Technical Paper

Achieving Better I/O Throughput Using SGIO in the Microsoft Windows Environment

Table of Contents

Introduction

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 Microsoft 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.

This paper explains how experienced users can recognize any loss of I/O throughput before they resort to hardware or system changes and how they can solve this problem by using the SGIO (scatter-read/gather-write input/output) feature.

I/O Throughput and the Windows Cache File

Reduction in I/O throughput occurs when the Windows cache manager does not find data in the Windows cache file. The *Windows file cache* is a temporary storage location in memory that is used by Windows to hold data that is read from disk or data that is being written to disk. Files opened for sequential access use the Windows file cache. When data is read from a file, the Windows cache manager searches the file cache first. The cache manager reads data from the beginning of the file into the file cache anticipating that the data will be needed. This process is called a *read-ahead* or a *look-ahead read*. Likewise, data that is being written to disk is stored in the file cache. That data is then written to disk by the cache manager at a convenient time. This process is called a *delayed-write* or a *lazy write*. Data that is not found in the cache causes the cache manager to read the data from disk. This behavior, known as a *cache miss,* reduces I/O throughput.

The file cache grows 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 that is being written to a file is retained in the file cache. The cache manager determines when to flush the data or to write this data to disk but retains the data in the cache. The file cache can shrink, too. For example, the file cache size is reduced when a file is deleted or when an application requests more memory.

Unfortunately, such shrinkage does not occur often when SAS processes large files (those measured in gigabytes). Reading a 30-gigabyte SAS data set with a DATA step can cause the file cache to grow by 2 or 3 gigabytes. If your system has 8 or more gigabytes of RAM, the file cache will grow and use as much of the available RAM as it can.

Reading several distinct SAS data sets of this size in succession causes the file cache to expand greatly. Even reading a single SAS data set of this size causes the file cache to expand because the cache manager attempts to retain as much of the data as possible. A huge file cache causes a latency each time you read from or write to a large file. The latency occurs because the cache manager has to search through the cached pages of data when it is accessing the file.

When the system reads or writes data, the latent period is longer when the system reads or writes data for a file of at least 2 GB than for a file that is only a few hundred megabytes in size. Pages of data are typically 4—64 K in size. So you can imagine how much the I/O throughput is reduced when the cache manager searches a 2—3 gigabyte cache when the pages average 16 K per page! SAS data sets of 4—10 GB create even more reduction in performance when the pages are found in the file cache.

Scatter-Read/Gather-Write/Input/Output (SGIO) Feature

The solution to regaining lost I/O throughput is to bypass the file cache. Windows has two application programming interface (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. The section “Using the SGIO= Data Set Option to Enable and Disable SGIO Processing” provides examples of how to use the option to enable and disable SGIO.

Using SGIO Processing

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 4K (4096). Files that are created on a 64-bit Windows system have a page size that is a multiple of 8K (8192).
- Files that are to be processed with SGIO are created in SAS Version 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.

Tuning the DATA Step and Procedures with the SGIO Feature

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 only affects the performance of 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. Only change 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 option 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 five 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

Comparison of Data-Set Creation with a Default (4K) 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 8K or 16K) in the BUFSIZE= option and re-run the DATA step with the same buffer numbers (1 to 10000) to see if you can gain better I/O throughput.

With the SGIO feature, you can tune as much as time allows. If time permits, try buffer sizes of 24 K, 48 K, 128 K, and so on with the same buffer number values. Also, try BUFNO= values between the buffer numbers from Table 1 that give the best throughput. The table above shows that the best throughput occurs with BUFNO=1000. Therefore, you should try values between 500 and 1000, and even between 1000 and 5000 because you might find a combination that provides better throughput.

Note: Windows 32-bit systems have a 64 MB I/O limitation, which means that the results of the BUFNO= value that you choose and the BUFSIZE= value must be less than 64MB. 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 result produced by these values is 40,960,000 bytes, which is well below the 64-MB limit.

The simple example of using SGIO processing that is illustrated in this section is able to outperform the default I/O processing that occurs if you use the Windows file cache. In addition, the size of the data set that is created is less than 1 GB. However, a case where SGIO processing can benefit a file less than 1 GB is unusual.

Using the SGIO= Data Set Option to Enable and Disable SGIO Processing

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 2GB. 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.

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.

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 this example 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 re-run your code with the same initial buffer numbers to see if you can find a range that enables maximum throughput.

   Note: You must re-create existing 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.

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.