Measuring the Workload on Drives on a Linux System Using the blktrace Tool

Technology Paper

Written by
Dan Lingenfelter, PhD

The workload seen by hard drives impacts performance. In most cases, measurements provide the most accurate characterization of the drive-level workload. There are many measurement tools available but reviewing them all is beyond the scope of this paper. This paper focuses on how to use the *blktrace* tool to measure the workload seen on a hard drive on a Linux system.

On each hard drive, each block of stored data is given a logical block address (LBA). When the host system sends a read or write command to the drive, it tells the drive to read or write the data located at one or more LBAs. Using the *blktrace* tool, it is possible to record the starting LBA, transfer size, timestamp, and many other attributes for each request sent to the drive. The recorded data can be used to compute the read/write percentages, sequential percentage, and transfer size distribution.

Generating and Recording Drive Workloads

For the purpose of a *blktrace* walkthrough, imagine an example where a 10GB file will be written to a disk. Use blktrace to record the requests to and from the drive. Assume there is a folder at /<trace folder> to record the trace to and another folder /<test folder> to write the test data to. In this setup, these folders are located on different physical drives to be sure that the recorded trace is as pure as possible.

1. To start recording, navigate to /<trace folder> and start blktrace with the following command, where sdX is the name of the drive where /<test folder> is physically located.

   ```
   sudo blktrace -d /dev/sdX
   ```

2. Now write a 10GB file to /<test folder>/ with a series of 64kB transfers. For example, use a separate terminal and fio as follows:

   ```
   sudo fio ten_gb_write.ini
   ```

   The contents of ten_gb_write.ini are included at the end of the post.

3. When the write finishes, press Ctrl+C to stop the trace in the terminal where blktrace was running.
4. To make the output of blktrace readable, we can use blkparse to extract the information that we need. For example:

   ```
   blkparse sdX -f "%5T.%9t, %p, %C, %a, %d,
   %S, %N\n" -a complete  -o output.txt
   ```

   This creates a file with the timestamp, requesting host process, operation type (read/write), LBA number, and number of LBAs read or written for each transfer. The `-a complete` filters the trace so that we only see commands that were completed.

Using the steps above, examine the workload a drive experiences for various system-level workloads generated with fio.

Interpreting the Results

The figure below shows the starting LBA of each transfer as a function of time. The large solid line in the low end of the LBA space is the file written in the example provided in the *Generating and Recording Drive Workloads* section. There are also a series of dots that are in other parts of LBA space that are caused by operations for the operating and file systems. The transfer of the 10GB file is not entirely sequential due to these other transfers in other ranges of the LBA space.

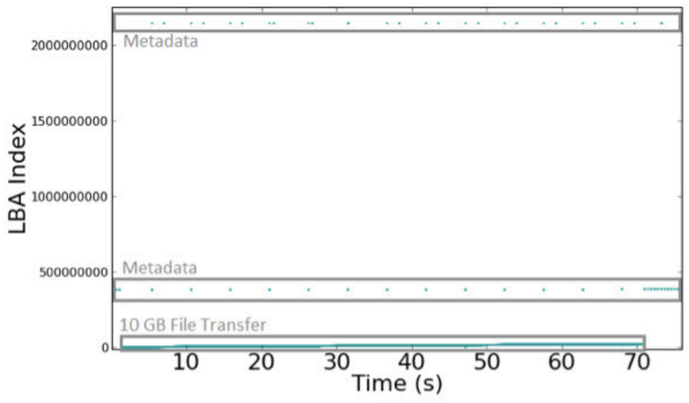

Figure 1 – The starting LBA of each transfer

Analysis of the trace data shows that 99.5% of the transfers were sequential, 100% were writes, and 97% had a size of 512kB. The dominant transfer size is 512kB and not 64kB as specified with fio because the Linux I/O scheduler will, by default, merge requests for adjacent LBAs until the transfer reaches a system-specified maximum, which in this case is 512kB. Analyzing the trace data shows the impact of metadata and the Linux I/O scheduler on overall performance.

Conclusion

This experiment showed an example of how to use *blktrace* to characterize a workload at the drive level. The measurement showed that the actual workload seen by the drive was different than the parameters used to generate the system-level workload would suggest. Metadata overhead interrupted the sequential large file write, while the optimizations performed by the Linux I/O scheduler increased the transfer size seen by the drive. Understanding the connection between system-level workloads and the I/O pattern that the drive experiences is essential to optimizing performance.

ten_gb_write.ini:

```
filename=/<test folder>/fio.tmp
iodepth=512
size=10000M
ioengine=libaio
direct=1

[job]
numjobs=1
bs=64k
rw=write
```

www.seagate.com