Paper 327-2009

Improving SAS® I/O Throughput by Avoiding the Operating System File Cache

Leigh Ihnen and Mike Jones, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® I/O processing (the reading and writing of SAS data files – data sets, catalogs, index files, and so on) is done by default via the UNIX or Windows operating system's file cache. When accessing SAS data files in the multi-gigabyte range, overall system I/O throughput can be negatively affected. One method that might avoid such an I/O bottleneck is to avoid the operating system's file cache by using the SGIO SAS parameter (that was introduced with SAS® 8) or the new DIO SAS parameter (that has been added to SAS® 9.2 on several UNIX platforms).

This paper will discuss when the use of SGIO and DIO is appropriate and how to specify and tune the two capabilities. In addition, the paper will present examples of potential system I/O throughput improvement, which can be very significant.

INTRODUCTION

With the SAS 9.2 release for UNIX and Windows, the ability to specify that sequential I/O avoid the use of the system file cache via Direct I/O in UNIX (DIO) or Scatter Gather I/O in the Windows OS (SGIO) can increase overall system throughput. The intent of this paper is to

- provide background information
- contrast the usage of SGIO and DIO in a relational database environment to usage in a SAS environment
- provide a generic description of how I/O is performed with and without DIO or SGIO
- present advice on when to use DIO and SGIO
- present examples of potential performance gains

For a SAS workload that can exploit DIO or SGIO, significant gains in overall system throughput can be achieved. The format of this paper is a little unusual as it consists of two separate papers by different authors combined after a generic introduction. Information on UNIX DIO will be presented first followed by information on Windows SGIO.

SAS AND I/O

The SAS System is based upon a virtual architecture created to support a wide range of operating systems. In keeping with the goal of a portable SAS language, features that were prevalent across multiple operating systems were incorporated. Incorporation of features which were specific to any one operating system were limited to the features which provided the most benefit. Over time, changes to hardware environments and operating systems have created opportunities for incorporating new features into the SAS language--in particular the ability of UNIX and platforms based on Windows to effectively support hundreds of gigabytes to multiple terabytes of data. One of these opportunities concerns the APIs which the SAS System uses to perform I/O.

In a typical SAS environment, multiple concurrent SAS process will be executing in support of a user workload of batch and interactive jobs. Since the SAS environment consists of large numbers of independent processes, I/O performance of SAS jobs is dependent upon optimizations implemented in the OS and file system. To increase I/O performance, operating systems have implemented file caching strategies which, when dealing with moderate size data, can obtain significant performance improvements from read-ahead, write-behind, and caching strategies. Heuristic algorithms are used to predict application behavior and adapt these caching strategies.

How accurately the workload's application behavior is predicted will determine the increase in system throughput. As with every algorithm there are overhead costs with predicting and managing resources. Hence I/O through the file cache has an upper bound that can be lower than the bandwidth which the physical I/O subsystem can sustain. In a UNIX environment the ceiling appears to be in the 600 Megabytes per second (MBs) to 900 MBs range for current server models. So, for a relatively small but very resource-intensive collection of activities in a SAS environment, performing I/O through the system file cache can limit the overall system throughput. The typical examples involve sequentially accessing a file or files that are larger in size than the file cache, and are not accessed by more than one SAS process concurrently. In this case the file cache pages are not reused, which eliminates the primary benefit of using the file cache. If the system is not saturated, the read-ahead optimization will still provide single job performance improvements.

RELATIONAL DATABASES AND I/O

In the relational database environment all I/O performed by the database application is coordinated by a small number of processes for its large number of client connections. The relational database application *internally* implements caching, read-ahead and write-behind strategies to maximize I/O performance. In the database I/O environment this data caching occurs at the application level. In the UNIX world the general recommendation is to specify DIO as a mount point option to maximize I/O performance and avoid double caching and overly aggressive read-ahead behavior.

SAS AND SUPPORT FOR DIRECT I/O UNDER UNIX

Before the Version 9.2 release of SAS, the use of direct I/O (DIO) for SAS data sets was controlled at the file system configuration level. Either the file system was mounted with DIO enabled or the file system discovered that DIO was appropriate based upon the *bufsize* value of a SAS data set. The I/O buffer used by SAS needed to meet an alignment and size criterion (usually the restriction is that the buffer is a multiple of 4 K and aligned on a 4 K boundary). This level of control forced the physical separation of SAS data sets on different file systems based on size and access patterns rather than on the usual logical application grouping (e.g., collocation of tables that are tightly coupled in usage). Given the limitations on configuring a SAS environment to use DIO most user-implemented environments were not designed and set up to effectively use DIO.

The primary benefit of DIO is reduced CPU utilization as reads and writes are performed by the Virtual Memory Manager (VMM) directly to/from the application to the storage device instead of copying the data through the VMM file cache. An additional benefit of using DIO is reduced file cache contention depending upon the system-wide I/O access patterns. From an individual user's prospective DIO may not reduce the total elapsed time of a single SAS job. However at a holistic system level, total system throughput may be increased. By moving the jobs that benefit from DIO (ostensibly those that least effectively use the system file cache) out of the file cache, the overall load is reduced on the system file cache, aiding the remaining SAS jobs that make effective use of file cache.

FILE I/O

Normally I/O processing to files (e.g., reading from/writing to) goes to or from the SAS application buffer to the VMM which uses memory to cache the buffers in a file cache. In the case of a read request (e.g., reading a SAS data set) the VMM copies the buffer from the file cache to the application buffer. If the buffer is not in the file cache the file system uses the VMM to read the buffer into the file cache from disk before copying it to the application buffer.

For a write request, the VMM copies the SAS application buffer to the file cache. The buffers are not written to disk until one of the following occurs: a sync or fsync call, the synchronization daemon runs, or a write-behind threshold is reached – forcing the write. When the file cache hit ratio is high (e.g., reads from and writes to system file cache are satisfied with pages resident in the cache), this type of cached I/O is very effective in improving overall I/O performance.

However, the use of cached I/O imposes a limit on the amount of physical I/O that a system can perform. The management of the file cache becomes the limiting factor. With DIO the transfer of data for a process is dependent only upon acquiring use of the I/O path to perform the transfer and updating file related data on disk when writing to a file. In the case of cached I/O additional activities must occur. The VMM must find memory in which to place the data by either finding available space in the file cache or creating available space. These activities require locking control structures to ensure consistency of the cache. In addition information used to manage what files have data in the cache and whether the data must be written to disk must be maintained. The overhead of managing the file cache and maintaining locks limits the maximum I/O throughput through the file cache even if the I/O subsystem is capable of achieving higher throughput rates.

SINGLE JOB PERFORMANCE

Most UNIX implementations provide for single process, sequential I/O performance enhancement via use of file caching strategies. The file system sequential read-ahead feature can enhance the performance of programs that access files sequentially. The file system detects sequential reads and has the VMM read larger and larger number of buffers in a single read. This "read-ahead" in effect increases the cache hit ratio. Similarly a "write-behind" strategy defers writes to disk until a larger contiguous write can be committed to disk as a single I/O.

USING DIO TO IMPROVE OVERALL SYSTEM I/O PERFORMANCE

Applications that have poor cache hit rates or applications that do very large I/Os may not get much benefit from the use of cached I/O via the system file cache. In these cases DIO may be helpful. As stated previously, one of the primary benefits of using DIO is that it allows applications to avoid diluting the effectiveness of the system file cache performance for other files. When a file is read or written, that file competes for space in memory (in this case – the system file cache portion of memory) which could cause other file data to get pushed out of memory. If an application developer knows that certain files have poor cache-utilization characteristics, then it could be engineered where only those particular files could be opened with DIO. DIO requires substantially fewer CPU cycles than regular cached I/O. In summary, I/O-intensive applications that do not get much benefit from the caching can enhance performance for the entire system by using DIO.

Applications that have large sequential I/Os are generally good candidates for DIO with little performance difference over the default behavior of utilizing the system file cache. Applications that do numerous small I/Os will typically see less performance benefit or a performance degradation, because DIO cannot do read ahead or write behind, which provides most of the enhanced application throughput.

SAS DATA SET I/O ACCESS PATTERNS AND DIO

SAS performs data set I/O via a fixed size I/O buffer which is associated with the file at data set creation time. If the buffer size is chosen by SAS (SAS option *BUFSIZE=0* which is the default value), the buffer is a multiple of 8 K. The user can specify a *bufsize* value via a global options statement or as a SAS data set option. It is strongly recommended that a multiple of 8 K be selected to avoid performance issues when using DIO. As stated previously sequential I/O may be a good candidate for utilizing DIO.

Most SAS data set I/O is sequential or skip sequential. When SAS data sets are created the I/O is predominantly sequential with the file being extended every time an I/O buffer is written. The exception to sequential writes is the DATA step MODIFY statement which can randomly access I/O blocks depending upon a key=index value or a point=record id clause on the *MODIFY* statement. Similarly, the predominant read access pattern is sequential unless a where clause, key= or point= is specified in the DATA step code or the procedure.

WHEN TO USE DIO FOR SAS DATA SETS

SAS data sets that are too large to fit in a file cache (bigger than the memory used for file cache) and are accessed sequentially are good candidates for using the data set DIO option to improve total system throughput. If only one SAS job will be accessing the data set at any given time then the use of DIO is indicated to improve total system performance. Depending upon the setting of the *TRANSFERSIZE*= option of the LIBNAME statement, DIO may provide nearly the same performance as going through the file cache when the SAS job is the only job running on the system.

If multiple, concurrent SAS jobs have been started that read a large data set sequentially, then using DIO is not recommended as the jobs are sharing data via the file cache, and would actually benefit from the high hit-rate associated with that specific situation. If DIO was utilized in that case, then the amount of physical I/O performed would be the size of the data set multiplied by the number of concurrent jobs. If the data set can be accessed randomly via indexes concurrently with SAS job sequentially accessing the data set then using DIO would also not be recommended. A data set can only be accessed concurrently in one file system mode. If the data set is opened with DIO, ALL access will be via DIO and not the file cache. An exception would be if the following are met.

- The data set is sorted.
- The jobs access the data in sort order or with firstobs= and obs=.
- The jobs access disjoint sets of data.

Creating SAS data sets that are larger than file cache could benefit from utilizing DIO with a large *TRANSFERSIZE*= value.

OBTAINING PERFORMANCE USING DIO

Since DIO disables read-ahead and write-behind optimizations for I/O going through the file cache, the *TRANSFERSIZE*= value has been added to the LIBNAME statement in SAS 9.2. The *TRANSFERSIZE*= option increases the size of the I/O buffer used by SAS but does not change the internal data set structure as specified by the *BUFSIZE*= value associated with the data set. The result is the read-ahead and write-behind behavior of cache I/O is mimicked by SAS. Using large values of 512 K or 1 M with DIO can provide I/O transfer rates for single SAS jobs equivalent to using cached I/O.

In the case of multiple SAS jobs using DIO for sequentially accessing large data sets, the observed system physical I/O throughput can be greatly increased over the I/O throughput obtainable with cached I/O. In one test case running 30 concurrent SAS jobs the sustained I/O rate went from 552 MBs through the file cache to 1270 MBs using DIO. The maximum elapsed time for a job dropped from 14 minutes 47 seconds to 6 minutes 26 seconds. In this example all of the jobs were sequentially accessing different data sets, each of which were individually larger than file cache. This is the best case scenario for utilizing DIO.

CONCLUSION

Using Direct I/O when an application sequentially accesses a large and very large SAS data set (multi-gigabyte and larger) can improve system I/O bandwidth significantly subject to the maximum sustained rate of the I/O subsystem. As DIO disables the file system read-ahead, a suitable large value for the LIBNAME *TRANSFERSIZE=* option should be used to avoid slow single job performance. If the system workload consists of a significant number of jobs which can effectively use direct I/O and the system has a very large I/O bandwidth then increases in the system throughput could be in the 200 to 300 percent range. Direct I/O is not appropriate for jobs which make effective reuse of data in the file cache, as every direct I/O operation translates into one or more physical I/Os. The best case scenario when doing I/O is that the operation does not perform a physical I/O.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Leigh Ihnen
SAS Campus Drive
SAS Institute Inc.
E-mail: Leigh.Ihnen@sas.com

SAS AND SGIO FOR WINDOWS

Before SAS 9.2, direct I/O on Windows was infrequent even though the direct I/O functionality has existed since SAS 8. Few customers saw the need to use direct I/O on Windows since most SAS data sets were several gigabytes or less. There were some customers with SAS data sets much larger, but they were unaware that direct I/O could be used to regain loss I/O throughput. Actually, there is a loss of I/O throughput when processing a 2 GB SAS data set sequentially. The loss of I/O throughput occurs because the data read by SAS is read into the Microsoft Windows File Cache. Likewise, data written by SAS is written to the Windows File Cache. Each page of a SAS data set has to be located in the file cache which introduces a latency for each page read from disk or written to disk. A 2GB SAS data set with 4 K pages (4,096 bytes) contains more than 524,200 pages. The more pages a SAS data set contains, the greater the latency for all of the reads. Direct I/O bypasses the file cache and reads several pages of data per I/O request. Locating a page has much less latency since the data resides within memory buffers that SAS owns. These buffers are reused when reading the next group of pages from the SAS data set.

FILE I/O FOR WINDOWS

The Windows file cache is temporary storage in memory used to hold data that is read from disk or data being written to disk. Files opened for sequential access will utilize the Windows file cache. When data is read from a file, the cache manager will search the file cache first before physically reading the data from disk. The Windows cache manager reads data from the beginning of the file into the file cache anticipating that the data will be needed. This is called *read-aheads* or *look-ahead reads*. Likewise, data being written to disk is stored in the file cache and written by the cache manager at a convenient time. This is called *delayed-writes* or *lazy writes*.

The file cache will grow as more data is read from a file. The cache manager retains previously read data even though new data is being read from disk via look-ahead reads. Likewise, data being written to a file is retained in the file cache until the cache manager determines it is time to flush or write this data to disk. The file cache can shrink too; deleting files or an application requesting more memory can cause the file cache size to be reduced. Unfortunately, the shrinking of the file cache does not occur often when SAS processes large GB files. Reading a 30 GB SAS data set with the DATA STEP can cause the file cache to grow to 2G B to 3 GB in size or even larger if the system contains more RAM. Reading several distinct files of this size in succession will cause the file cache to balloon in size. When reading these large GB files, a latency is introduced each time data is read or written.

The latency occurs because the file cache manager searches through the cached pages of data for the file being accessed. If the file is 1 GB in size or larger, then the latency is longer than if the file is only a few hundred MB in size. That is, pages of data are typically 16 K – 64 K in size. Picture the cache manager searching the file cache for a 16 K-page of data for EVERY page read from the file when the cache is 2 to 3 GB in size. See the latency now! Even files only 2 GB to 4 GB in size suffer in performance when the pages are found in the file cache.

SAS I/O FOR WINDOWS

In the SAS System, SAS I/O, the most common input/output (I/O) processing, is performed on SAS files such as SAS data sets, SAS catalogs, and SAS index files. Most SAS I/O processing is *sequential I/O*, which means that the DATA step and SAS procedures read a data set from the beginning of the file to the end of the file. The data is read from or written to pages that are typically 4-64 K in size.

A SAS job can contain numerous DATA steps and SAS procedures that read and process the same data set or distinct data sets, or it can write or rewrite numerous new and existing data sets. With standard SAS options, each data set that is read from disk is read into the Windows file cache. Likewise, before a data set is written to disk, it is written to the Windows file cache.

A large SAS job can cause numerous SAS data sets to reside in the Windows file cache. However, processing large SAS data sets (data sets greater than 2 GB) reduces I/O throughput. This loss of I/O throughput is not obvious typically because data sets tend to slowly increase in size. Eventually, the processing of such a job seems much slower than is expected, but you might not recognize that reduction in I/O throughput is the root cause. Often, customers spend many hours investigating the hardware of the system by running the SAS job as well as upgrading or enhancing the system, only to learn that none of these changes solve the problem.

DIRECT I/O FOR WINDOWS (SGIO)

The solution to regaining lost I/O throughput is to bypass the Windows file cache with direct I/O. Direct I/O on Windows is called Scatter Read/Gather Write I/O (SGIO). Windows has two application programming interfaces (APIs) that you can use to bypass the file cache. These APIs enable applications to read and write multiple pages of data sequentially with a single Windows file system call (*the scatter-read/gather-write/input/output feature*). The *scatter-read feature* reads data directly from the disk. The *gather-write feature* writes data directly to the disk. To activate scatter-read/gather-write file access globally in SAS, specify the SGIO system option either on the command line or in the SAS configuration file. For example, specify the SGIO system option on a command line when you invoke SAS, as follows:

```
c:\SAS> sas -sgio
```

SAS 9.1.3 SP4 and later also contain the SGIO= data set option to enable and disable SGIO processing for specific data sets.

SGIO processing (activated with the SGIO system option) provides each SAS I/O file with the ability to use scatter-read/gather-write access. However, not all SAS I/O files will be opened for SGIO processing even if the SGIO option is specified when you invoke SAS. The following criteria must be satisfied in order to open a file for SGIO processing:

- SAS I/O files that are created on 32-bit Windows system have a page size (in kilobytes) that is a multiple of 4 K (4096). Files that are created on a 64-bit Windows system have a page size that is a multiple of 8 K (8192).
- Files that are to be processed with SGIO are created in SAS 7 or later because files created in earlier releases do not use SGIO processing.
- The file access pattern must be sequential (for example, non-indexed data sets).
- The SAS file is created in the Windows environment.

Note: If a SAS I/O file does not meet these criteria, the SGIO processing remains inactive for that file even if you specify the SGIO system option or data set option. If a file is not using SGIO processing, no warning message appears in the SAS log.

To obtain maximum throughput, use the SGIO feature as a tuning process to obtain maximum I/O throughput. Apply the BUFSIZE= and the BUFNO= system options. SAS data sets contain pages of data. When you create a data set, the page size of each page of data is the value for the BUFSIZE= option. The *BUFSIZE= option* specifies the size of each page (or buffer) of data to read or write for each I/O operation. For an existing SAS data set, the value for the BUFSIZE= option is the page size that was used when the SAS data set was created.

The *BUFNO= option* specifies the number of buffers (or pages) to read or write for each I/O operation. You rarely obtain any improvement in I/O throughput by simply specifying the SGIO system option because the software assumes a default value of 1 for the BUFNO= option. Therefore, SAS reads or writes only one page directly to the disk, which results in poor performance.

When data is read from a SAS data set, the number of pages read is the number that is specified in the BUFNO= option. Likewise, when data is written to a SAS data set, the number of pages written is the number that is specified in the BUFNO= option. Because the page size is defined when the file is created, specifying the BUFSIZE= option affects the performance of only those data sets that are being created.

In the following example, a simple DATA step is used to create a SAS data set sample:

```
options bufno=3;

data sample;
do i=1 to 100000000;
output;
end;
run;
```

In this example, the SGIO feature uses three buffers, as specified by the BUFNO= option. Three pages of data are written with one call to the Windows file system. If the DATA step is run on a Windows 32-bit system, the software uses the default page size of 4 K for the data-set sample. To determine if you can obtain better throughput for this simple DATA step, increase the page size or the number of buffers. Change only one of these options for each run of the DATA step. As shown in the next example, the value for the BUFNO= option remains the same, but the BUFSIZE= option has been added.

```
options bufno=3 bufsize=8k;

data sample;
do i=1 to 100000000;
output;
end;
run;
```

The other possibility is to keep the same value for the BUFSIZE= option while you change the value for the BUFNO= option.

Suppose that you test for throughput improvements by changing only the BUFNO= value, as illustrated here:

```
options bufno=n;

data sample;
do i=1 to 100000000;
output;
end;
run;
```

If you submit this DATA step nine times using BUFNO= option values that range from 1 to10,000, you obtain the following time results, measured in seconds.

SAS Run	Run 1	Run 2	Run 3	Run 4	Run 5	Average
NOSGIO BUFNO=1 (default)	40.68	40.81	37.56	43.23	40.74	40.60
SGIO BUFNO=3	57.27	59.10	57.99	58.10	59.71	58.43
SGIO BUFNO=20	31.01	35.31	31.03	31.04	30.73	31.82
SGIO BUFNO=50	29.35	30.06	30.26	30.23	30.02	29.98
SGIO BUFNO=100	29.23	28.68	28.50	29.57	29.73	29.14
SGIO BUFNO=500	29.51	27.73	27.79	27.88	27.93	28.17
SGIO BUFNO=1000	27.43	28.10	27.54	28.04	28.23	27.87
SGIO BUFNO=5000	28.78	28.37	28.39	28.92	28.74	28.64
SGIO BUFNO=10000	28.68	28.76	28.65	28.67	28.81	28.71

Table 1: Comparison of Data-Set Creation with a Default (4 K) Page Size and Varying BUFNO= Option Values on a 32-Bit Platform (Time Measured in Seconds)

These results show that various I/O throughput improvements can be obtained from simply changing the number of buffers. For this scenario (BUFSIZE=4K), BUFNO=1000 had the best overall time average. Notice that specifying BUFNO=3 does not outperform the case that uses the default case, NOSGIO BUFNO=1.

Notice that the performance degrades when the number of buffers reaches 5,000 and 10,000. Obviously, this is not a typical DATA step, but the same strategy also works for complex DATA steps.

The next step is to change the page-size value (for example, to 8 K or 16 K) in the BUFSIZE= option and rerun the DATA step with the same buffer numbers (1 to 10000) to see if you can gain better I/O throughput.

Note: Windows 32-bit systems have a 64 MB I/O limitation, which means that the product of the BUFNO= value that you choose and the BUFSIZE= value must be less than 64 MB. For Windows 64-bit systems and Windows on Itanium, the I/O limitation is 2 GB. In the previous example, the largest number of buffers used is 10,000 and the default buffer size is 4 K. The product produced by these values is 40,960,000 bytes, which is well below the 64-MB limit.

SAS Run	16K Average	32K Average	64K Average
Default (NO SGIO)	Note1	Note 1	Note 1
SGIO BUFNO=3	39.49	34.06	29.40
SGIO BUFNO=20	30.59	28.47	27.82
SGIO BUFNO=50	28.70	27.11	27.15
SGIO BUFNO=100	27.45	27.24	26.92
SGIO BUFNO=500	27.40	28.16	28.67

SGIO BUFNO=1000	27.77	29.10	41.50
SGIO BUFNO=5000	Note 2	Note 2	Note 2
SGIO BUFNO=10000	Note 2	Note 2	Note 2

Table 2: Comparison of Data set Creation with 16 K, 32 K, and 64 K Page Size and Varying BUFNO Values on a 32-bit Platform (Time Measured in Seconds)

From these results, you can see the absolute best average times for each page size used. The overall best average time was for a 64 K page size using 100 buffers. However, more I/O throughput may be obtained with a larger page size or with slightly fewer or slightly more than 100 buffers. If time permits, try the same test with buffer sizes of 24 K, 48 K, 128 K and so on with the same BUFNO= values as above. Also, try the same test with a buffer size of16 K and BUFNO= values between 100 and 500 as well as 500 and 1000. A more optimal BUFNO= may exist in these ranges. Likewise, for 32 K and 64 K page sizes, try the test with BUFNO= value below and above 50 and below and above 100 respectively. With the SGIO feature, you can tune as much as time allows.

In Table 2, there are some tests with no results. An explanation for these results follows:

Note 1: No elapsed time was obtained for these runs since the system used began paging data to disk. Since paging was present, the elapsed time(s) were discarded. You should choose a system with more memory to possibly find a more optimal number of buffers value and/or a buffer size value.

Note 2: The DATA STEP exited prematurely from an I/O error because the maximum Windows I/O limitation (64 MB) was surpassed. Therefore, no elapsed time was obtained for these runs.

It is important to note that SGIO can, in fact, reduce data throughput and reduce performance for small files less than 2 GB. When you specify the SGIO option on the command line or in the configuration file, the option has a global effect in that all SAS I/O files (of any size) will use SGIO if the files meet the criteria. If you have a mix of large and small files, that processing can degrade the throughput for files that are less than 2 GB. Therefore, it is strongly recommended that you do not use SGIO processing for such files, but that instead you allow them to be processed, by default, through the Windows file cache.

SGIO - SAS DATA SET OPTION

When a SAS job has a combination of small and large SAS data sets, the solution is to invoke SAS without the global SGIO feature. Instead, you can use the SGIO= data set option to specifically target those files that will benefit from SGIO processing, as shown in the following example that processes multiple SAS data sets:

```
data master(sgio=yes);
set NYmonthly(sgio=yes) LAmonthly CHImonthly;
. . . more SAS statements . . .
run;
```

In this example, SAS is invoked without the SGIO option. The data sets LAmonthly and CHImonthly are less than 1 GB. Therefore, SGIO processing is activated only for the data sets master and NYmonthly.

If most of the data sets are large enough for SGIO processing, then you can also use the SGIO data set option to disable SGIO processing for the small data sets. The following example illustrates the use of the SGIO= data set option in that manner:

```
data salesreport;
set west midwest(sgio=no) mideast(sgio=no) east;
. . . more SAS statements . . .
run;
```

Here, SAS is invoked with the SGIO option specified. The data sets salesreport, east, and west are processed using SGIO. The data sets midwest and mideast are less than 1 GB, so these data sets are processed without SGIO processing.

RECOGNIZING WHEN TO USE SGIO

Most SAS jobs are a combination of numerous DATA steps and procedures. Some DATA steps and procedures process small SAS data sets and some process data sets that are several GBs in size. The overall performance of the large SAS jobs may be poor but caused by only a few of the DATA steps and/or procedures within the job. This was the case in a SAS ETL Studio job that showed poor performance. The job contains numerous DATA steps and SAS procedures. A review of the SAS log of this job shows that there were six DATA steps and procedures that appeared to be the performance areas.

DATA STEP/PROC	Default Real Time	Default CPU Time
Step 1 (DATA STEP)	16:55:24.42	12:33:57.09
Step 2 (SORT)	8:13.92	1:41.43
Step 3 (DATA STEP)	4:36:15.45	54:43.75
Step 4 (SORT)	12:53:05.94	6:52:32.21
Step 5 (APPEND)	5:14:49.45	3:45:56.90
Step 6 (SQL)	1:02:45.41	7:21.89

Table 3: ETL STUDIO Job Results (Default Real and CPU Times) (All times in hh:mm:ss format)

This ETL Studio run took around 41 hours to complete. Table 3 shows the six DATA steps and procedures that show substantial I/O times. All of these six DATA steps and procedures were processing SAS data sets that were several GBs in size. Since each of these was processing the data sequentially, the Windows file cache was huge and was affecting the I/O throughput. The solution is to use SGIO to regain the loss I/O throughput.

PROC/DATA STEP	Default (NO SGIO)	BUFNO=100	BUFNO=150	BUFNO=200	BUFNO=250
Step 1 (DATA STEP)	16:55:24	11:07:17	10:29:38	10:22:11	10:34:44
Step 2 (SORT)	8:13	5:30	4:14	4:12	4:10

Step 3 (DATA STEP)	4:36:15	2:13:12	2:12:47	2:07:26	2:05:01
Step 4 (SORT)	12:53:05	11:52:12	11:12:09	11:18:24	11:30:58
Step 5 (APPEND)	5:14:49	5:17:23	5:41:08	6:01:14	6:26:14
Step 6 (SQL)	1:02:45	19:12	18:25	18:04	17:38

Table 4: ETL STUDIO Job Results (BUFSIZE=64 K) (SGIO and BUFNO Options Specified) (All times are Real times and in hh:mm:ss format)

Table 4 shows the real time results of using SGIO for each of the six DATA steps and procedures. The SAS data sets contained a page size of 64 K. For Step 1 (DATA step), the best real time was for BUFNO=200 which reduced the real time by over 6 hours. For Step 2 (PROC SORT), Step 3 (DATA step), and Step 6 (SQL), the BUFNO=250 time was the best which cut the real time in half. However, Step 4 (PROC SORT) had the best time with a BUFNO=150 improving the throughput by over 1 hour. But not all of these were improved by using SGIO. Step 5 (PROC APPEND) loss I/O throughput by using SGIO when using these BUFNO values and a 64 K page size. However, APPEND could have required fewer than 100 buffers or more than 250 buffers to obtain maximum I/O throughput. Given the data set's page size was 64 K, the maximum BUFNO= value is about 1000. PROC APPEND had just less than 1.5 hours of I/O time so it is a good candidate for SGIO. If APPEND could not be improved, the loss of 3 minutes is worth regaining several hours of I/O throughput for the other five steps.

Again, there may be better results for each step using a different page size greater than 64 K or even smaller than 64 K. If time permits, try a variety of BUFSIZE= values and BUFNO= values to find the optimal combination.

CONCLUSION

The Windows file cache does a good job of processing files less than 2 GB. When the file size is above 2 GB, however, you should use SGIO tuning to obtain maximum I/O throughput. In a large SAS job, tune DATA steps and procedures separately.

Remember, the level of tuning is your decision. The tuning example in this paper uses a wide range of buffer numbers, which is a good start in most circumstances. The results for these examples suggest a range whereby the maximum throughput can be obtained. It is usually worth the extra effort to tune your code in order to gain improvements in performance.

When you tune your code, be sure to follow these steps:

1. Try different values for the BUFNO= option to find the optimal number that will provide improved throughput.
2. Change the buffer size and rerun your code with the same initial buffer numbers to see if you can find a range that enables maximum throughput.

Note: You must recreate existing SAS data sets to change the page size. In addition, make sure that you always increase the buffer size for a new run since SAS selects the smallest optimal page size by default.

3. The SGIO system option affects all SAS I/O files (data sets, catalogs, indexes, etc.) if they meet the criteria for SGIO.
4. Be aware of any small SAS I/O files less than 2 GB because they rarely benefit from SGIO processing. Performance can even be reduced in some cases. You should allow such files to be processed by the Windows file cache. If you have a combination of small and large files, use the SGIO= data set option to target which files should receive SGIO processing.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mike Jones
SAS Campus Drive
SAS Institute Inc.
E-mail: Mike.Jones@sas.com