing the execution of a program into intervals bounded by synchpoints is a significant technological advance. The synchpoint concept simultaneously fulfills three needs: (1) It provides points from which a transaction processing progran can be conveniently restarted in the event it ends ab-

normally (which is classical checkpoint/restart facility); (2) It provides points at which shared data resources acquired by a program can be released by the system, to prevent excessive queuing on these resources; and (3) It provides points at which actions of a program are committed to the program's environment. Synchpoints might also be used as points at which the system provides consistency of the data base, thus allowing application programs the convenience of creating temporary inconsistencies in the data base between synchpoints. Intervals between synchpoints are fundamental units of work in transaction processing systems, and are comparable in significance to jobs and job steps in batch processing environments.

Along with a number of other generalized data base management systems, IMS/VS solves the problem of sharing data among concurrently executing programs, in a manner that preserves the integrity of the data without creating excessively long queues on shared resources. In early releases of IMS, data sharing was controlled through processing intent scheduling, i.e., a transaction processing program would not be scheduled if it intended to update segment types that were being updated by currently running programs. Under this regime, the granule of sharing was effectively the segment type. With the introduction of program isolation in IMS/VS, the granule of sharing has been reduced to the segment instance. Programs that are updating the same segment types are allowed to run concurrently, with segments that are being acquired and released as required to prevent interference. Deadlocks over data resources are resolved by forcing one or more of the deadlocked programs to terminate abnormally, and using the synchpoint recovery facilities already in place to restart the terminated programs.

GENERAL REFERENCES

1. IBM Corporation, *IMS/VS Version 1 System/Application Design Guide*, Document SH20-9025, IBM Corporation, Data Processing Division, White Plains, New York 10604.

2. IBM Corporation, *IMS/VS Version 1 Fast Path Feature General Information Manual*, Document GH20-9069, IBM Corporation, Data Processing Division, White Plains, New York 10604.

3. IBM Corporation, *IMS/VS Version 1 Installation Guide*, Document SH20-9081, IBM Corporation, Data Processing Division, White Plains, New York 10604.

4. IBM Corporation, *IMS/VS Version 1 Operator's Reference Manual*, Document SH20-9028, IBM Corporation, Data Processing Division, White Plains, New York 10604.

5. IBM Corporation, *IMS/VS Version 1 Utilities Reference Manual*, Document SH20-9029, IBM Corporation, Data Processing Division, White Plains, New York 10604.

6. IBM Corporation, *IMS/VS Version 1 Application Programming Reference Manual*, Document SH20-9026, IBM Corporation, Data Processing Division, White Plains, New York 10604.

Reprint Order No. G321-5050.