Concurrent VSAM access for batch and CICS: A white paper series

Transactional VSAM (DFSMStvs)

A white paper from:
IBM, CICS, Db2, and z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. SYSB-II is a registered trademark of H&W Computer Systems, Inc. All other products mentioned herein may be trademarks or registered trademarks of their respective companies.
CICS, batch, and Transactional VSAM

CICS® applications and VSAM data drive a large percentage of the transactional processing at the core of worldwide commerce and information systems. According to current estimates, CICS applications handle more than 30 billion transactions per day and process more than $1 trillion worth of business every week. When these systems are offline or data is inaccessible due to batch processes, productivity stalls, crucial information is unavailable, and decisions are delayed.

With the goals of IT and business units focused on delivering web access, electronic transactions, and interconnectivity as the basis for revenue growth and customer satisfaction, it increases the pressure on the VSAM data to be more available as enterprises push toward a 24/7 environment. This makes availability of VSAM-reliant CICS applications more important than ever.

This white paper explores issues relevant to batch/CICS file sharing and Transactional VSAM (DFSMSstvs). It is designed as a concise guide for IT personnel if they are considering Transactional VSAM as a solution for concurrent sharing of VSAM files between CICS and batch processing.

This white paper is one of four in the H&W series titled *Concurrent VSAM Access for Batch and CICS*. The other white papers in the series are:

- **Transparent VSAM File Sharing via CICS and Batch**. This explores a method for sharing VSAM files inside CICS so that batch jobs appear to CICS like any other online transaction.
- **VSAM Record-Level Sharing (RLS)**. This explores a method for accessing VSAM files that gives multiple CICS address spaces concurrent READ/WRITE access to recoverable VSAM data sets. Challenges are explored in detail.
- **General Overview of Conventional and Current Alternatives** — This white paper explains the conventional and previous approaches, which include SHAREOPTIONs, file open-and-close products, shadow files, memo posting and EXCI, VSAM RLS and Transactional VSAM.
VSAM historical context

VSAM, CICS, and parallel Sysplex environments grew in popularity throughout the 1970s, 1980s, and 1990s. VSAM was originally shipped by IBM® in 1973, and it became the file access method of choice when much of the CICS application development occurred.

By the 1990s, IBM had already been using the coupling facility to share Db2® and IMS data across operating system images. Sharing VSAM data amongst CICS systems or even between CICS and batch in this manner was still an issue. It made sense to use the same technology on the CICS VSAM platform as well. The first step toward greater sharing of CICS VSAM data came with the announcement of VSAM RLS. Changes were made to both CICS and VSAM toward the goal of sharing VSAM files between multiple CICS regions and batch. While a good first step in reaching that goal, VSAM RLS still had significant restrictions that meant the challenge of sharing VSAM files between batch and CICS persisted, and even magnified the need for full continual access to VSAM data.

In mid 2003, IBM announced the availability of Transactional VSAM. This optionally priced feature enables batch jobs and CICS transactions to update shared VSAM data sets concurrently.

Transactional VSAM: an overview

Transactional VSAM is an optional z/OS® feature that enables batch jobs and CICS transactions to concurrently update shared VSAM data sets. Multiple batch jobs and CICS transactions can process with the same VSAM data sets. Data integrity is ensured for both CICS and concurrent batch updates. This product is designed to solve the following business problems:

• Reduce or eliminate the batch window for CICS and other VSAM applications. This is done by allowing concurrent batch and online updating of recoverable VSAM data sets.
• Improve system availability.
• Simplify the scheduling of batch jobs because batch can be run concurrently in multiple z/OS images in a parallel sysplex, rather than serially in a single image.
• The sharing of VSAM data sets at the record level with integrity, and delivering commit and rollback functions for non-CICS applications.

Transactional VSAM is an extension of VSAM RLS and comes with several prerequisites, including a system coupling facility. For a current list of these prerequisites, review the DFSTVSTv Overview and Planning Guide at www.ibm.com.
Transactional VSAM — Is it right for you?

The batch program perspective

Most batch programs need source code adjustments to work with Transactional VSAM. This should be done so that each unit of work can be viewed as a single transaction. This has many repercussions for the redesign of a batch program. In order to execute batch programs this way, the program must assume that the VSAM resources are not only being accessed by CICS online users, but are also being accessed and updated by other transactional batch processes. For this reason alone, transactional batch processes need to be aware of the impact they have on other concurrent users of the resources they are accessing.

Transactional VSAM is a recoverable resource manager, and by itself does not manage sync point or backout processing. To provide support for sync processing, Transactional VSAM interfaces with z/OS Resource Recovery Services (RRS). RRS is the sync point manager and is responsible for providing a single point of coordination for the commit processing of resources owned by different resource managers such as Transactional VSAM. RRS provides the interfaces that an application uses to either commit changes or to back out changes. Other resource managers such as IMS, MQ, and Db2 can also utilize RRS services to coordinate commit or backout requests. Because a Transactional VSAM batch job has the capability of accessing data owned by one or more of these additional resource managers, all of these resource managers must use RRS services to coordinate commit and backout activity. Failure to use RRS services by all of the resource managers results in an uncoordinated unit of work in which only part of the updates are committed or rolled back.

To exploit the capabilities of Transactional VSAM without causing resource contention issues, a batch program should issue frequent sync points. More discussion on sync processing follows later in the paper.

During transaction processing, RLS obtains exclusive locks on records that are updated and holds these locks until the end of the transaction. These locks are used to ensure that any process requesting a consistent read is prevented from viewing the changes until they are committed. Locking also ensures that no other process can change those locked records until they are committed or backed out.
Processing in this manner, a transaction is isolated from the effects of other programs sharing the data. However, the implications of locking, logging, and sync processing must be considered:

- **Locking other users out of the shared data** — Locks should be held for as short a time as possible, and an application should commit changes frequently. Because other users of the data might be CICS transactions, it is particularly important to commit frequently so that CICS transactions do not suffer long waits for locks.

- **Avoiding deadlocks** — A deadlock occurs when two concurrent processes need to obtain a lock that the other process is holding. The usual way of avoiding such deadlocks is to implement a programming standard that enforces the order in which updates should be made.

- **Handling a failure to obtain a lock** — Any program running in a transactional environment must be able to handle a failure to obtain a lock.

- **Log implications** — If too much work remains uncommitted, the undo log records associated with those changes must be kept. This means logs can grow substantially and cause performance issues.

- **Loss of position after a sync/commit** — Transactional VSAM forces a loss of positioning during sync processing.

- **Job failure** — When restarting from a job failure, the restart process must avoid reapplying changes that have been committed in an earlier execution.

While it might be possible to modify a batch job step to exploit Transactional VSAM without making source code changes, you need to be sensitive of the current and future implications of doing so. Using this approach means the entire job step would be treated as a single Transactional VSAM transaction. This has the potential to introduce a multitude of locking, logging, and concurrency and backout issues. In certain cases, this might be acceptable, such as for jobs that only perform a small number of updates. In this instance, the cost of changing these programs might outweigh the benefit of reducing lock contention. Such cases are in all probability the exception rather than the norm. If you consider this approach, add a process to periodically check the volume of updates that this program is processing. If the volume grows, you might need to make the source code changes described earlier.

It might also be possible to split updates and replace a single batch job step with multiple concurrent batch processes by exploiting RLS and Transactional VSAM. For example, if the program has a sequential set of updates (each independent of any other), it might be possible to split the updates and run smaller sets of updates in multiple concurrent batch job steps, each executing a different instance of the same program.
Syncpointing considerations

Batch applications need to be re-examined to determine the frequency of their commit logic in order to avoid elongating CICS response times. How often you commit is determined by the reason for the commit. One reason might be to release locks while another is to reduce the cost of a rollback. There is always a trade-off between the cost of issuing the commit, and the effects of not doing a commit. Excessive commits consume additional resources and elongate job execution time. Insufficient commits cause concurrency issues because of lock contention.

Alternatives to treating the job step as a single transaction with an implicit commit at a successful step end include the following:

- Issue a commit after a single logical unit of work has been processed.
- Issue a commit after a number of logical units of work have been processed.
- Issue commits based on a count of the number of records updated.
- Issue commits based on elapsed time.
- Base the commit on multiple factors such as time and update count.

In order to issue sync points in the application, the code must be modified to invoke RRS commit services, which is a service called SRRCMIT. If backout is required, the program must be modified again to call SRRBACK, which is the MVS callable service for backout. Whether you decide to commit based on logical units of work, a timer, or a combination of factors, good programming practice will allow for easy adjustment of the mechanism used to control the commit frequency.

Sync points (both commit and backout) cause all locks to be released, which means the application cannot depend on locking across the sync process. For example, an application cannot issue a read just prior to a commit and then follow the commit with an update. In addition, POINTs used to establish position when using repeatable read (CRE) can cause a subsequent read to fail, if a sync call is inserted between the two processes. These examples of positioning issues occur because Transactional VSAM forces a loss of positioning during sync point processing. Loss of positioning is forced to avoid integrity issues. Locks are released by the sync process, and an application cannot assume that any lock obtained prior to the sync is still held.

Transactionalizing batch applications and issuing sync points at appropriate points during the execution raises the question of how to handle situations when the batch process abends, and restart is required.
Recovery considerations

In the past with nontransactional batch work, the process for recovering from a job failure usually meant restoring the data sets used by the job step back to a copy that was taken prior to starting the batch process. The error could then be corrected, and the job step rerun. This process is not practical in a shared environment. Restoring a file to a point in time prior to the batch process removes changes made and committed by other transactional programs with shared access to it.

When a batch Transactional VSAM job fails, all uncommitted, inflight changes are backed out by Transactional VSAM. Committed changes remain. Committed changes are those changes made in a prior unit of work and made permanent by the sync process. If you simply rerun the batch job that failed, committed changes are repeated. If the type of change made is simply an update to a customer's address or e-mail ID, repeating the change would have no impact. However, if a change adds or subtracts money to or from a customer's bank balance, you would not want those changes repeated. To address this, batch programs that participate in a Transactional VSAM environment must be modified to ensure that changes are not reapplied when the application restarts.

When using Transactional VSAM, the traditional, recommended approach to addressing this issue is to modify your application, so that it can be restarted from the last good sync point. In order to design a restartable batch application, you must consider many factors. Coding techniques can be used to avoid reprocessing all of the work that has already been committed in a prior execution, and thereby avoid the data integrity issues that arise in this situation.

Batch application programs can be rewritten to use a positioning data set. To ensure data integrity, changes made by the batch application need to be written to the positioning data set and kept synchronized with the actual work being committed by the application. Program restart can then use the positioning file to find the last record committed. This allows the program to skip all input records already processed, and process only records not processed during a previous execution.

If the batch program is driven by a recoverable input file, another technique to avoid reprocessing records is possible. As the recoverable input file is processed, records can be deleted from the file, leaving only unprocessed records in the event of a batch job step ABEND. While this can simplify the restart process, it requires additional program changes that add overhead to the I/O process, and render the input driver file unusable for subsequent processing.
Prior to the introduction of Transactional VSAM, a batch application could choose to close a file during processing and know that any changes made to the file would be permanent. This is not the case with Transactional VSAM batch applications. Closing a file in a Transactional VSAM batch program does not signal a commit to take place. If changes are made to the file prior to close, record locks remain held until a commit is issued either explicitly or implicitly at the end of the program. Record locks held in this fashion are likely to be converted to retained locks. With retained locks, any other applications that attempt to access the records can abend rather than being informed that the resource is unavailable, and the lock cannot currently be obtained.

Closing files during program execution without first issuing a commit call can also cause indoubt units of recovery. The recovery log records of indoubt units are moved to the shunt log. This increases the possibility of manual intervention to either commit the unit of work or back out the inflight changes made in the unit of recovery. If this proves necessary, you can issue the IDCAMS SHCDS LISTSHUNTED command to list the shunted entries, and the IDCAMS RETRY or PURGE command to resolve the indoubt situation. As a good programming practice and to avoid this situation, issue commits prior to closing any files.

Implementing Transactional VSAM

JCL changes needed to request Transactional VSAM

To access a file using Transactional VSAM you must modify the JCL to add (or update) the RLS parm on the DD statement associated with the file, or where appropriate, update the MACRF parameter used on the ACB macro for the file specified in the program.

Be careful with an application that accesses the same data set using multiple RPLs. Using different RPLs to access the same record can cause lock contention within a unit of recovery.

RLS introduced two read integrity options for batch jobs:

- **No Read Integrity (NRI)** — Does not obtain a share lock on the record (dirty read).
- **Consistent Read (CR)** — Obtains a share lock (waits for any exclusive lock held by another process to be released).
Transactional VSAM provides a third option:

- **Consistent Read Explicit (CRE)** — Repeatable read (obtains a share lock and keeps the lock until the end-of-transaction).

RLS by itself can utilize CRE, but in this instance it is valid for use only with CICS.

Any process specifying the CRE option on the JCL RLS parameter, or on the ACB RLSREAD parameter, utilizes Transactional VSAM access.

When a non-CICS application requests RLS access and OPENs a recoverable file for OUTPUT, the file is opened for Transactional VSAM access. Prior to Transactional VSAM, the OPEN would have failed.

**Preparation for logging**

Transactional VSAM uses the same locking and buffering techniques as VSAM RLS. However, to provide logging for Transactional VSAM you must fulfill the following additional requirements:

- Update the CFRM policy to define structures for the system logger.
- Define log structures and log streams.
- Update data sets to add LOG (NONE/UNDO/ALL) and LOGSTREAMID.
- Update SYS1.PARMLIB (IGDSMSxx) with Transactional VSAM parameters.
- Ensure RACF authorization to the log streams.
- Update SMS configuration.

**Transactional VSAM log policies**

Log structures are defined in two places: the coupling facility resource management (CFRM) policy and the system logger (LOGR) policy. Additionally, you must define log streams for the different kinds of logs used by Transactional VSAM. Also, if a data set is forward recoverable, a log stream for forward recovery must be assigned to this data set.

The CFRM policy divides coupling facility space into structures and enables you to define how MVS should manage coupling facility resources. The system logger component manages log streams based on the policy information that installations place in the LOGR policy. Both are defined by means of a batch job.
Transactional VSAM log streams

A log stream is a VSAM linear data set that simply contains a collection of data. Each log stream is assigned to a structure defined in the LOGR policy. Transactional VSAM needs UPDATE RACF access to the log streams. Many log streams can write data to a single coupling facility structure, and the log data stays segregated by the log stream. The log streams are contained in list structures in the coupling facility (except for DASDONLY log streams).

Each Transactional VSAM instance requires its own pair of system logs: a primary (undo) log and a secondary (shunt) log. You must define these two logs before you can use Transactional VSAM. Additionally, you can define forward recovery logs and a log of logs.

Transactional VSAM logging

Each z/OS system has its own logger address space known as IGXLOGR. Any CICS address spaces and any Transactional VSAM processes running on this z/OS system write their log records to the system logger. The system logger merges log records for shared data sets that have been updated across multiple sysplex images.

The z/OS system logger used by the Transactional VSAM logger reuses the design of the CICS logger and contains much of the code of the CICS logger. In fact, forward recovery log streams for VSAM recoverable files are shared across both CICS and Transactional VSAM.

When files defined to use Transactional VSAM are accessed within CICS, RLS locking protocols are used, and CICS provides logging for backout and for forward recovery. Logging for backout in this fashion also allows CICS to participate in the two-phase commit process. The two-phase commit process allows applications to identify consistency points during processing and to force data synchronization by use of sync point or commit logic.

If the commit process fails for any reason, data can be backed out to the last point of consistency by using before images stored in the log to back out any inflight changes to the data. Forward recovery is also possible by reapplying changes that have already been committed on top of a down-level copy of a data set.

The use of the z/OS logger and RRS allows Transactional VSAM to enhance VSAM RLS and provide not only batch logging for backout and forward recovery support in the same fashion as CICS, but also to support a two-phase commit protocol for batch.
Any data set accessed by VSAM RLS (and therefore Transactional VSAM) must use a LOG parm. The LOG parm can be specified when the data set is defined, or the data set can be altered to specify LOG at a later date. However, the data set must have this option specified before participating in either VSAM RLS or Transactional VSAM. The value and meaning of the LOG parm is as follows:

- **LOG (NONE)**
  
The data set is nonrecoverable. Neither CICS nor Transactional VSAM log any changes for the data set and neither is able to provide transactional or data set recovery.

- **LOG (UNDO)**
  
The data set is backward recoverable. That is, any inflight changes made by a unit of work that do not succeed (are uncommitted) are backed out. Typically this is done by CICS where changes are logged to the CICS journal and used to back out any inflight unit of work that failed. This is also known as transactional recovery.

- **LOG (ALL)**
  
The data set is both backward and forward recoverable. When ALL is used, the logging protocols record both the before and after images of any change. In addition to transactional recovery, ALL allows products such as CICSVR to rebuild data in the event of hardware failures or software problems against the data set. This type of recovery is known as data set recovery. Data sets defined as LOG (ALL) must also have a LOGSTREAMID specification.

**Other considerations**

If you are evaluating Transactional VSAM, consider the following.

**Cost**

- The Transactional VSAM software is fee-based, and as your sysplex grows, plan for software upgrade charges.
- You should budget for training operations, systems, and development staff.
- If the RRS is new to your facility, your recovery process will change. Be sure to add time for this training, and to update documentation describing the recovery process.
- Transactional VSAM requires an investment, which includes acquiring coupling facilities.
- Developers will need to change the source code for each batch program using Transactional VSAM. You should plan for adequate time to test and QA this work.
- You should plan for mainframe upgrade costs in the near term, especially if using the LPAR(s) approach for the coupling facility.
Performance

- The Performance Effects section of the DFSMSdvs Overview and Planning Guide also states that there will be a noticeable increase in processor use by applications using these services. Remember that each VSAM record that is updated by batch also has the following:
  - A record written to one or two log entries.
  - Incremental overhead for the two-phase commit processing.
  - Additional cycles taken to obtain and check locks.
- Expect a potential increase in batch run times as well as a potential increase in response time for CICS transactions using files defined for Transactional VSAM usage.

Conclusion

Transactional VSAM can be an effective tool for reducing or eliminating the batch window for CICS VSAM applications. With this tool, you can expect increased system availability and simplified scheduling of batch jobs because they can run concurrently in multiple z/OS images in a parallel sysplex. However, Transactional VSAM can pose design and coding challenges that can affect performance and increase costs. Before implementing Transactional VSAM you should carefully weigh the benefits against these costs.

VSAM file-sharing software, on the other hand, allows you to maintain investments in the existing mainframe hardware, existing application design and code, and existing people and skills that support everything.

If you do not have programming resources or source code access, review other installments in this series, specifically Transparent VSAM File Sharing via CICS and Batch. Consider also reading VSAM Record-Level Sharing (RLS) and Overview: Conventional and Current Alternatives.

About H&W

Headquartered in Boise, Idaho, H&W has been a leading provider of quality software solutions since 1979. H&W creates reliable, technically sound solutions like SYSB-II® that provide long-term value. Today corporations worldwide, including many Global 500 companies, trust H&W for their IT software and services needs. Call 1-800-338-6692 or visit www.hwcs.com.
References

The authors consulted the following sources in creating this white paper.

• Lovelace, Dave, Rama Ayyar, Alvaro Sala, Valeria Sokal, and IBM Technical Support Organization. 
  VSAM Demystified. 3rd Ed. (Publication number SG24-6105-02.) IBM International Technical Support 

• Menendez, Terri. VSAM Record Level Sharing (RLS Overview), Parts 1 and 2. IBM System Storage and 

• Morton, Nigel, Kazumasa Kawaguchi, Guido Thielemans, Dave Lovelace, and IBM Technical Support 
  Organization. DFSMSdss Overview and Planning Guide. 1st Ed. (Publication number SG24-6971-00.) 

• Morton, Nigel, Ron Bretschneider, Ruth Ferziger, Dave Lovelace, Mary Lovelace, and IBM Technical 
  Support Organization. DFSMSdss Presentation Guide. 1st Ed. (Publication number SG24-6973-00.) IBM 
  International Technical Support Organization, 2003

• Rogers, Paul, Redlef Janssen, Andre Otto, Rita Pleus, Alvaro Salla, and Valeria Sokal. ABCs of z/OS 

• Software Announcement 203-131, dated May 13, 2003 (IBM z/OS V1.4: Support for IBM eServer 
  zSeries z990 Multipurpose On Demand Servers and Preview: z/OS V1.5), http://www-01.ibm.com/
  common/ssi/rep_ca/1/897/ENUS203-131/ENUS203-131.PDF