BUFFERING THE SAS® SYSTEM VSAM PERFORMANCE HEADACHES

Michael A. Raithel, Marriott Corporation

ABSTRACT

The Virtual Storage Access Method (VSAM) has had a strong and popular career since it was introduced by IBM in 1973. It is the cornerstone of online applications such as IMS and CICS, and is widely used in vendor packages and batch applications. Its flexibility and internal data management have made it an access method of choice.

Those who choose to use VSAM can significantly improve their programs efficiency by properly managing the VSAM buffers. There is a specific methodology for determining what kind, and how many VSAM buffers to allocate to a program. If used correctly, this buffering methodology will greatly reduce program CPU time and disk I/O's and lead to better job turnaround time.

The SAS System allows programmers to load, read, and update VSAM files. SAS programmers can implement the VSAM buffering methodology via the BUFND and BUFNI keywords on the VSAM file's INFILE statement. The SAS Guide To VSAM Processing states: "The systems programming staff at your installation can help you determine what values to assign..." to BUFND and BUFNI. The author of this paper, a systems programmer with years of VSAM tuning experience, will discuss how to determine the optimal values for BUFND and BUFNI.

INTRODUCTION

VSAM buffering is a powerful tool SAS programmers can utilize to improve the efficiency of programs accessing VSAM files. This paper details the methodology for exploiting VSAM buffers in SAS programs for each of the three types of VSAM files.

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 gaining 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 in graphs for all three of the VSAM data set types.

This paper is not intended to be a detailed VSAM tutorial. Simplications 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 References section at the end of the paper.

VSAM DATA SET TYPES

There are three types of VSAM data sets the SAS programmer may have to access: The Key Sequenced Data Set (KSDS), the Entry Sequenced Data Set (ESDS), and the Relative Record Data Set (RRDS). Each of these data set types differs in its internal organization. Thus, 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 data set types is stored in what is known 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 is first defined, the entire data set is automatically divided into CI's containing free space. (The CI's are grouped together into Control Areas, which are not important to this discussion). When records are loaded into the file, 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 methods of accessing records in a KSDS: Sequential, Skip Sequential, and Direct. A SAS programmer must choose one of these access methods 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 sequenced 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. 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. 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 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 depend upon the way the data is being accessed. When VSAM files 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 area 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 file 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 is clear 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 THE BUFFER INFORMATION

The SAS programmer can override the default VSAM allocations via the BUFND and BUFNI keywords. These keywords are coded in the SAS program on the INFILE statement of the VSAM file that is to be accessed. (See FIGURE 1, below). The values a SAS programmer codes depend upon the type of VSAM file being accessed, and the way the programmer intends to process the file. 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.

```
INFILE VSAMFILE VSAM KEY=SORTKEY GENKEY SKIP
BUFND=21 BUFNI=2;
```

Figure 1

There are two tools a SAS programmer needs to determine what the buffers should be set to for a particular VSAM data set. The first is a LISTCAT of the VSAM data set that is to be accessed. The second is knowledge about the track characteristics of the Direct Access Storage Device (DASD) device the VSAM data set resides on.

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 (FIGURE 2). Unfortunately, a LISTCAT listing is too large to have one included as an illustration in this paper. The reader is urged to use either of the two methods shown in FIGURE 2 to generate one for examination.

LISTCAT VIA A BATCH JOB:

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

LISTCAT COMMAND IN TSO:

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

Figure 2

Though the LISTCAT listing contains a plethora of information, only four fields pertain to setting buffer values. The four fields are: the data CISIZE, the data CI/CA, the data HI-USED-RBA and the index REC-TOTAL. 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 field that would be of interest would be the data CISIZE field. These fields are referenced in the algorithms in the next section.

Once the data CISIZE has been obtained from the LISTCAT listing, the SAS programmer needs to know about the storage characteristics of the DASD containing the VSAM data set. Particularly, the number of data CI's per track is needed. This can be obtained by taking the data CISIZE from the LISTCAT and looking up the number of blocks of that size that will fit on a track for the particular DASD being used. FIGURE 3 has the common data CISIZE's listed with the number of CI's per track for IBM 3350 and 3380 devices.

Obtaining accurate information from the LISTCAT, and about the disk device is crucial to properly setting the buffers. If there is uncertainty as to device type or how to read the LISTCAT listing, check with a DASD Administrator or a Systems Programmer.

NUMBER OF DATA CI's/TRACK

CISIZE	3380	3350
512	46	27
1024	31	15
2048	18	8
4096	10	4
8192	5	2

Figure 3

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, or 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 * (NUMBER OF DATA CI'S PER TRACK)) + 1

BUFNI = DO NOT SPECIFY.

KSDS DIRECT PROCESSING

BUFND = DO NOT SPECIFY.

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

KSDS SKIP-SEQUENTIAL PROCESSING

BUFND = (2 * (NUMBER OF DATA CI'S PER TRACK)) + 1

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

ESDS AND RRDS SEQUENTIAL PROCESSING

BUFND = (2 * (NUMBER OF DATA CI'S PER TRACK)) + 1

BUFNI = DO NOT SPECIFY.

ESDS AND RRDS DIRECT PROCESSING

BUFND = DO NOT SPECIFY.

BUFNI = DO NOT SPECIFY.

BUFFERING BENCHMARKS

The graphs in this section compare running a simple SAS program against buffered and unbuffered VSAM files. The emphasis is on buffering the KSDS in all three of the ways it can be accessed; sequentially, direct and skip sequentially. ESDS and RRDS examples are also presented.

These benchmarks were run on an IBM 3090/200 processor under MVS/XA SP2.2.0. The VSAM data sets resided on IBM 3380 disk packs in an ICF catalog environment. The keys for the random and skip sequential processing of the KSDS were created by utilizing a random number generator to extract random key values from the VSAM file and store them in a 'key' file. Then several 'bare

bones' SAS programs were executed to use the key file to access the VSAM data sets with and without buffers.

None of the benchmarks were run as "pure research", but rather as illustrations that the VSAM buffering methodology could be utilized in SAS programs. Therefore, simplistic KSDS, ESDS, and RRDS of 35 cylinders, containing 14,604, 14,604 and 17,550 records, respectively, were used. Extrapolations can be made on the CPU time and DISK I/O savings from buffering SAS programs accessing far larger VSAM data sets.

Graph 1 contrasts a buffered sequential read of every record in a VSAM file with an unbuffered read of the same file. Employing the VSAM buffering methodology cut the CPU seconds by 12.5% and the disk I/O's by 45.5%.

Graph 2 compares buffered and unbuffered direct accesses of a KSDS. Three different key files, containing 25, 50 and 75 percent of the keys to records in the VSAM file were used. This was done to simulate light to heavy random accesses of a KSDS. The graph shows, the CPU savings varying from 18 to 32% and the I/O savings remaining constant at 48%. Clearly, using the BUFNI parameter on the SAS VSAM INFILE statement results in significant savings for direct access processing of a KSDS.

Graph 3 represents a buffered and unbuffered KSDS that is read in a skip-sequential manner. The CPU seconds for the buffered run actually rose by 3%, while the I/O's dropped by 33%. Care should be exercised in shops that employ CPU charge-back schemes, so that a rise in CPU charges does not nullify a drop in disk I/O charges.

A study of the three preceding charts reveals a subtle point: it is far more expensive to randomly access a KSDS VSAM file than it is to read it in sequential or skip sequential mode. A SAS programmer who has a file of keys to records that must be extracted from a KSDS, should consider coding skip-sequential reads instead of direct reads. This would make sense if the number of records in the key file approached the number of records in the VSAM file.

Graph 4 illustrates the skip-sequential versus direct access reasoning put to the test. The same key file that was used to perform random accesses on the KSDS was sorted. Then, with proper use of the BUFNI and BUFND keywords, the file was read skip-sequential using the keys as 'partial' keys. This resulted in a dramatic reduction in the disk I/O's, especially as the number of keys approached the number of records in the VSAM file. Clearly, it is far more efficient to sort a file of keys and read a KSDS skip-sequentially than to read the file randomly.

Though the SAS system can be used to initially load a VSAM data set, it should not be recommended. It is far more efficient to use the VSAM REPRO command to load the data set. Graph 5 compares the IDCAMS REPRO with a buffered SAS VSAM load program. The SAS program consumes about four times more CPU seconds and disk I/O's than the REPRO does.

Graphs 6 and 7 show the CPU time and DISK I/O savings from buffering sequential access of an ESDS and an RRDS. Both graphs indicate a low reduction in CPU seconds and a major gain in cutting the DISK I/O's by fifty percent or more.

Graph 8 illustrates the futility of buffering direct access of an ESDS and an RRDS, There were no real reductions in either CPU time or DISK I/O's when either data set was buffered.

CONCLUSIONS

SAS programmers can significantly improve the efficiency of SAS programs accessing VSAM files via the VSAM buffering methodology. By determining the values for BUFND and BUFNI and coding them on the VSAM file's INFILE statement, they can cut DISK I/O's and CPU time, and improve job turnaround time. All that is needed is a LISTCAT, DASD track information, and the buffering algorithms. Armed with these, the SAS programmer can begin buffering away VSAM performance headaches.

ACKNOWLEDGEMENTS

SAS is a registered Trademark of SAS Institute, Inc., Cary, NC, USA.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.

The author would like to thank Mary Krobath for her word processing acumen in revising this paper, and Carolyn Ostrowski for her skillful layout of the graphics.

REFERENCES

1. Keller, R. J. (1985) SAS Guide To VSAM Processing, Version 5 Edition, Cary, North Carolina: SAS Institute, Inc.

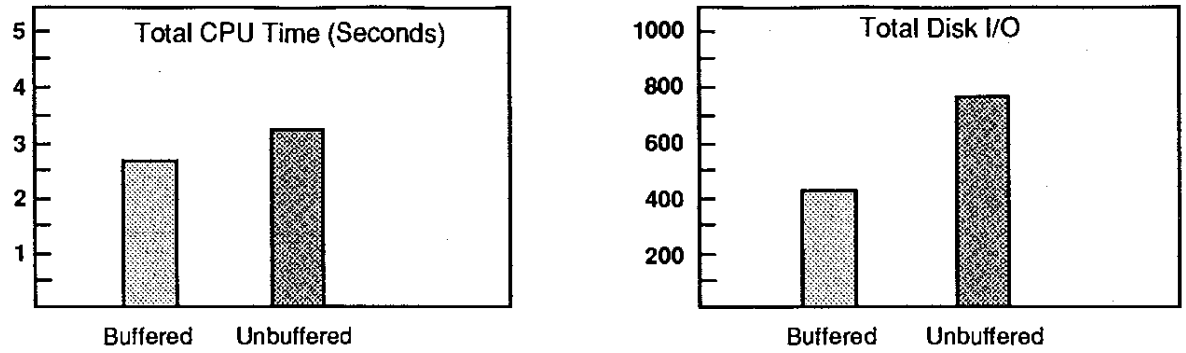
2. International Business Machines Corporation (1985), MVS/370 Integrated Catalog Administration: Access Method Services Reference, San Jose, California: International Business Machines Corporation

3. International Business Machines Corporation, (1985), MVS/370 VSAM Administration Guide, San Jose, California: International Business Machines Corporation.

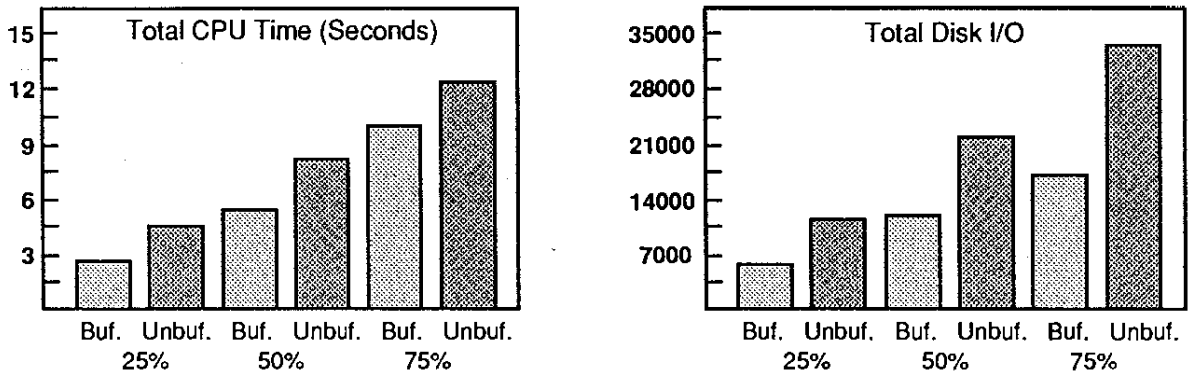
CONTACT INFORMATION

Michael A. Raithel
Marriott Corporation
1201 Seven Locks Road, Room 213
Rockville, Maryland 20854

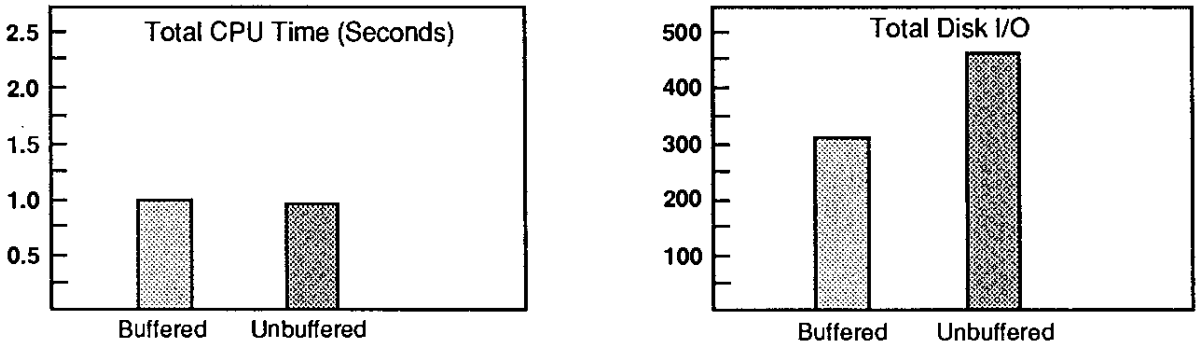
Telephone: (301) 738-1089



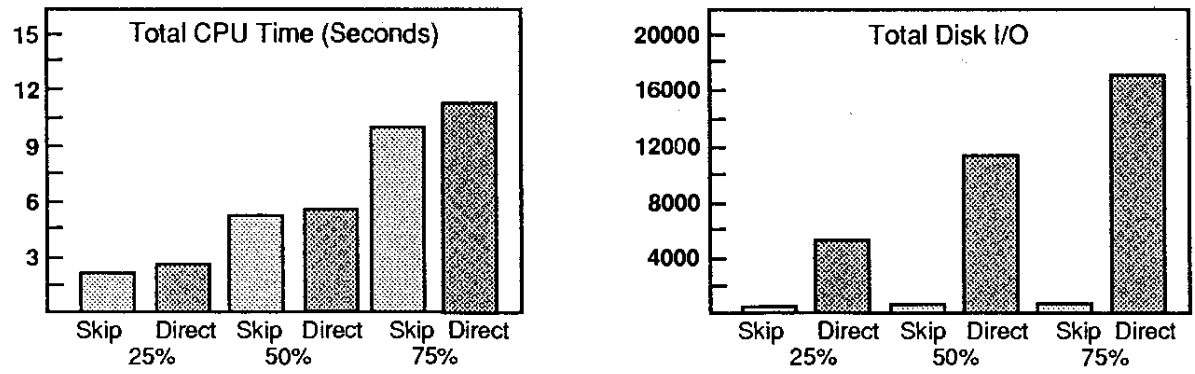
Graph 1. Sequential Access of a VSAM KSDS -- Buffered and Unbuffered



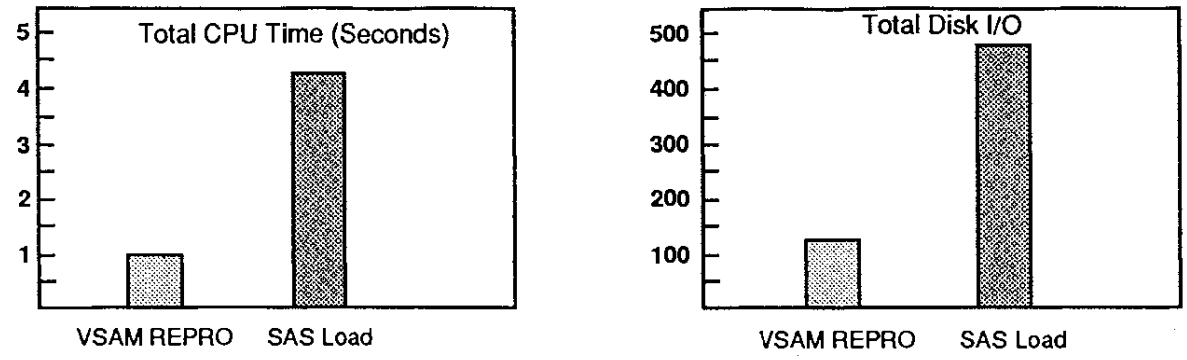
Graph 2. Direct Access of a VSAM KSDS -- Buffered and Unbuffered



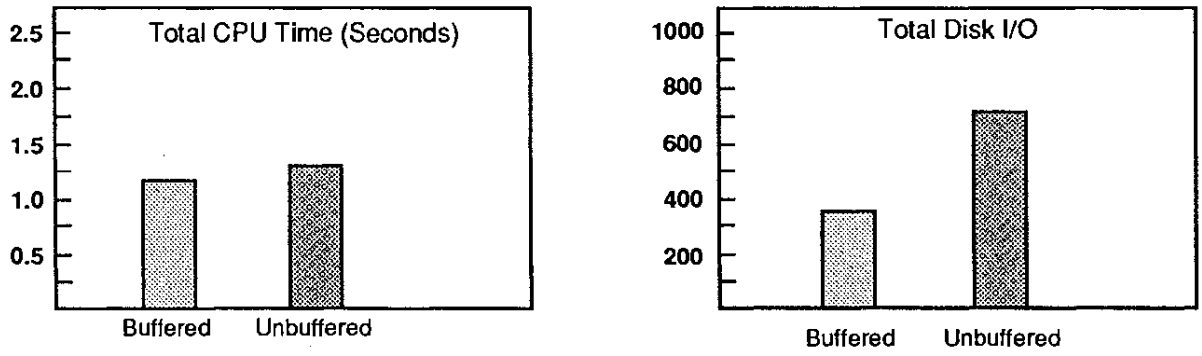
Graph 3. Skip-Sequential Access of a VSAM KSDS -- Buffered and Unbuffered



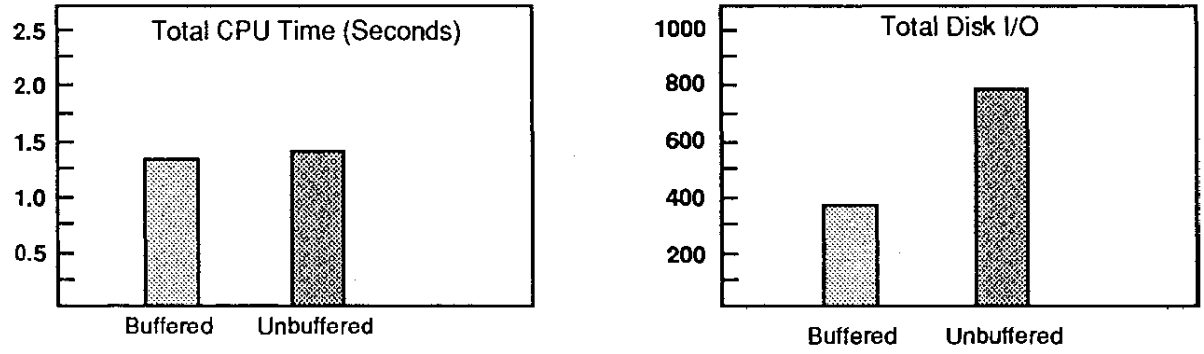
Graph 4. Skip-Sequential vs. Direct Access of a VSAM KSDS -- Both Fully Buffered



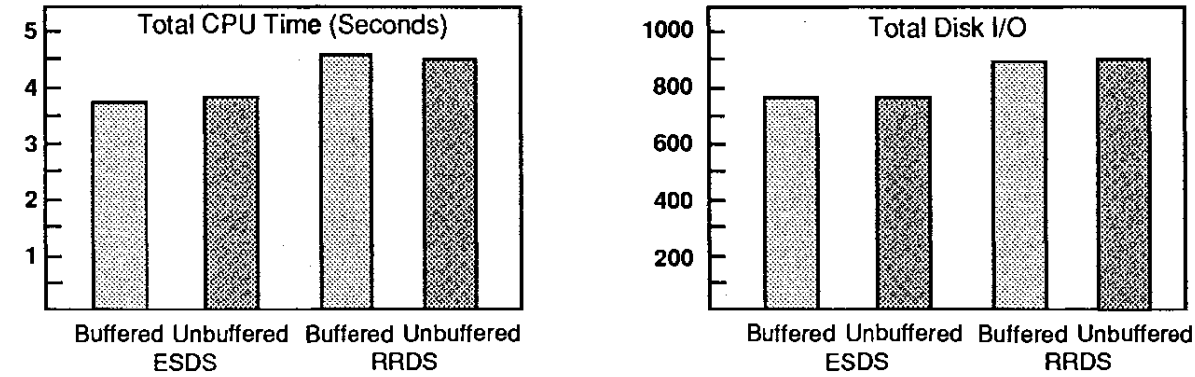
Graph 5. VSAM REPRO vs. SAS Load of a VSAM KSDS



Graph 6. Sequential Access of a VSAM ESDS -- Buffered and Unbuffered



Graph 7. Sequential Access of a VSAM RRDS -- Buffered and Unbuffered



Graph 8. Direct Access of VSAM ESDS AND RRDS -- Buffered and Unbuffered