We've been fans of H&W for years after hearing how highly they're thought of by their customers, and are delighted that they've agreed to share their extensive expertise in VSAM and RLS with our readers.

VSAM Record-Level Sharing (RLS) is a more modern buffering and serialization technique for achieving higher availability for applications that use VSAM files. This article is the first in a series of articles about VSAM RLS, written by Roy and his colleagues in H&W, that examines the VSAM RLS foundation in native VSAM, and how the evolution of CICS from a single region to a multi-region, multi-system configuration, affected the environment. The article then discusses the advent of VSAM RLS and the needs it addresses, as well as noting considerations for its use.
A Brief History of Native VSAM

Unsurprisingly, VSAM RLS traces its origins to VSAM, which is a file management system for IBM mainframe platforms that IBM introduced in the early 1970s as part of OS/VS1. VSAM enables programs to store and access records that are organized into files, or data sets, categorized into one of these four main types:

- **Key Sequenced Data Set (KSDS).** A KSDS consists of a data component and an index component. Records in a KSDS have a primary key that ensures uniqueness of data records. A task can access the records directly, using the key, or sequentially. It is also possible to create alternate indices to access the data set using other fields in the record as secondary keys.

- **Entry Sequenced Data Set (ESDS).** With an ESDS, records are stored and accessed in sequential sequence, or in the order they were written to the data set. Finding and accessing a particular record normally involves scanning the data set from the start.

- **Relative Record Data Set (RRDS).** Records using the RRDS structure are accessed according to their relative position in the file. For example, a task could read record number 32,000 directly by specifying its logical position. While KSDS data sets have keys, RRDS files do not.

- **Linear Data Set (LDS).** An LDS is a special, unstructured, VSAM file that does not use the standard VSAM access technique. Systems like DB2, that want to perform their own record management within a control interval (CI), use LDS data sets.

VSAM RLS can work with any of these types of data sets except LDS. Any of these data sets can be defined as extended format data sets. Extended format data sets may have one or more optional attributes: data striping, data compression, and extended addressability. Only extended format data sets are eligible for partial space release, candidate volume space, and system-managed buffering (SMB).

serialization and Buffering With Native VSAM

Native VSAM has its own serialization techniques for concurrent access and buffering techniques for performance. These techniques traditionally served native VSAM well. To understand VSAM serialization and buffering, you first need to know that records are organized inside VSAM data sets in fixed-sized blocks called control intervals (CIs). You specify the CI size on the definition when you create the data set. Each CI in a VSAM data set typically will hold multiple records. Data is written to and read from VSAM files in units of CIs.

Buffering

The concept of buffering is really a method of reducing the I/O costs associated with data set access. It is much more efficient to retrieve a record from an in-memory buffer than to perform an I/O request to retrieve the record from disk. Buffer allocation is
probably the single most important factor that influences VSAM performance. Generally speaking, the larger the number of buffers, the more CIs can be kept in storage, thereby delivering better performance.

Traditionally, buffer allocation was controlled by the user, who specified the number of buffers as part of the cluster definition. Today, you can optionally select to have the system manage VSAM data set buffers dynamically. System-managed buffering is only available for extended format VSAM files, but should be investigated for any performance-critical VSAM data sets used by batch jobs. Some users report elapsed time reductions of up to 80% for system-managed buffering compared to the default buffer values. For more information on system-managed buffering, refer to the section titled 'Determining I/O Buffer Space for Non-Shared Resources' in the IBM Manual SC23-6855, z/OS DFSMS Using Data Sets.

There are also a number of vendor products that dynamically manage VSAM buffer allocation based on file access characteristics. Some of these are:

- CA / Hyper-Buf from CA
- Performance Essentials from Rocket Software / Mainstar
- Veloci-Raptor from Dino Software
- Mainview Batch Optimizer from BMC

Grouping VSAM records into CIs means that the amount of time that the system has to go back and forth to get data is shortened, thereby improving performance. The CI size, which is typically around 4K, determines the amount of data that is transferred from the disk into the VSAM buffers that get allocated when the data set is opened. Depending on the type of processing you're doing, you can choose one of three main types of buffering for VSAM data sets:

- **Non-Shared Resources (NSR).** NSR is the default buffering technique that VSAM uses. When using NSR, buffers are located in the address space’s private area and are not shared among VSAM data sets. This technique is best suited to sequential or skip sequential processing, and it is typically the mode that batch programs use.

- **Local Shared Resources (LSR).** LSR allows tasks in the same address space to share VSAM buffers among VSAM data sets, enabling a more centralized method of managing buffer pools. CICS uses LSR because the many tasks that are concurrently accessing the data sets that CICS manages can share a single, larger, buffer pool, resulting in more effective use of the buffers. LSR is best suited for direct-access processing, and it is typically the mode that CICS tasks use.

- **Global Shared Resources (GSR).** GSR is similar to LSR but is somewhat more restrictive because it can enable VSAM buffer pools to be shared among multiple address spaces. This mode is not widely used, and CICS does not support it. The only user of GSR that I am aware of is IMS.
For batch, you could also implement Batch Local Shared Resource (BLSR) buffering to improve VSAM buffer management. The more modern logical replacement for BLSR is system-managed buffering (SMB).

These modes are not intrinsic to the data set, but instead refer to the way a task can open them. Therefore, a task can open a data set in one mode at one time and that same data set in a different mode at a different time.

For most batch programs where buffers are not being shared, and for online access where a single CICS File Owning Region (FOR) manages concurrent access, these buffering techniques work well. However, to address customers' need for higher availability and cross-system sharing a new mode – RLS mode – was introduced. This is the mode in which multiple tasks, typically spread across multiple systems, can read and write VSAM data sets concurrently. With the introduction of RLS mode, NSR, LSR, and GSR became known as non-RLS modes.

**STRINGS**

STRINGS can be thought of as a method of handling I/O requests against a VSAM data set when the I/O request needs to track the current record position. VSAM uses a control block called a Placeholder (PLH) for this purpose.

When you define a VSAM data set, you specify the maximum number of such concurrent I/O requests using the STRING attribute. The number of strings therefore can control the number of concurrent processes that can access the file. A too-small number of strings can cause a bottleneck for heavily accessed files.

The STRINGS value only applies to non-RLS mode access.

**Serialization**

Native VSAM serializes at the CI level, so a task needs exclusive control over the CI if it wants to make a change to any record in that CI. Therefore, multiple tasks attempting to access data within the same CI can cause contention issues. To prevent these contention issues, you could apply certain techniques to each task so they collectively can have serialized access to the VSAM data sets. However, this is an inefficient and error-prone process that relies on every single task having to implement and strictly adhere to the serialization scheme.

The VSAM data set SHAREOPTIONS parameter (SHAREOPT) can eliminate the need to serialize every task, depending on the setting you choose. This parameter (specified when the data set is created) sets the level of serialization at the VSAM data set level. You can choose one of four SHAREOPTIONS levels of access to a data set, with the numerically lowest level being the most restrictive. A brief summary of SHAREOPTIONS settings and their effects follows.
SHAREOPT (xregion,xsystem)

- With xregion set to 1, VSAM allows a single task to open a data set for read/write processing (also known as r/w, or update access), or it allows multiple tasks to open a data set for read only. This option provides for high data integrity and performance, but it severely limits the amount of concurrency for the data set.

- With xregion set to 2, VSAM ensures that a single task can open a data set for r/w and multiple tasks can concurrently open the data set for read only. While VSAM ensures write integrity for the task that has the data set open for r/w, those tasks that open the data set for read only have no read integrity as it is possible for them to read out of date information. This option provides for high data integrity for the updating task and good performance, and it is by far the most widely used option.

- With xregion set to 3, VSAM allows many tasks to concurrently open a data set for r/w. Tasks are responsible for their own serialization and integrity handling; otherwise, data corruption could ensue. This option provides for flexibility, but performance is slower, and it could result in one task overwriting the changes of another task unless you are meticulous about each and every task adhering to the same serialization technique.

- Like xregion 3, xregion set to 4 allows multiple tasks to open a data set for r/w. As with xregion 3, tasks are responsible for their own serialization and integrity handling, or data corruption could ensue. The difference between xregion option 4 and 3 is that VSAM refreshes the buffer pools associated with the data set when a task changes a record if you use xregion 4. This ensures that tasks running with their own buffers get the latest copy of data as soon as the first task changes it (although there are restrictions and limitations that you must be aware of). Again, this option provides for flexibility, but it could result in one task overwriting the changes of another task unless you are meticulous about applying the serialization to each and every task.

The cross-system option comes into play when you have multiple distinct processors or a sysplex environment where each processor has its own operating system. In this mode of sharing, VSAM permits only two SHR parameter specifications:

- SHR(xregion,3) or
- SHR(xregion,4)

For shareoptions '3 3' or '4 3', VSAM implements a process known as Control Block Update Facility (CBUF) as an attempt to provide some limited integrity assistance. When in use at these higher shareoptions levels, certain critical VSAM control blocks for shared VSAM data sets are maintained by VSAM Record Management in CSA. This allows accurate VSAM Control Block information for shared VSAM data sets to be available to all z/OS users accessing the shared resource, but only if a strict user-provided serialization process is implemented by each user (ENQ/DEQ and or RESERVE). So VSAM doesn’t really ensure data integrity, but it enables the user to provide integrity by implementing their own serialization techniques. This is an incredibly complex process. More detail can be found in the CBUF section of the Sharing.
VSAM Data Sets chapter in the IBM Manual SC23-6855, z/OS DFSMS Using Data Sets.

In summary, the higher the numerical value of the SHAREOPTIONS xregion, the more responsibility the task has to ensure data integrity.

SHAREOPTIONS settings originally controlled the level of access that concurrent batch jobs had to VSAM data sets. After IBM introduced CICS, these options applied to CICS tasks as well, with the change being that batch jobs and CICS tasks could not access the VSAM files for r/w at the same time. Doing so would have required all batch and CICS tasks to have the same serialization mechanism applied to them, which isn’t possible, and could cause performance issues. This situation led to the introduction of the batch window, which meant that CICS online tasks had control of the VSAM files during the traditional workday. At night, r/w access for the CICS tasks was halted while batch tasks had control of the VSAM files for r/w.

As CICS evolved, new buffering and serialization techniques like VSAM RLS became necessary. Because access is controlled by VSAM RLS, the shareoptions values are irrelevant when the data set is being accessed in RLS mode. However they are still important if you have batch or other work that continues to access the files in non-RLS mode.

---

CICS and Its Interaction With VSAM

To understand VSAM file management fully, it’s important to be familiar with the basic evolution of CICS and its interactions with VSAM files.

Originally, a single CICS region fulfilled multiple roles. The region could be a terminal-owning region (TOR), which handled all communication to and from the outside world. That same region could also be the application-owning region (AOR), which ran all of the CICS applications (tasks). And, that region would also be the file-owning region (FOR), owning all of the VSAM files the CICS tasks needed to access. This configuration is shown in Figure 1.
As systems grew larger, companies commonly began to split workloads across multiple CICS regions, and those related regions would communicate with each other through a process known as ‘function shipping’. Function shipped allowed multiple CICS regions to send their read and write requests for a VSAM file to the FOR that owned that file. This effectively enabled all the CICS regions to access the FOR-owned data as if it was their own.

Because CICS owned and used all of the VSAM data sets during the workday, it provided the necessary serialization for concurrency and locking for data integrity when the SHAREOPTS parameter was set to (1,3) or (2,3). This meant you didn’t have to serialize at the task level. It also meant that the FOR owned the data set while still allowing other CICS tasks to access the data concurrently. Multiple CICS tasks could access the data set for read only or even r/w by function shipping their requests to the FOR. (Batch tasks could still access the data sets directly, but for read only.) This is why CICS became, and continues to be, the transaction manager of choice for many mainframe sites with large amounts of data in VSAM.

CICS provides other means of enabling concurrency. As mentioned earlier, native VSAM serializes at the CI level when updating records in a VSAM file, so a task needs exclusive control over the CI to update it. Therefore, multiple tasks attempting to access data within the same CI can cause contention issues. To compensate for this, CICS mimics record-level sharing within the scope of a single CICS region. This is controlled by the CILOCK initialization parameter in each CICS region. When this parameter is set to the default of NO, CICS issues a lock for a certain record in a CI but allows other CICS tasks to update other records in that CI.
So, if CICS is providing record-level sharing at the local level, why would you need a solution at a more global level? The simple answer is that a single CICS FOR is a bottleneck because the FOR performs all r/w processing against a VSAM data set. In addition, the CICS FOR is a single point of failure. If there is a problem with the FOR or the system the FOR runs on, or if either is down for a planned outage, then all access to the data is lost until the problem is resolved. Providing record-level sharing on a global level prevents these issues, and that’s where VSAM RLS comes in.

How VSAM RLS Works

Introduced in DFSMS/MVS V1.3, VSAM RLS allows tasks running in several CICS regions to share VSAM files, at the record rather than the CI level, and with full r/w access.

Before VSAM RLS, a task such as a CICS FOR owned a VSAM data set and opened it for r/w. In environments running VSAM RLS, VSAM RLS uses a Coupling Facility connected to all the systems in the Parallel Sysplex to perform data-set-level locking, record locking, and data caching. Now, the VSAM RLS (SMSVSAM) address space, instead of the CICS FOR, owns the VSAM files, so multiple CICS regions on multiple systems in the sysplex can update the data concurrently with full integrity. This eliminates the CICS FOR as a potential bottleneck and single point of failure. Updates are not restricted to a single CICS region or to a single system. A typical VSAM RLS configuration is shown in Figure 2.

Figure 2 - Typical VSAM RLS Configuration
While VSAM RLS allows multiple applications in multiple CICS regions to share VSAM files with integrity, limitations still exist for allowing CICS tasks and batch tasks to share this data.

**Recoverability of RLS-Managed Data Sets**

Data integrity for data sets that VSAM RLS manages involves their recoverability status.

Within CICS, traditional VSAM data sets were treated as either recoverable or non-recoverable, based on the setting of the BACKOUT parameter when you defined the data set to CICS. Similarly, VSAM RLS can recognize data sets as recoverable or non-recoverable based on the setting the LOG parameter on the VSAM cluster definition. This parameter has three possible settings:

- **UNDO.** With UNDO, the data set is backward recoverable. This means that CICS can back out any inflight changes made by a failed unit of work (uncommitted changes). CICS backs out these changes using the records' before-image that it journaled using the System Logger. This is also known as transactional recovery.

- **ALL.** The data set is both backward and forward recoverable. When you choose ALL, CICS journals both the before and the after images of a record. Backward, or transactional recovery, functions as it does with the UNDO setting. However, ALL also allows products such as IBM's CICS VSAM RECOVERY (CICSVR) to rebuild data in the event of hardware or software failures involving the data set. This type of recovery is known as forward recovery, or data set recovery. Data set recovery is only possible when you are using another product such as CICSVR.

- **NONE.** This value identifies the data set as non-recoverable. CICS does not log any changes for the data set and is unable to provide transactional or data set recovery.

**Note:** If you set the recoverability for a data set that VSAM RLS manages using the LOG parameter, CICS will honor that setting and ignore the BACKOUT parameter.

**VSAM RLS and Data Integrity**

When VSAM RLS manages recoverable VSAM data sets, the SMSVSAM address space protects the integrity of the VSAM data it owns by placing sysplex-wide locks in its lock structure in the Coupling Facility (CF), and registering each system's interest in each piece of data in cache structures in the CF. SMSVSAM buffers can also be used by batch jobs to have read only access to recoverable files that VSAM RLS manages while CICS has r/w access to those same data sets.

Batch jobs can specify RLS mode access to a VSAM file by adding the RLS= parameter to the DD statement for the file. This parameter also lets you specify what level of integrity the job requires when reading the data set. There are 2 options:

- The first is CR, or Consistent Read, which can be used to ensure that batch
jobs read the most current information in the file.

- The second option is NRI, or No Read Integrity, which could allow a batch job to access out-of-date information.

If batch needs to update recoverable data sets that VSAM RLS manages, the situation changes. In order to open a recoverable data set in RLS mode, a task must register with RLS as being able to provide transactional recovery. Because native batch tasks aren’t capable of transactional recovery, they can’t open recoverable data sets for update in RLS mode, so the data sets have to revert to non-RLS mode. You must close (or quiesce) the recoverable data sets to VSAM RLS and CICS and then batch can open them natively for r/w in non-RLS mode. When batch finishes accessing the data sets for r/w, you re-open them as recoverable data sets to VSAM RLS and CICS for r/w.

If CICS needs to read the recoverable data sets that VSAM RLS manages during the batch window, you can set the SHAREOPTIONS to (2,3). CICS would then open the recoverable data sets in non-RLS mode for inquiry only, which you typically control by providing a separate read-only FCT entry.

**Data Integrity and Non-Recoverable VSAM Data Sets**

Data integrity with non-recoverable data sets that VSAM RLS manages becomes decidedly more complicated and difficult to ensure. With the exception of SHAREOPTIONS (2,x), VSAM RLS ignores the SHAREOPTIONS setting. This means that both CICS and batch can read and update non-recoverable data sets under VSAM RLS’s control, at the same time. In this situation, data integrity problems are likely. The main likelihood of a problem is if the batch job abends. These files are non-recoverable, so no BACKOUT occurs. Normally for a batch step that updates a VSAM file, the process would be to take a backup of the file prior to it being updated. In the event of an abend, the file would be restored from that backup. However, if the file is being updated concurrently by batch and CICS transactions, a restore of the file would regress the updates made by the CICS transactions subsequent to the backup.

If a CICS transaction or batch program updates multiple records and then fails before it successfully updates the last record, the unit of work is only partially committed. For example, consider a transaction that is going to transfer $100 from someone’s savings account to his or her checking account. A single transaction issues two updates. The first update adds $100 to the checking account and adjusts the account’s balance. The second update subtracts the $100 from the savings account and adjusts that account’s balance accordingly. If the data sets are defined as non-recoverable and the transaction fails before it finishes, the $100 might be added to the checking account, but the $100 might not be successfully subtracted from the savings account. Clearly, this would cause a big problem for a financial institution.

The potential for data integrity issues escalates when both CICS transactions and
batch jobs are trying to update non-recoverable VSAM files that VSAM RLS manages. Records are committed with every successful write, rewrite, or delete request. In the event of an abend, the data sets may be in a state that is less than ideal when your auditors come by. You have to take care to back out only changes that the batch task made. This situation is even more complex and prone to error if you have multiple CICS and batch tasks making changes in parallel. For batch tasks that you can rerun at the point of failure or without adverse effects to the data, this recovery dilemma may not be an issue. However, this is not typically the case.

So, recovery becomes your responsibility when batch and CICS share update access for non-recoverable data sets that VSAM RLS owns. If you choose to restore the data set with a point-in-time backup, what happens to the updates that occurred after the chosen point in time? The implications of sharing non-recoverable VSAM data sets for update between batch and CICS needs to be carefully considered.

VSAM RLS is a good first step, allowing multiple CICS tasks to share VSAM files with integrity. However, it still has significant limitations in relation to batch that magnify the need for full continual access to VSAM data. VSAM RLS has to work in conjunction with another solution in order to ensure data integrity in an environment where both CICS and batch need r/w access to VSAM files during the same times. The IBM product that addresses this need is Transactional VSAM Services (TVS). H&W offer a product called SYSB-II that addresses the same requirement. Other products may be available from other vendors.

### Identifying VSAM RLS Candidates

Because VSAM has been around for a long time, there are some functions that are supported for non-RLS access that are not widely used any more and therefore are not supported when accessing a data set in RLS mode. And there are functions that are not compatible with accessing VSAM data sets based on records, rather than on CIs. Therefore, there are some restrictions that you need to be aware of. The IBM Manual GC52-1388, z/OS DFSMStvs Administration Guide, lists the following restrictions:

- You cannot specify VSAM RLS access for a VSAM data set using the ISAM interface to VSAM.
- You cannot open individual components of a VSAM cluster for VSAM RLS access.
- You cannot specify a direct open of an alternate index for VSAM RLS access, but you can specify a VSAM RLS open of an alternate index path.
- VSAM RLS open does not implicitly go to the beginning of the data set. For sequential or skip-sequential processing, you must specify a POINT or GET DIR, NSP request to establish a position in the data set.
- VSAM RLS does not support a request that is issued while the caller is executing in any of the following modes: cross-memory mode, SRB mode, or under an FRR.

Additionally, note that VSAM RLS does not support the following options and capabilities:
Linear, KEYRANGE, IMBED, and temporary data sets
- Addressed access to KSDS data sets
- Control interval access
- User buffering (UBF)
- GETIX and PUTIX requests
- ACBSDS (system data set) specification
- Hiperbatch

For more advice about identifying applications that support VSAM RLS, refer to the section titled 'Determining applications that can use VSAM RLS' in IBM Manual SC23-6860, z/OS DFSMSdfp Storage Administration.

You might have some files or some jobs that will bump into these restrictions. However we believe that the majority of VSAM files in most installations should be able to exploit and benefit from VSAM RLS. Also, the following IBM products and functions support RLS:

- CICS (obviously)
- DFSMShsm (for its CDSs)
- SCLM (Source Control Library Manager) component of ISPF, providing the ability to share its accounting, cross reference, and audit and versioning databases across systems in a sysplex
- Info/Man
- IMS
- IBM File Manager (with APAR PI37043)

However, before you proceed, it would be prudent to speak to the suppliers of any applications you have purchased that use VSAM files. There are so many CICS/VSAM applications, it would not be possible for us or even for IBM to maintain a list of which ones do or not support VSAM RLS, so speak to your vendors. If they don't explicitly support VSAM RLS today, they might be willing to test their application with RLS or let you know about the experiences of their other customers.

Conclusion

Migrating to VSAM RLS can appear somewhat daunting for anyone that has many years of experience with CICS and VSAM. You have your own processes built up over the years, your own backup and recovery methods, and you are used to managing the co-existence between CICS and batch. And now there is this new environment that you need to learn and get familiar with.

However, RLS is worth the effort to get there. The elimination of the FOR as a single point of failure is critical for anyone with very high availability requirements. And the improvement in performance and the reduced overhead for anyone doing function
shipping to FORs on another system will be significant. Add to that the fact that RLS enablement is IBM's strategy for delivering catalog performance enhancements, and there is a compelling case for getting started with your migration to VSAM RLS. We hope that this series of articles will be valuable to you during that migration.

**RECOMMENDATION**: Unlike other database managers, VSAM RLS is included in z/OS at no additional charge. It can be implemented for all or a subset of your VSAM applications. Because it takes some getting used to, get started with it now so that you are positioned and ready to go when you application developers come looking for it.

Our next article will focus on preparing for RLS, including hardware and software requirements. It will also provide information on installation and customization steps necessary to enable VSAM RLS, as well as examining candidates for RLS exploitation, tolerance, and intolerance.

### References

VSAM RLS has been around for a long time now, so there are plenty of documents and experience to help you set it up and manage it. In fact, because it has been available for so long, some of the documents will seem quite old; however the information they contain is largely still valid.

- IBM CICS SupportPac CP13: *IBM CICS TS record level sharing (RLS) performance study*. This contains a program to format the SMF type 42 records, shows the performance impact of using RLS, and describes how you can measure the impact.
- IBM Manual - GC52-1388 - z/OS DFSMStvs Administration Guide
- IBM Manual - SC23-6855 - z/OS DFSMS Using Data Sets
- IBM Manual - SC23-6860 - z/OS DFSMSdfp Storage Administration Reference
- IBM Manual - SC23-6877 - z/OS DFSMStvs Planning and Operating Guide
- IBM Redbook - SG24-4765 - CICS and VSAM Record Level Sharing: Planning Guide
- IBM Redbook - SG24-4766 - CICS and VSAM Record Level Sharing: Implementation Guide
- IBM Redbook - SG24-4768 - CICS and VSAM Record Level Sharing: Recovery Considerations
- IBM Redbook - SG24-6108 - VSAM Demystified
- IBM Redbook - SG24-6971 - DFSMStvs Overview and Planning Guide
- IBM Redbook - SG24-6973 - DFSMStvs Presentation Guide
- SHARE in Anaheim 2011 - Session 8277 - CICS and VSAM RLS User Experience by Glenn Schneck
- SHARE in San Francisco 2013 - Session 12994 - VSAM Boot Camp, Part 1 of 3, An Introduction to VSAM by Michael Friske
- SHARE in San Francisco 2013 - Session 12998 - VSAM Boot Camp, Part 2 of 3, Using IDCAMS to Manage VSAM Data Sets by Michael Friske
- SHARE in San Francisco 2013 - Session 12999 - VSAM Boot Camp, Part 3 of 3,
Buffering, Record Level Sharing, and Performance Basics for VSAM Data Sets
by Michael Friske