Paper 302

Optimizing the Processing of VSAM Data Sets With The SAS® System

Michael A. Raithel, WESTAT

ABSTRACT

Ever since its introduction in 1973, the Virtual Storage Access Method (VSAM) has been a popular data storage construct on MVS systems. VSAM is the cornerstone of online applications such as IMS and CICS, and is widely used in vendor packages and in-house-written batch applications. Its flexibility and automatic internal data management routines have long made it an access method of choice.

This paper focuses on how SAS programmers can optimize the processing of VSAM data sets by manipulating VSAM buffers. It presents specific techniques and methodologies for determining what kind, and how many VSAM buffers to allocate to a SAS program. If implemented correctly, these buffering methodologies will greatly reduce disk I/O's, reduce CPU time, and lead to better job turnaround time.

INTRODUCTION

VSAM buffering is a powerful tool SAS programmers can employ to improve the efficiency of programs that access VSAM data sets. This paper details the methodology for exploiting VSAM buffers in SAS programs for each of the three types of VSAM data sets.

The paper begins by reviewing the three types of VSAM data sets and their processing characteristics. It examines VSAM's default buffering and explains the general buffer strategy. It describes the basic tools for obtaining the information needed to determine how many buffers to use for processing a VSAM data set. Then, the buffering algorithms for each access of each type of VSAM data set are presented.

The paper concludes by presenting benchmarks made by running SAS programs accessing buffered and unbuffered VSAM data sets. The resulting reductions in disk I/O's and job CPU time are illustrated for all three of the VSAM data set types.

Readers should note that this paper is not intended to be a detailed VSAM tutorial. Simplifications of VSAM concepts and internal workings have been made to facilitate the discussion on how to manage the VSAM buffers. Readers interested in more in-depth VSAM theory should refer to the publications listed in the reference section at the end of the paper.

VSAM DATA SET TYPES

There are three types of VSAM data sets that SAS programmers may have to access: Keyed Sequential Data Sets (KSDS), Entry Sequential Data Sets (ESDS), and Relative Record Data Sets (RRDS). Each of these data set types differs in its internal organization. Consequently, the buffering strategy differs for each. It is important to know how each type of VSAM data set is organized to understand how its buffering technique works.

The data in each of the three VSAM data set types is stored in units that are know as "Control Intervals" (CI). A CI is the VSAM

term for a block of data. A CI may contain one or more VSAM records or it may contain free space. When a VSAM data set (of any type) is first created, the entire data set is automatically divided into CI's containing free space. (VSAM aficionados will know that the CI's are grouped together into Control Areas, CA's, but this is not important to our discussion). When records are loaded into the data set, VSAM inserts them into the CI's. The order in which they are inserted, and subsequently how they are accessed, depends on whether the data set is a KSDS, an ESDS, or an RRDS.

The KSDS is divided into two parts: An index component and a data component. The index component contains compressed keys and pointers to where the records are stored in the data component. The data component contains the entire record, including the key. The keys in a KSDS contain unique values, so no two records have the same key. The KSDS keeps records stored in key sequence order. New records are inserted in the proper key sequence in the data component.

There are three modes of accessing records in a KSDS: Sequential, Skip Sequential, and Direct. A SAS programmer must choose one of these access modes to process a VSAM KSDS. As illustrated later, there is a different buffering algorithm for sequential, skip sequential, and direct access of a KSDS.

Sequential Access is when the entire KSDS is read, record-by-record, in the proper key sequence. When this happens, VSAM reads each Index CI into memory, one-by-one, and uses the compressed keys and pointers to retrieve the records stored in the Data CI's. The only way a record can be retrieved is if every record with a key sequence less than it has already been retrieved.

Skip Sequential Access is when groups of records in a KSDS are read sequentially as long as they match a partial key value. When a record does not match the partial key value, the SAS program logic either halts processing, or obtains the next partial key value, as the programmer sees necessary. VSAM begins skip sequential accessing by reading the Index CI's until it finds the pointer to the first Data CI containing the record with the partial key value. Then it reads the Data CI into memory and gives the program the first record with the matching partial key (if it exists in the KSDS). Subsequent SAS program KSDS reads return the next records in sequence in the Data CI. This means that the first read in skip sequential access is a direct read, while subsequent reads with the same partial key are sequential.

Direct Access is when KSDS records are retrieved solely by a match on the entire value of the key. The SAS programmer supplies the key value and invokes VSAM to return the KSDS record with this key. When this happens, VSAM reads the Index CI's until it finds the one with the pointer to the Data CI containing the matching record. Then it reads the Data CI into memory and returns the record (if it exists) to the SAS program. The program processes this record and then obtains a new key value to use in accessing the KSDS. This continues until there are no more key values to be used in obtaining records from the KSDS.

The ESDS is composed of only one component; a data component. Records exist in the ESDS in the order in which they were inserted. There is no regard for a key and completely duplicate records can exist in the data set. The most common way of accessing an ESDS is sequentially, but it may be accessed by Relative Byte Address (RBA) or by direct access via an alternate index.

The RRDS is also composed of only a data component. The position of a record in an RRDS is used as the "key" for the record. That is, the key to accessing the fourth record

is four; the twenty-seventh, twenty-seven, and so on. Records are inserted and deleted by their Relative Record Number (RRN) and may contain completely duplicate data, if necessary. An RRDS can be read sequentially, skip sequentially, or by direct access via the record's RRN.

THE VSAM BUFFERS

When a VSAM data set is opened, buffers are automatically allocated, by MVS, to hold the CI's as they are read. These buffers reside in main memory and remain in effect for the entire execution of the program. Each buffer holds one CI. If the programmer does not specify the number of buffers, VSAM defaults to a specific number of fixed buffers. The default buffer allocation for a KSDS is one index and two data buffers. The ESDS and the RRDS default to two data buffers. Though default allocations may be all that is needed for some applications, they are usually not sufficient.

The number that is sufficient for minimizing processing overhead depends upon the way the data is being accessed. When VSAM data sets are accessed sequentially, VSAM automatically uses a read-ahead function to fill its data buffers with the next group of Data CI's. This is done in a single I/O operation. If the default of two Data CI's is used, one I/O operation is performed for each Data CI. If the programmer specifies more than the default (i.e. Twenty data buffers), then many Data CI's are read into memory with each VSAM data I/O operation. Since the chances of wanting the records in the next Data CI are very high in true sequential processing, this results in a great reduction in disk I/O's.

Conversely, direct access of a VSAM data set does not need more than the VSAM default of two data buffers. Programs using direct access to process an ESDS or an RRDS do not need to specify index buffers. Programs processing a KSDS need to maximize the number of index buffers held in main memory. This is done because every direct access of a KSDS causes Index CI's to be read into memory and searched until the one containing the programmer supplied key value is located. (Then an I/O is issued to move the Data CI containing the sought after record into a data buffer in memory). The default of one index buffer causes an I/O for every Index CI read into memory during the search. When a programmer allocates multiple index buffers, multiple Index CI's remain resident in main memory. If the Index CI's needed for a search exist in the index buffers already in memory, then no Index CI I/O's will be necessary. Thus, the default of one KSDS index buffer is usually not sufficient for direct access.

From this discussion, it should be clear that there is a definite VSAM buffering strategy. For sequential access of KSDS, ESDS, and RRDS, the strategy is to maximize the data buffers. For direct access of all three, the default data buffers are sufficient, but the KSDS index buffers should be maximized. When skip-sequential processing is to be done for an ESDS or an RRDS, the data buffers should be maximized. For the KSDS being accessed by skip-sequential processing, both the data and the index buffers should be maximized.

GATHERING BUFFER INFORMATION

SAS programmers can override the default VSAM allocations via the BUFND and BUFNI keywords. BUFND declares the number of VSAM data buffers that should be allocated; BUFNI, the number of VSAM index buffers. These keywords are coded in the SAS program on the INFILE or FILE statement of the VSAM data set that is to be processed. (See **SPECIFYING VSAM BUFFERS**, below). The values a SAS programmer codes depend upon the type of VSAM data set being accessed, and the way the programmer intends to process the data

set. The next section will describe the algorithms for setting BUFND and BUFNI while this section will explain where to obtain the information needed in those algorithms.''

All of the information that a programmer needs for determining what the buffers should be set to for a particular VSAM data set are contained in a "LISTCAT" of the VSAM data set. A LISTCAT is a VSAM catalog listing containing the complete information about a VSAM data set. A LISTCAT can be obtained by submitting a batch job or by entering the LISTCAT command in TSO. (See examples, below). Unfortunately, a LISTCAT listing is too large to have one included as an illustration to this paper. The reader is urged to use either of the two methods shown in the examples to generate one for examination.

LISTCAT VIA BATCH JOB

```
//STEP01      EXEC PGM=IDCAMS
//SYSPRING DD SYSOUT=*
//SYSIN       DD *
 LISTC ENT(vsam.data.set.name) ALL
//
```

LISTCAT VIA TSO

TSO LISTC ENT('*vsam.data.set.name*') ALL

Though the LISTCAT listing contains a plethora of information, only five fields pertain to setting buffer values. The five fields are: the data CISIZE, the data CI/CA, the data HI-USED-RBA, the index REC-TOTAL, and the PHYRECS/TRK. If the LISTCAT is for an ESDS or an RRDS there will not be an index portion to the LISTCAT. In that case, the only fields that would be of interest would be the data CISIZE and the PHYRECS/TRK fields. These fields are referenced in the algorithms in the next section.

Obtaining accurate information from the LISTCAT is crucial to properly setting the buffers. If you are uncertain as to how to read the LISTCAT listing, check with a DASD Administrator or a Systems Programmer.

THE BUFFERING METHODOLOGY

Now that the background has been set, and the sources of information explained, the buffering methodology can be detailed. As covered earlier, the number of data buffers need to be maximized in sequential and skip-sequential processing for all three VSAM data types. In direct processing, the number of index buffers should be maximized for the KSDS. There is no need to maximize data buffers for the KSDS, ESDS, and RRDS in direct access processing.

Once the SAS programmer has identified the type of VSAM data set that will be accessed, and the type of access that will take place, the following algorithms should be followed to set the values for BUFND and BUFNI:

KSDS SEQUENTIAL PROCESSING

BUFND = (2 * (Data PHYRECS/TRK)) + 1
BUFNI = DO NOT SPECIFY

KSDS DIRECT PROCESSING

BUFND = DO NOT SPECIFY

$$\text{BUFNI} = 1 + \text{Index REC-TOTAL} - \left(\frac{\text{Data HI-USED-RBA}}{\text{(Data CISIZE*Data CI/CA)}} \right)$$

KSDS SKIP SEQUENTIAL PROCESSING

BUFND = (2 * (Data PHYRECS/TRK)) + 1

$$\text{BUFNI} = 1 + \text{Index REC-TOTAL} - \left(\frac{\text{DATA HI-USED-RBA}}{\text{(Data CISIZE*Data CI/CA)}} \right)$$

ESDS AND RRDS SEQUENTIAL PROCESSING

BUFND = (2 * (PHYRECS/TRK)) + 1
BUFNI = DO NOT SPECIFY

ESDS AND RRDS DIRECT PROCESSING

BUFND = DO NOT SPECIFY
BUFNI = DO NOT SPECIFY

SPECIFYING VSAM BUFFERS

Once you have determined how many and what type of buffers that you need to utilize, it is fairly easy to specify them in your SAS program. BUFND and BUFNI are specified on the INFILE or FILE statement used to identify the VSAM data set you are processing. Below, you will find three examples of specifying buffers on the INFILE statement.

Here is an example of specifying the buffers for direct access of a VSAM KSDS:

```
Infile vsmfile1 vsam key=keyvar
        bufnd=25 bufni=2;
```

Here is an example of specifying buffers for sequential access of a VSAM ESDS:

```
Infile vsmfile2 vsam bufnd=25;
```

Here is an example of specifying buffers for skip sequential access of a VSAM RRDS:

```
Infile vsmfile3 vsam rrn=relrecno skip
        feedback=fback bufnd=25;
```

Note that in the examples above, BUFNI is not used for access of VSAM ESDS and RRDS data sets since they do not contain indexes. Also, the examples illustrate buffering on INFILE statements, but the same method of coding the buffers holds true for FILE statements of VSAM data sets.

BUFFERING BENCHMARKS

Tables 1, 2, and 3, at the end of this paper, present the results of benchmarking the use of buffers in SAS program that process VSAM data sets. Refer to these tables in the following discussion.

In the benchmarks, the VSAM KSDS, ESDS, and RRDS each contain one million records. The records in each VSAM data set are 90 bytes long. The keys in the VSAM KSDS are 22 bytes long. The CISIZE of all three VSAM data sets is 4096 bytes.

Table 1 illustrates the differences in CPU time and EXCP Count (I/O's) from unbuffered and buffered sequential processing of VSAM KSDS, ESDS and RRDS data sets. The difference in CPU time is modest, ranging from 14 to 16 percent. Those modest savings may be significant in organizations with IS Chargeback systems that charge for CPU time. The difference in EXCP Count is stunning. For all three data sets there was a reduction in EXCP Count of 90%. This is truly a computer resource savings that is worthwhile!

Table 2 compares buffered and unbuffered direct access of a VSAM KSDS. Four different key files, containing 10, 25, 50, and 75 percent of the keys to records in the KSDS are used. This is done to simulate light to heavy random accesses of a KSDS. The table shows that the CPU savings range from 16%, to 24%, to 29%. The EXCP Count savings are again dramatic: they are reduced by 33%. Clearly, using BUFNI on the INFILE statement results in significant resource savings for direct access processing of a VSAM KSDS.

A study of these two tables reveals a subtle point: it is far more resource intensive to randomly access a VSAM KSDS data set than it is to read it sequentially. (It is also more resource intensive to access a KSDS randomly than to access it skip sequentially,

though that is not covered in this paper). A SAS programmer who has a file of keys to records that must be extracted from a KSDS should consider coding skip sequential reads or reading the entire KSDS instead of direct reads. This makes more and more sense as the number of records in the key file approaches the number of records in the VSAM data set.

Table 3 contrasts initially loading VSAM data sets with a VSAM REPRO command against loading VSAM data sets with the SAS System in a DATA step. In every instance--in the REPRO's and in the SAS loads--the VSAM data sets were fully buffered. For all three types of VSAM data sets, there is a significant reduction in CPU time from using a VSAM REPRO for the initial load. For a KSDS and an ESDS there is no significant difference in EXCP Count. However, there is an 88% reduction in EXCP Count in loading a VSAM RRDS with a SAS DATA step. Whether the 88% reduction in EXCP Count is more important than a 48% rise in CPU time is a site dependent decision. Consequently, for initially loading VSAM KSDS and ESDS data sets, use the VSAM REPRO command rather than a SAS DATA step; it will save you CPU time. For an initial load of a VSAM RRDS data set, determine whether you want to save CPU time or EXCP's and use the appropriate load software.

CONCLUSION

SAS programmers can significantly improve the efficiency of SAS programs that access VSAM data sets by exploiting the VSAM buffers. By determining the values for BUFND and BUFNI and coding them on the VSAM data set's INFILE or FILE statement, they can cut CPU time and EXCP's, and improve their batch job turnaround time. All that is needed is a VSAM LISTCAT and the buffering algorithms. Armed with these, SAS programmers can drastically reduce the amount of computer resources needed to process the data stored in their organizations' VSAM data sets.

TRADEMARKS

REFERENCES

Raithel, Michael A.
Tuning SAS Applications in the MVS Environment
Cary, NC: SAS Institute Inc., 1995
303pp

SAS Institute, Inc.
SAS Guide to VSAM Processing, Version 5 Edition
Cary, NC: SAS Institute Inc., 1985
185 pp.

CONTACT INFORMATION

Michael A. Raithel
WESTAT
1650 Research Boulevard
Rockville, MD 20850-3129
E-mail: maraithel@erols.com

	CPU Time (Sec)	EXCP Count
KSDS Unbuffered	23.25	25,780
KSDS Buffered	19.52	2,690
KDSD Differences	**-16%**	**-90%**
ESDS Unbuffered	22.20	22,829
ESDS Buffered	19.14	2,334
ESDS Differences	**-14%**	**-90%**
RRDS Unbuffered	22.38	23,466
RRDS Buffered	19.33	2,384
RRDS Differences	**-14%**	**-90%**

Table 1. Sequential read of a KSDS, ESDS, and RRDS, buffered and unbuffered. Each file contains one million records.

KSDS Direct Access	**CPU Time (Sec)**	***EXCP Count***
10 % Unbuffered	52.80	375,054
10% Buffered	40.07	249,321
Difference	**-24%**	**-33%**
25% Unbuffered	105.10	750,070
25% Buffered	79.80	498,553
Difference	**-24%**	**-33%**
50% Unbuffered	189.23	1,500,116
50% Buffered	159.11	997,188
Difference	**-16%**	**-33%**
75% Unbuffered	307.54	2,250,149
75% Buffered	218.48	1,495,986
Difference	**-29%**	**-33%**

Table 2. Direct access read of 10%, 25%, 50%, and 75% of a KSDS, buffered and unbuffered. The KSDS contains one million records.

	CPU Time (Sec)	EXCP Count
KSDS REPRO	15.60	7,791
KSDS SAS Load	36.47	7,846
Difference	**+134%**	**+.7%**
ESDS REPRO	14.4	5,391
ESDS SAS Load	36.83	5,447
Difference	**+156%**	**+1%**
RRDS REPRO	22.8	49,140
RRDS SAS Load	33.77	5,667
Difference	**+48%**	**-88%**

Table 3. VSAM REPRO versus a SAS Data Step load of a VSAM KSDS, ESDS, and RRDS. Each file was loaded with one million records.