Exploring Linux Memory Usage and Disk IO performance

Frits Hoogland
Yugabyte
Topic: disk IO and disk IO performance

- The main topic of this presentation is disk IO performance on Linux.

- In order to understand disk IO performance, a detailed understanding of disk IO and related technologies is necessary.

- This presentation explains some the mechanics, in order to let the attendee understand buffered disk IO performance better.

- Memory usage is quite fundamentally closely related to cached IO performance.
Disk IO and memory

- Any regular disk IO is performed *buffered*.
- Buffered means: using the operating system memory for caching.

- You can do IO *without* using the operating system for caching.
  - Only if you explicitly request it: O_DIRECT.
  - Makes sense if you don't want to stage memory in two caches.

- If you are not sure which you are using you are quite probably doing *buffered IO*. 
Where does buffered IO go?

• Linux does not have a dedicated memory area as 'page cache'.
  • Traditional Unix such as HPUX and AIX have that.

• Buffered IO must allocate memory to store the IO.
  • Even if that means it will get removed immediately b/c memory pressure(!)
  • Writes are special.

• Therefore it competes with regular memory usage.
Okay: but where does buffered IO go?

- Linux provides an insight into its memory usage via `/proc/meminfo`
  - Which is a messy gathering of memory related statistics.

- Named values in 'meminfo' do contain overlapping memory allocations, and can contain multiple, different allocations.

- Roughly put, it should be in 'Cached', 'Dirty' and 'Mapped', which can contain other allocated memory, such as shared memory.
You are not really making it understandable!

• I know.

• I think it's wrong to try to capture page cache size.

• You must have memory that is usable for buffering purpose.
  • Which is also memory for application usage.

• The best way to assess usable memory is use `MemAvailable`
How about 'MemFree'?

- There also is the MemFree statistic in 'meminfo'?

- MemFree is not 'free' as in available.
  - It is a small amount of memory pre-cleaned for direct usage.
  - There will be lots after startup, because it was never touched.

- Linux tries to do the bare minimum, and thus keep used memory around.
  - And thus to reduce MemFree to a minimum (vm.min_free_kbytes).
  - The swapper force-frees memory. (Page daemon)
  - Processes explicitly freeing memory will add to MemFree.
  - https://dev.to/yugabyte/what-is-free-memory-in-linux-18km
MemAvailable

- Statistic in `/proc/meminfo`.

- Kernel estimation of available memory without requiring swapping.

- Many of the other statistics (in `/proc/meminfo`) contain information, are useful, but do not provide a full picture to assess available memory.
Why is this important actually?

- Buffering can do miracles for IO performance*.

- Equally it can do "miracles" for container/application performance.
Let's test!

- Tests done on Amazon EC2:
  - c5.large VM (20000/4000 IOPS, 594/82 MBPS)
  - EBS: GP3 250M (3000 IOPS, 125 MBPS)
- I am not running into my bursting limits so concrete:
  - IOPS: 3000
  - MBPS: 125
  - Not easy to find.
Read: 2G

- Drop the page cache
- Validate available memory
- Run a fio **read** test reading 2G:

```bash
fio --name test --filename /tmp/fiotest
   --ioengine sync --rw randread --bs 8k
   --invalidate 0 --filesize 2G
```
```bash
[sudo@ip-172-158-19-16 ~]$ sudo su -c "echo 1 > /proc/sys/vm/drop_caches"
[sudo@ip-172-158-19-16 ~]$ /usr/bin/swapinfo -r
available memory :  3200 MB
total memory :  3664 MB, free memory :  3376 MB, used memory :  161 MB
total swap :  0 MB, free swap :  0 MB, used swap :  0 MB
```
Read: 2G

• This is a summary from the run:

IOPS=2609, BW=20.4MiB/s (21.4MB/s) (2048MiB/100465msec)

• My limits are 125 MBPS and 3000 IOPS.
• Why didn't we reach any of these? Is AWS lying?
• No: look at the latency:
  clat (usec): min=195, max=27385, avg=382.22, stdev=263.41
• 382 (avg usec) * 2609 (IOPS) \approx 996638 \approx 1 \text{ second}: \text{latency bound!}
Read: 2G

• Now let's perform the exact same run again
[centos@ip-172-158-19-16 ~]$ sudo su - -c "echo 1 | /proc/sys/vm/drop_caches"
[centos@ip-172-158-19-16 ~]$ /mnt/rafs/target/release/ntomemory -q
available memory : 1320 MB
total memory : 3664 MB, free memory : 3376 MB, used memory : 161 MB
total swap : 0 MB, free swap : 0 MB, used swap : 0 MB
[centos@ip-172-158-19-16 ~]$ fio --name test --filename /tmp/fiotest --ioengine sync --rw randread --bs 8k --invalidate 0 --filesize 2G
test: (rw=0): readspeed=bs=8K 81928-81928, (r) 81928-81928, (w) 81928-81928, (rw) 81928-81928, ioengine=sync, iodepth=1
Fio-3.7
Starting 1 process
Jobs: 1 ([1])...end
[centos@ip-172-158-19-16 ~]$ iwconfig
ether: wireless Lans are disabled

---

bw (Kb/s): min=50040, max=33252, per=100.00%, avg=28874.93, stddev=1870.07, samples=200

tcpp: min=1330, max=2590, avg=2009.35, stddev=233.77, samples=200

log (sec): 100.00%, 80.50%, 60.00%, 40.00%, 20.00%, 10.00%, 5.00%, 1.00%, 0.00%

lat (sec): 2.14, 4.08, 8.19, 16.39, 32.79, 65.58, 129.91

xre: 85.85% sys 15.15% ctime 2.6145, xff=0, m=25

10 depth: hello=0.0%, goodbye=0.0%, 2000.0%, 4000.0%, 6000.0%, 8000.0%, 10000.0%

submit: 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%
complete: 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%, 0.0%

issued nets: total=20124,0,0,0,0,0,0,0,0,0,0

latency: target=0, window=0, percentile=0.00%, depth=1

Run status group 0 (all jobs):
READ: bw=20.4MB/s (21.0MB/s), 20.4MB/s, 20.4MB/s (21.0MB/s, 20.4MB/s, 20.4MB/s), run=100455-100455

Disk stats (read/write):
/dev/h0: io=29125857193, merge=111, ticks=9957979, in_queue=97555, util=96.99%
[centos@ip-172-158-19-16 ~]$
Read: 2G

• This is quite much different, isn't it?

read: IOPS=585k, BW=4571MiB/s (4793MB/s)(2048MiB/448msec)

• My limits are 125 MBPS and 3000 IOPS.
• Now I did 585000 IOPS and 4571 MBPS!
  clat (nsec): min=893, max=19771, avg=1342.90, stdev=406.95
• It was all cache, no physical IOs were performed:
  ios=0/0, merge=0/0, ticks=0/0, in_queue=0, util=0.00%
Read: 4G

- Drop the page cache
- Validate available memory
- Run a fio `read` test reading 4G:

```
fio --name test --filename /tmp/fiotest --ioengine sync --rw randread --bs 8k --invalidate 0 --filesize 2G
```
Centos@ip-172-158-19-16:~$ sudo su -c "echo \1 > /proc/sys/vm/drop_caches"
Centos@ip-172-158-19-16:~$ /usr/bin/printf "\0"

Available memory : 372 MB
Total memory : 3664 MB, free memory : 3267 MB, used memory : 166 MB
Total swap : 0 MB, free swap : 0 MB, used swap : 0 MB
Starting 1 process
Jobs: (1/1)

<table>
<thead>
<tr>
<th>Total Memory</th>
<th>Free Memory</th>
<th>Used Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>3664 MB</td>
<td>3267 MB</td>
<td>166 MB</td>
</tr>
</tbody>
</table>

Available memory: 372 MB
Total swap: 0 MB, free swap: 0 MB, used swap: 0 MB

Test: (ps=0x13) read=81928-81928, write=81928-81928, ioengine=sync, iodepth=1, FIO-3.7
Read: 4G

• This is a summary from the run:

read: IOPS=2702, BW=21.1MiB/s (22.1MB/s) (4096MiB/193996msec)

• My limits are 125 MBPS and 3000 IOPS.
• IOPS rate identical to 2G run, indicates being latency bound again.
• Time and disk physical IOs roughly doubled, as expected.
Read: 4G

• Now let's perform the exact same run again

• Caveat: I had to slightly alter the fio statement.
  • Add option `--norandommap`
  • This prevents every 8k IO offset from being touched exactly once.
Read: 4G

- This is a summary from the run:

  read: IOPS=4206, BW=32.9MiB/s (34.5MB/s) (4096MiB/124633msec)

- My limits are 125 MBPS and 3000 IOPS.
- IOPS rate increased, because of caching
- Still had to do a lot of IO:
  ios=330920/105
  issued rwts: total=524288,0
Reality

- Let's take a look at the memory figures again:

  [centos@ip-172-158-19-16 ~]$ ./eatmemory-rust/target/release/eatmemory -q

  available memory : 3179 MB
  total memory : 3664 MB, free memory : 3267 MB, used memory : 166 MB
  total swap : 0 MB, free swap : 0 MB, used swap : 0 MB

- Having 166MB used is not a realistic scenario.
- A server would typically have an application running!
- Which is what reads that data to serve it, right?
- What if we occupy 50% of memory?
• I build a tool that can do that: eatmemory

[centos@ip-172-158-19-16 ~]$ ./eatmemory-rust/target/release/eatmemory -s 2000
done. press enter to stop and deallocate

• Credits to original eatmemory.c tool (https://github.com/julman99/eatmemory.git)

• Let's try the same 2G run again!
```
[centos@ip-172-158-19-36 ~]$ sudo su -c "echo 1 > /proc/sys/vm/drop_caches"
[centos@ip-172-158-19-36 ~]$ /usr/bin/anymemory -n "target/Release/anymemory" -q
available memory : 1335 MB
total memory : 3664 MB, free memory : 1228 MB, used memory : 2215 MB
total swap : 0 MB, free swap : 0 MB, used swap : 0 MB
```

Starting 1 process

Jobs: 1

Disk stats (read/write):
  /dev/nvme0n1: io=262560/146, merge=4/19, ticks=96084/185, in_queue=96082, util=97.3%
Read: 2G / 50% of 4G memory taken

• This is a summary from the run:

read: IOPS=2671, BW=20.9MiB/s (21.9MB/s) (2048MiB/98136msec)

• My limits are 125 MBPS and 3000 IOPS.

• Time is slightly less (98136 vs. 100465), but generally equal.
• Because despite the memory allocation, there was no change: bound by IO.
Read: 2G / 50% of 4G memory taken

- Now let's perform the same run again
  - Add option `--norandommap`
Starting 1 process

Jobs: 1 (0-1): [r13] [r11] [r8] [r7] [r6] [r5] [r4] [r3] [r2] [r1] [r0]


bw ( KB/s): min=13332, max=3744, avg=99.999, op=23165.87, stdev=1330.25, samples=196
cpu: min=574, max=2965, avg=2067.73, stdev=166.38, samples=196

load (sec): min=0.54, max=0.87, avg=0.64, 100%=0.64, 1000%=0.64

run status group 0 (all jobs):

READ: bw=20.9MB/s (21.9MB/s), 20.9MB/s:20.9MB/s (21.9MB/s:21.9MB/s), to=2048MB (2147MB), run=98136:98136

Disk stats (read/write):
novelti: io=255650/146, merge=0/17, ticks=96684/105, in_queue=96892, util=97.37%

[sudo@ip-172-158-19-36 ~]$ i
Starting 1 process
Jobs (1/1): [r1(1)] 0.00K [r1=21.0m/s, w=40.0Kb/s] [r2=779m, w=40.0Kb/s]
test: [groups=1, jobs=1, err=0] pid=25675 Thu Apr 18 14:36:21 2022
read: IOP=2671, BM=20.0m/s (21.0m/s) [2444584/81336sec]
cct (usec): win=715, nm=24206, avg=375.64, std=14.75
lat (usec): win=24206, avg=375.69, std=14.75
cct (usec): win=715, nm=24206, avg=375.64, std=14.75
cct (usec): win=715, nm=24206, avg=375.69, std=14.75
cct (usec): win=715, nm=24206, avg=375.64, std=14.75
cct (usec): win=715, nm=24206, avg=375.69, std=14.75
bw (KiB/s) = 11332, max=3744, per=99.99, avg=21365.87, stddev=1330.25, samples=106
lats: win=1674, max=2633, avg=2670.73, stddev=160.38, samples=106
lat (usec): win=250.876, 500.659, 2.28, 7500.54, 1066.82, 2.83
lat (usec): win=2.14, 4.0, 0.81, 3.8, 0.81, 0.81
cpu: win=31.5%, avg=49.6%, cts=82434, rcpu=16.9%, min=36
20 depths: 1:0.00, 2:0.00, 4:0.00, 8:0.00, 16:0.00, 32:0.00, 64:0.00, 128:0.00
subbit: 64:0.00, 128:0.00, 256:0.00, 512:0.00, 1024:0.00, 2048:0.00, 4096:0.00, 8192:0.00
completed: 64:0.00, 128:0.00, 256:0.00, 512:0.00, 1024:0.00, 2048:0.00, 4096:0.00
issued msgs: total=405244, 0, short=0, short=0, 0 dropped=0, 0, 0, latency: target=0, window=0, percentile=100.0000, depth=1

Run status group # (all jobs):
READ: bm=20.0m/s (21.0m/s), 20.0m/s-20.0m/s (21.0m/s), io=2444584 (214748), run=408336-512366

Disk stats (read/write):
nvme0s: con=2556564, merge=17, ticks=998984/105, in=3456, out=99692, utl=97.37%
Centos:17-158-19-16 -s /sys/class/blkdev/root/blk[0-9]* -bks invalidate 0 --norandommap --filesize 2G
test: [Go] = /mnt/raid, bs=1K 1024K-1024K, (0) 1024K BI 1024K, (T) 1024K BI 1024K, loeignnet-smp, iod=1, depth=1

File 1.7
Starting 1 process
Jobs (1/1): [r1(1)] 0.00K [r1=21.0m/s, w=40.0Kb/s] [r2=779m, w=40.0Kb/s]
test: [groups=1, jobs=1, err=0] pid=37575 Thu Apr 18 14:36:29 2022
read: IOP=3525, BM=27.0m/s (28.9m/s) [2444584/74366sec]
cct (usec): win=516, nm=23106, avg=23139.64, stddev=250678.87
lat (usec): win=538, avg=23132.54, stddev=250789.30
lat (usec): win=516, nm=23106, avg=23139.64, stddev=250678.87
lat (usec): win=538, avg=23132.54, stddev=250789.30
lat (usec): win=538, avg=23132.54, stddev=250789.87
lat (usec): win=516, nm=23106, avg=23139.64, stddev=250678.87
lat (usec): win=538, avg=23132.54, stddev=250789.30
20 depths: 1:2.00, 2:0.00, 4:0.00, 8:0.00, 16:0.00, 32:0.00, 64:0.00, 128:0.00
subbit: 64:0.00, 128:0.00, 256:0.00, 512:0.00, 1024:0.00, 2048:0.00, 4096:0.00, 8192:0.00
completed: 64:0.00, 128:0.00, 256:0.00, 512:0.00, 1024:0.00, 2048:0.00, 4096:0.00, 8192:0.00
issued msgs: total=405244, 0, short=0, short=0, 0 dropped=0, 0, 0, latency: target=0, window=0, percentile=100.0000, depth=1

Run status group # (all jobs):
READ: bm=27.0m/s (28.9m/s), bm=27.0m/s-27.0m/s (28.9m/s), io=2444584 (214748), run=74366-74366sec

Disk stats (read/write):
nvme0s: con=1012004, merge=0, ticks=72973, in=1024, utl=98.02%
Centos:17-158-19-16 -s /sys/class/blkdev/root/blk[0-9]* -bks invalidate 0 --norandommap --filesize 2G
test: [Go] = /mnt/raid, bs=1K 1024K-1024K, (0) 1024K BI 1024K, (T) 1024K BI 1024K, loeignnet-smp, iod=1, depth=1
Read: 2G / 50% of 4G memory taken

read: IOPS=3525, BW=27.5MiB/s (28.9MB/s) (2048MiB/74366msec)

• My limits are 125 MBPS and 3000 IOPS.
• This gone beyond the limits (IOPS, 3000 <> 3525).
• Time difference with previous 2nd 2G run: 74.3 <> 0.4 second (!)

• Reason: physical IO had to be performed:
  ios=192264/64, merge=0/6, ticks=72937/55, in_queue=72978, util=98.01%
Write: 2G

- Validate available memory
- Run a fio **write** test writing 2G:

```bash
fio --name test --filename /tmp/fiotest
   --ioengine sync --rw randwrite --bs 8k
   --filesize 2G
```
**Write: 2G**

- This is a summary from the run:

  IOPS=22.1k, BW=173MiB/s (181MB/s) (2048MiB/11840msec)
  clat (usec): min=2, max=18861, avg=44.34, stdev=597.77

- My limits are 125 MBPS and 3000 IOPS.
- IOPS = 22100, which is significantly more than 3000 IOPS.
- Reason: only 24% was written;

  ios=0/61815
  issued rwts: total=0,262144
Write: 2G -- write details

- Why aren't all writes cached, like all reads were?
  - Writes are special!
  - Writes cannot be discarded like reads can, they **must** be written first.
  - Writes can/should not exhaust available memory.
  - Therefore: \texttt{vm.dirty_background_ratio}, \texttt{vm.dirty_ratio}, others.
  - Ratio is taken from \textit{available memory}, unlike popular believe of total mem.
  - \url{https://dev.to/fritshooglandyugabyte/linux-buffered-write-latency-10mc}
- In linux, processes performing buffered writes do not actually write to disk.
  - Produce dirty pages, and get throttled (wait in write()) to balance.
Write: 500M

- Validate available memory
- Run a fio **write** test writing 500M:

```bash
fio --name test --filename /tmp/fiotest --ioengine sync --rw randwrite --bs 8k --filesize 500M
```
Write: 500M

- This is a summary from the run:
  
  IOPS=193k, BW=1506MiB/s (1579MB/s) (500MiB/332msec)

- My limits are 125 MBPS and 3000 IOPS.
  
  IOPS = 193000, MBPS = 1506.
  
  Reason; no write (throttling):
  
  ios=0/0, merge=0/0, ticks=0/0, in_queue=0, util=0.00%

Reality

- The writes so far were also conducted with no memory used.
- Let's occupy 50% and perform the same tests again.
Write: 2G / 50% of 4G memory taken

- Validate available memory
- Run a fio **write** test writing 2G:

```bash
fio --name test --filename /tmp/fiotest
   --ioengine sync --rw randwrite --bs 8k
   --filesize 2G
```
Write: 2G / 50% of 4G memory taken

• This is a summary from the run:

IOPS=6581, BW=51.4MiB/s (53.9MB/s)(2048MiB/39832msec)

• My limits are 125 MBPS and 3000 IOPS.
• IOPS = 6581, MBPS = 51      (vs . 22100 IOPS, 173 MBPS no mem pressure)
• Reason; write throttling:
  ios=216/151456
• Why? Available: 1018 MB, vm.dirty_ratio: 30% = 305MB
Write: 500M / 50% of 4G memory taken

- How about writing 500M? That was really fast previously?

- Run a fio write test writing 500M:

```sh
fio --name test --filename /tmp/fiotest --ioengine sync --rw randwrite --bs 8k --filesize 500M
```
Write: 500M / 50% of 4G memory taken

- This is a summary from the run:

  IOPS=25.1k, BW=196MiB/s (206MB/s) (500MiB/2549msec) (332ms)

  - My limits are 125 MBPS and 3000 IOPS.
  - IOPS = 25100, MBPS = 196 (vs. 193000 IOPS, 1506 MBPS no mem pressure)
  - Despite feeling fast, performance was severely impacted!!
  - Reason; write throttling:
    ios=17/20846

Conclusion

• If you are using buffered IO, do you rely on caching for performance?
• Are you keeping track of Available Memory?
• Available memory \approx \text{memory acting as/available for cache + regular alloc.}

• Understand the differences between read and write cache properties:
  • Data must be read before it can be cached and reused.
  • A variable proportional limit is imposed on # dirty buffers.
    • Kernel applies write throttling when # dirty pages increases.
Conclusion

• You have to understand your *active dataset*
  • Which consists of reads and writes.
  • The cache effectivity is relative to available memory.

• This means you might seem to suffer random IO performance issues.
  • Which can be caused by either:
    • Change in the active dataset.
    • Change in available memory
PS

- The tests were performed on Linux without swap.
  - Buffered IO needs buffers which is a memory allocation.
  - It competes with regular IO allocations.

- Therefore, Linux will evaluate available memory using a LRU mechanism.
  - Therefore, bursts of IO buffers usage
  - Could push seldom used mapped allocations to swap.