The Case for and Value of Transactional VSAM

A CICS/Batch File Sharing Enhancement

Session TSS03
Ruth Ferziger
ruthf@us.ibm.com

What is VSAM RLS?

VSAM record level sharing (RLS) was introduced in DFSMS/MVS 1.3.0. It works with the CICS Transaction Server as an alternative to 'traditional' record management. The intent was to enhance cross system data sharing by providing a locking mechanism at the level of the individual record in a VSAM data set rather than at the control interval or control area level.

VSAM RLS requires no changes to existing VSAM data sets, except to define them with a LOG parameter and possibly a LOGSTREAMID parameter. Other than that, they are standard VSAM data sets in the same format as other KSDSs, RRDSs, VRRDSs, and ESDSs and use the same VSAM interfaces.

Note that to use VSAM RLS with a recoverable data set, that data set must be SMS-managed. This is because there is information in the storage class assigned to the data set which is used to determine how the data set is managed in the coupling facility. Since it is the association of a storage class with a data set that makes a data set SMS-managed, these data sets must be SMS-managed.

What is a Recoverable Data Set?

Recoverable data set:
- Accessing application must be recognized as a commit protocol application if recovery function is required (CICS or Transactional VSAM)
- Transactional back out and forward recovery capability (if needed)
- LOG(UNDO|ALL) attribute

Non-recoverable data set
- Applicable to any application which can tolerate multiple updaters (including CICS)
- No assumption on application recovery environment
- LOG(NONE) or undefined attribute

The concept is to maintain a log of changed records for a recoverable data set and use the log to provide atomic commit or back out of a unit of work's (also known as transaction or unit of recovery) changes to the data set. For VSAM RLS, CICS maintains logs of its changes to recoverable data sets, and VSAM RLS inhibits batch jobs from updating recoverable data sets in RLS mode. Batch readers may concurrently read a recoverable data set, but they cannot update it.

Batch updaters and CICS File Control may concurrently share non-recoverable data sets since these data sets do not require logging.

Transactional VSAM is the second stage of the DFSMS/MVS recoverable data sets strategy. It provides transactional recovery within VSAM, rather than deferring this capability to callers of VSAM, by providing both logging and two phase commit and back out protocols, in addition to the locking functions already provided by VSAM RLS.

RLS introduced to VSAM the concept of Recoverable File. This attribute is specified via AMS (Access Methods Services) DEFINE or ALTER. The parameter and options are:

- LOG(NONE) This declares the file non-recoverable.
- LOG(UNDO) This declares the file recoverable.
- LOG(ALL) This declares the file is recoverable and also requests forward recovery logging (redo) of changes.

The attribute only applies when the file is accessed in RLS or Transactional VSAM mode. When the file is accessed in NSR/LSR mode, VSAM ignores the attribute.

The recoverable attribute means that when the file is accessed in RLS or Transactional VSAM mode, transactional recovery is provided. With RLS, the recovery is only provided when the access is through CICS File Control, so RLS does NOT permit a batch (non-CICS) job to OPEN a recoverable file for OUTPUT.

RLS does provide read access to a recoverable file by batch jobs. The batch job may request RLS read locking to avoid seeing uncommitted changes made by sharing CICS applications. The batch job may declare via a JCL parameter that it wants to access the file through RLS.

The parameter and options are:

- RLS=NRI This declares RLS access without read locking.
- RLS=CR This declares RLS access with read locking.

RLS supports non-recoverable files. It provides record locking and file integrity across concurrently executing CICS and batch applications. Transactional recovery is NOT provided. This is because VSAM RLS does not provide:

- undo logging
- two-phase commit/back out support

Most batch jobs that modify VSAM files are NOT designed to share data and can NOT use this form of data sharing.

What is Transactional Recovery?

The transaction program execution model provides data sharing of recoverable resources. During the life of a transaction, its changes to recoverable resources are NOT seen by other transactions. The transaction may request that its changes be rolled back (backed out). If the transaction fails,

its changes are backed out. This capability is called transactional recovery. It is provided by the resource managers.

Applications that are designed to the transaction model are able to easily share the recoverable resources. The resource managers provide the sharing isolation and recovery when a transaction fails or when the execution environment fails.

IMSDB and DB2 are resource managers that provide transactional recovery for their databases. CICS File Control provides transactional recovery for VSAM recoverable files. Now, Transactional VSAM provides transactional recovery for VSAM recoverable files as well.

This is a simple example that illustrates transactional recovery. The application wants to make a change to two different data items. A field in one data item is decremented from 200 to 100. A field in the other data item is incremented from 700 to 800. Transactional recovery means that either both changes are made or neither change is made. When the application requests Commit, both changes are made atomically.

If the application makes these changes to non-recoverable data and the application or the application environment fails, one or both of the changes may be lost.

The use of recoverable data means all data and other recoverable resources that are within the same commit scope are always commit-consistent. A commit scope is the set of recoverable resource managers that participate in a commit.

CICS Function Shipping Before RLS

Before VSAM RLS, there were CICS application owning regions and CICS file owning regions. Each FOR owned specific files. If an AOR needed access to a specific file, it had to function ship the request to the correct FOR which would then process that request and return the data. If the FOR or the system on which it was running became unavailable, access to the data was lost until the system or FOR could be restored.

Parallel Sysplex CICS with VSAM RLS

CICS AORs (Application Owning Regions) directly access VSAM RLS. They do not have to function-ship VSAM requests to a CICS FOR (File Owning Region). This means that the failure of a single system does not make data sets unavailable until that system can be restored. Instead, CICS can still access that data via VSAM RLS server on its own system.

The coupling facility (CF) provides the base multisystem functions required for sysplex sharing.

Using VSAM RLS

VSAM RLS did NOT introduce any new data formats and uses, for the most part, the existing VSAM interfaces. The major difference is the addition of RLS as a possible value on the MACRF keyword of the ACB. Where before you might have specified NSR, LSR, or GSR, now you may also specify RLS. Alternatively, you may also use the RLS keyword on the JCL. Once you have opened a data set for RLS access, you may then use all the existing VSAM interfaces to access the data: POINT, GET, PUT, ERASE, ENDREQ, etc.

The scope of the sharing is the MVS sysplex, not the individual system. And the serialization, which had previously been done at the CI level, is now done at the record level. This means that two different applications, online systems, or z/OS images can be accessing different records within a single CI at the same time without interfering with each other. VSAM RLS manages the serialization and takes responsibility for keeping buffers up to date.

It manages this by maintaining a lock structure in the coupling facility. By doing so, all activity for all systems in the sysplex is always visible to all of the other systems in the sysplex, so it can never be the case that two applications or online systems or z/OS images are updating the same record at the same time.

VSAM RLS

- Buffers are stored (currently) in the SMSVSAM data space and are also cached in the coupling facility.
- Record locks are stored in a lock table, which also resides in the coupling facility.

RLS Read Integrity

NRI, or dirty read, does not provide any means of locking. Because of this, it may result in reading uncommitted changes made by other applications (which currently have the record locked using an exclusive lock).

CR provides a clean read and provides read integrity. It does this by obtaining a shared lock on the record before reading it, and releasing the locks as soon as the record is read. If the record is locked exclusively by another applications, the CR read must wait until the locks is freed. When the record is available, RLS serializes the record in shared mode and then immediately frees the lock. Using this option, the application will never see uncommitted changes.

CRE also provides a clean read and provides read integrity. The major difference is that after reading the record, the shared locks is not released, but remains held until the end of the transaction (commit or back out). During this time frame, no other transactions can update the record. This means that the application can read the record multiple times and it is guaranteed that it will always see the same thing. In a pure RLS world, this option is available only to CICS. This is because RLS has no concept of a transaction and therefore has no mechanism for releasing the lock. With Transactional VSAM, this option becomes available to batch as well.

RLS Sharing of Recoverable Files

RLS introduced to VSAM the concept of Recoverable File. This attribute is specified via AMS (Access Methods Services) DEFINE or ALTER. The parameter and options are:
LOG(NONE) This declares the file non-recoverable.
LOG(UNDO) This declares the file recoverable.
LOG(ALL) This declares the file is recoverable and also requests forward recovery logging (redo) of changes.

The attribute only applies when the file is accessed in RLS or Transactional VSAM mode. When the file is accessed in NSR/LSR mode, VSAM ignores the attribute.

The recoverable attribute means that when the file is accessed in RLS or Transactional VSAM mode, transactional recovery is provided. With RLS, the recovery is only provided when the access is through CICS File Control, so RLS does NOT permit a batch (non-CICS) job to OPEN a recoverable file for OUTPUT.

RLS does provide read access to a recoverable file by batch jobs. The batch job may request RLS read locking to avoid seeing uncommitted changes made by sharing CICS applications. The batch job may declare via a JCL parameter that it wants to access the file through RLS.

The parameter and options are:
RLS=NRI This declares RLS access without read locking.
RLS=CR This declares RLS access with read locking.

RLS Sharing of Non-Recoverable Files

RLS supports non-recoverable files. It provides record locking and file integrity across concurrently executing CICS and batch applications. Transactional recovery is NOT provided. This is because VSAM RLS does not provide:
- undo logging
- two-phase commit/back out support

Most batch jobs that modify VSAM files are NOT designed to share data and can NOT use this form of data sharing.

What is the Batch Window Problem?

The majority of customers in the CICS Joint User Group have batch windows ranging from two to ten hours. The programs run during the batch window consist of both in house applications and vendor written applications. These customers would like to be able to do batch haring to enable them to reduce or eliminate the batch window.

While VSAM record level sharing was a starting point for this goal, it does not fully enable the necessary sharing. The possibility of augmenting it with Transactional VSAM capability was reviewed with the Customer Design Council in May of 1995 to validate that it addressed a real requirement and that the approach was viable. The consensus from the council was that the project should be allowed to proceed, and several customer indicated a strong need for the capabilities provided by Transactional VSAM.

The batch window is a period of time in which online access to recoverable data sets must be disabled. During this time, no transaction processing can be done. This is normally done because it is necessary to run batch jobs or other utilities which do not properly support recoverable data. As such, to allow these jobs or utilities to safely update the data, it is first necessary to make a copy of the data. In event that the batch job or utility fails or encounters an error, this copy can be restored and online access can be reenabled.

If the batch job completes successfully, then the updated copy of the data set can be used because only the batch job had access to the data while it was being updated. Therefore, the data cannot have been corrupted by interference from online transaction processing or other batch jobs.

Transactional VSAM Background

IBM originally issued a statement of direction regarding Transactional VSAM back in May of 2000. At that time, no statement was made as to the intended ship vehicle of this new functionality or how it would be delivered.

Transactional VSAM can be thought of as VSAM RLS plus. It builds on the capabilities provided by VSAM RLS by continuing to support access to VSAM data sets at the record level and by adding the necessary commit, backout and logging to support full transactional recovery.

DFSMStvs is a new member of the DFSMS family of products which includes DFSMShsm, DFSMSdss, and DFSMSrmm.

What is DFSMStvs?

Transactional VSAM (TVS)

Transactional VSAM supports the System/390 Coupled Systems Strategy. It builds on the locking and data caching functions provided in VSAM RLS using Coupling Facility (CF) hardware to provide a shared data storage hierarchy for VSAM data. A new Transactional VSAM access mode builds on the access to shared VSAM data sets via CF locking and caching provided by VSAM RLS. It adds the logging and two-phase commit and backout protocols required for full transactional capability and sharing.

TVS is a major step toward the enablement of continuous operations (24x7) of CICS VSAM online applications.

When accessing a data set using VSAM RLS, RLS provides the record level locking and buffer coherency that allows for data sharing. CICS retains the responsibility for doing all logging and providing the two-phase commit and back out protocols. This means that while applications using CICS can read and update recoverable VSAM data sets, batch jobs cannot, because they have no way to provide the necessary recovery capabilities.

Transactional VSAM gives these capabilities to batch job by doing the logging for them and by providing the necessary two-phase commit and backout capabilities. It does this by using the System Logger to perform all of its loggings, and by using the system sync pointer manager to provide two-phase commit and back out coordination across all participating resource managers. This is the RRS (Recoverable Resource Services) component of RRMS (Recoverable Resource Management Services).

In this environment, Transactional VSAM can be thought of as a resource manager which manages records in recoverable VSAM data sets.

The Value of Transactional VSAM

The objective of Transactional VSAM is to provide transactional recovery directly within VSAM. It is an extension to VSAM RLS. It allows any job or application that is designed for data sharing to read/write share VSAM recoverable files.

Transactional VSAM is a follow-on project/capability based on VSAM RLS (record level sharing). RLS provides a sysplex-wide server for sharing VSAM files. It provides CF (coupling facility) based locking and data caching with local buffer cross-invalidate. RLS supports CICS as a transaction manager. This provides sysplex data sharing of VSAM recoverable files when accessed through CICS. CICS provides the necessary unit-of-work management, undo/redo logging, and commit/backout functions. VSAM provides the underlying sysplex-scope locking and data access integrity.

Transactional VSAM adds logging and commit/backout support to VSAM RLS. Transactional VSAM requires/supports the OS/390 RRMS (Recoverable Resource Management Services) component as the commit or sync point manager.

Transactional VSAM provides a level of data sharing with built-in transactional recovery for VSAM recoverable files that is comparable to the data sharing and transactional recovery support for data bases provided by DB2 and IMSDB.

Extending the Availability of CICS Applications

The majority of customers in the CICS Joint User Group have batch windows ranging from two to ten hours. The programs run during the batch window consist of both in house applications

and vendor written applications. These customers would like to be able to do batch haring to enable them to reduce or eliminate the batch window.

While VSAM record level sharing was a starting point for this goal, it does not fully enable the necessary sharing. The possibility of augmenting it with Transactional VSAM capability was reviewed with the Customer Design Council in May of 1995 to validate that it addressed a real requirement and that the approach was viable. The consensus from the council was that the project should be allowed to proceed, and several customer indicated a strong need for the capabilities provided by Transactional VSAM.

The batch window is a period of time in which online access to recoverable data sets must be disabled. During this time, no transaction processing can be done. This is normally done because it is necessary to run batch jobs or other utilities which do not properly support recoverable data. As such, to allow these jobs or utilities to safely update the data, it is first necessary to make a copy of the data. In event that the batch job or utility fails or encounters an error, this copy can be restored and online access can be reenabled.

If the batch job completes successfully, then the updated copy of the data set can be used because only the batch job had access to the data while it was being updated. Therefore, the data cannot have been corrupted by interference from online transaction processing or other batch jobs.

TVS Customer Value

IBM customers who have considerable assets in VSAM require a rapid way to integrate this data "directly" with e-applications. The Web application has no awareness of VSAM in the same sense as a COBOL program for instance. Instead, the Web application requests a service from the Web. This Web "service" may require VSAM assets in order to compose the result set to return to the requester. Knowledge of VSAM exists in the tools and structures - "infrastructure" - of the WebSphere Application Server "container" that contains the program - i.e. Enterprise Java Bean (EJB) - that fields the inbound request, converts it into the appropriate parameterized invocations, and drives the VSAM J2EE Connector to gather the raw VSAM data that it - the EJB - will render and pass back to the Web requester.

This describes in thumbnail fashion, a multi-tier application server approach to database access that decouples the back end legacy system from the Web e-application. An Enterprise Java Bean (EJB’s) layer provided in the WebSphere container can be loosely equated to an access method that can "directly" access legacy data - via "connectors" - to back end Enterprise Information Systems (EIS). The VSAM connector is complements the other connectors such as DB2 and CICS and IMS, etc. in that it makes the WebSphere solution more complete. For IBM customers with significant non-database VSAM assets, it is the most cost effective path to an e-business solution.

VSAM Data Sharing - RLS

With VSAM RLS, batch jobs could share non-recoverable files for read and update while CICS was using them. Assuming the share options were correctly defined, they could also share recoverable files, as long as they only wanted to read them.

R/O = read only
R/W = read or write

VSAM Data Sharing - Transactional VSAM

With Transactional VSAM added to the picture and built on top of VSAM RLS, full sharing of recoverable files becomes possible. Batch jobs can now update the recoverable files without first quiesicing CICS' access to them.

System Logger is a z/OS facility to write logs from multiple address spaces in multiple systems into the coupling facility which can then be written to DASD

CF -- Coupling Facility required for locking, caching, and buffer coherency in a sharing environment.

Accessing a Data Set with Transactional VSAM

For the most part, Transactional VSAM only supports those data sets that are defined as recoverable. That is, the log attribute for the data set is either UNDO (back out logging only) or ALL (back out and forward recovery logging). When a batch job opens a recoverable data set for update, the open is done in Transactional VSAM mode. This allows Transactional VSAM to provide the necessary transactional recovery for the data set.

Note that data sets opened for input with the CRE option specified are also open in Transactional VSAM mode. This is because the CRE (consistent read explicit, also known as repeatable read) locks are sync point duration locks. Without the Transactional VSAM support, VSAM RLS would know nothing about sync points, and the locks would never get released.

In either case, read or update access, the application is responsible for defining the sync points by invoking the RRS commit or back out function. It is not possible for Transactional VSAM to define the sync points because it knows nothing about what the application is actually doing. If it tried to imply a sync point every so many operations any of the following could happen:

1. Transactional VSAM could insert the sync point between paired operations (that is, a GET UPD and its paired PUT UPD or ERASE).

2. Transactional VSAM could insert the sync point in the middle of two pieces of work which were meant to be atomic (for example, between subtracting 100 dollars from a checking account and adding it to a savings account).

3. Transactional VSAM could decide to commit a piece of work that the application would have realized should have been backed out (or vice versa).

Using Transactional VSAM

Transactional VSAM (sometimes abbreviated TranVSAM) permits a batch job to OPEN a recoverable file for OUTPUT. However, most existing batch jobs that modify VSAM files are NOT designed to permit data sharing. Each job assumes the file is NOT being changed by another concurrently executing program.

Transactional VSAM provides the necessary transactional recovery to enable data sharing. Batch jobs that are designed to use the transactional programming model may use Transactional VSAM to read/write shared VSAM recoverable files.

What is a Transaction?

This is an example of the usage of interfaces to Transactional VSAM. Notice the only change to the already existing VSAM application interface is the new repeatable read option on GET.

Call SRRCMIT invokes the RRS component of RRMS to commit the changes made by the application. RRS interfaces with Transactional VSAM to commit the VSAM file changes and release the corresponding VSAM locks.

If the application also changed other recoverable resources managed by resource managers (like DB2) that support RRMS, the commit applies atomically across Transactional VSAM and the other resource managers.

Unit of Recovery

The set of changes processed by a single commit or backout (between commits or back outs, or between the start of the program and the first commit or backout) is called a unit of recovery. A unit of recovery is an indivisible (atomic) entity which permits resource consistency to be maintained.

Change for a single unit of recovery occur atomically. Atomic operations are those that execute in the same manner as machine instructions: either all the changes are made or none are made. The point at which the applications makes the changes permanent is called the atomic instant of commit, or more simply commit or a sync point. If for some reason the applications desires that the changes not be made or an error occurs which makes it impossible for the changes to be made atomically, then the changes are backed out.

We strongly recommend that your not allow your programs to end without committing and that you not convert your applications to use Transactional VSAM without inserting sync points. If

you use only the implicit sync points (commit for normal end of step or end of job, and back out for abnormal end of step or end of job) you may end up creating retained locks which can cause problems for other jobs. When these jobs request the retained locks, instead of waiting for the locks, they will receive retained lock rejects, which can only be cleared by the application which caused them committing or backing out.

In addition, not adding commits or back outs to your application can cause the entire program to be viewed as a single unit of recovery. This can cause it to hold a large number of locks and to have a large amount of active data in the undo log. This can have two very negative impacts on your sysplex operations. The first is that by holding a large number of locks, your program or application is locking out any other batch applications or on line systems that need to access those records. This can negatively impact your on line response time. Having a large amount of active data on the undo log can cause it to spill from the coupling facility onto DASD, negatively impacting system performance.

Note: This example is extremely simplified since it leaves out any GET operations, as well as other operations such as POINT and ENDREQ.

Transactional VSAM Logstreams

Each Transactional VSAM instance has a primary system log (undo log) and a secondary system log (shunt log). Both are implemented as MVS system logger log streams. These logs are intended for use only for recovery purposes--for example, during back out or restart. They are not meant to be used for any other purpose. The primary system log stream holds data for most normal in-flight units or recovery (URs). The secondary system log stream holds information for URs that cannot be completed, normally due to back out failures, or which have been determined to be long running.

System log stream names are generally qualified names, of which the high level qualifier is the Transactional VSAM instance name. Transactional VSAM requires the log stream names to be IGWTVnnn.IGWLOG.SYSLOG and IGWTVnnn.IGWSHUNT.SHUNLOG respectively.

Transactional VSAM instance names must be unique throughout the sysplex. Each Transactional VSAM instance supports the system log for its exclusive use. All other logs are kept separate from the system log. Their stream names are checked to ensure that they are different from that of the system log.

You must define the log streams for the Transactional VSAM primary and secondary system logs to the MVS system logger before starting Transactional VSAM.

Forward recovery logs are kept separate from system logs. Transactional VSAM obtains the log stream name of a VSAM forward recovery log from the ICF catalog entry for the data set. Transactional VSAM only writes to forward recovery logs. It is the responsibility of products

which provide forward recovery capability to read them.

If you associate forward recovery log stream ids with VSAM data sets, you must ensure that you define those log streams to the MVS system logger before you open the data sets. If the log stream name is not defined to the MVS system logger, the connect request fails. Note that the log stream definition determines the MVS log stream structure in the coupling facility to which the log stream is written.

If the use of a log-of-logs is requested via the IGDSMSxx member of SYS1.PARMLIB, Transactional VSAM writes a log-of-logs. It contains copies of the start of run records, and the tie-up records and file close records for recoverable data sets, and log stream exception information. This provides data set recovery products such as CICSVR with the information required for to control forward recovery. If you use both Transactional VSAM and CICS, it is recommended that you use the same log-of-logs for both.

If you do not want Transactional VSAM to write a log-of-logs, omit the LOG_OF_LOGS parameter from the IGDSMSxx member of SYS1.PARMLIB. If you use Transactional VSAM in a sysplex environment, and you run with a log-of-logs, then the log-of-logs should be a single log stream shared by all Transactional VSAM instances that are used to access the same set of recoverable data sets.

Forward Recovery Logging

Forward recovery is the process of applying the records contained in a redo (or forward recovery) log to redo the changes made to a data set in event that the data set is lost or damaged and must be recovered from a back up copy. This process is only available to those data sets which are defined as being forward recoverable -- that is, have LOG(ALL) specified and also have a LOGSTREAMID specified.

The forward recovery log records written by Transactional VSAM were designed to be as similar as possible as those written by CICS. There are some minor differences so that it is possible to differentiate the writer of the records, but those differences are so minor that any forward recovery utility which can perform forward recovery for data sets accessed by CICS can also do so for those accessed in Transactional VSAM mode.

Transactional VSAM Logging

Transactional VSAM logging uses the z/OS System Logger. The Transactional VSAM logger is a reuse of the design and much of the code of the CICS logger. Forward recovery logstreams for VSAM recoverable files will be shared across CICS and Transactional VSAM. The forward recovery log stream is specified as an attribute of the file. It is specified via AMS. The parameter is: LOGSTREAMID=name.

CICS/Transactional VSAM Logstreams

Transactional VSAM performs the logging for VSAM RLS data sets accessed in Transactional VSAM mode. You can share a forward recovery log stream between multiple data sets - you do not need to define a log stream for each forward-recoverable data set. Your decision is a balance of transaction performance, rapid recovery, and the work involved in managing a large number of log streams.

The MVS logger merges all the forward recovery log records from the various Transactional VSAM instances onto the shared forward recovery log. Some points to consider are:

1. All data sets used by one transaction should use the same log stream (to reduce the number of log streams written to at sync point).
2. Share a forward recovery log stream between data sets that:
   - Have similar security requirements
   - Have similar backup frequency
   - Are likely to need restoring in their entirety at the same time.
3. Log stream names should relate to the data sets. For example, PAYROLL.data_sets could be mapped to a forward recovery log named PAYROLL.FWDRECOV.PAYLOG.
4. Don't mix high update frequency data sets with low update frequency data sets, because this causes a disproportionate amount of unwanted log data to be read during recovery of low frequency data sets.
5. Don't put all high update frequency data sets on a single log stream because you could exceed the throughput capacity of the stream.
6. If you define too many data sets to a single log stream, you could experience frequent structure-full events when the log stream can't keep up with data flow.

7. Redundant data should be deleted from log streams periodically so that the log streams do not become excessively large. Typically, for a forward recovery log, deletion of old data is related to the data backup frequency. For example, you might keep the 4 most recent generations of backup, and when you delete a redundant backup generation you should also delete the corresponding redundant forward recovery log records. These are the records older than the redundant backup because they are no longer needed for forward recovery.

The log of logs is written to provide information to forward recovery programs such as CICS VSAM Recovery (CICSVR) The log of logs contains copies of the start of run records, and the tie-up and file close records for forward recoverable data sets written to forward recovery logs. Thus it provides a summary of which recoverable VSAM data sets Transactional VSAM has used, when they were used, and to which log stream the forward recovery log records were written.

If you have a forward recovery product that can utilize the log of logs, you should ensure that all Transactional VSAM instances sharing the recoverable data sets write to the same log of logs log

stream. Do not share the log of logs between test and production Transactional VSAM instances, because it could be misused to compromise the contents of production data sets during a restore.

Transactional VSAM and z/OS RRS

Recoverable Resource Management Services consists of three pieces. The first piece is registration services. This is the set of services that allow resource managers to register with RRMS and let it know that they are present and managing recoverable resources. When Transactional VSAM initializes it uses these services to register as a resource manager for recoverable VSAM data.

The second piece is context services. This is the piece of RRMS that manages things like TCBs and makes sure that if a task terminates without having taking action on an outstanding unit of recovery (transaction) that the system takes the appropriate action. Transactional VSAM registers its interest with context services when it detects that a unit of recovery running under a TCB is doing Transactional VSAM work.

The third piece is resource recovery services. This is the z/OS sync point manager and provides the interfaces that applications use to invoke commit or back out.

Transactional VSAM is a recoverable resource manager. It is NOT a commit or sync point manager. Transactional VSAM interfaces with the z/OS sync point manager. When an application issues a commit request directly to z/OS or indirectly through a sync point manager that interfaces with the z/OS sync point manager, Transactional VSAM is invoked to participate in the 2-phase commit process. Other resource managers (like DB2) whose recoverable resources were modified by the transaction are also invoked by the z/OS sync point manager thus providing a commit scope across the multiple resource managers.

SYS1.PARMLIB

Some Transactional VSAM parameters apply only to the system on which they are found. Others apply across the sysplex. Regardless of which type a parameter may be, values are not remembered across IPLs. Therefore, your IGDSMSxx member of SYS1.PARMLIB must always specify a complete set of the parameters you wish Transactional VSAM to use.

The RLSTMOUT parameter is optional. It specifies the maximum time in seconds that a VSAM RLS or Transactional VSAM request is to wait for a required lock before the request is assumed to be in deadlock and aborted with VSAM return code 8 and reason code 22(X'16'). RLSTMOUT is specified as a value in seconds in the range of 0 to 9999. The default is 0. A value of 0 means that the VSAM RLS or Transactional VSAM request has no time out value; the request will wait for as long as necessary to obtain the required lock.

VSAM RLS detects deadlocks within VSAM and Transactional VSAM. It cannot detect deadlocks across other resource managers, and uses the time out value to determine when such deadlocks may have occurred. The installation may specify a global time out value in the IGDSMSxx member of SYS1.PARMLIB, a step level time out value on the JCL, and specific applications may specify a time out value on the RPL passed for each VSAM request.

For a particular VSAM RLS or Transactional VSAM request, the value used for time out is:

1. the value specified in the RPL, if any
2. the value specified in JCL at the step level, if any
3. the value specified in the IGDSMSxx member of SYS1.PARMLIB, if any

RLSTMOUT is both a VSAM RLS and a Transactional VSAM parameter. Therefore if RLSTMOUT is found but no Transactional VSAM instance names are specified, then the value is used only by RLS. RLSTMOUT may be specified only once in a sysplex and applies across all systems in the sysplex. The first instance of Transactional VSAM brought up within the sysplex determines the value. Subsequent Transactional VSAM instances use the value established by the first system, regardless of what may be specified in their members of SYS1.PARMLIB. To change the value, use the SETSMS command. This will cause the value to be changed on all systems in the sysplex.

The SYSNAME parameter is optional. It specifies the name of the systems to which the preceding or subsequent Transactional VSAM instance names apply. Up to 32 system names may be specified. The system names must be specified in the same order as the Transactional VSAM instance names. SMS examines the system names specified and compares them to the system name in the CVT. When a match is found, it stores the value of the TVSNAME parameter in the matching position as the Transactional VSAM instance name for the system. The combination of SYSNAME. and TVSNAME should be used when the PARMLIB member is shared between systems.

If no SYSNAME. parameter is specified, then the TVSNAME applies to the system on which the PARMLIB member is read. This parameter is supported only in PARMLIB. It is not supported on the SETSMS command.

If SYSNAME. is found without TVSNAME, it is treated as a syntax error. If SYSNAME. is found, but the system is not listed, then Transactional VSAM will not be started on the system.

The TVSNAME parameter is optional. It specifies the identifiers which uniquely identify instances of Transactional VSAM running in the sysplex. Up to 32 identifiers may be specified. The identifiers must be unique within the sysplex. They must be a numeric value from 0 to 255, which Transactional VSAM uses as the last byte of its instance name (although it will be displayed as three bytes).

If the TVSNAME parameter is specified without the SYSNAME. parameter, then only a single value may be specified, and the name applies to the system on which the PARMLIB member is read. The numeric digits are appended to the characters IGWTV to form the Transactional VSAM instance name. The TVSNAME parameter may be used without the SYSNAME. parameter when the PARMLIB member is not shared between systems. This parameter is supported only in PARMLIB. It is not supported on the SETSMS command, and there is no default.

If no TVSNAME parameter is found, then Transactional VSAM processing will not be available on the system.

The TV_START_TYPE parameter is optional. It specifies the type of start Transactional VSAM is to perform. Up to 32 TV_START_TYPE values may be specified. TV_START_TYPE values must be specified in the same order as Transactional VSAM instance names. If WARM is specified, then Transactional VSAM reads its undo log and processes the information it finds in accordance with the information RRS has about any outstanding URs. If COLD is specified, then Transactional VSAM deletes any information remaining in the undo log and starts as if the log were empty. COLD should be used when the Transactional VSAM undo log has been damaged. The default is WARM.

You can allow some values to default and others to be specified by entering something like:

TV_START_TYPE(COLD,,COLD)

This would cause the Transactional VSAM instance in the second position to warm start, while the first and third Transactional VSAM instances would cold start.

If TV_START_TYPE is found with only the TVSNAME parameter, then it applies to the system only which the parmlib member is being read. If it is found with both TVSNAME and SYSNAME parameters, then it applies to the system specified on the SYSNAME parameter preceding or following it. If the TVSNAME parameter is not specified and TV_START_TYPE is, it is treated as a syntax error.

This parameter is supported only in PARMLIB. It is not
supported on the SETSMS command.

The AKP parameter is optional. It specifies the activity key point trigger value, which is the number of logging operations between the taking of keypoints. Up to 32 activity key point values may be specified. AKP values must be specified in the same order as Transactional VSAM instance names. Valid values are 200 to 65535. The default is 1000.

You can allow some values to default and others to be specified by entering something like:

AKP(800,,3000)

This would cause the value for the system/Transactional VSAM instance in the second position to default to 1000, and set the values for the first and third systems/Transactional VSAM instances to 800 and 3000 respectively.

If AKP is found with only the TVSNAME parameter, then it applies to the system only which the parmlib member is being read.

If AKP is found without TVSNAME, it is treated as a syntax error.

The log of logs parameter is optional. It specifies the log stream to be used as the log of logs. This log contains copies of the tie up records written to forward recovery logs and is used by forward recovery products. If it is not specified, then no log of logs is used. The default is to use no log of logs.

If LOG_OF_LOGS is found but no Transactional VSAM instance names are specified, then it will be treated as a syntax error.

This parameter is unique to each system in the sysplex. As each instance of Transactional VSAM comes up, it uses the log of logs name found in its member of SYS1.PARMLIB.

If you are running test and production systems within the same sysplex, it may be desirable to use different logs of logs. In this case, use a separate parmlib member for the test system and specify a different log of logs in it.

This parameter is supported only in PARMLIB. It is not supported on the SETSMS command.

The QTIMEOUT parameter is optional. It specifies the quiesce exit timeout value in seconds. Only one quiesce timeout value may be specified and applies to all systems in the sysplex. The first instance of Transactional VSAM brought up within the sysplex determines the value. Subsequent Transactional VSAM instances use the value established by the first system, regardless of what may be specified in their members of SYS1.PARMLIB. To change the value, use the SETSMS command. This will cause the value to be changed on all systems in the sysplex.

The quiesce timeout value specifies the amount of time the Transactional VSAM quiesce exits will allow to elapse before concluding that a quiesce cannot be completed successfully. Valid values are 60 to 3600. The default is 300.

If QTIMEOUT is found but no Transactional VSAM instance names are specified, then it is treated as a syntax error. Unlike other Transactional VSAM parameters, it is specified only once and applies across all systems.

The MAXLOCKS parameter is optional. It specifies two values: the maximum number of unique lock requests that a single unit of recovery may make before warning messages are issued and an increment value. Once the maximum number of unique lock requests is reached, the warning messages will be issued every time the number of unique lock requests over and above the maximum increases by a multiple of the increment. The messages include the job name of the job which is holding the locks. The installation can then determine if the application should be allowed to proceed or if the job should be canceled. If the job is canceled, the UR will be backed out, and the locks will remain held until the back out completes.

MAXLOCKS is specified as a pair of values in the range of 0 to 999999. The default for both values is 0. A value of 0 indicates that warning messages should not be issued.

If MAXLOCKS is found but no Transactional VSAM instance names are specified, then it will be treated as a syntax error. Unlike other Transactional VSAM parameters, it is specified only once and applies across all systems.

Performance

VSAM performs best when running in non-shared resource (NSR) mode. Because no resources are shared, no serialization effort is required and, therefore, there is no cost for serializing except for some basic support through VSAM's share options. But, as the name implies, the NSR resources are not shared with other users or applications. With local shared resources (LSR) and global shared resources (GSR), VSAM provides some extra code to allow sharing to some degree and to manage buffer pools. Record-level sharing (RLS) goes another step further and provides sharing support across a Parallel Sysplex on a logical record level.

With the advent of DFSMStvs, another step is taken by providing serialization support through record locking, logging support for commit or backout, as well as forward recovery functionality through forward recovery logging. All of this does not come for free, and requires some extra effort in various system areas. Compared with NSR or LSR, DFSMStvs requires some extra processing cost from the following areas:

Cross address space communication -- all requests are passed to the SMSVSAM server for processing
Record locking -- shared locks are obtained for any records which are read with integrity and exclusive locks are obtained on any records which are obtained for update or added
Data logging for commit or backout -- without logging, it would be impossible to provide atomicity and recoverability because there would be no way to perform the backout function
Commit or backout processing -- this two-phase process ensures that either all updates are committed or all the changes are backed out

Forward recovery logging -- for those data sets which are forward recoverable, there is additional overhead associated with writing copies of the changed records to the forward recovery log
Coupling facility cache access -- the coupling facility is used for locking, buffering and logging
Loss of chained sequential I/O -- Transactional VSAM is built on top of VSAM RLS, which is optimized for direct access rather than sequential access
Loss of deferred write -- in order to provide the level of recoverability required, VSAM RLS and Transactional VSAM do not support deferred write

Taking these into consideration, DFSMStvs performance cannot be compared directly with the performance of NSR access to VSAM files because so much more work is done on behalf of the application. Batch applications that use DFSMStvs will take longer than traditional batch applications that were designed to run in a batch window without a concurrent online system. The motivation to use DFSMStvs comes from the need to access VSAM files concurrently, especially with a transaction server which is up and providing online service even during batch update processes.

Performance -- Parallelizing the Workload

Even though the additional overhead associated with locking, logging, and two phase commit and back out results in increased elapsed time, it should be noted that this can be offset by the parallelism that Transactional VSAM allows. Of course, this assumes that the batch jobs are independent of one another, and that one batch job is not waiting for the output of another batch job to provide its input.

In this example created by the ITSO, the master file contained 1,000,000 records. The job which updated this master file was run in three different ways: as a single non-shared resources (NSR) job that updated all 1,000,000 records, as a single Transactional VSAM job that update all 1,000,000 records, and as four separate Transactional VSAM jobs that each updated 250,000 of the records.

This chart shows a comparison of the elapsed time. Note, however, that these results were obtained in an uncontrolled test environments an should by no means used as benchmarks.

When splitting the Transactional VSAM work load into four parallel batch jobs, each jobs ends after about the same elapsed time because each job is processing the same number of records. They complete in about 41% of the time it took to run the updates as a single Transactional VSAM job. Running them as four parallel jobs even caused them to run slightly faster than the traditional NSR environment when comparing wall clock time. The number of EXCPs and the CPU consumption vary slightly between jobs in the DFSMStvs environment. In addition, CPU time, elapsed time, and the number of EXCPs are higher when using Transactional VSAM. This is due to the extra overhead noted earlier: locking, logging, and two phase commit and back out.

Transactional VSAM Summary

Transactional VSAM provides general VSAM recoverable file sharing. This addresses a long standing requirement for sharing of VSAM files across CICS and batch jobs.