MATEUSZ PIOTROWSKI

OTTAWA 2025-06-13

ZFS Direct IO Benchmarking Pitfalls

- Mateusz Piotrowski <0mp@FreeBSD.org>
- Based in Berlin
- FreeBSD user since 2016
- Committer since 2018

Areas of interest:

Ports | rc(8) | ZFS | Benchmarking

Not holding direct IO wrong is a great first step to potential performance gains!

ZFS 101

1

Pitfalls

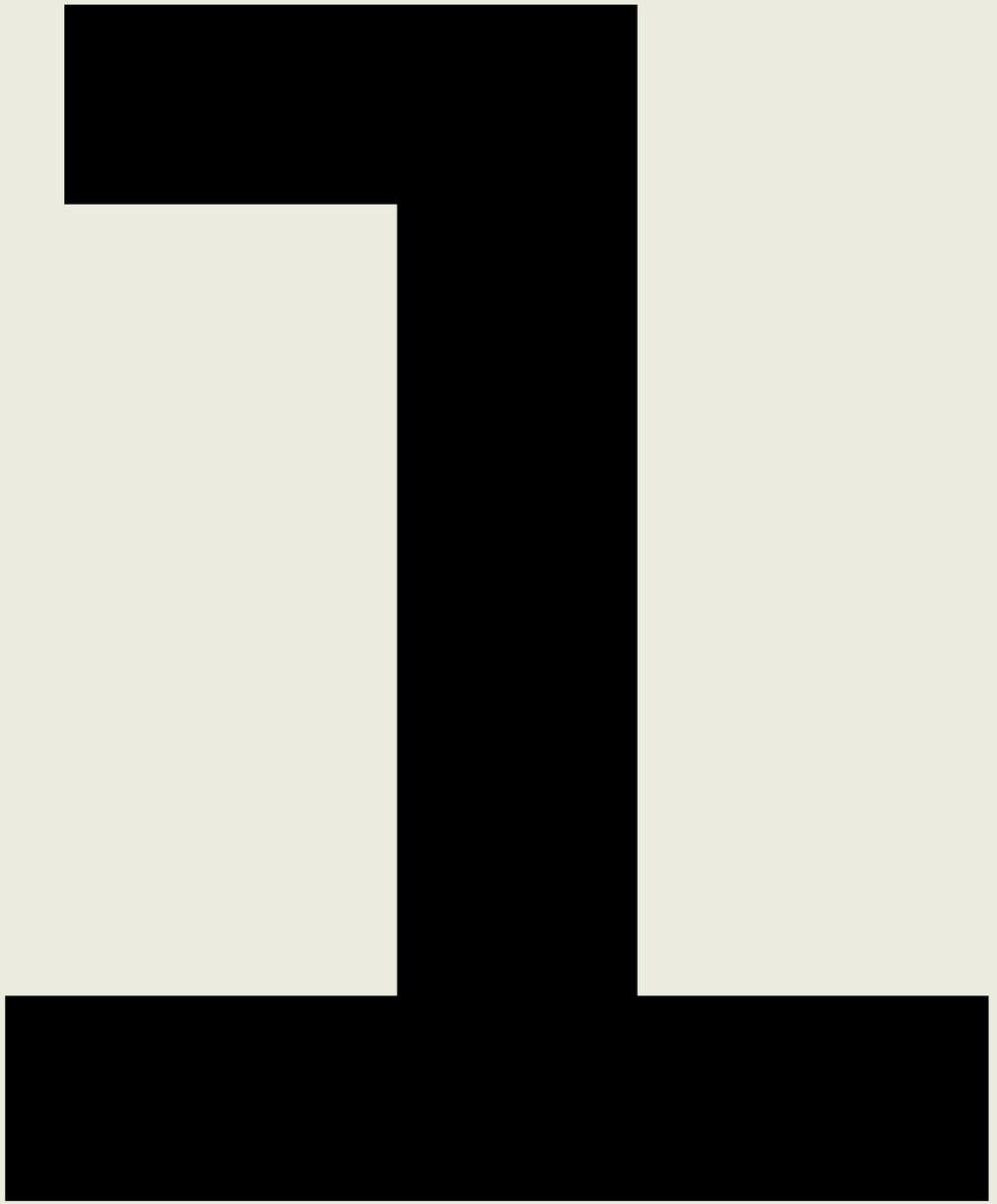
2

Implementation

3

ZFS 101

MATEUSZ PIOTROWSKI



OTTAWA 2025-06-13

ZFS Architecture Overview

MATEUSZ PIOTROWSKI

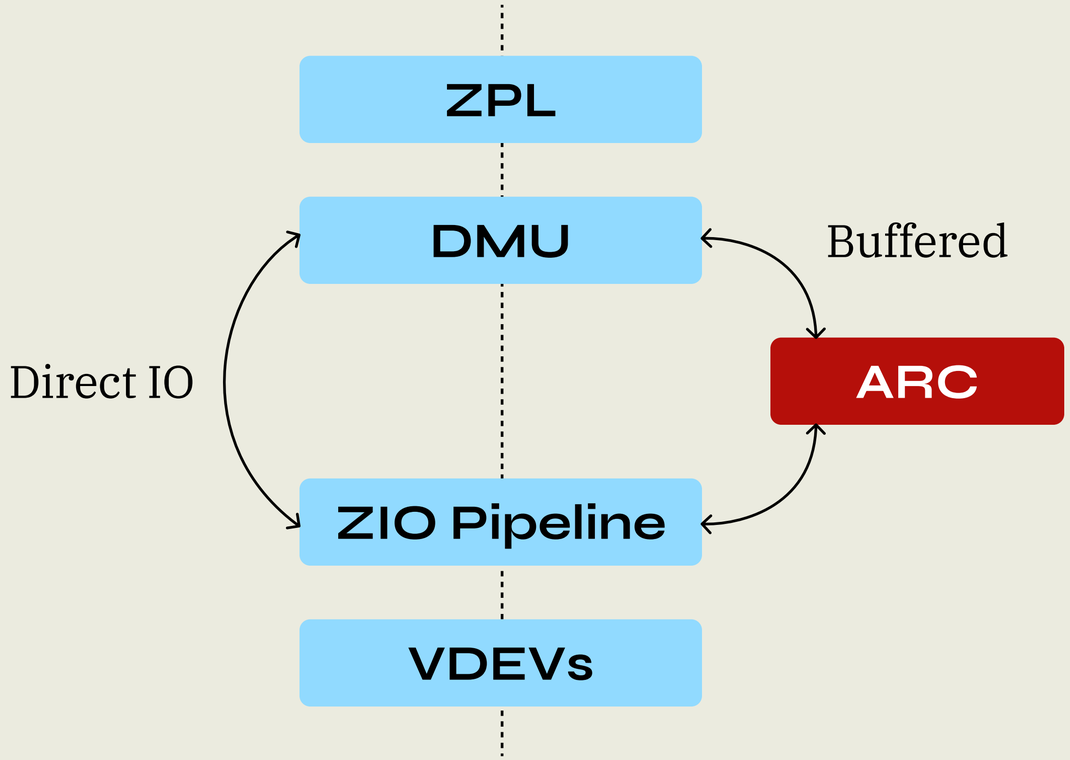
OTTAWA 2025-06-13

- Copy-on-write file system
  - Unlocks pretty exciting features!
- History
  - Developed at Sun Microsystems in 2001
  - Imported into FreeBSD 2008
  - As of 2025, FreeBSD is using OpenZFS

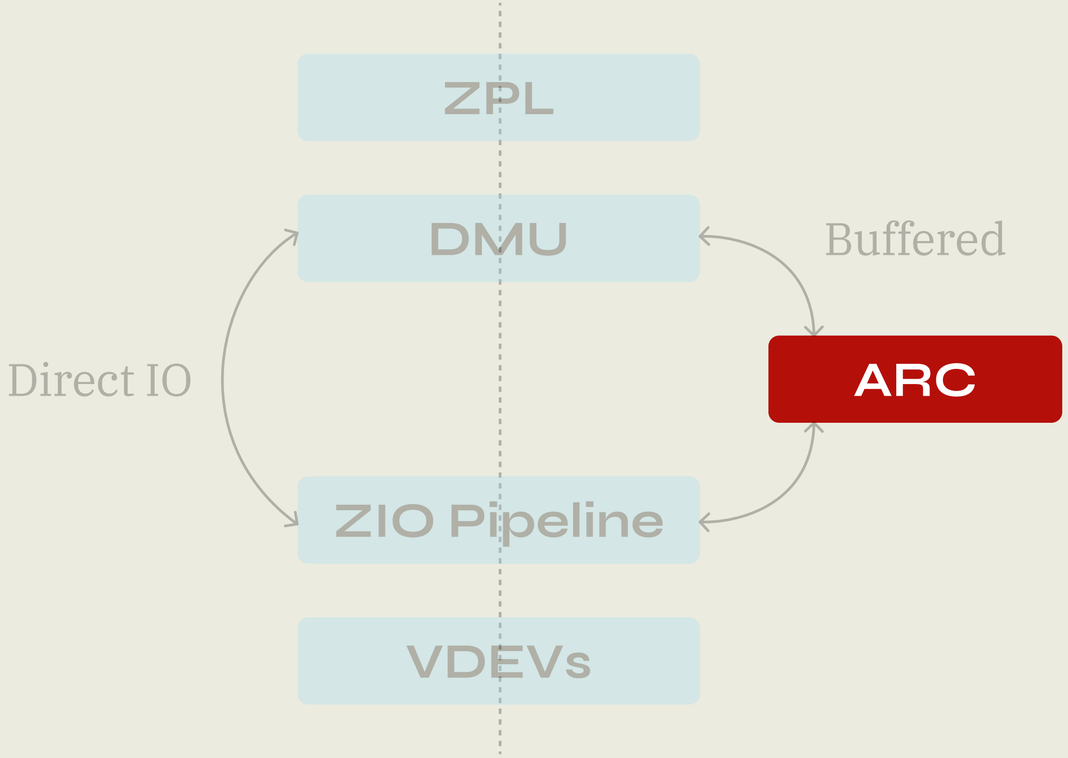
- Data integrity
  - Checksummed blocks
  - Silent data corruption detection and correction
- Data consistency
  - State gets updated at checkpoints
- Pooled storage
  - No need to partition disks in advance
  - Easier partition creation, growing, and shrinking

- Snapshots
- Efficient remote replication
- Compression
- Encryption
- Deduplication
- RAIDZ
- Quotas
- Boot environments
- ...

- Essentially:
  - ZPL: VFS⇆ZFS
  - DMU: manages dnodes (e.g., filesystems, snapshots, ...)
  - ARC caches blocks
  - ZIO orchestrates all read and write operations
  - VDEVs aggregate disks into RAIDZ groups



- Lots of useful caching
  - Balances between Most Recently Used & Most Frequently Used
  - Ghost lists
  - Uses unused RAM
- Tradeoff:
  - Better latency and throughput
  - Higher CPU and RAM use



- O_DIRECT
- Introduced in XFS by IRIX for database workloads
- Purpose:
  - Bypass caches to prevent unnecessary IO
- Not standardized
  - OpenZFS's implementation is based on documents like Linux open(2) and conventions

- Application with its own caching *(e.g., databases)*
- Write-and-forget *(e.g., application checkpointing)*
- Read-and-forget *(e.g., restoring a backup)*
- NVMe pools (but not only!)

Pitfalls

MATEUSZ PIOTROWSKI

2

OTTAWA 2025-06-13

Setup

- **Hardware**
  - Intel Core i7-8550U (4 cores + 4 HT)
  - RAM: 16 GB
  - Disk: 64-GB CFast card connected via USB 3
- **Software**
  - FreeBSD 15.0-CURRENT amd64
  - ZFS 2.3.99-323-FreeBSD_g246e5883b
    - Record size: 1m
  - Fio 3.38
  - Hyperfine 1.19.0

- ***Let's take a look at a sample workload.***

```
# Revision 1

[global]
group_reporting=1
end_fsync=1
rw=write
numjobs=4
bs=128k
filesize=128k
nr_file=1000

[raw]
filename=/dev/da0
direct=1

[direct]
directory=/tank
direct=1

[buffered]
directory=/tank
direct=0
```

```
Benchmark 2: make buffered
  Time (mean ± σ):     12.140 s ±  0.314 s    [User: 0.034 s, System: 0.040 s]
  Range (min … max):   11.781 s … 12.364 s    3 runs
```

w/o direct IO

```
Benchmark 3: make direct
  Time (mean ± σ):     12.342 s ±  0.220 s    [User: 0.057 s, System: 0.050 s]
  Range (min … max):   12.153 s … 12.583 s    3 runs
```

w/ direct IO

```
$ sysctl kstat.zfs.tank.misc.iostats | grep direct
kstat.zfs.tank.misc.iostats.direct_write_bytes: 0
kstat.zfs.tank.misc.iostats.direct_write_count: 0
kstat.zfs.tank.misc.iostats.direct_read_bytes: 0
kstat.zfs.tank.misc.iostats.direct_read_count: 0
```

```
# sysctl vfs.zfs.dio_enabled
vfs.zfs.dio_enabled: 0

# sysctl vfs.zfs.dio_enabled=1
vfs.zfs.dio_enabled: 0 -> 1
```

```
$ sysctl kstat.zfs.tank.misc.iostats | grep direct
kstat.zfs.tank.misc.iostats.direct_write_bytes: 0
kstat.zfs.tank.misc.iostats.direct_write_count: 0
kstat.zfs.tank.misc.iostats.direct_read_bytes: 0
kstat.zfs.tank.misc.iostats.direct_read_count: 0
```

```
$ zfs get recordsize,direct -Ho value tank
1M
standard
```

```
# Revision 1

[global]
 group_reporting=1
 end_fsync=1
 rw=write
 numjobs=4
 bs=128k
 filesize=128k
 nr_file=1000

[raw]
 filename=/dev/da0
 direct=1

[direct]
 directory=/tank
 direct=1

[buffered]
 directory=/tank
 direct=0
```

Iteration #2

- O_DIRECT is ignored if a file is smaller than the record size

- ***Let's try something else:***
  - Just 1 file per fio job
  - Big files: 4 GB
  - io_size makes the benchmark run shorter
  - offset_increment helps with spreading writes on raw disk

```
# Revision 2

[global]
group_reporting=1
end_fsync=1
rw=write
numjobs=4
bs=128k
size=4g
io_size=10m
offset_increment=25%

...
```

```
Benchmark 2: make buffered
  Time (mean ± σ):      4.289 s ±  0.195 s    [User: 0.028 s, System: 0.047 s]
  Range (min … max):    4.156 s …  4.513 s    3 runs
```

w/o direct IO

```
Benchmark 3: make direct
  Time (mean ± σ):      4.182 s ±  0.049 s    [User: 0.038 s, System: 0.059 s]
  Range (min … max):    4.132 s …  4.230 s    3 runs
```

w/ direct IO

```
$ sysctl kstat.zfs.tank.misc.iostats | grep direct
kstat.zfs.tank.misc.iostats.direct_write_bytes: 0
kstat.zfs.tank.misc.iostats.direct_write_count: 0
kstat.zfs.tank.misc.iostats.direct_read_bytes: 0
kstat.zfs.tank.misc.iostats.direct_read_count: 0
```

```
# Revision 2

[global]
group_reporting=1
end_fsync=1
rw=write
numjobs=4
bs=128k
size=4g
io_size=10m
offset_increment=25%

[raw]
filename=/dev/da0
direct=1

[direct]
directory=/tank
direct=1

[buffered]
directory=/tank
direct=0
```

Iteration #3

- O_DIRECT is ignored if the issued writes are not record size aligned
  - Also must be PAGE_SIZE aligned

- ***Let's try something else:***
  - Big block size of 16m

```
# Revision 3

[global]
group_reporting=1
end_fsync=1
rw=write
numjobs=4
bs=16m
size=4g
io_size=300m
offset_increment=25%

...
```

```
Benchmark 1: make raw
  Time (mean ± σ):     22.379 s ±  0.822 s    [User: 0.044 s, System: 0.078 s]
  Range (min … max):   21.690 s … 23.289 s    3 runs

Benchmark 2: make buffered
  Time (mean ± σ):     21.011 s ±  0.357 s    [User: 0.039 s, System: 0.037 s]
  Range (min … max):   20.728 s … 21.412 s    3 runs

Benchmark 3: make direct
  Time (mean ± σ):     20.815 s ±  0.699 s    [User: 0.039 s, System: 0.068 s]
  Range (min … max):   20.385 s … 21.621 s    3 runs

Summary
  make direct ran
    1.01 ± 0.04 times faster than make buffered
    1.08 ± 0.05 times faster than make raw
```

```
$ sysctl kstat.zfs.tank.misc.iostats | grep direct
kstat.zfs.tank.misc.iostats.direct_write_bytes: 1270874112
kstat.zfs.tank.misc.iostats.direct_write_count: 1212
kstat.zfs.tank.misc.iostats.direct_read_bytes: 0
kstat.zfs.tank.misc.iostats.direct_read_count: 0
```

Iteration #4

- Quite exciting!
- Let’s try to find a workload where direct IO performs better than buffered

- ***Let’s try something else:***
  - Block size of 1m

```
# Revision 4

[global]
group_reporting=1
end_fsync=1
rw=write
numjobs=4
bs=1m
size=4g
io_size=300m
offset_increment=25%

...
```

```
Benchmark 1: make raw
  Time (mean ± σ):     24.401 s ±  3.476 s    [User: 0.078 s, System: 0.050 s]
  Range (min … max):   21.286 s … 28.151 s    3 runs

Benchmark 2: make buffered
  Time (mean ± σ):     43.981 s ±  8.581 s    [User: 0.033 s, System: 0.049 s]
  Range (min … max):   34.112 s … 49.687 s    3 runs

Benchmark 3: make direct
  Time (mean ± σ):     30.582 s ±  2.279 s    [User: 0.058 s, System: 0.051 s]
  Range (min … max):   28.092 s … 32.565 s    3 runs

Summary
  make raw ran
    1.25 ± 0.20 times faster than make direct
    1.80 ± 0.44 times faster than make buffered
```

Finally, we see a difference.

1

Direct IO Activation

sysctl vfs.zfs.dio_enabled=1 is rather important.

2

No Small Files

Direct IO is ignored for files smaller than the record size.

3

PAGE_SIZE Alignment

Direct IO must be PAGE_SIZE aligned.

4

Record Size Alignment

Direct IO writes must be record size aligned

5

Disk’s Raw Performance

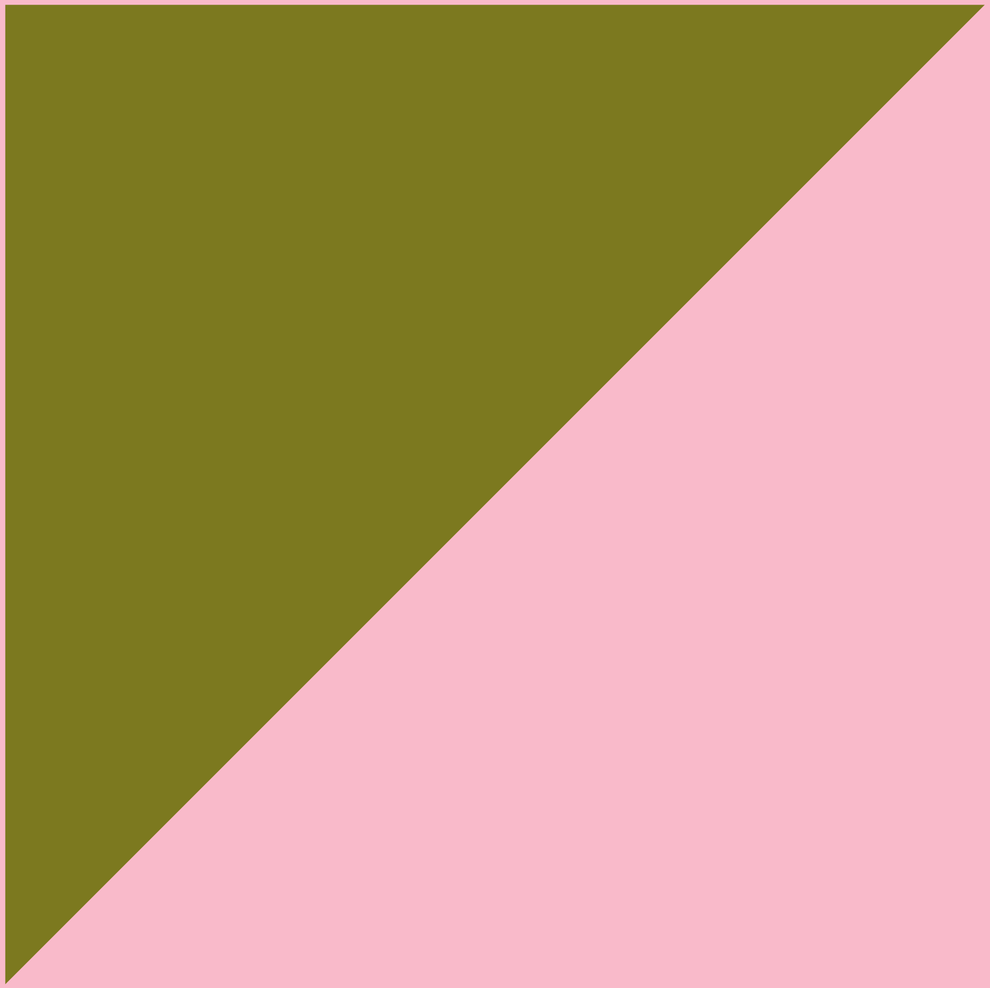
Direct IO will not be faster than your disk.

6

Fast Hardware

Direct IO probably will not have a positive impact on slow hardware.

Implementation



3

MATEUSZ PIOTROWSKI

OTTAWA 2025-06-13

How Does Direct IO Fit ZFS?

MATEUSZ PIOTROWSKI

OTTAWA 2025-06-13

- Direct IO does not imply O_SYNC
  - Need to sync to ensure that asynchronous calls made it to disk
- Indirect blocks are still buffered
  - Direct IO is about user buffers (i.e., the actual data)
- direct=standard vs direct=always
  - “standard” will honor O_DIRECT and error out if used incorrectly
  - “always” will try its best to make all IO skip the ARC

- Write IO
  - Record size & PAGE_SIZE alignment
- Read IO
  - PAGE_SIZE alignment
- ZFS will make sure to keep our data coherent
- mmap’ed files: always buffered
- Modifying inflight buffers is safe, but causes redirection to the ARC

- Unimplemented cases:
  - Direct IO to zvols
  - Mixing deduplication with direct IO writes

- Direct IO is
  - relatively easy to use
  - definitely not something to be enabled without testing
- Right alignment is critical
- Try tuning the ARC or adding an L2ARC first
- ZFS + Direct IO will not eat your data

0mp@FreeBSD.org

Any questions?

BSDCan 2025

Thanks!

MATEUSZ PIOTROWSKI

OTTAWA 2025-06-13