Paper 479-2013

Benchmarking SAS® I/O: Verifying I/O Performance Using fio

Spencer Hayes, DLL Consulting

ABSTRACT

Input/Output (I/O) throughput is typically the most important computing aspect of a SAS® environment. Bandwidth requirements ranging from 25MB/sec/core to 135MB/sec/core are common in a high-performance SAS system. Insuring that the storage subsystem can meet the demands of SAS is critical to delivering the performance required by the business and user community. Ideally, SAS administrators could run real-world SAS jobs to benchmark the I/O subsystem. However, technical and logistical challenges frequently make that option impractical. The open source software tool called "fio" provides a method for accurately simulating I/O workloads. It is configurable to match closely the existing or expected I/O profile for a SAS environment.

INTRODUCTION

The author's goal of this paper is to explain how to perform accurate, repeatable I/O benchmarking using the fio tool.

The paper will discuss I/O performance and how it relates to SAS. It will review high level SAS architecture and discuss why I/O is a common bottleneck. Further it will explore the various components of the I/O subsystem and how each may impact performance.

This document will cover the fio tool and how it operates. The paper will review a typical SAS workload and how to create a fio job file to simulate an example SAS program. It will describe the details of the fio job file and how to customize a job file for a target I/O workload. It will review the output from fio and explain the various fields. It will describe how to evaluate a benchmarked system for adequate I/O performance. Finally, the paper will compare the fio results to an actual SAS program to validate that the simulated workload is accurate.

I/O: INPUT AND OUTPUT

At a high level, I/O is simply any data going into or coming out of a computer system. I/O throughput is most commonly discussed in reference to network or storage (disk) bandwidth. Certain situations may also raise the topic of memory bandwidth between the CPU and physical memory, although not as frequently.

Two common bottlenecks in computing occur when moving data across a network between computer systems and when moving data between physical memory and disk within a single system. Such a condition is referred to as being "I/O bound" or in other words, the application is constrained by available I/O bandwidth. SAS in particular is very often storage I/O bound.

SAS ARCHITECTURE AND DISK I/O

SAS is designed to read and write data sets one record at a time. The advantage of this approach is that SAS is able to process and create data sets of seemingly unlimited size. The drawback of the design -- compared to traditional in-memory applications such as R or Microsoft Excel -- is that SAS demands a large amount of consistent, sustained storage I/O bandwidth. As data sets scale into the multi-terabyte and petabyte range, the requirements for bandwidth will continue to increase.

The filesystem that holds the SASWORK library is very sensitive to I/O throughput. SAS uses the WORK library to store temporary files created by users, as well as internal temporary system files. By default, intensive operations, like sort and merge, will utilize the WORK library for intermediate steps including writing the entire source data sets before performing the operation.

Because the bandwidth between the system's Central Processing Unit (CPU) and its physical memory is many times greater than the bandwidth to disk, the "one record at a time" design creates the situation where the CPU is idle or underutilized as it waits for data to be written to or read from disk. Increasing the storage I/O throughput will increase the amount of data available for the CPU to process and in-turn increase SAS job performance.

STORAGE SUBSYSTEM COMPONENTS

The layout of a storage subsystem is fundamentally the same on all computer systems from a small desktop PC to a large enterprise class server connected to a Storage Area Network (SAN). While the performance, reliability and security characteristics may change as the storage system scales up, all disks appear similarly at the Operating System (OS) level.

Each component in the chain from disk to computer system may impact performance.

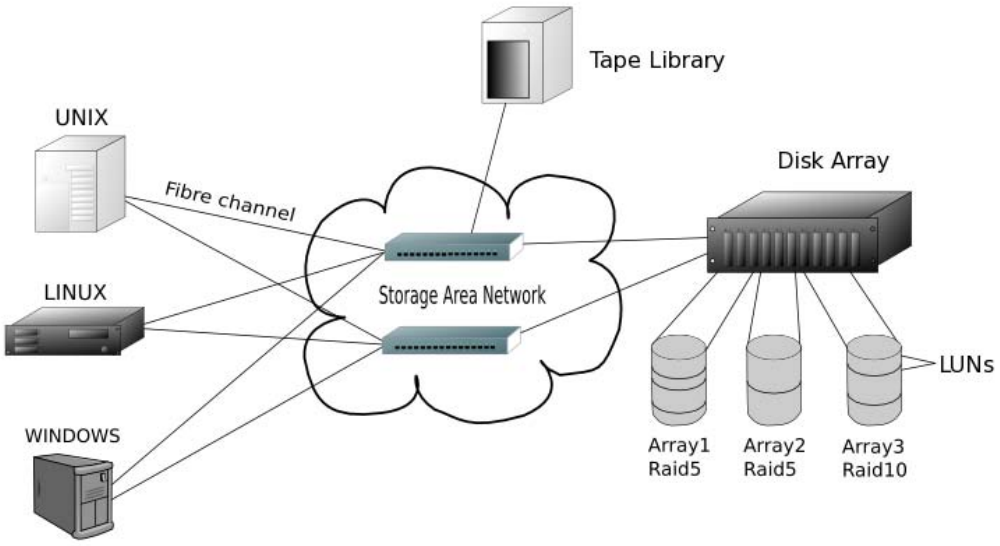

Figure 1: Layout of a typical Storage Area Network (SAN)

The performance of traditional spinning disks is largely dependent on the rotational speed of the platters and the time it takes to move the head to the correct position. 7200 RPM disks are common in desktop PC's while 10k or 15k RPM disks are typically configured in large enterprise storage arrays. Spinning disks perform very well for single, sequential read and write operations. However, when multiple users access the same disk, performance rapidly degrades due to the seek time associated with moving the head from place to place on the disk. Combining or pooling many disks into an array is a common way to increase storage I/O performance by aggregating the bandwidth of several disks at once.

A "bus" connection attaches disks to computer systems. Examples of a bus connection are the internal wiring for locally-attached disks within a server, or fiber optic cables running from a storage array to a SAN switch or from the SAN switch to a Host Bus Adapter (HBA) card in a server. The bandwidth of the bus depends on the technology used. Fibre Channel, Serial Attached SCSI and Serial ATA are three common bus technologies.

OS level components may also affect I/O performance. Multipathing software allows a host to send data to a disk via multiple bus connections. Volume management software such as Veritas Volume Manager or Linux LVM may be used to create logical volumes that combine multiple disks into a higher-level device on which a filesystem is built. A common volume type is a stripe which reads and writes chunks of a file to each disk in order, effectively aggregating the performance of multiple disks.

Finally, the filesystem itself contributes to system performance. Various filesystems have different characteristics, some of which may or may not make them well suited for a SAS system. On RedHat Enterprise Linux, the ext3, ext4 and xfs filesystems are available, but the SAS best practices dictate that only ext4 and xfs should be used due to performance impacts from the ext3 journaling feature.

FIO: FLEXIBLE I/O TESTER

Benchmarking I/O is an important, but often difficult exercise. Storage configurations will vary greatly from environment to environment and even between applications. Often corporate IT standards will determine the baseline storage configuration, but that configuration may not be optimized for a SAS environment. To insure high performance for SAS, it is crucial to determine whether or not the storage subsystem is up to par. However, given high degrees of variation and complexity, making this determination can be a challenge.

The fio (pronounced fee-yo) tool is open source software and under active development. fio provides a method for simulating arbitrary I/O workloads in an accurate manner. The tool is driven off of a job file that specifies how many processes to spawn and what kind of work each process must do. It is possible to have a single job performing one type of I/O, many jobs performing the same type of I/O or many jobs performing all different types of I/O.

fio is available under the GPLv2 license at http://freecode.com/projects/fio and supports Linux, Solaris, AIX, HP-UX and Windows operating environments among others. Pre-built packages may be available, but most users will simply download the source code and compile a local version. Documentation is provided in a UNIX "man" page as well as

in the README and HOWTO files within the source archive.

FIO JOB FILE FORMAT

The HOWTO document included with the tool provides a clear and detailed description of all the settings available for fio as well as how to configure the job file. From section 4.0 of the HOWTO:

The job file format is the classic ini file, where the names enclosed in [] brackets define the job name. You are free to use any ascii name you want, except 'global' which has special meaning. A global section sets defaults for the jobs described in that file. A job may override a global section parameter, and a job file may even have several global sections if so desired. A job is only affected by a global section residing above it. If the first character in a line is a ';' or a '#', the entire line is discarded as a comment.

The following is an example file with a single job that will perform a 100MB sequential read operation:

```
[job1]
rw=read
size=100m
directory=/sasdata1
```

In this example, we set up a single job and defined all the options within the [job1] section. In order to simulate real-world workloads, we will often want to have multiple jobs executing concurrently. The next example shows a fio job file with three jobs. The first sequentially reads a 100MB file, the second randomly reads a 20MB file and the third sequentially writes a 50MB file:

```
[global]
directory=/sasdata1

[seq_read]
rw=read
size=100m

[rand_read]
rw=randread
size=20m

[seq_write]
rw=write
size=50m
```

In this example, we use the special [global] section to define options that apply to all jobs. In this case we want each job to locate their files in the /sasdata1 directory. The amount of tuning offered by fio's job file format provides very granular control over the test I/O workloads.

CREATING A JOB FILE TO SIMULATE A SAS WORKLOAD

Generally SAS I/O workloads are sequential reads and writes, and fall into the percentage range of 50/50 to 60/40 reads versus writes. For the purposes of this presentation, we will use the 50/50 read/write split. SAS will automatically adjust the I/O size based on the data set sizes. For data sets larger than a few MB, SAS will use 128KB chunks.

The following SAS program interleaves two 834MB data sets to create a third data set of 1.7GB:

```
data work.censusmerge;
  set work.alla work.allb;
  by serialno;
run;
```

The following fio job file will simulate the SAS program above:

```
[interleave]
directory=/sasdata1
direct=0
invalidate=1
blocksize=128k
rw=readwrite
size=3335m
```

The [interleave] job section defines the directory to use as well as the 'direct=0' and 'invalidate=1' options. The 'direct=0' option tells fio to use the OS file system cache for reads and writes. In a real-world scenario, there's no guarantee that the data sets will reside in the OS cache. To account for this, the 'invalidate=1' option is provided to invalidate the cache for the fio files prior to starting I/O. SAS will take advantage of the OS file system write cache and that usage accurately reflects the actual storage subsystem performance in the majority of scenarios. To simulate a true worst-case scenario where the OS file system cache is full, use "direct=1" to bypass it.

The 'blocksize=128k' section specifies 128KB blocks for I/O. The 'rw=readwrite' option sets the I/O pattern to mixed sequential reads and writes. The default mix for 'rw=readwrite' is 50% reads, 50% writes.

The total data read will be 1.7GB and the total data written will be 1.7GB. Using the 'rw=readwrite' I/O pattern, fio concurrently reads 1.7GB and writes 1.7GB of data. That characteristic is important given the SAS design of "one record at a time". SAS will read one record from each BY group in each data set, then output the appropriate record to the new data set, read another row, write another row and repeat until the program completes.

ANALYZING OUTPUT FROM FIO

To execute the test, we simply run the fio program with the job file as the first argument. The following examples show the output from a desktop system running RedHat Enterprise Linux. The first output has the contents of the job file as well as the intermediate output while fio is executing:

```
[shayes@gordon ~]$ cat sgf_demo2.fio
[interleave]
directory=/sasdata1
direct=0
invalidate=1
blocksize=128k
rw=readwrite
size=3335m

[shayes@gordon ~]$ /opt/fio_2.0.9/bin/fio sgf_demo2.fio
interleave: (g=0): rw=rw, bs=128K-128K/128K-128K, ioengine=sync, iodepth=1
fio-2.0.9
Starting 1 process
Jobs: 1 (f=1): [M] [14.8% done] [80446K/87585K /s] [628 /684  iops] [eta 00m:23s]
```

The first line of output provides the job name as well as several parameters and their values. The second and third lines state the version of fio and how many processes (jobs) are configured.

The fourth line provides data on the status of the job and is refreshed slightly faster than once per second. The number of currently running jobs is displayed as well as their job types in the first set of brackets. In this case our mixed sequential read/write job is represented by an 'M'. The remaining sections show the percentage complete of the total fio program, the average amount of data being read and written per second, the read and write operations per second and finally the estimated time left to complete the program.

Once the fio program completes, a significant amount of detailed, verbose output is displayed. The output has been trimmed for readability:

```
interleave: (groupid=0, jobs=1): err= 0: pid=13774: Fri Sep 21 16:16:13 2012
  read : io=1676.0MB, bw=36839KB/s, iops=287 , runt= 46587msec
...
  write: io=1659.0MB, bw=36465KB/s, iops=284 , runt= 46587msec
...

Run status group 0 (all jobs):
   READ: io=1676.0MB, aggrb=36839KB/s, minb=36839KB/s, maxb=36839KB/s,
mint=46587msec, maxt=46587msec
  WRITE: io=1659.0MB, aggrb=36465KB/s, minb=36465KB/s, maxb=36465KB/s,
mint=46587msec, maxt=46587msec

Disk stats (read/write):
    dm-6: ios=25653/373464, merge=0/0, ticks=68580/181800744, in_queue=181869348,
util=89.15%, aggrios=13118/6696, aggrmerge=12993/366972, aggrticks=64771/3232191,
aggrin_queue=3296957, aggrutil=90.39%
  sda: ios=13118/6696, merge=12993/366972, ticks=64771/3232191, in_queue=3296957,
util=90.39%
```

The key data points to analyze are in bold. The first bolded line denotes the beginning of the section for the specific job and the statistics pertaining to it. If multiple jobs are used, there will be multiple sections containing data about each individual job. The designation of the start of the job-specific section and the timestamp are the two important pieces of data for this line.

The next two bolded lines provide data describing the read and write performance for the job. The 'io=*' field indicates the amount data transferred. The 'bw=*KB/s' field describes the bandwidth of the read operation in terms of KB per second. Understand that these statistics are only for this one job and may vary greatly if multiple jobs are running concurrently. The 'iops=*' field describes the number of I/O operations per second ("IOPS"). This statistic is generally more useful when observing high transaction-rate systems such as Online Transaction Processing ("OLTP") databases. SAS performance tuning does not typically target IOPS as a main factor. Finally the run time of the job is listed in the 'runt= *msec' field.

The lines at the bottom of the output are the most important for the majority of test cases. These summarize and aggregate the data for all the jobs executed. In this example, the data matches the individual read and write lines contained in the "interleave" job section above. However, when executing multiple jobs, these lines provide the overview of the total throughput and utilization.

In the summary section the 'io=*' fields designate the total amount of data read and written for all jobs. The 'aggrb=*' fields are the average bandwidth of all jobs and are the key to understanding the storage subsystem performance. In this example, these numbers add up to 73304KB/s or approximately 72MB/s. As noted above, SAS requires I/O bandwidth in the range of 25-135MB/s. The "Disk Stats" section indicates that the disk was 90.39% utilized during our job. Any additional concurrent jobs would very likely saturate the disk and degrade overall performance on this small system. This desktop Linux workstation appears to be sufficient as a single-user SAS system with data set sizes of a few GB.

COMPARING FIO TO A REAL-WORLD SAS PROGRAM

A simulation tool is only valuable if the simulated workload is comparable. The following scenario was created to validate that fio produces the same I/O patterns as a real-world SAS program. The two source data sets, work.alla and work.allb, use census data from the census.gov website.

```
sas /sasdata1/work/SAS_work1E5700005C87_gordon > ls -alhtr
total 4.0G
...
-rw-rw-r-- 1 sas sas 834M Sep 21 16:38 alla.sas7bdat
-rw-rw-r-- 1 sas sas 834M Sep 21 16:38 allb.sas7bdat
```

The execution of the following SAS data step creates the merged data set work.censusmerge:

```
data work.censusmerge;
  set work.alla work.allb;
  by serialno;
run;
```

The program produced the following output in the SAS log:

```
2917  data work.censusmerge;
2918    set work.alla work.allb;
2919    by serialno;
2920  run;

NOTE: There were 3877316 observations read from the data set WORK.ALLA.
NOTE: There were 3877316 observations read from the data set WORK.ALLB.
NOTE: The data set WORK.CENSUSMERGE has 7754632 observations and 28 variables.
NOTE: DATA statement used (Total process time):
      real time           46.93 seconds
      user cpu time       6.85 seconds
      system cpu time     9.89 seconds
      memory              487.13k
      OS Memory           10720.00k
      Timestamp           09/21/2012 05:44:25 PM
      Page Faults                      0
      Page Reclaims                    0
      Page Swaps                       0
      Voluntary Context Switches       5730
      Involuntary Context Switches     8520
      Block Input Operations           0
      Block Output Operations          0
```

As well as the merged data set:

```
sas /sasdata1/work/SAS_work1E5700005C87_gordon > ls -alhtr
total 4.0G
...
-rw-rw-r-- 1 sas sas 1.7G Sep 21 17:44 censusmerge.sas7bdat
```

The SAS program interleaving two data sets required 46.93 seconds to complete. The fio job above simulating this SAS program took 46587 milliseconds, or 46.59 seconds. Subsequent runs of both the SAS program and the fio job will produce slight variations on the program durations, but overall the fio job appears to be a very close fit for the example SAS scenario it was designed to simulate.

As an additional test of scalability, the scenario was re-run with larger data sets. The second test used source data sets of 4.1GB to produce a merged data set of 8.2GB. fio was configured with a job file identical to the first with the exception of the ‘size=16674m’ option. The same SAS code along with a larger sample of the census data were used to produce the SAS data sets.

The SAS program finished in 3:59.35 or roughly 239 seconds. The fio job completed in 235485msec or roughly 235 seconds. As before, the fio job appears to simulate accurately the real-world SAS program.

CONCLUSION

SAS performance tuning is a tricky and difficult problem to solve. Without the best tools and methodologies, it can be a black hole of lost time and productivity. However, administrators can quickly identify bottlenecks in storage I/O by using fio to model and evaluate this major constraint on SAS performance. In addition fio provides a repeatable process that allows businesses not only to benchmark baseline configurations, but also compare performance against new IT purchases and quantify the return on investment of capital expenditures.

REFERENCES

Augustine, Bob. 2012. “Storage 101: Understanding Storage for SAS® Applications “. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings12/416-2012.pdf

Axboe, Jens. 2012. “fio HOWTO”. Available at “http://git.kernel.dk/?p=fio.git;a=blob;f=HOWTO”

Figure 1. Moll, Michael. 2006. “Schema of a Storage Area Network (SAN)”. Available at http://commons.wikimedia.org/wiki/File:Schema_SAN_german.png

ACKNOWLEDGMENTS

I would like to thank Bob Augustine for writing the cited paper above that gave me the inspiration for this paper. Also thanks to Jens Axboe and the fio developers for creating an outstanding and useful tool.

Credit also goes to Don Hayes, Rebecca Hayes, Jennifer Hayes, Roger Hayes and Deborah Hayes, also known as my dad, sister, wife, uncle and mom, several of whom either are or plan to be SAS professionals as well. Their encouragement and support was invaluable. Finally, thanks to Jeff Holoman and Jeremy Reynolds for their valuable input and advice.

CONTACT INFORMATION

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

Name: Spencer Hayes
Enterprise: DLL Consulting
City, State ZIP: Johns Creek, GA 30005
Work Phone: 404-668-0830
E-mail: spencer.hayes@dllbi.com