



Improving VSAM Performance

The Basics of Good NSR
Buffering

Elliot Hamilton and Ron Ferguson

Good buffering works wonders – and can change your experience of VSAM from "The Very Slow Access Method" to "The Very Speedy Access Method." Find out how you can optimize NSR buffering to improve performance.

Mainstar
Software
Corporation

P.O. Box 4132
Bellevue, WA 98009

www.mainstar.com

info@mainstar.com

Background on Buffering

Mainstar's VSAM instructors have always taught that blazing-fast VSAM file performance can be achieved with good buffering. Unfortunately, this improved performance wasn't available automatically and required some user effort. This led to widespread use of ISV products – in some cases, very expensive ones – to automatically implement better buffering. Now, with the advent of System Managed Buffering (SMB), high performance can be achieved through standard z/OS system interfaces with virtually no application programmer effort and with no JCL changes. Before we discuss how System Managed Buffering works and how you can take advantage of it, it might be useful to understand the overall buffering picture for VSAM files.

NSR Buffering

NSR means that each VSAM file in a task will have its own dedicated buffers assigned within the program address space.

NSR – one of those terms that is about as non-descriptive as it can possibly be – stands for Non-Shared Resource buffering. Since it begins with 'non', it's a negative term, and unless you know what its opposite, LSR (Local Shared Resource buffering) is all about, there's no hope of understanding anything about NSR. Unfortunately, for much of the applications and system programming world, NSR remains as much a mystery today as it's been for all of VSAM's existence. (Why else would the IBM VSAM Redbook be entitled [VSAM Demystified](#), with a large part of it specifically devoted to buffering?).

In simple terms, NSR means that each VSAM file in a task will have its own dedicated buffers assigned within the program address space, and hence, will not share them with any other VSAM file that is open within the task. NSR is also the automatic default type of VSAM buffering logic, and for many years it was the only type of standard buffering available for high-level language (COBOL) access to VSAM files.

So, let's start with the basics of NSR to see how things work with this simplest type of VSAM buffering.

On the next page, Figure 1 shows a sample DEFINE CLUSTER command for a typical production KSDS. Figure 2 is a LISTCAT for this data set, and as the STATISTICS show, it is now loaded with records and considerable processing has already been performed on it.

```

DEFINE CLUSTER(NAME(TEST.KSDS) -
              RECSZ(450 450) -
              KEYS(10 0) -
              FREESPACE(0 0) -
              CYLINDERS(350 50) -
              VOLUME(*) -
              SPEED) -
DATA(CISZ(12288)) -
INDEX (CISZ(1024))
    
```

Figure 1: DEFINE CLUSTER

| | | | |
|-----------------------------|------------------------|------------------------|------------------------|
| DATA ----- TEST.KSDS.DATA | | | |
| ATTRIBUTES | | | |
| KEYLEN-----10 | AVGLRECL-----450 | BUFSPACE-----25600 | CISIZE-----12288 |
| RKP-----0 | MAXLRECL-----450 | EXCPEXIT----- (NULL) | CI/CA-----60 |
| SHROPTNS(2,3) SPEED | UNIQUE NOERASE | INDEXED NOWRITECHK | NOIMBED NOREPLICAT |
| UNORDERED NOREUSE | NONSPANNED | | |
| STATISTICS | | | |
| REC-TOTAL-----522181 | SPLITS-CI-----95 | EXCPS-----385755 | |
| REC-DELETED-----8 | SPLITS-CA-----3 | EXTENTS-----1 | |
| REC-INSERTED-----1192 | FREESPACE-%CI-----0 | SYSTEM-TIMESTAMP: | |
| REC-UPDATED-----118731 | FREESPACE-%CA-----0 | X'B74B82FA4D625B42' | |
| REC-RETRIEVED--8028177 | FREESPC-----19759104 | | |
| ALLOCATION | | | |
| SPACE-TYPE----CYLINDER | HI-A-RBA-----258048000 | | |
| SPACE-PRI-----350 | HI-U-RBA-----239616000 | HURBA | CASZ = CISIZE x CI/CA |
| SPACE-SEC-----50 | | | |
| VOLUME | | | |
| VOLSER-----VSAM01 | PHYREC-SIZE-----12288 | HI-A-RBA-----258048000 | EXTENT-NUMBER-----7 |
| DEVTYPE---X'3010200F' | PHYRECS/TRK-----4 | HI-U-RBA-----239616000 | EXTENT-TYPE-----X'00' |
| VOLFLAG-----PRIME | TRACKS/CA-----15 | | |
| EXTENTS: | | | |
| LOW-CCHH---X'01800000' | LOW-RBA-----0 | TRACKS-----5250 | |
| HIGH-CCHH---X'074E000E' | HIGH-RBA-----258047999 | | |
| INDEX ----- TEST.KSDS.INDEX | | | |
| ATTRIBUTES | | | |
| KEYLEN-----10 | AVGLRECL-----0 | BUFSPACE-----0 | CISIZE-----1024 |
| RKP-----0 | MAXLRECL-----1017 | EXCPEXIT----- (NULL) | CI/CA-----33 |
| SHROPTNS(2,3) RECOVERY | UNIQUE NOERASE | NOWRITECHK NOIMBED | NOREPLICAT UNORDERED |
| NOREUSE | | | |
| STATISTICS | | | |
| REC-TOTAL-----343 | SPLITS-CI-----3 | EXCPS-----78401 | INDEX: |
| REC-DELETED-----0 | SPLITS-CA-----1 | EXTENTS-----6 | LEVELS-----3 |
| REC-INSERTED-----0 | FREESPACE-%CI-----0 | SYSTEM-TIMESTAMP: | ENTRIES/SECT-----7 |
| REC-UPDATED-----103 | FREESPACE-%CA-----0 | X'B74B82FA4D625B42' | SEQ-SET-RBA-----0 |
| REC-RETRIEVED-----0 | FREESPC-----66560 | | HI-LEVEL-RBA----106496 |

Figure 2: IDCAMS LISTCAT – Freshly-Loaded KSDS Test File

By explicit specification, this particular KSDS has a data component CISZ of 12,288, with an index component CISZ of 1,024. Note the BUFFERSPACE field in the LISTCAT (Figure 2) – for the data component, it shows a value of 25,600, and in

the index component is a value of 0 (zero). Since the BUFFERSPACE keyword was not specified in the DEFINE CLUSTER command, the LISTCAT values were filled in by IDCAMS during DEFINE CLUSTER command processing.

Here are the rules:

- If the BUFFERSPACE keyword is coded on the DEFINE CLUSTER command, the specified value is plugged into the data component BUFSPACE field in the LISTCAT, regardless of where it was specified on the DEFINE command (either cluster or data component level – it's not allowed at the index level). The value in the index component BUFSPACE field always shows zero, as there is no way for it to be specified.
- If the BUFFERSPACE keyword is not coded on the DEFINE CLUSTER command, IDCAMS sets the value in the data component BUFSPACE field equal to the formula:

| | | |
|-----------------------|----------|---------------|
| Data CISZ x 2 | = | 24,576 |
| Index CISZ x 1 | = | 1,024 |
| Total | = | 25,600 |

And, again, the index component field is set to zero.

Here's what's going on.

At OPEN time for the data set, if the BUFFERSPACE value is not overridden by other means of specification (for example the AMP keyword in the JCL – discussed in the next topic), the data component value is allocated via GETMAIN, divided up into buffers, some assigned to data component access and others assigned to index component access. The general logic for dividing up the buffers is:

- If sequential access is specified by the program issuing OPEN, *maximum* buffers are allocated to the data component, and *minimum* buffers to the index component.
- If key direct processing is specified, *maximum* buffers are allocated to the index component, and *minimum* buffers to the data component.

In the case of our example data set, with 25,600 shown in the BUFSPACE field, this will result in two data buffers and one index buffer being allocated. Without going into more detail than necessary, this is totally inadequate. This default value *results in the worst possible performance that a VSAM data set can possibly have* and is the result of the overly conservative buffer management for small MVS systems that we had in the early 1970s (and never upgraded).

The solution is to never let this default value in the BUFSPACE field take effect.

Optimizing NSR Buffering

There's a JCL keyword called AMP ("Access Method Parameters" – reserved exclusively for VSAM files) and one of its best uses is to override the BUFSPACE value at execution time. AMP is coded on the DD statement like this:

```
//DDNAME DD DSN=cluster.name,DISP=SHR,  
// AMP='BUFND=x,BUFNI=y'
```

Where:

BUFND specifies the number of *buffers* (not bytes, as in BUFFERSPACE) to be allocated for the data component.

BUFNI specifies the number of buffers to be allocated for the index component.

The 'million-dollar' question is: what are the optimum values to specify, for any type of processing?

BUFNI for Key Direct Processing

Your goal is to set up enough buffers so that the entire index set (that is, all records above the index sequence set) can be held resident, to ensure that maximum lookasides will be successful as key direct searches are made through the index. (Note: there's a rule in NSR buffering set up by IBM, which is not something we can modify, that simply does not allow lookasides for any sequence set record other than the last one brought into storage. There is no benefit for more than a single buffer for the sequence set.)

To calculate BUFNI, use the following formula:

$$\text{TIREC} - (\text{HURBA} / \text{CASZ}) + 1$$

Where:

TIREC is the total number of index records.

HURBA is the data component Hi-Used RBA (taken from the ALLOCATION paragraph of the LISTCAT).

CASZ is the product of data component CISZ times CI/CA.

Here is the example from the LISTCAT (see Figure 2 on page 3):

$$\begin{aligned}
 &343 - (239616000 / (12288 \times 60)) + 1 \\
 &343 - (239616000 / 737280) + 1 \\
 &343 - 325 + 1 \\
 &18 + 1 \\
 &19
 \end{aligned}$$

Based on this calculation, the JCL specification should be BUFNI=19. Since the index CISZ is 1024, this will cost 19K from the address space, as each index record in this data set is the size of one index CI, and will exactly fit in one buffer of that size.

The result will be that once each index record is brought in during a key direct search, it will remain in storage throughout further processing – and no further I/O will be required.

BUFND could be left at its default of 2, as it's not likely you'll achieve very many data component lookasides from key direct processing.

What happens if the KSDS expands beyond 18 index set records? (In the calculation, "TIREC - (HURBA / CASZ)" determines the size of the index set.) If the index expands due to additions in the data component, the suggested value of BUFNI=19 will be inadequate, resulting in more EXCPs than expected. Then again, will it be noticed? For this reason, even though BUFNI=19 is the "ideal" value for direct processing, adding a few additional buffers would be reasonable.

BUFNI for Sequential Processing

For sequential processing, the goal becomes optimizing data buffer 'read-ahead' logic, coupled with the concept of 'flip-flop' buffering management that VSAM provides.

Remember that, by default, two data buffers are allocated. One of these is set aside by VSAM for 'work area' (split processing and CI pre-formatting are two examples of what this work area is used for). With this setup, each data CI to be read in, costs an I/O. *The result would be the worst possible performance for the data set.*

Theoretically, at the other extreme, one I/O can bring in an entire CA if you give VSAM enough buffers. For our example data set, that would require 60 buffers (because the file has 60 CIs per CA), of 12K each (720K). To do this would require 120 buffers, because flip-flop buffering initiates an asynchronous I/O to fill the other 'set' of buffers, while the application program is cruising through the first set (issuing read-next, read-next, and so on, sequential requests) – that's what is

meant by flip-flop buffering. While the 1.4M of storage required to do this wouldn't be a problem with today's systems, considerably fewer buffers yield almost the same results. It should be noted that applications will exhibit a decrease in execution time and a decrease in EXCP counts. However the CPU time may increase from more conservative buffer values!

Performance tests show that full track I/O is about the fastest, most cost-effective buffering that you can set up. To do this, use the formula:

$$(2 \times \text{CIs per track}) + 3$$

Where:

2 provides two sets of buffers for flip-flop buffer management.

CIs per track is taken from the LISTCAT PHYREC/TRK field.

3 provides at least one work area buffer – sometimes VSAM uses more than one work area, and this allows for that possibility.

Here is an example from our LISTCAT (see Figure 2 on page 3):

$$(2 \times 4) + 3$$

$$11$$

Based on this calculation, the JCL specification should be BUFND = 11. Since the data CISZ is 12,288, this will cost 132K from the address space.

BUFNI could be left at its default of 1, as VSAM will only allow one sequence set record in storage at a time, and since we're only using the sequence set for sequential processing, we only need one buffer.

A word of caution is needed concerning the "CIs per track" value in the above paragraphs. Most control interval sizes correspond one to one with their physical record sizes, therefore using the PHYREC/TRK value as the "CIs per track" value is correct. Certain size control intervals, such as 14336, are placed on two 7168 physical records and there are 7 of these physical records per track. Let's see: if each CI requires 2 physical records, and there are 7 physical records per track, that means there are 3.5 CIs per track! However, the formula is the same for buffers, "(2 x CIs per track) + 3". For the 14336 CI, it would require (2 x 3.5) + 3, which is BUFND=10.

Alternate Index Buffering

The previous considerations for base cluster processing hold true for alternate indexes (AIXs): more index buffers for direct processing or more buffers for sequential processing.

The problem with increasing buffers for AIXs is that, except for special situations, the AIX name is never coded in JCL, making it rather difficult to add an AMP=BUFNI=nnn value. Applications will have the base cluster and PATH names coded on the JCL, but not the AIX name.

This is one of the moments to take advantage of an AIX's "BUFSPACE" value. VSAM will apportion excess BUFSPACE according to the "more index buffers when direct" and "more data buffers when sequential" rule, which is the basis for all NSR buffering.

Invariably, AIX processing is direct access so it makes sense that additional index buffers will be available. Much like the calculation for direct processing of a cluster, the same arithmetic should be performed for the AIX. Rather than adding an AMP=BUFNI=nnn to the JCL, the effect of using that AMP value needs to be converted to "bytes required" and that value be used in an ALTER aix_data_component BUFSP(...) command.

To illustrate, assume that the LISTCAT from Figure 2 is really an alternate index (not a base cluster). Ideal direct processing was calculated to be 19 index buffers and by default, 2 data buffers.

If we told VSAM to use BUFNI=19 and BUFND=2 via the AMP keyword, VSAM would multiply the "index CI size times 19" and add that to the "data CI size times 2".

$$\begin{aligned} &(19 \times 1024) + (2 \times 12288) \\ & \quad (19456) + (24576) \\ & \quad \quad 44032 \end{aligned}$$

In other words, VSAM would request 44,032 bytes from z/OS, and divide that into 19 index buffers and 2 data buffers.

By altering the AIX's data component BUFSPACE with the IDCAMS "ALTER ... BUFFERSPACE (44032)" command, VSAM will give the excess bufferspace, (that is, the bufferspace not consumed by 2 data buffers and 1 index buffer) to the AIX's index component and we achieve minimum EXCPs on the index.

Conclusion

If you haven't implemented System Managed Buffering yet, you achieve remarkable performance improvements for batch processing by applying the recommendations in this article. If your data sets are currently in EXTENDED format, or will be converted to EXTENDED in the near future, you should be able to implement System Managed Buffering, which can help you achieve even better performance than available with NSR buffering. To learn more about improving VSAM performance, read the IBM Redbook entitled, *VSAM Demystified*. The PDF is available for download at www.redbooks.ibm.com.

Finally, if you're staying with NSR buffering, the Map functionality in Mainstar's Catalog RecoveryPlus (CR+) product automatically makes all of the NSR buffering calculations for you. All you have to do is plug the numbers into your JCL and you're ready to go. To see how CR+ MAP can help you improve performance, request a trial at www.mainstar.com or contact us at experts@mainstar.com to arrange a personal briefing.

Elliot Hamilton

Elliot Hamilton has been involved with IBM's MVT, VS1, SVS, XA, and numerous variations of MVS since the mid-1970s. He gained VSAM and catalog expertise through experience as a systems Programmer and as a System Engineer and Instructor for Amdahl Corporation. As one of the early instructors for Mainstar Software Corporation (formerly Software Information Services, Inc.), Elliot presented VSAM and ICF catalog classes around the world. When Mainstar's customers' needs changed, Elliot worked with the Mainstar Software Development staff on numerous projects, including Catalog RecoveryPlus (CR+) and VSAM Manager. Currently, Elliot provides Level 3 Support for CR+, using his expertise in identifying tricky and difficult problems within ICF catalogs to search out brilliant CR+ solutions.

Ron Ferguson

Mainstar's founder, Ron Ferguson, has a technical background in large-scale OS/390 systems. As a software instructor for 20+ years, he has presented over 600 courses on VSAM and ICF catalogs, and is recognized worldwide as an expert in these areas.

Mainstar is a registered trademark of Mainstar Software Corporation. Catalog RecoveryPlus is a trademark of Mainstar Software Corporation. Mainstar Software Corporation is a wholly owned subsidiary of Rocket Software, Inc. 000-0513-02 (10/10/06)