SAS6600-2016

Using SASIOTEST to Measure I/O Throughput

Mike Jones, SAS Institute Inc., Cary, NC

ABSTRACT

Have you questioned the Read throughput or Write throughput for your Windows system drive? What about the input/output (I/O) throughput of a non-system drive? One solution is to use SASIOTEST.EXE to measure the Read or Write throughput for any drive connected to your system. Since SAS® 9.2, SASIOTEST.EXE has been included with each release of SAS for Windows. This paper explains the options supported by SASIOTEST and the various ways to use SASIOTEST. It also describes how the I/O relates to SAS I/O on Windows.

INTRODUCTION

If you have used SAS for Windows, then you have probably questioned the I/O throughput of reading or writing a SAS data set. You can see the real time and CPU time displayed in the SAS log for DATA step or PROC statement runs. When the real time increases and the CPU time remains fairly constant for the same DATA step or PROC statement, this causes concern. Further investigation does not explain why the DATA step or PROC statement has suddenly become slower. Next, you check the Windows event logs to see whether there are any hardware issues. However, there are no problems with the disk drive where the I/O occurs. Maybe the I/O throughput for the WORK library is fine, but the I/O throughput for a different SAS library is much slower. And, this SAS library exists on a different disk drive than the WORK library. Again, no obvious explanations for the slower I/O throughput can be found.

The SAS system is complex. Numerous threads are running within SAS in addition to the DATA step or PROC statement. Some of these threads can be reading and/or writing to other files on the same disk drive as your SAS data set. Obtaining a more accurate measurement of I/O throughput for your SAS data set is a difficult task. You need to be able to measure the I/O throughput for a file the same size as your SAS data set without SAS performing the I/O. This is where SASIOTEST.EXE can be used to measure the I/O throughput.

SASIOTEST is an external utility to the SAS system used to measure I/O on any Windows platform. Since SASIOTEST is a stand-alone utility, you can copy SASIOTEST.EXE to any system to measure I/O throughput. SAS does not have to be installed on the system in order to run SASIOTEST.EXE. SASIOTEST opens files and reads and writes data just as the SAS I/O code does within SAS. That is, SASIOTEST uses the same WIN32 API routines to do the I/O just as the SAS I/O code for Windows.

SASIOTEST: THE SAS I/O TEST UTILITY

FEATURES

As mentioned, SASIOTEST performs file I/O just as the SAS I/O code in SAS. SASIOTEST reads and writes data but performs no actions on the buffer used for reading and writing data. This provides a more accurate measurement of I/O throughput. SASIOTEST has the following characteristics:

- runs on Windows 32-bit and Windows x64 systems
- writes or reads data in blocks/pages just as SAS I/O does
- opens files for reading or writing using the Windows API CreateFile()
- writes to a file using the Windows API WriteFile() for sequential cache writes
- reads from a file using the Windows API ReadFile() for sequential cache reads
- uses the Windows API CloseFile() to close the file
- contains a Help facility to describe the usage of the utility
- for direct I/O reads and writes, uses the Windows APIs ReadFileScatter() and WriteFileGather()
- supports the reading of pages randomly from existing file
- displays the elapsed real time
- displays the I/O throughput in megabytes per second (MB/second)

EXAMPLES AND SAMPLE OUTPUT

SASIOTEST.EXE is launched from a command prompt. SASIOTEST.EXE exists in your *sasroot* folder. The file, sas.exe, also exists in your *sasroot* folder. Here is the default *sasroot* folder:

C:\Program Files\SASHome\SASFoundation\9.4\

Displaying SASIOTEST Help

A simple invocation of SASIOTEST displays the HELP facility. To display the HELP facility, invoke SASIOTEST with no filename specified and no options:

C:\Program Files\SASHome\SASFoundation\9.4\>sasiotest

The following examples show how to use SASIOTEST.

Creating a File Using SASIOTEST

SASIOTEST can be used to create files of different sizes. If the -filesize option is not specified, then SASIOTEST creates a 1MB file by default. If the –pagesize option is not specified, SASIOTEST defaults to an 8K page size. A page is simply a buffer that is allocated memory, which is used to read or write the data. When writing a file, SASIOTEST allocates a page/buffer to write the data to the file, which, by default, is 8k bytes. Here is an example:

```
 sasiotest test.out –w

SAS I/O Test Utility: v2.0

WARNING: Filesize option not specified. Defaulting to 1 MB file size.
WARNING: Pagesize option not specified. Defaulting to 8K page size.
Sequentially writing 1048576 bytes to file: test.out with an 8192 page size.
Write Throughput: 707.741722 MB/Sec.
Elapsed time: 0.001413 Secs
```

Here, SASIOTEST opens the file, test.out, and sequentially writes 1MB of data. Writing the data sequentially means that page 1 is written first, and then page 2 is written, and so on. Notice the Write throughput is over 707MB/sec. The Write throughput is far greater than the maximum rating for the disk drive. The Write throughput displayed is high because the data is actually written to the Windows File Cache. Writing to the file cache is extremely efficient. SASIOTEST opens the file, specifying flags to use the file cache. The Windows File Cache Manager writes the pages of data to the disk drive whenever it sees fit, sometimes even after the process has completed.

Note: For this example, SASIOTEST creates the file, test.out, in the current folder on the current disk drive.

Here is a SAS DATA step example similar to the I/O in the preceding SASIOTEST example:

```
data test(bufsize=8K);
  do i=1 to 1000000;
     output;
  end;
run;
```

This DATA step creates a SAS data set with 1,000,000 OBS and 1 numeric variable using an 8K page size. The total file size for the SAS data set is greater than 1MB, but SAS I/O uses the same Open mode and flags to sequentially write the data. The file is opened to use the file cache, so the throughput you observe here might seem inflated.

Creating a File Using SASIOTEST with Additional Options

In the previous example, SASIOTEST used the default file size and page size values. Likewise, the previous example wrote the file, test.out, to the current folder on the current disk drive. However, you are encouraged to try various page size and file size values. The Write throughput varies with different file sizes and different page sizes. And, writing the same size file to a different drive could vary the Write throughput. Here is an example:

```
  sasiotest d:\test.out -w -pagesize 64k -filesize 512K

SAS I/O Test Utility: v2.0

Sequentially writing 524288 bytes to file: d:\test.out with a 65536 page
size.
Write Throughput: 375.276280 MB/Sec.
Elapsed time: 0.001332 Secs
```

Here, SASIOTEST creates a 512KB file, but the file is written to drive D instead of the default drive. The page size used is 64K instead of the default 8K page size. All pages are written sequentially to the file cache as in the previous example. The Write throughput, 375MB/sec, is higher than expected for drive D but lower than the previous example. The larger page size and smaller file size could explain the lower throughput. When you see an example such as this one, try writing a different file to the same location but with various page sizes and file sizes. Also, ensure that no other processes are reading or writing data as well.

This example of SASIOTEST is similar to SAS I/O when invoking a SAS DATA step:

```
   data test(bufsize=64K);
     do i=1 to 500000;
        output;
     end;
   run;
```

This DATA step creates a SAS data set with 500,000 OBS and 1 numeric variable using a 64K page size. The total file size for the SAS data set is greater than 512K bytes, but SAS I/O uses the same Open mode and flags to sequentially write the data. Again, the throughput you observe here might be higher than expected since the data is written to the file cache.

Reading a File Using SASIOTEST

In addition to creating files, SASIOTEST can be used to read existing files. For reading a file, the file size and page size default to 1MB and 8K, respectively. So, SASIOTEST allocates an 8K page to read the data from the file. An example for reading an existing file using SASIOTEST is as follows:

```
  sasiotest D:\myvolume\test.out -r

SAS I/O Test Utility: v2.0

WARNING: Filesize option not specified. Defaulting to 1 MB file size.
WARNING: Pagesize option not specified. Defaulting to 8K page size.
Sequentially reading 1048576 bytes from file: c:\test.out with an 8192 page
size.
Read Throughput: 1682.549010 MB/Sec.
Elapsed time: 0.000594 Secs
```

For this example, SASIOTEST opens the file, D:\myvolume\test.out, and sequentially reads 1MB of data using an 8k page. Reading the data sequentially means that page 1 is read first, and then page 2 is read, and so on, until the last page is read. For each Read operation, 8K bytes are read. Notice the Read throughput is extremely high (over 1682MB/sec) and is far greater than the maximum rating for the disk drive. The Read throughput is high because the data is read from the Windows File Cache. Like writing data to the file cache, reading data from the file cache is extremely efficient. For reading data, SASIOTEST opens the file, specifying flags to use the file cache. This causes the Windows File Cache Manager to pre-read several pages of data from the disk into the file cache if the data isn't already in the file cache. This ensures the next page of data read exists in the file cache and improves the performance. Here is a similar DATA step:

```
data _null_;
  set test;
run;
```

This DATA step reads the SAS data set: test. The same flags that SASIOTEST uses to read a file are used to open the data set so that the file cache would be used. The data set is read sequentially beginning with the first page, and then the second page, and so on, until all pages have been read. Since the data is read from the file cache, the throughput for the DATA step here might be higher than expected.

To match the preceding SASIOTEST example with this DATA step example, create a SAS data set approximately 1MB in size with an 8K page size. Use the BUFSIZE= option to define the page size when creating the SAS data set. For existing SAS data sets, run PROC CONTENTS to determine the page size defined when the data set was created.

Note: For reading files, SASIOTEST requires the file size to be a multiple of the page size. This guarantees that SASIOTEST reads all data from the existing file. If the file is too small for the given page size, then SASIOTEST displays an error message. If the file is larger than the specified file size, then SASIOTEST reads only the specified file size amount of data. That is, if the file being read is 1GB and the file size specified for SASIOTEST is only 1MB, then SASIOTEST reads only the first 1 MB of the file.

Reading a File Using SASIOTEST with Additional Options

In the previous example, SASIOTEST read a 1MB file using an 8K page size. For reading files with SASIOTEST, you should try various page size and file size values since the Read throughput will vary. Reading data from a different drive could vary the Read throughput as well. Here is a SASIOTEST example:

```
 sasiotest c:\test.out -r -pagesize 64k -filesize 4G
```

```
SAS I/O Test Utility: v2.0

Sequentially reading 4294967296 bytes from file: c:\test.out with a 65536
page size.
Read Throughput: 63.986464 MB/Sec.
Elapsed time: 64.013539 Secs
```

The Read throughput for the preceding SASIOTEST invocation suggests that some disk I/O occurred. With a larger file size being read, as well as a larger page size for each read, the Read throughput decreased significantly. Try this invocation on your system with a 4GB file, but vary the page size as follows: 4K, 8K, 16K, 32K, 64K, 128K, and even 256K to see if different page size values affect the Read throughput. Smaller page sizes require more Read operations, which usually leads to slower throughput. The larger page sizes reduce the number of Read operations and improve throughput, but page sizes much larger than 512K (that is, 2MB and above) might result in less optimal performance.

Here is an example of an equivalent SAS DATA step that sequentially reads a SAS data set:

```
data _null_;
  set four_gig;
```

```
   run;
```

SAS I/O uses the same Open mode and flags to sequentially read the data from disk. To match the same amount of I/O as the SASIOTEST example, create a SAS data set of 4GB (that is, approximately 4 billion OBS and a single numeric variable) with a BUFSIZE=64K.

Now, a file can exist on the disk but might not exist in the file cache because the file cache manager discarded the file to provide space for a different file that was populated in the file cache. To see this effect on your system, run these three SASIOTEST examples from the same command prompt consecutively:

```
sasiotest c:\test.out -w -pagesize 64k -filesize 12G

sasiotest c:\junk.out -w -pagesize 64k -filesize 12G

sasiotest c:\test.out -r -pagesize 64k -filesize 12G
```

The first SASIOTEST invocation creates a 12GB file called c:\test.out. This invocation populates the file cache with the file C:\test.out. If your system does not have enough available memory for a 12GB file, then reduce the size for the file c:\test.out accordingly. If your system has more than enough available memory for a 12GB file, then increase the file size for the file c:\test.out accordingly. The second SASIOTEST creates a “dummy” file called c:\junk.out and populates the file cache as well. The objective here is to force the file cache manager to discard the data for the file: c:\test.out. The data for file c:\test.out is discarded from the file cache if there is not enough available memory to populate the file cache with the data for file c:\junk.out. Now, the third SASIOTEST will read the file c:\test.out, but the file does not exist in the file cache. Once SASIOTEST opens the file for sequential reading, the file cache manager detects that the file does not exist in the file cache and begins reading multiple data pages from the disk into the file cache. When SASIOTEST reads the first data page, the data will reside in the file cache. As SASIOTEST reads more data pages from the file c:\test.out, the file cache manager continues to read multiple data pages from the disk and stores them into the file cache. The file cache manager continues reading multiple data pages until the end of the file is reached. The performance you see with this scenario is more optimal than SASIOTEST reading each data page directly from the disk. Yes, there is disk I/O occurring in the background via the file cache manager, but SASIOTEST is not delayed when reading the data.

If the entire file exists in the file cache when the file is opened, then the file cache manager detects this and no data is read from the disk. If this is the case, the Read throughput is even better since there is no physical disk I/O performed. Let’s try the following. Consider the two following SASIOTEST examples run from the same DOS prompt consecutively:

```
sasiotest c:\test.out -w -pagesize 64k -filesize 12G

 sasiotest c:\test.out -r -pagesize 64k -filesize 12G
```

Adjust the size of the file c:\test.out to the amount of available memory on your system. The first SASIOTEST creates a 12GB file called c:\test.out to populate the file cache. And, the second SASIOTEST reads the file c:\test.out from the file cache. Since the entire file exists in the file cache, the file cache manager recognizes this and does no background reads from the disk. This scenario is the most optimal because there is no disk I/O occurring. If you read the file again a second time, the Read throughput would be optimal providing no other process on the system forced this file from the file cache.

Your system might not have enough memory to allow the file cache to contain the entire file. For files that exceed the amount of available memory on your system, the file is read from disk via the file cache manager, but data pages near the end of the file replace data pages near the beginning of the file within the file cache. For example, the file to be read is 20GB, but only 15GB of memory is available. SASIOTEST begins by opening the file, which instructs the file cache manager to begin reading multiple data pages from the disk. SASIOTEST begins reading the pages sequentially, so the first page is read, and then the second page, and so on. Eventually, the file cache manager reads another block of data pages from the disk, and SASIOTEST continues to read the pages as expected. Remember the pages that have been read by SASIOTEST still exist in the file cache at this point even though SASIOTEST has already read them. *The pages aren't discarded immediately.* Eventually, the file cache fills to the capacity of the available memory, so the file cache manager begins discarding the file's first data pages from the file cache. This memory is used to retain the last group of data pages in the file. For this scenario, the data pages read by SASIOTEST are file cache reads, but disk I/O occurs during the entire process of reading the file. If this file is subsequently read again, the first group of data pages are not in the file cache, so the file cache manager reads them from disk, as was the case when the file was read the first time. The Read throughput for this scenario varies, as you might guess, since there is disk I/O occurring in the background every time you read the file.

Using SASIOTEST to Randomly Read a File

So far, SASIOTEST has been used to read and write files sequentially. However, you can use SASIOTEST to read and write data pages randomly. That is, SASIOTEST generates a page number randomly and reads that page from the file. The algorithm used by SASIOTEST determines the number of pages within the file based on the file size and page size specified. This value is called the total number of pages. SASIOTEST randomly reads at most this total number of pages from the file. For random reads, the SASIOTEST algorithm might result in all pages being read in the file once. Some pages might be read multiple times, and some pages might not be read once. However, the read pattern is random (that is, non-sequential). A SASIOTEST innovation to read a file's pages randomly is as follows:

```
  sasiotest c:\test.out -r -ran -pagesize 64k -filesize 1M
```

```
SAS I/O Test Utility: v2.0

Random Reads: Reading 16 pages from file: c:\test.out with a 65536 page size.
Read Throughput: 944.664671 MB/Sec.
Elapsed time: 0.001059 Secs
```

First, SASIOTEST determines there are sixteen 64K pages within the 1MB file. There is a total of sixteen random pages read from the file. SASIOTEST randomly reads the 16 pages from the file c:\test.out with a 64K page. Another invocation of the same preceding SASIOTEST example could result in a similar random pattern or a completely different pattern. The Read throughput is dependent on the data page existing in the file cache. For random reads, the Windows File Cache is used some but not as much as when reading data sequentially. When SASIOTEST opens the file for random reads, the file cache manager detects this and reads some of the file's first few pages from the disk, storing them into the file cache. The number of pages read by the file cache manager is much less than when the file is opened sequentially. After the initial read by the file cache manager, the read pattern by SASIOTEST determines the pages that the file cache manager reads from disk to populate the file cache. The file cache manager reads pages before and after the page requested by SASIOTEST. Hence, some data pages read could exist in the file cache resulting in faster I/O throughput, and some data pages might have to be read from disk. Reading from the disk degrades I/O throughput.

If the entire file exists in the file cache, then Read throughput is optimal. To experiment with this scenario on your system, run the following SASIOTEST examples from the same DOS prompt consecutively:

```
 sasiotest c:\test.out -w -pagesize 64k -filesize 5G

   sasiotest c:\test.out -r -ran -pagesize 64k -filesize 5G
```

Again, adjust the size of the file according to the amount of available memory on your system to ensure the file will exist entirely in the file cache. The first SASIOTEST invocation populates the file cache with the file c:\test.out. The second SASIOTEST randomly reads the data pages from c:\test.out. Since all of the data exists within the file cache, the Read throughput is optimal.

Likewise, if the file being read randomly does not exist in the file cache, then Read throughput varies based on the number of data pages that are found within the file cache and the number of data pages that have to be read from disk by SASIOTEST. Remember, the file cache manager reads pages before and after the page requested by SASIOTEST. If the next read request from SASIOTEST is a page that is in the file cache, then the read for that page is optimal. However, if the read request from SASIOTEST is a page that is not in the file cache, then the page is read from disk. Reading from the disk degrades I/O performance. To observe this behavior with SASIOTEST, run these three invocations from the same DOS prompt consecutively:

```
 sasiotest c:\test.out -w -pagesize 64k -filesize 5G

   sasiotest c:\junk.out -w -pagesize 64k -filesize 12G

   sasiotest c:\test.out -r -ran -pagesize 64k -filesize 5G
```

Here are the results:

- The first SASIOTEST invocation writes the file to disk and populates the file cache. Skip this invocation if the file already exists on the disk.
- The second SASIOTEST invocation writes a dummy file to the disk and populates the file cache with the dummy file. Ensure that the size of the dummy file is enough to discard all pages of the file c:\test.out. Also, adjust the file sizes here according to the amount of available memory on your system.
- The third SASIOTEST invocation randomly reads the file, so the Read throughput varies based on whether a page exists in the file cache or has to be read from disk. Remember, after the third SASIOTEST has completed, some of the data pages for the file c:\test.out will exist in the file cache. To see if I/O throughput varies, try invocations 2 and 3 above. Or, try only the third SASIOTEST to see if the Read throughput improves or degrades.

An example DATA step similar to the preceding SASIOTEST example is nearly impossible. First, there is no way to know what data pages SASIOTEST read without some advanced I/O analysis. Likewise, a new invocation of the same SASIOTEST would result in different data pages being read. To force a SAS data set to be opened in random I/O mode, use the POINT= option with the SET statement in the DATA step. SAS I/O opens the SAS data set using the same flags that SASIOTEST used in the preceding example.

Using SASIOTEST to Randomly Read and Write Data

In the previous example, SASIOTEST was used to randomly read data pages from a file. Another feature of SASIOTEST with random reads is to write the data page back to disk after the page has been read. This is also called Read/Write mode. Use the –rw option for SASIOTEST for Read/Write mode. This type of I/O is similar to UPDATE mode within SAS I/O. The file is opened by SASIOTEST for reading and writing data. Again, the algorithm used for Read/Write mode is the same as for random reads, so all data pages in the file could be read and written once. Some data pages might be read and written multiple times. Again, the read/write pattern is random (that is, non-sequential). Even though the reads and writes are random, the data page could exist in the file cache and result in faster I/O throughput. Here is an example invocation of SASIOTEST:

```
   sasiotest c:\test.out -rw -pagesize 8k -filesize 1M
```

```
SAS I/O Test Utility: v2.0

Random Reads and Writes: Reading and Writing 128 pages to file: c:\test.out
with an 8192 page size.
Read/Write Throughput: 710.778588 MB/Sec.
Elapsed time: 0.001407 Secs
```

SASIOTEST will read/write a total of 128 data pages. SASIOTEST randomly reads a page from the file and then writes the same page back to the file in the same location. For each page read, the same data is written back to the file. The data is not modified by SASIOTEST. If the entire file exists in the file cache, the Read/Write throughput is optimal. Some disk I/O might occur immediately after a data page is written back to the file, but this is dependent on the file cache manager. You can experiment with this scenario by invoking the SASIOTEST examples listed for Random Reads, but use the –rw option for the last

SASIOTEST invocation in each scenario. Populate the file cache with the file and then access the data pages using the Read/Write mode:

```
 sasiotest c:\test.out -w -pagesize 64k -filesize 5G
   sasiotest c:\test.out -rw -pagesize 64k -filesize 5G
```

Adjust the file size accordingly so that the entire file resides within the file cache. The second SASIOTEST invocation randomly reads a page from the file that resides in the file cache and then writes that same page back to the file cache. The file cache manager chooses when to write the page to disk. The Read/Write throughput here is optimal with the only disk I/O occurring via the file cache manager possibly writing any pages to disk.

Next, try the scenario where the file does not exist in the file cache. That is, run these SASIOTEST invocations from the same DOS prompt consecutively:

```
 sasiotest c:\test.out -w -pagesize 64k -filesize 5G
   sasiotest c:\junk.out -w -pagesize 64k -filesize 12G
   sasiotest c:\test.out -rw -pagesize 64k -filesize 5G
```

Here are the results:

- The first SASIOTEST invocation writes the file to disk and populates the file cache. You can skip this step if the file already exists on the disk.
- The second SASIOTEST invocation writes a dummy file to the disk and populates the file cache with the dummy file. Ensure that the size of the dummy file is enough to discard all pages of the file c:\test.out, and adjust the file sizes here according to the amount of available memory on your system.
- The third SASIOTEST invocation randomly reads the data pages and then writes each page back to the file before reading the next page. The Read/Write Throughput varies based on whether a page exists in the file cache or has to be read from disk. Some of the data pages for the file c:\test.out exist in the file cache after the third SASIOTEST completes.

Note: The second preceding SASIOTEST does not have to write a dummy file to force the file cache manager to discard the pages of the file c:\test.out. You can accomplish this as well by reading a dummy file sequentially from disk. That is, your second SASIOTEST invocation could be as follows:

```
   sasiotest c:\junk.out -r -pagesize 64k -filesize 12G
```

Here, the file cache is populated with the data pages of the dummy file since SASIOTEST sequentially reads the file from disk.

A DATA step that opens a SAS data set in Read/Write (UPDATE) mode is as follows:

```
data master;
  modify master point=<nobs>;
  replace;
run;
```

Since there is no easy way to know which data pages SASIOTEST updated, the I/O pattern for the preceding DATA step is different. However, the DATA step reads the page that contained the OBS and then writes that page back to the SAS data set. This scenario is similar to the I/O performed by SASIOTEST. In both scenarios, SASIOTEST and SAS I/O open the files with the same flags.

Using SGIO with SASIOTEST to Write a File

SASIOTEST also allows you to read a file or write a file using the Scatter Read or Gather Write (that is, SGIO) functionality. When SAS reads or writes large multi-gigabyte files sequentially, the I/O throughput degrades since the file cache can grow up to the size of available memory. There is overhead in searching/finding a data page within the file cache for each I/O operation performed. SAS I/O for Windows allows you to bypass the file cache using the SGIO option.

With SAS, you specify the BUFNO= option as well as the BUFSIZE= option when using SGIO to write a SAS data set. The BUFNO= option defines the number of buffers/pages to write for each gather Write operation. The BUFSIZE= option defines the size of each page/buffer. For each gather Write operation, multiple pages are written to the file. This same functionality also exists in SASIOTEST. SASIOTEST uses the same Win32 APIs that SAS I/O uses for SGIO functionality. With SASIOTEST, use the –sgio option to use the SGIO functionality. Use the –pagesize option to define the size of each page/buffer. Use the –numpages option to define the number of buffers/pages to write for each gather Write operation. When using the sgio option, the file size must be a multiple of the product of the page size and the number of pages. To use the SGIO functionality in SASIOTEST, run the following SASIOTEST invocation from a DOS prompt:

```
 sasiotest test.out -w -sgio -pagesize 4k -filesize 2G -numpages 256
```

```
SAS I/O Test Utility: v2.0

Gather Writing 256 pages - 2048 iterations to file: test.out with a 4096 page
size.
Write Throughput: 33.330783 MB/Sec.
Elapsed time: 61.444702 Secs
```

SASIOTEST is invoked to write a 2GB file using a 4K page size and 256 pages. Hence, the file size is a multiple of the product of the page size and number of pages. SASIOTEST bypasses the Windows File Cache and writes multiple data pages to disk per I/O operation. The -numpages option specifies the number of pages written to disk per I/O operation. SASIOTEST sequentially writes the data pages to disk but writes 256 data pages per I/O operation. The results of this SASIOTEST indicate there were 2048 iterations. This is the total number of I/O operations, which in this case are Write operations. The -sgio option should be used only for writing or reading files that are at least 2GB in size.

A similar DATA step is as follows:

```
  data test1(sgio=yes bufsize=4K bufno=256);
    do i=1 to 250000000;
       output;
    end;
  run;
```

This DATA step creates a SAS data set with 250,000,000 OBS and 1 numeric variable using a 4K page size. The total file size for the SAS data set is approximately 2,000,000,000 bytes. Both SASIOTEST and SAS I/O use SGIO to write the 4K pages to the file. Each SGIO operation writes 256 data pages to the file.

Using SGIO for SAS requires some experimentation since choosing the right BUFNO= value and BUFSIZE= value is not obvious to obtain maximum I/O throughput. This is where you can use SASIOTEST to experiment with different buffer sizes and number of buffers. Let's assume your SAS data set is approximately 40GB. From a command prompt, invoke the following SASIOTEST:

```
   sasiotest test.out -w -sgio -pagesize 8k -filesize 40G -numpages 256
```

```
Observe the Write Throughput and continue invocations with various page size
values. That is,
```

```
   sasiotest test.out -w -sgio -pagesize 16k -filesize 40G -numpages 256

   sasiotest test.out -w -sgio -pagesize 32k -filesize 40G -numpages 256

   sasiotest test.out -w -sgio -pagesize 64k -filesize 40G -numpages 256
   sasiotest test.out -w -sgio -pagesize 128k -filesize 40G -numpages 256

   sasiotest test.out -w -sgio -pagesize 256k -filesize 40G -numpages 256
```

After several invocations, note the best Write throughput and try various number of pages. Once you have determined the optimal page size and number of pages, try these values with a SAS DATA step.

Using SGIO with SASIOTEST to Read a File

When reading a SAS data set using SGIO, you specify the BUFNO= option to define the number of pages SAS I/O will read per I/O operation. The size of each page is defined within the SAS data set when the data set is created, so the BUFSIZE= option has no effect. SASIOTEST allows you to scatter read multiple data pages from a file. However, SASIOTEST does not store any data in the file, so you must specify a page size when using SGIO. Requiring a page size for SASIOTEST and SGIO has its advantages though. You can run numerous SASIOTEST invocations specifying a different page size with each invocation while holding the number of pages constant. Recall that the file size must be a multiple of the product of the page size and the number of pages specified. Run the following SASIOTEST command in a DOS command prompt:

```
  sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 256

SAS I/O Test Utility: v2.0

Scatter Reading 256 pages - 4096 iterations from file: test.out with a 4096
page size.
Read Throughput: 64.339907 MB/Sec.
Elapsed time: 63.661889 Secs
```

With SGIO, the file cache is not used, so each Read operation reads numerous pages of data from the disk.

Here is an example DATA step that uses SGIO to read the SAS data set:

```
   data _null_;
      set test1(sgio=yes bufno=256);
   run;
```

The assumption is the SAS data set, test1, was created with a page size of 4K and approximately 4GB of data. This ensures that approximately 4096 total scatter Read operations are performed. Recall that the page size for a SAS data set is defined when the data set is created. However, the number of pages can vary with different invocations of the DATA step. Both SASIOTEST and SAS I/O use SGIO to read the data pages from the file.

Using SGIO to scatter read the data from a file requires some experimentation. However, using SASIOTEST makes this process simpler because you don't have to re-create the file to change the page size when using SASIOTEST. Run several invocations varying the page size but hold the number of pages constant:

```
   sasiotest test.out -r -sgio -pagesize 8k -filesize 4G -numpages 256

   sasiotest test.out -r -sgio -pagesize 16k -filesize 4G -numpages 256

   sasiotest test.out -r -sgio -pagesize 32k -filesize 4G -numpages 256

   sasiotest test.out -r -sgio -pagesize 64k -filesize 4G -numpages 256

   sasiotest test.out -r -sgio -pagesize 128k -filesize 4G -numpages 256

   sasiotest test.out -r -sgio -pagesize 512k -filesize 4G -numpages 256
```

Notice how a different page size affects the Read throughput. Notice the larger page sizes require fewer scatter Read operations to read the 4GB file. Next, vary the number of pages holding the page size constant:

```
   sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 512
   sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 1024
   sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 2048
   sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 4096
   sasiotest test.out -r -sgio -pagesize 4k -filesize 4G -numpages 8192
```

Now create a much larger file that allows you to experiment with page sizes over 1MB as well as specify the number of pages over 10,000.

Using SASIOTEST with Unbuffered I/O

As explained in previous examples, SASIOTEST is used to sequentially read or write a single page of data. Both sequential reading and sequential writing data uses the file cache, which can improve the I/O throughput significantly. By using SASIOTEST and SGIO, you can bypass the file cache, but SASIOTEST uses numerous pages for each I/O operation. And with SGIO, there are other differences. In addition, SASIOTEST allows you to sequentially read or write a single page per Read operation and eliminate the file cache with unbuffered I/O. Using SASIOTEST and the -unbuffered option sequentially reads or writes a single page of data and bypasses the file cache. Each page of data is read directly from the disk or written directly to disk.

Note: The functionality of the -unbuffered option is not used in SAS I/O. The functionality is provided in SASIOTEST simply for experimental purposes.

You might want to experiment with unbuffered reads and writes to see the effect on your system. To read data unbuffered with SASIOTEST, run the following command:

```
   sasiotest test.out -r  -pagesize 8k -filesize 4G -unbuffered
SAS I/O Test Utility: v2.0

Sequentially reading 4294967296 bytes from file: test.out with a 8192 page
size.
Read Throughput: 55.080833 MB/Sec.
Elapsed time: 74.363436 Secs
```

To write data unbuffered with SASIOTEST, run the following command:

```
   sasiotest test.out -w -pagesize 8k -filesize 4G -unbuffered
SAS I/O Test Utility: v2.0

Sequentially writing 4294967296 bytes to file: test.out with a 8192 page
size.
Write Throughput: 32.051893 MB/Sec.
Elapsed time: 127.792765 Secs
```

In these examples, SASIOTEST uses an 8K page to read or write the data. Since this is unbuffered I/O, the file cache is not searching for any of the pages of data. This is a direct read from the disk or a direct write to the disk, so the I/O throughput is obviously slower.

Using SASIOTEST and Read with History

SASIOTEST and unbuffered I/O enable you to experiment with random reads. When the –ran option and the –unbuffered option are specified, SASIOTEST opens the file for reading, but the access pattern is neither sequential nor random. This access pattern for reading data is called Read with History. Using the Read with History access pattern uses the file cache. However, the file cache manager fetches additional data pages into the file cache based on the pages read from the file. Hence, the read-history (that is, pages read from the file) controls which data pages the file cache manager reads into the file cache. The file cache manager does not read any data pages into the file cache until the first few pages of data are read from the file. An example of using the read with history access pattern with SASIOTEST is as follows:

```
   sasiotest test.out -r -ran -pagesize 8k -filesize 4G -unbuffered
```

```
SAS I/O Test Utility: v2.0

Read with History: Reading 524288 pages from file: test.out with a 8192 page
size.
Read Throughput: 7.700519 MB/Sec.
Elapsed time: 531.912234 Secs
```

The performance for the Read with History I/O is unpredictable. The data pages read from the file are read in a random order as described for the –ran option, but the file cache is not populated the same by the file cache manager. Given this, there is a greater chance the data page to be read is not in the file cache and must be read from disk. You can use this access pattern to compare the random Read throughput.

Since unbuffered reads and writes are not supported in SAS I/O, there are no equivalent DATA step examples for any of these unbuffered examples with SASIOTEST.

Using SASIOTEST to Write a File with Writethru Option

When writing data sequentially with SASIOTEST, a single page of data is written to the file cache as shown in previous examples. By writing the data to the file cache, the Write throughput improves significantly. Actually, the file cache manager writes the data in large blocks (that is, numerous data pages) to the disk when appropriate. However, forcing the data to disk when the write occurs can be achieved using SASIOTEST and the -writethru option. Using SASIOTEST and the -writethru option writes a single page of data to the file cache, and then the data is written to disk. There’s no delay in writing the data by the file cache manager.

Note: The functionality of the -writethru option is not used in SAS I/O. The functionality is provided in SASIOTEST simply for experimental purposes.

To use SASIOTEST and the -writethru option, run the following command:

```
   sasiotest c:\test.out -w -pagesize 64k -filesize 2G –writethru
```

```
SAS I/O Test Utility: v2.0

Sequentially writing 2147483648 bytes to file: c:\test.out with a 65536 page
size.
Write Throughput: 39.501309 MB/Sec.
Elapsed time: 51.846383 Secs
```

Here, SASIOTEST sequentially writes the 2GB file using a 64k page, but the –writethru option forces the data pages to be written to the Windows File Cache and then written directly to disk. This results in slower throughput. You might want to experiment with the -writethru option to see how it affects your system.

Multiple Invocations of SASIOTEST

A single invocation of SASIOTEST can be used to measure the I/O throughput for a file. However, there are other ways to use SASIOTEST that might benefit you. You can simultaneously run multiple SASIOTEST invocations from different DOS command prompts or run multiple SASIOTEST sessions with a .bat file. This allows you to measure the effects of multiple processes accessing the same disk drive. Here are some examples of simultaneously running multiple SASIOTEST invocations:

- Simultaneously run two or more invocations of SASIOTEST accessing distinct files on the same disk drive to measure the Write throughput. Launch multiple DOS command prompts. Invoke SASIOTEST creating new distinct files to the same drive with each invocation. For example, three invocations of SASIOTEST from different DOS command prompts creating three distinct files on drive C:

  ```
  1. sasiotest c:\output_file1 -w -pagesize 256k -filesize 3G
  2.  sasiotest c:\output_file2 -w -pagesize 256k -filesize 3G
  3.  sasiotest c:\output_file3 -w -pagesize 256k -filesize 3G
  ```

  Since the I/O will target the same disk drive, this gives you an indication of how writing to multiple files on the same disk drive affects your system's throughput. Also, try different page size values for each invocation as well as different file sizes.

- Simultaneously run multiple invocations of SASIOTEST to the same file to measure the Read throughput. Here are three invocations of SASIOTEST simultaneously reading data from the same file:

  ```
  1. sasiotest c:\output_file1 -r -pagesize 256k -filesize 3G
  2.  sasiotest c:\output_file1 -r -pagesize 256k -filesize 3G
  3.  sasiotest c:\output_file1 -r -pagesize 256k -filesize 3G
  ```

  When SASIOTEST opens a file for Read mode, the file is opened so that multiple processes can also open the file for reading the data. This is consistent with SAS I/O on Windows when reading a SAS data set.

  Note: Only one process is allowed to write to the same file with SASIOTEST.

- Simultaneously run multiple invocations of SASIOTEST to read and/or write but access different disk drives. SAS on Windows allows this as well. The WORK library and SASUSER library can be defined to different paths as well as disk drives. Determine if this type of I/O is a performance degradation on your system.

- Simultaneously run multiple invocations of SASIOTEST to read and/or write but access different network paths. While reading and writing to a file that exists on a network volume is permitted, keep in mind that the I/O throughput is dependent on your network. Remember, the filename specified can contain a UNC path as well as a disk drive. Here is an example:

  ```
    sasiotest \\myvol\rnd\testing\master_output_file1 -r -pagesize 256k -
  filesize 3G
  ```

- Using SGIO, run SASIOTEST with a constant file size but vary the page size and the number of pages to see how this affects I/O throughput. A different page size might be better for drive D than what you use for drive C. Likewise, more pages and a smaller page size might be a better choice for your disk drive. When using SGIO, vary only one of the parameters for a series of SGIO runs. That is, keep the file size and page size constant, and try various number of pages.

- Experiment with random reads and random writes using multiple SASIOTEST invocations to see if this type of file I/O has a large effect on I/O throughput.

Finally, experiment with a mixture of random and sequential I/O using multiple SASIOTEST invocations.

CONCLUSION

As you can see, SASIOTEST is a valuable tool. When you need to measure I/O throughput without using the DATA step or other SAS procedures, SASIOTEST is an excellent choice. SASIOTEST can be used to measure the I/O throughput for any disk drive accessible to your Windows system as well as for reading and writing files across your network.

Reading or writing a file sequentially using SASIOTEST places the file in the Windows File Cache. The file remains in the file cache until the file is deleted from disk or the Windows File Cache Manager needs the memory for a new file.

Sequentially reading or writing data when all data resides in the file cache affects the I/O throughput in a positive manner. Usually, the throughput is greatly increased beyond the drive's capabilities. If you experience this effect, then use SASIOTEST to populate the file cache with a file or multiple files. Then, try reading the original file from disk to get a more accurate I/O throughput.

As for SGIO, you should try SGIO if you are reading or writing SAS data sets larger than 2GB. While the file cache does a good job of managing data pages for files that are less than 2GB, the latency in searching the file cache increases for files larger than 2GB. Use SASIOTEST and SGIO to measure the I/O throughput for files greater than 2GB and compare to the throughput when using the file cache.

As previously mentioned, the -unbuffered and -writethru options do not exist in SAS I/O. These options were implemented in SASIOTEST to allow you to experiment with them if you wish. This is direct reading or writing data with no file cache assistance.

Also, SASIOTEST is a stand-alone utility. This means you can copy SASIOTEST to new systems where SAS has not been installed and run some I/O tests.

For more information about SGIO on Windows, see the paper listed in the "References" section. The second half of this paper discusses SGIO on Windows in much greater detail.

REFERENCES

Ihnen, Leigh, and Mike Jones. 2009. "Improving SAS® I/O Throughput by Avoiding the Operating System File Cache." *Proceedings of the SAS Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings09/327-2009.pdf.

USAGE

SASIOTEST *filename* [*options*]

where:

filename – This option is required and specifies the filename to read or write. The filename MUST be the first argument.

options – Specify one or more of the following SASIOTEST options:

-r - Specifying -r will cause SASIOTEST to read from the specified file. –r is the default.

-w - Specifying -w will cause SASIOTEST to write to the specified file.

-rw – Specifying –rw will cause SASIOTEST to read a page from the specified file and then write that page to the specified file. NOTE: For -rw, the reads are always random reads of a page and the writes are always writing the same page that was read.

-seq - Specifying -seq will cause SASIOTEST to read the data sequentially from the specified file.

-ran - Specifying –ran will cause SASIOTEST to read the data randomly from the specified file. The default access pattern is –seq.

-filesize N - defines the total size of file to be read or created by SASIOTEST. For example, 800k, 512K, 5m, 40m, 2G, 10g, etc. The default file size is 1MB.

-pagesize N - defines the size of the page for each I/O operation. SASIOTEST will read or write this number of bytes. For example, 8k, 128K, 256k, 1M, etc. The default page size is 8k.

-sgio - causes SASIOTEST to use the Scatter Read and/or Gather Write APIs for I/O operations. Using these APIs will force the data to be read from disk or a write to disk avoiding the Windows File Cache. When using the -sgio option, the page size must be a multiple of 4K on Windows x86 systems and a multiple of 8K on Windows x64 systems.

-numpages M - specifies the number of pages that SASIOTEST will read or write for each SGIO operation. This option is REQUIRED if -SGIO option present.

-unbuffered - causes SASIOTEST to read or write pages bypassing the file cache. If -r, -ran, and -unbuffered are specified, the file will be read randomly and opened without the RANDOM ACCESS flag.

-writethru - causes SASIOTEST to write the pages through the Windows File Cache and directly to disk.

CONTACT INFORMATION

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

Mike Jones
100 SAS Campus Drive
SAS Institute Inc.
Cary, NC 27513
919-531-7939
Email: Mike.Jones@sas.com