Course Project #3: SSD Performance Profiling

Charles Clarke
Advanced Computer Systems
10/15/20204

Introduction:

The description of this project made it sound pretty daunting with the whole "will destroy all your data!" disclaimer. However I followed all the necessary steps and it didn't turn out to be too dangerous after all. I attempted to download fio via the github link provided, but the directions were terrible. ChatGPT was able to walk me through an easy enough installation via WSL which was much more helpful. I then took some time to create a 5GB partition on my SSD to prevent any bad things from happening, especially on my personal computer. This took some time as I had some files which were in the way of the partition, however I got those sorted out thanks to some youtube videos. Before I knew it, I had fio working, and a partition to test it on. From there it was almost a breeze.

Data Access Size:

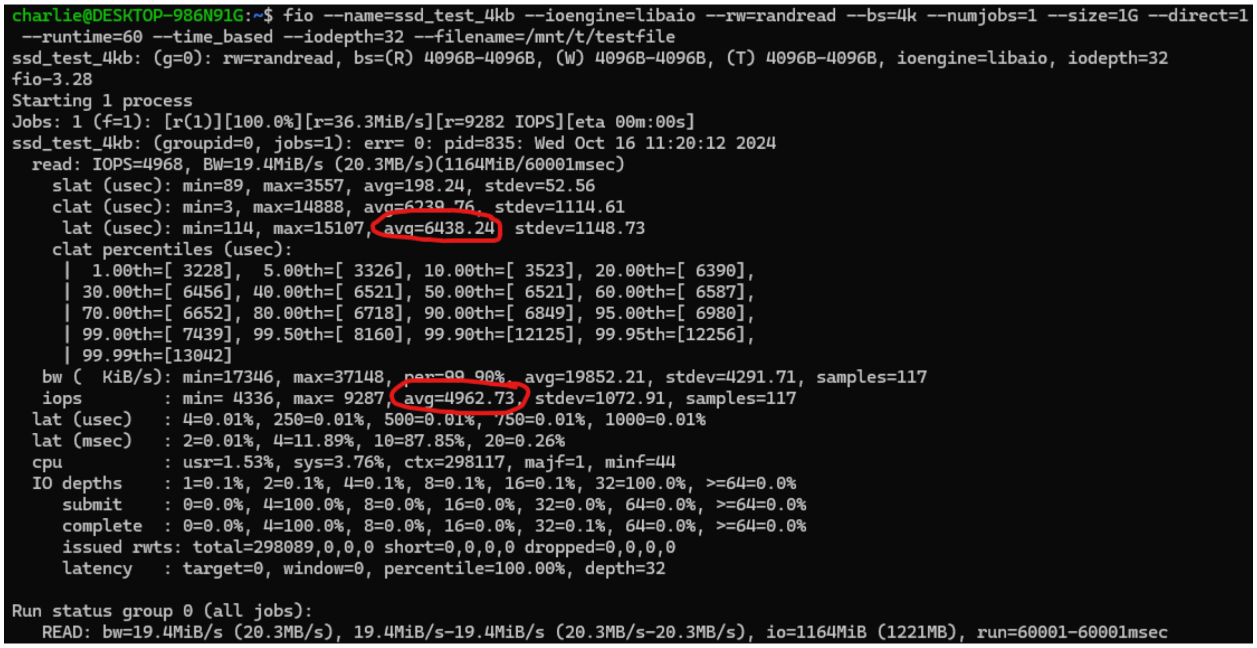
I started out the experiments by testing the throughput and latency at different data access sizes. From 4KB up to 128KB, I recorded the outputs in google sheets to visualize the data. There were clear trends in the change in latency and throughput when compared to data access size. The data can be seen below.

Size(KB)	Latency(ms)	Throughput(IOPS)
4	3.67	4350
16	3.94	4055
32	5.02	3190
128	16.7	960

As you can see, as size goes up, latency also goes up and throughput decreases which is expected. I obtained this data using the following fio command:

fio --name=ssd_test_4kb --ioengine=libaio --rw=randrw --rwmixread=50 ***--bs=4k*** *--numjobs=1 --size=1G --direct=1 --runtime=60 --time_based --iodepth=32 --filename=/mnt/t/testfile*

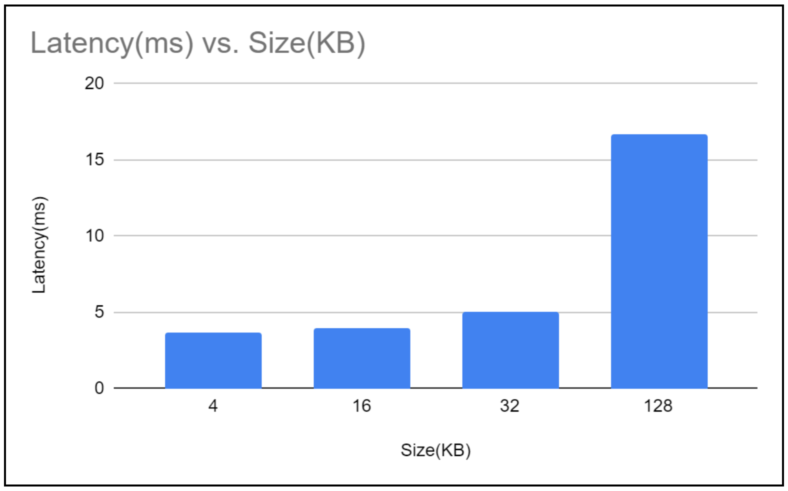
This command outputs a whole lot of data which must be sifted through to find average throughput and latency over the 60 second runtime, this command line output can be seen below.

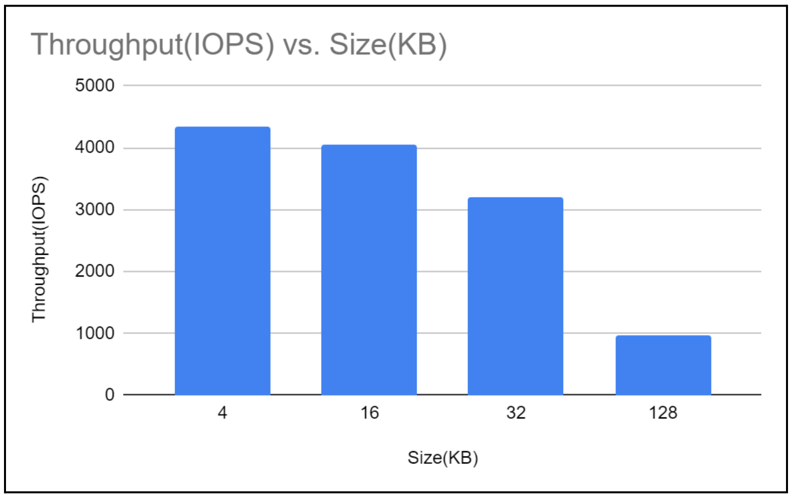


```
charlie@DESKTOP-986N91G:~$ fio --name=ssd_test_4kb --ioengine=libaio --rw=randread --bs=4k --numjobs=1 --size=1G --direct=1
 --runtime=60 --time_based --iodepth=32 --filename=/mnt/t/testfile
ssd_test_4kb: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.28
Starting 1 process
Jobs: 1 (f=1): [r(1)][100.0%][r=36.3MiB/s][r=9282 IOPS][eta 00m:00s]
ssd_test_4kb: (groupid=0, jobs=1): err= 0: pid=835: Wed Oct 16 11:20:12 2024
  read: IOPS=4968, BW=19.4MiB/s (20.3MB/s)(1164MiB/60001msec)
    slat (usec): min=89, max=3557, avg=198.24, stdev=52.56
    clat (usec): min=3, max=14888, avg=6239.76, stdev=1114.61
     lat (usec): min=114, max=15107, avg=6438.24, stdev=1148.73
    clat percentiles (usec):
     |  1.00th=[ 3228],  5.00th=[ 3326], 10.00th=[ 3523], 20.00th=[ 6390],
     | 30.00th=[ 6456], 40.00th=[ 6521], 50.00th=[ 6521], 60.00th=[ 6587],
     | 70.00th=[ 6652], 80.00th=[ 6718], 90.00th=[ 6849], 95.00th=[ 6980],
     | 99.00th=[ 7439], 99.50th=[ 8160], 99.90th=[12125], 99.95th=[12256],
     | 99.99th=[13042]
   bw (  KiB/s): min=17346, max=37148, per=99.90%, avg=19852.21, stdev=4291.71, samples=117
   iops        : min= 4336, max= 9287, avg=4962.73, stdev=1072.91, samples=117
  lat (usec)   : 4=0.01%, 250=0.01%, 500=0.01%, 750=0.01%, 1000=0.01%
  lat (msec)   : 2=0.01%, 4=11.89%, 10=87.85%, 20=0.26%
  cpu          : usr=1.53%, sys=3.76%, ctx=298117, majf=1, minf=44
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=298089,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=19.4MiB/s (20.3MB/s), 19.4MiB/s-19.4MiB/s (20.3MB/s-20.3MB/s), io=1164MiB (1221MB), run=60001-60001msec
```

All of these parameters do something important to the test. First of all, the job is named “ssd_test_nkb” and it is run on libaio for the test. It also specifies a 50/50 ratio of mixed random reads and writes. I can then set the data access size to nkb, n being amount of KB. The next few parameters limit the command from running too long or making too large of a test file. I run this command for each data access size I tested in the table above to obtain the results. I visualized these results in graphs shown below.





This increased data access size has these effects for a couple reasons. Latency rises because when the block size is small, the ssd can handle many operations in parallel because each is relatively lightweight. However once the block size starts to increase, the ssd has more data to process per request, slowing down each operation. Throughput decreases for a similar reason, the ssd can process fewer operations per second as block size increases and parallelism decreases, meaning throughput drops with increased data access size.

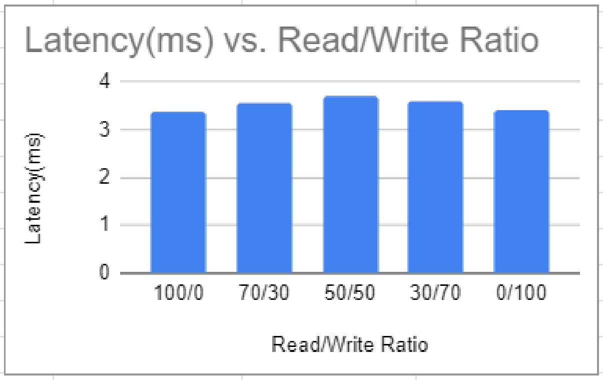
Read vs. Write Ratio:

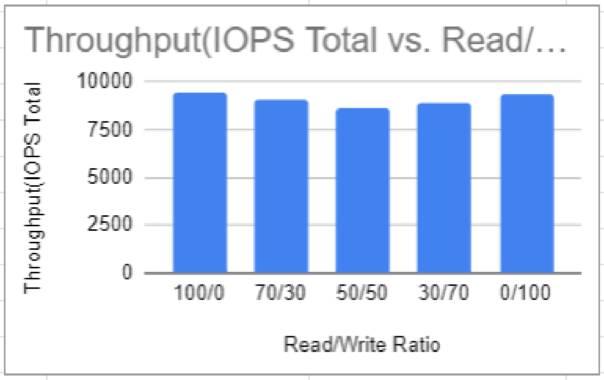
The next part of this experiment was to keep the data access size and queue depth constant, and change the read vs write ratio to see the effect of each type of operation of the ssd's performance. This would be done using some parameters that allow me to configure the ratio of reads to writes, the command I used can be seen below.

*fio --name=ssd_test_4kb --ioengine=libaio --rw=randrw **--rwmixread=30** --bs=4k --numjobs=1 --size=1G --direct=1 --runtime=60 --time_based --iodepth=32 --filename=/mnt/t/testfile*

The parameter –rwmixread=n gives the percentage of reads compared to writes during the 60 second time window of the test. I tested a variety of ratios, from 100% read to 70% read to 50% read, to 30% read, to eventually all writes. The data came out very well and showed a clear trend which made a lot of sense and was very relieving. I tabulated all of the data, which can be seen below.

Read/Write Ratio	Latency(ms)	Throughput(IOPS Read)	Throughput(IOPS Write)	Throughput(IOPS Total)
100/0	3.38	9460	0	9460
70/30	3.54	6318	2716	9034
50/50	3.71	4309	4315	8624
30/70	3.59	2682	6245	8927
0/100	3.4	0	9362	9362





These trends seen above make sense because as the balance of reads and writes is equal, there is a heavy demand for both types of operations which often require different internal processes. This means the ssd has to work harder to manage resources between the two types of operations. This also means that the ssd can't fully optimize for one type of operation as it can when doing 100% reads or writes. Switching from reads to writes is just more overhead needed to complete all the processes. You also see slightly higher throughput at 100% read compared to 100% write, which makes sense as well. Writes are generally more complex processes, managing wear leveling and handling erase cycles.

I/O Queue Depth:

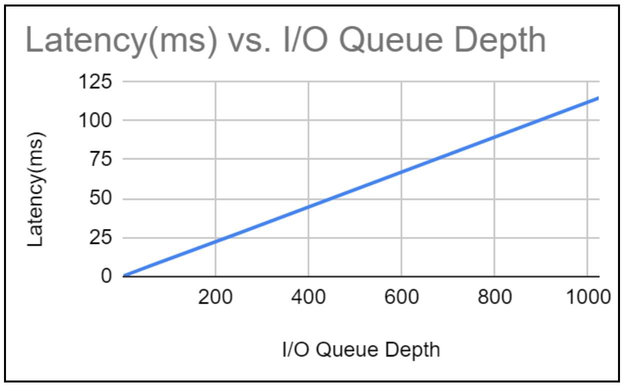
Finally to wrap this experiment up, I tested the effect of I/O queue depth on my ssd's latency and throughput. I/O queue depth is the number of operations that the system can queue up for the ssd to process at any given time. It’s basically the number of pending requests. Higher queue depth usually relates to higher throughput as it processes requests more concurrently. However it can increase latency because each request has to wait longer in the queue. I altered the queue depth using the parameter –iodepth=n in the command, this can be seen below.

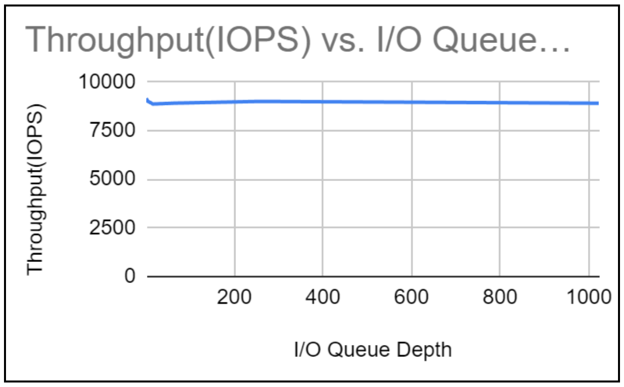
fio --name=ssd_test --ioengine=libaio --rw=randread --bs=4k --numjobs=1 --size=1G --direct=1 --runtime=60 --time_based ***--iodepth=1024*** *--filename=/mnt/t/testfile*

The results I found from this experiment weren’t quite what I was expecting. The data I collected can be seen below.

I/O Queue Depth	Latency(ms)	Throughput(IOPS)
1	0.11	9162
4	0.44	9007
16	1.81	8864
64	7.19	8908
256	28.46	8997
1024	114.79	8904

As you can see the latency increases with queue depth as expected, because as more operations are queued up, the longer each one has to wait. The throughput however did not increase like I thought it would. This is probably because my ssd is hitting its maximum throughput around the value of 9000, so adding more values to the queue doesn’t speed it up. It seems like my ssd is quite efficient at lower queue depths, so the queue depth doesn’t matter much for my ssd in particular. I graphed these values to visualize them effectively, these can be seen below.





As you can see the latency to io depth graph is incredibly linear showing that there is a completely causal relationship between the two. However on the right, throughput doesn’t really do anything special as io depth is changed, showing that it’s almost completely independent on my ssd. This is most likely not true for other devices.

Conclusion:

Throughout this experiment I’ve learned that there are many factors that affect the performance of my ssd, for example, the size of the data that is accessed at once, the ratio of read operations to write operations, or the I/O queue depth of the ssd. All of these have effects on throughput and latency within the ssd. First of all larger data access sizes lead to higher latency and less throughput, so it is important to keep that value small in order to process more operations in parallel. Next the ratio of read to write operations is an important metric to be aware of. The ssd will work harder overall when resources are shared between read and write processes. Throughput and latency will always be better when prioritizing one or the other. Read operations also tend to have an edge in performance compared to write because they require less overhead to execute. Finally, I/O queue depth can have some massive effects on the latency of single processes, however it didn't seem to affect the overall performance of my ssd at all which was interesting.