Evaluating Tools that Measure Storage Device Performance

By Angelo Di Sena, Software Developer Manager
Procolo Carannante, Software Engineer
Fabio Salvati, Software Engineer
Massimo Cirillo, Software Engineer

The number of available storage device options has continued to grow and now includes hard disks, Flash memory, *e*·MMC™, solid state drives (SSDs), phase change memory (PCM), and so forth. The importance of storage devices to the overall performance of a system is also growing due to the increasing speed of systems in terms of CPU speed, network bandwidth, and graphical rendering. More and more, the storage device represents the bottleneck in the system, making evaluations of its performance (and consequent tuning) a crucial activity.

Evaluating the performance of a storage device is often considered a straightforward, if not time-consuming task. Executing a set of performance tests produces objective measurements, giving a definitive picture of device performance.

While there is a general consensus on performance measurements for storage devices, there are several ways to generate such measurements that can produce different results on the same storage medium. This discrepancy is mainly due to the different tools used to evaluate performance and the method the tool uses to generate performance measurements.

This article describes common measurements used to evaluate device performance and some typical tools used to measure the performance of a storage device.

Typical Performance Tests

To assess Flash memory performance, a set of tests is executed to produce specific performance measurements. Typical performance tests include:

- **Sequential READ**: A previously written file (typically greater than or equal to 16MB) is read starting from offset 0, and the offset is increased by the sector size until the end of the file is reached. At the end of process, the sequential READ throughput is calculated.
- **Random READ**: Multiple sectors are read from a previously written file. The starting offset of the sector to be read is randomly selected. Some tools (such as IOzone and fio) can be used to set the sector size (see “Sector/Chunk Size Selection”).
- **Sequential WRITE**: A file (typically greater than or equal to 16MB) is sequentially written and sectors of the same size are continuously written, appending each sector to the previous one.
- **Random WRITE**: Multiple sectors of the same size are written in an empty region. The starting offset of each sector to be read is randomly selected. Some tools (such as IOzone and fio) can be used to set the sector size (see “Sector/Chunk Size Selection”).
- **Sector/Chunk Size Selection**: As previously stated, some tools (such as IOzone and fio) perform tests at the file system or Flash translation layer (FTL) level. This enables a user to set the sector/chunk size of the test session. The sector/chunk size

is important both when choosing the correct file system or FTL and when choosing the optimum cache buffer size.

Choosing the Right File System/FTL

When optimizing the performance of an application that writes to Flash using a fixed chunk size, running performance benchmarks on several file systems/FTLs using a fixed sector/chunk size could be the optimal way to choose the best file system/FTL for that application.

With a single IOzone test session (using the `iozone –a` syntax), you can quickly evaluate performance results for each sector size by reading the Microsoft® Excel file provided as an output by the tool.

Choosing the Optimum Cache Buffer Size

A cache buffer can reduce the number of I/O memory accesses when using an operating system on a particular hardware platform. However, an incorrect configuration can generate a significant number of unnecessary memory operations, causing performance to decrease. When this occurs, running tests on a file system/FTL and changing the sector/chunk size for each test session is the best way to identify and set the correct buffer size and optimize performance.

Typically, performance values are expressed in two measurement units:

- KB/s
- IOPs

IOPs is the number of I/O operations per seconds, which is used to measure random READ/WRITE throughput. For example, with a chunk size of nKB, the relationship between IOPs and KB/s is: KB/s = IOPs x n chunk size.

Performance Measurement Tools

A number of tools can be used to measure storage device performance. In the following sections we discuss:

- IOzone
- Iometer
- dd
- fio

We provide a comparison of these tools in Table 4 on page 12.

IOzone Overview and Features

IOzone is a file-system-level benchmark tool that is useful when selecting the correct file system and/or operating system for a particular hardware platform. IOzone is also used to measure any changes in performance after optimization/modifications in any operating system or hardware platform.

IOzone features include:

- POSIX asynchronous I/O
- Mmap() file I/O
- Normal file I/O
- Single stream measurement
- Multiple stream measurement
- Distributed file server measurements (cluster)
- POSIX Pthreads
- Multi-process measurement
- Output can be imported into Excel for graph generation
- Latency plots
- 64-bit compatible source
- Large file compatible
- Configurable processor cache size

IOzone Performance Tests

IOzone measures and evaluates I/O performance using the following tests:

Table 1. IOzone Performance Tests

Test	Description
Read	Reads an existing file.
Write	Writes a new file.
Re-read	Reads a file that was recently read.
Re-write	Writes a file that already exists.
Read Backwards	Reads a file backwards.
Read Strided	Reads a file with a strided access behavior. For example, read at offset 0 for a length of 4KB, seek 200KB, read for a length of 4KB, seek 200KB, and so forth.
Fread	Reads a file using the fread() library function, which is a library routine that performs buffered and blocked READ operations.
Fwrite	Writes a new file using the fwrite() library function, which is a library routine that performs buffered WRITE operations.
Freread	Reads a file using the fread() library function that was recently read.
Frewrite	Writes an existing file using the fwrite() library function.
Random Read	Reads a file with accesses being made to random locations within the file.
Random Write	Writes a file with accesses being made to random locations within the file.
Aio_read	Reads a file using an asynchronous I/O POSIX mechanism.
Aio_write	Writes a file using an asynchronous I/O POSIX mechanism.
Mmap	Uses the mmap() mechanism for performing I/O.

Supported Level of Automation

IOzone is a command line tool. Execution is automated using:

- `Iozone –a`

This option enables all tests listed in Table 1 to be run by performing I/O operations on files from 64KB to 512MB using chunk sizes from 4KB to 16MB. The command line option is also used to run specific tests with different file and chunk sizes.

In addition, IOzone generates graphs and Excel spreadsheets that can be exported and analyzed. Visit www.iozone.org for a complete list of options.

Supported Operating Systems

IOzone is an open source project with binary files built for:

- Oracle® AIX
- BSDi
- HP® HP-UX
- IRIX
- FreeBSD
- Linux
- OpenBSD
- NetBSD
- OSFV3
- OSFV4
- OSFV5
- SCO® OpenServer®
- Oracle® Solaris
- Apple® OS X
- Microsoft® Windows (95/98/Me/NT/2K/XP)

For other operating systems or hardware-specific platforms, source code can be downloaded and compiled from www.iozone.org.

Iometer Overview and Features

Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. It measures performance under a controlled load. Iometer is both a workload generator (it performs I/O operations to stress the system) and a measurement tool (it examines and records the performance of its I/O operations and their impact on the system). It can be configured to emulate the disk or network I/O load of any program or benchmark, or can be used to generate entirely synthetic I/O loads.

Iometer is considered the de facto tool for generating and measuring storage performance and is typically used to compare the performance of different disk drivers.

Iometer measures and characterizes:

- Performance of disk and network controllers
- Bandwidth and latency of buses
- Network throughput to attached drives
- Shared bus performance
- System-level hard drive performance
- System-level network performance

Iometer consists of two programs.

- **Iometer**: The controlling program. Iometer's graphical user interface is used to configure the workload, set operating parameters, and start and stop tests. Iometer provides instructions to Dynamo, collects the resulting data, and summarizes the results in output files. Only one copy of Iometer should be running at a time. Iometer is typically run on the server.
- **Dynamo**: The workload generator. It has no user interface. At Iometer's command, Dynamo performs I/O operations and records performance information, then returns the data to Iometer. More than one copy of Dynamo can be running at a time. Typically, one copy runs on the server and one additional copy runs on each client machine. Dynamo is multithreaded and each copy can simulate the workload of multiple client programs.

Supported Performance Tests

- Sequential and random READ/WRITE I/O operations.

Supported Level of Automation

- The series of tests can be specified as a group via cycling options. As a result, data can be automatically collected on a variety of different loads (test cases).
- Results can be written to a comma-separated values (CSV) file for analysis.

Measurement Units

- Total I/Os per second
- Total megabytes per second (MB/s)
- Average I/O response time (ms)
- Maximum I/O response time (ms)
- CPU utilization (total)
- Total error count

Supported File Chunk Size

The supported file chunk size is 512 bytes, 2KB, 4KB, 8KB, and so forth.

The number of bytes read/written in each I/O request is configurable in bytes (0MB + 0KB + 1 byte) to almost 1GB (1023MB + 1023KB + 1023 bytes), and is limited only by the amount of virtual memory available. The default chunk size is 2KB.

Support Operating Systems

The primary operating systems supported by Iometer are:

- Windows (the main platform)
- Linux
- OS X

A complete list of supported operating systems is provided in Table 2. For a mapping of supported operating systems and processors, see http://www.iometer.org/doc/matrix.html.

Table 2. Supported Operating Systems for Iometer

OS	OS Version
Windows	Windows XP Pro (MSVC++ .NET
	Windows 2000 (MSVC++ 6.0)
	Windows NT 4 (MSVC++ 6.0)
	Windows 2003 (WIN2003 DDK)
Linux	Debian 3.0 (2.4.18 kernel, gcc 2.95.4, glibc 2.2.5)
	Red Hat 9 (2.4.20 kernel, gcc 3.2, glibc 2.3.2)
	Red Hat 8 (2.4.18 kernel, gcc 3.2, glibc 2.2.93)
	SuSE 8.1 (2.4.19 kernel, gcc 3.2, glibc 2.2.5)
	(2.6.x kernel with ARMLinux patch, XScale patch, and CCNT patch)
Netware	NetWare 6.0 with all service packs (MetroWorks 7.0, NetWare SDK 4.0, libc July 30, 2003)
	NetWare 6.5 with all service packs (MetroWorks 7.0, NetWare SDK 4.0, libc July 30, 2003)
Solaris	Mac OS 10.3.5 (G5)
	Solaris 9 (gcc 3.3.2)
	Solaris 8 (gcc 2.95.3)

Measurement Level (File System Level, LLD Level)

- File system level and block devices

Required Code Instrumentation (Add Print Trace, Add Special Hooks)

Iometer can be used as a benchmark and troubleshooting tool and can be configured to replicate the behavior of many popular applications, such as database and client-server applications. Iometer enables the creation of access patterns that vary according to the transfer request size, random/sequential distribution percentage, and READ/WRITE distribution percentage.

At the time this article was written, Iometer had no recent updates or bugs fixes and was a native Windows application.

dd Overview and Features

The dd tool is not a formal benchmark tool. It is a Linux command that is often used as a benchmarking tool. This tool is used to measure file system and driver performance using the Linux operating system in a particular hardware platform. In addition, dd can be used to measure any changes in performance after optimization/modifications in any Linux file system/driver or hardware platform. It is important to note that dd is not useful when a complete performance profile is required.

The syntax of the dd command is:

```
dd if=input_file of=ouput_file bs=block_size count=number_of_blocks
```

The dd command reads from the file indicated by *if* (input file) and writes read data into an *of* (output file). The file is written by first reading a block of *bs* (block size) bytes from *if* and then writing it into *of*. Count is the number of times the bs bytes of a block are written.

The dd command returns statistics about the executed operation. For example, the command *dd if=/dev/zero of=./file-out bs=2k count=64* writes a file (file-out) of 128KB, writing 64 chunks of 2KB of data (the contents of output file will be 0x00). It then returns:

```
64+0 records in
64+0 records out
131072 bytes (131 kB) copied, 0.00228877 s, 57.3 MB/s
```

This operation completed writing at 57.3 MB/s. It can be also used to evaluate READ performance. To evaluate READ performance, instruct dd to write to the /dev/null device. A write to /dev/null requires a negligible amount of time and as a result, the time used for the dd operation is the unique read time. For example, *dd if=./file-in of=/dev/null bs=2k count=51200* returns the following:

```
51200+0 records in
51200+0 records out
104857600 bytes (105 MB) copied, 1.1581 s, 90.5 MB/s
```

This command reads a 100MB file (file-in) in blocks of 2KB 51,200 times at 90.5 MB/s.

Additional dd Features

- Supported performance test
  - Only sequential READ and WRITE
- Supported level of automation
  - No automation support
- Measures READ/WRITE performance in MB/s
- Enables custom sector granularity
- Can measure performance at the file system and LLD level
- Required code instrumentation
  - No instrumentation required
- Can be used for simple tests that sequentially WRITE/READ data from a device

Fio Overview and Features

Fio is an open source (GPLv2 license) tool that enables benchmarking the performance of specific I/O operations. Fio was written and is currently maintained by Jens Axboe, who maintains the current Linux block layer and has authored the blktrace utility, which provides a method for tracing block I/O activity in the Linux kernel.

Fio binaries are available for several Linux distributions (Ubuntu, Mandriva, SuSE, Red Hat, and Debian), Solaris, and Windows. Web site references are provided in the readme file included in the fio distribution.

Fio enables benchmarking specific disk I/O workloads. The huge number of fio options provides the ability to issue precisely defined I/O loads. This enables the simulation of a large variety of I/O operation scenarios, from mobile application databases to storage servers and from embedded file systems to desktop storage. Fio targets benchmarking when the I/O load description is both large and precise, and a greater effort to set the tool is acceptable when determining the correct values for each parameter.

Porting Availability

- Operating system
  - Linux
  - Solaris
  - AIX
  - OSX
  - NetBSD
  - Windows
  - FreeBSD
- Processor architecture
  - ALPHA
  - ARM
  - HP/PA (PA RISC)
  - IA64 (ITANIUM)
  - MIPS
  - PowerPC (PPC)
  - S390 (IBM ESA/390)
  - SuperH (SH)
  - SPARC
  - SPARC64
  - X86
  - X86-64

Fio Resources

- Fio resides in a Git repository. The standard location is: http://git.kernel.dk/?p=fio.git;a=summary.
- Frequently generated snapshots can be downloaded here: http://brick.kernel.dk/snaps/.
- The primary project page is located here: http://freshmeat.net/projects/FIO/.

Supported Performance Tests

The fio tool is aimed at reproducing several types of loads on the Linux I/O subsystem. There are several different parameters you can configure. The following list provides a description of the main parameters defined for a workload. In addition, there are a number of parameters that modify other aspects of the desired I/O load. The HOWTO file in the fio distribution provides a full list of the options that can be used to specify benchmark workloads.

- **I/O type**: Defines the I/O pattern issued for files. Reads from the file can be done sequentially, randomly, or a as mix of sequential and random reads.
- **Block size**: The chunk size of each I/O, which may be a single value or a range of block sizes.
- **I/O size**: Defines how much data will be read/written.
- **I/O engine**: Defines how the I/O is issued. For example, the file could be memory mapped, a regular READ/WRITE could be used, or a splice, asynchronous I/O, syslet, or SG (SCSI generic SG) could be used.
- **I/O depth**: If the I/O engine is asynchronous, this defines how large a queuing depth is to be maintained.
- **I/O type**: Defines whether the I/O is buffered or direct/raw.
- **Num files**: Defines how many files are spread over the workload.
- **Num threads**: Defines how many threads or processes should be spread over the workload.

Measurement Level (File System Level, LLD Level)

Performance can be measured at the file system level, and the file system being tested depends on the path of the target file. If the target file is the name of a special file (such as a device node), fio measures the performance without the file system overhead. In this case, only the layer that manages the device is characterized (that is, the block device layer together with the LLD involved). Depending on the target, not all parameters and parameters values are possible.

Supported Level of Automation

Fio is a command line tool that is run by issuing the executable’s name followed by parameters in a shell on the target system. The parameters describe the target workload and can be explicitly written or (most often) included in a job file. More than one job file can be given on the command line. Fio serializes the running of multiple job files.

Typically, a job’s contents include a global section defining shared parameters and one or more job sections describing the jobs involved. When running, fio parses this file and sets everything up as described.

Benchmark Results Format

Fio produces a large amount of output. The workload that fio submits is described in a job file. A job file can include several groups. Each group (identified by a number) is made up of one or more jobs. Each job can fork more than one process or spawn more than one thread. Each thread/process can involve both reads and writes.

When fio has completed or is interrupted by ctrl+c, it shows the output related to each thread/process, each group, and each disk, in that order.

Table 3. Output for Each Thread/Process

Output	Description
io	The number of megabytes I/O performed.
bw	The average bandwidth rate.
runt	The runtime of the thread.
slat	The submission latency (avg is the average and stdev is the standard deviation), which is the time it took to submit the I/O. For synchronous I/O, the slat is the completion latency since queue/complete is one operation in this instance.
clat	The completion latency, which indicates the time from submission to completion of the I/O pieces. For synchronous I/O, clat is typically equal (or very close) to 0, as the time from submission to completion is essentially the CPU time (I/O has already completed – see “slat” description).
bw	The bandwidth average in KB/s or bytes/s. Also, an approximate percentage of the total aggregate bandwidth this thread received in this group is output. This last value is only useful if the threads in this group are on the same disk, since they are competing for disk access.
cpu	The CPU usage. This includes the user and system time combined with the number of context switches this thread went through, usage of system and user time, and the number of major and minor page faults.
IO depths	The distribution of I/O depths over the job lifetime. The numbers are divided into powers of 2. For example, the 16= entries includes depths up to that value, but higher than the previous entry. In other words, it covers the range from 16 to 31.
IO submit	The number of I/O pieces that were submitting in a single submit call.
IO complete	This is the number of I/O pieces that were completed in a single submit call.
IO issued	The number of READ/WRITE requests issued and the number that were short.
IO latencies	The distribution of I/O completion latencies. This is the time from when the I/O leaves fio until it is completed. The numbers follow the same pattern as the I/O depths. This means that 2=1.6% indicates that 1.6% of the I/O completed within 2ms and 20=12.8% indicates that 12.8% of the I/O took more than 10ms, but less than (or equal to) 20ms.

After fio has completed and displays the output, the group statistics are printed:

```
Run status group 0 (all jobs):
READ: io=64MB, aggrb=22178, minb=11355, maxb=11814, mint=2840msec,
maxt=2955msec
WRITE: io=64MB, aggrb=1302, minb=666, maxb=669, mint=50093msec,
maxt=50320msec
```

For each data direction, fio prints:

Output	Description
io	The number of megabytes I/O performed.
aggrb	The aggregate bandwidth of threads in this group.
minb	The minimum average bandwidth of a thread.
maxb	The maximum average bandwidth of a thread.
mint	The smallest runtime of the threads in that group.
maxt	The longest runtime of the threads in that group.

Finally, disk statistics are printed:

```
Disk stats (read/write):
sda: ios=16398/16511, merge=30/162, ticks=6853/819634, in_queue=826487,
util=100.00%
```

Each value is printed for each READ and WRITE, with READs printed first. The numbers indicate:

Output	Description
Ios	The number of I/Os performed by all groups.
Merge	The number of merges in the I/O scheduler.
ticks	The number of ticks that kept the disk busy.
io_queue	The total time spent in the disk queue.
util	The disk utilization. A value of 100% indicates that the disk was kept busy constantly, while 50% indicates a disk idling half of the time.

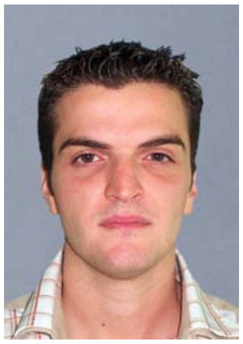
Appendix A: Tool/Feature Comparison

A comparison of the tools and features described in this article is provided in Table 4.

Table 4. Benchmark Tool/Feature Comparison

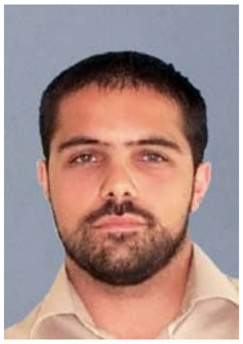
Tool	Unit	OS	Measures File System Performance?	Measures LLD Performance?	Code Change Required?	Write Granularity	Supported Test	Automation
IOzone	• IOPs • KB/sec	• Windows • Linux • OSX	Yes	No	No	Any	• Sequential READ/WRITE • Random READ/WRITE • Re-write • Fread/fwrite • Read backwards • Aio_read/Aio_write • Mmap	Yes
Iometer	• IOPs • MB/s • %	• Windows • Linux • OSX	Yes	Yes	No	Any	• Sequential READ/WRITE • Random READ/WRITE	Yes
dd	• MB/s	• Linux	Yes	Yes	No	Any	• Sequential READ/WRITE	No
fio	• KB/sec	• Linux • Solaris • AIX • OSX • NetBSD • Windows • FreeBSD	Yes	Yes	No	Any	• Sequential READ/WRITE • Random READ/WRITE • Re-write • Fread/fwrite • Read backwards • Aio_read/Aio_write • Mmap • Other	Yes

About the Authors



With over 8 years of experience with Flash/storage software, ***Angelo Di Sena****, Micron software developer manager, specializes in NAND Flash management in Linux, Symbian, and Android;* e·*MMC, integration in Android; and customer support for platform integration of nonvolatile memory. Angelo was a key architect and developer of Micron's NAND Flash Translation Layer (NFTL) software and has authored patents for optimizing WRITE/ERASE operations in memory devices, controlling Flash memory operations, and wear leveling in Flash-based storage devices.*

As a senior software engineer at Micron, ***Massimo Cirillo*** *specializes in Flash software driver development, Flash data and file system management, and Linux kernel development (Flash driver, block, file system layers). With more than a decade of software development experience, Massimo has been instrumental in driver and block Linux kernel optimization for Micron's Technology for Memory Optimization (TMO).*



Procolo Carannante *is a senior software engineer specializing in NAND, NOR, and phase change memory (PCM) management Software; the Linux kernel; Nucleus; and the Android operating system. Procolo has been involved in the development of the NAND Sector Manager (NSM), NAND Flash Translation Layer (NFTL), and Hybrid FTL (NAND plus PCM) project. He has also been worked on an* e·*MMC optimization project for Android. Prior to Micron, Procolo worked as a NAND application engineer and firmware engineer.*

Micron software engineer ***Fabio Salvati*** *specializes in software tools for Flash memory management; the Linux, Symbian, and Android operating systems; and file systems such as FAT, EXT3, and EXT4. With more than 10 years of software development experience, Fabio has worked on Micron's NAND Flash Translation Layer (NFTL) and Sector-Based Compact File System (SCFS) software.*