Fragging Rights

A Tale of a Pathological Storage Workload

Eric Sproul, Circonus

ZFS User Conference
March 2017

CIRCONUS

Eric Sproul

Circonus Release Engineer

Wearer of other hats as well (ops, sales eng., support)

ZFS user since Solaris 10u2 (>10 years ago now??)

Helped bring OmniOS to the world @ OmniTI

@eirescot

esproul

A Tale of Surprise and Pain

ZFS handled it well… until it didn't.

We'll talk about free space, COW, and the hole we dug for ourselves:

- How ZFS manages free space
- Our workload
- Problem manifestation
- Lessons learned and future direction

Free Space in ZFS

ZFS tracks free space with **space maps**

Time-ordered log of allocations and frees

Top-level vdevs divided into a few hundred **metaslabs**,
each with its own space map

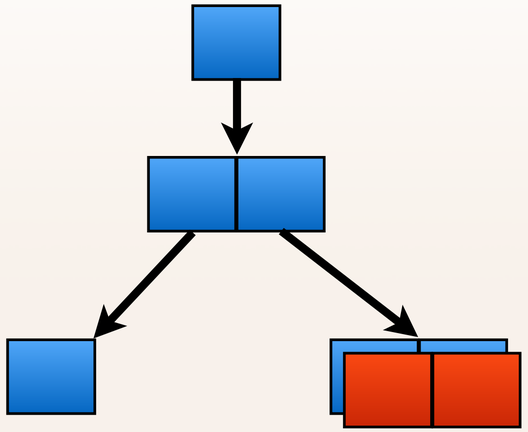
At metaslab load, the map is read into AVL tree in memory

For details, see Jeff Bonwick's excellent explanation of space maps:
http://web.archive.org/web/20080616104156/http://blogs.sun.com/bonwick/entry/space_maps

Copy-on-Write

Never overwrite existing data.

“Updating a file” means new bits written to previously free space, followed by freeing of old chunk.



The Workload

“It seemed like a good idea at the time…”

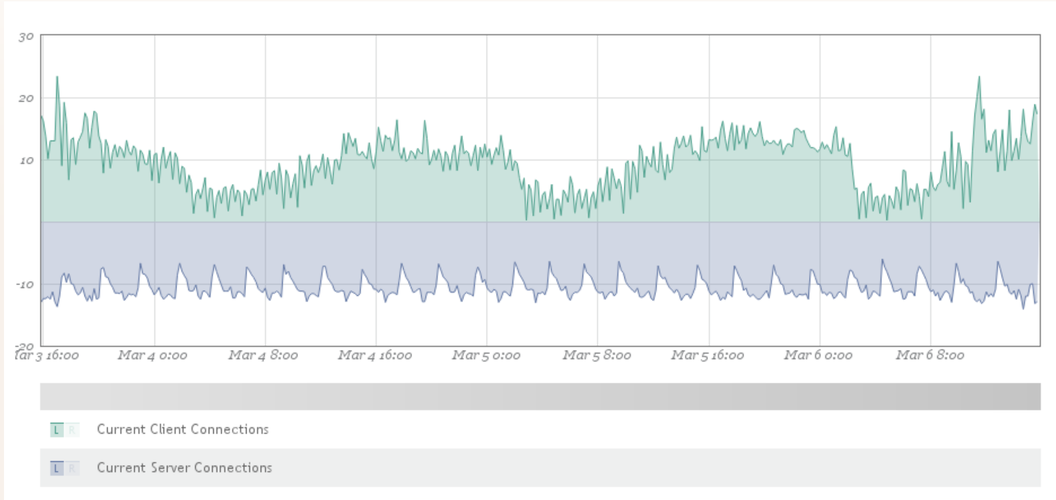
Scalable time-series data store

Compact on-disk format while not sacrificing granularity

Enable rich analysis of historical telemetry data

Store *value* plus 6 other pieces of info:

- variance of *value* (stddev)
- derivative (change over time) and stddev of derivative
- counter (non-negative derivative) and stddev of counter
- count of samples that were averaged to produce *value*



The Workload

File format is columnar and offset-based, with 32-byte records.

Common case is appending to the end (most recent measurement/rollup).

No synchronous semantics used for updating data files
(this is a cluster and we have write-ahead logs).

5x 1m records = 32 * 5 = 160 bytes append
1x 5m rollup = 32 bytes append
1x 3h rollup = 32 bytes append, then update w/ recalculated rollup

The Workload

With **copy-on-write**, every Circonus record append or overwrite modifies a ZFS block, freeing the old copy of that block.

ZFS has **variable block sizes**: minimum is a single disk sector (512b/4K), max is recordsize property (we used 8K).

When the tail block reaches 8K, it stops getting updated and a new tail block starts.

The Workload

ZFS sees tail block updated every 5 minutes for:

1m files: 8192/160 = 51 times (~4 hours)

5m files: 8192/32 = 256 times (~21 hours)

3h files: **tragic**

- last record written, then rewritten 35 more times
  (as 3h rollup value gets recalculated)
- 256 records to fill 8K, ZFS sees 256*36 = 9216 block updates (32 **days**)

Alas, we also use compression, so it's actually ~2x worse than this.

The Problem

After “some time”, depending on ingestion volume and pool/vdev size, performance degrades swiftly.

TXGs take longer and longer to process, stalling the ZIO pipeline.

ZIO latency bubbles up to userland; application sees increasing write syscall latency, lowering throughput.

Customers are sad. 😔

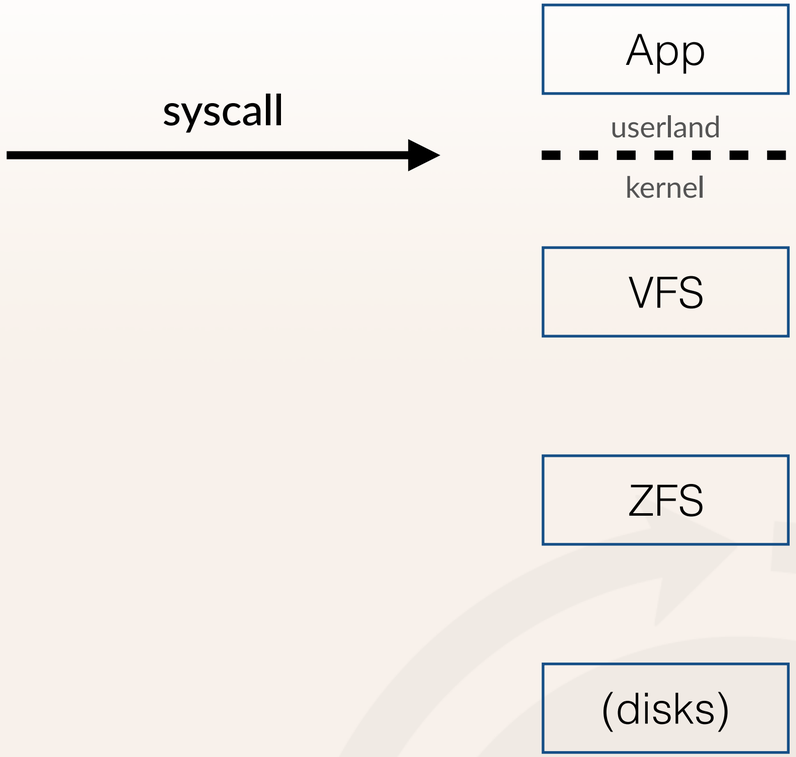
Troubleshooting

Start with what the app sees: DTrace syscalls

```
(us) syscall: pwrite bytes: 32
           value  ------------- Distribution ------------- count
               4 |                                         0
               8 |                                         878
              16 |@@@@@@@@@@@@                             27051
              32 |@@@@@@@@@@@@@@@@                         34310
              64 |@@@@@@                                   13361
             128 |@                                        2586
             256 |                                         148
             512 |                                         53
            1024 |                                         39
            2048 |                                         33
            4096 |                                         82
            8192 |                                         534
           16384 |@@@@                                     8614
           32768 |                                         474
           65536 |                                         22
          131072 |                                         0
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         36
         2097152 |                                         335
         4194304 |                                         72
         8388608 |                                         0
```

bad

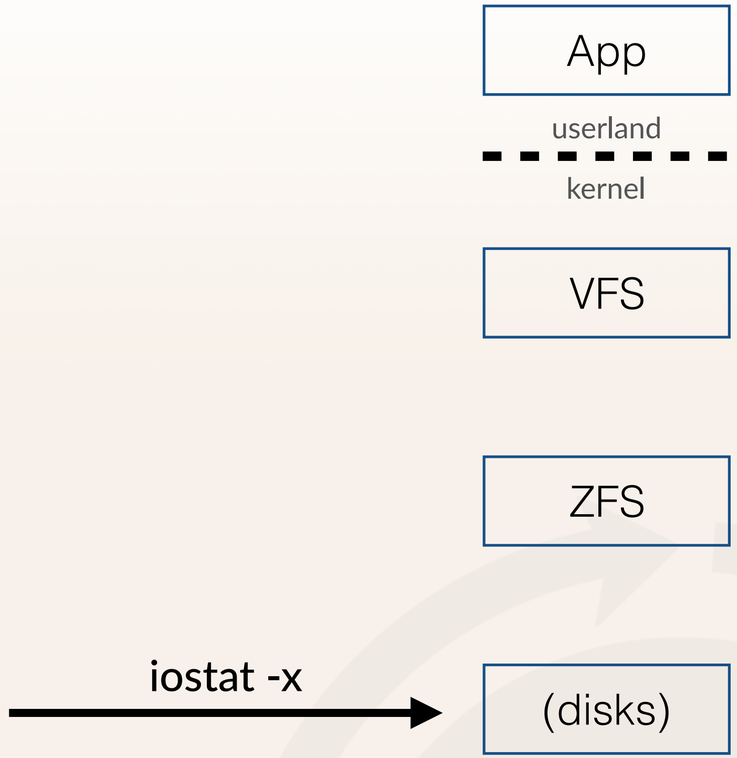
lolwut?!?



Troubleshooting

Slow writes must mean disks are saturated, right?

```
                 extended device statistics
device     r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
data     918.3 2638.5 4644.3 32217.0 1362.7 11.8  386.4  21  40
rpool      0.9    4.4    3.5   22.3  0.0  0.0    2.0   0   0
sd0        0.9    4.4    3.5   22.3  0.0  0.0    0.5   0   0
sd6       67.6  175.1  347.9 2299.8  0.0  0.6    2.6   0  13
sd7       64.4  225.5  335.9 2300.7  0.0  0.7    2.3   0  14
sd8       67.3  167.8  314.5 2300.4  0.0  0.6    2.6   0  13
sd9       65.3  173.8  326.7 2299.8  0.0  0.6    2.6   0  13
sd10      66.1  226.6  332.2 2300.7  0.0  0.6    2.2   0  13
sd11      67.2  153.9  338.8 2301.4  0.0  0.4    2.0   0  11
sd12      69.4  154.6  345.5 2301.4  0.0  0.4    2.0   0  11
sd13      64.0  162.0  321.9 2300.8  0.0  0.4    2.0   0  11
sd14      65.5  163.7  328.7 2300.8  0.0  0.4    2.0   0  11
sd15      64.5  221.4  343.9 2303.5  0.0  0.8    2.7   0  15
sd16      61.1  222.6  318.1 2303.5  0.0  0.8    2.8   0  15
sd17      63.5  211.3  338.1 2303.2  0.0  0.7    2.7   0  14
sd18      63.8  213.0  330.6 2303.2  0.0  0.7    2.6   0  14
sd19      68.7  170.0  321.9 2300.4  0.0  0.6    2.5   0  13
```



CIRCONUS

What Now?

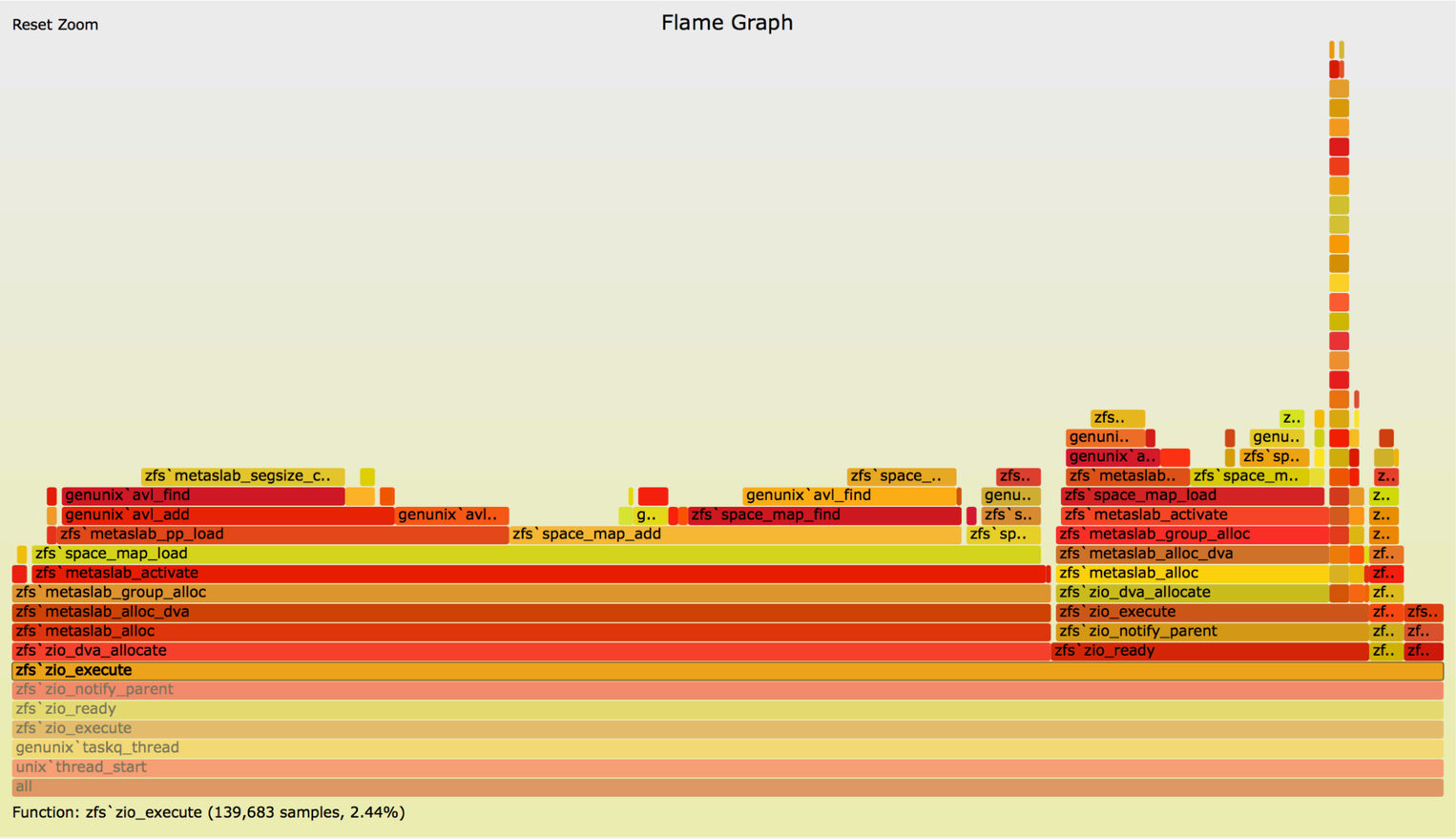
We know the problem is in the write path, but it's not the disks.

TL;DR is that ZFS internals are *very* complicated and difficult to reason about, especially with legacy ZFS code (OmniOS r151006, circa 2013).

We flailed around for a while, using DTrace to generate kernel flame graphs,
to get a sense of what the kernel was up to.

Kernel Stack Profiling

Flame graphs!



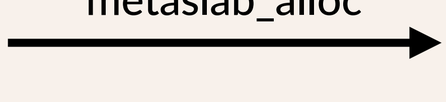
https://github.com/brendangregg/FlameGraph

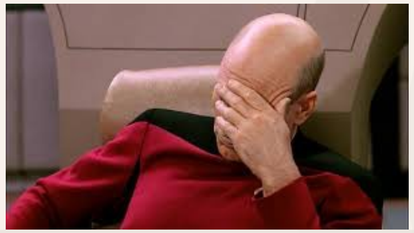
System mostly idle
(not shown),
but when not, it's ZFS.

Slab Allocation

>1s to find free space?!?

```
metaslab_alloc
time (nsec)
         value  ------------- Distribution ------------- count
          1024 |                                         0
          2048 |                                         1166
          4096 |@@@@@                                    41714
          8192 |@@@@@@@@@@@@@@@@                         134407
         16384 |@@@@@@@@@@@@@                            107140
         32768 |@@                                       17603
         65536 |@                                        10066
        131072 |@                                        10315
        262144 |@                                        8144
        524288 |                                         3715
       1048576 |                                         1598
       2097152 |                                         581
       4194304 |                                         75
       8388608 |                                         49
      16777216 |                                         0
      33554432 |                                         0
      67108864 |                                         0
     134217728 |                                         0
     268435456 |                                         0
     536870912 |                                         0
    1073741824 |                                         1276
    2147483648 |                                         0
```





Troubleshooting: Recap

What do we know now?

Starting with application-perceived latency:

- pwrite(2) syscall is slow.
- Slow because ZFS takes so long to allocate new space for writes.
- We don't know precisely why these allocations are bogged down, but it likely involves the nature of the pool's free space.

Quantity vs. Quality

Aggregate free space isn't everything

Needed to visualize existing metaslab allocations:

```
# zdb -m data

Metaslabs:
        vdev          0
        metaslabs   116   offset                spacemap          free
        ---------------   -------------------   ---------------   -------------
        metaslab      0   offset            0   spacemap     38   free    2.35G
        metaslab      1   offset    100000000   spacemap    153   free    2.19G
        metaslab      2   offset    200000000   spacemap    156   free    1.70G
        metaslab      3   offset    300000000   spacemap    158   free     639M
        metaslab      4   offset    400000000   spacemap    160   free    1.11G
```

Visualizing Metaslabs

vdev0												mslab
58%	57%	42%	16%	28%	34%	43%	53%	26%	9%	5%	65%	12
64%	65%	66%	64%	62%	66%	66%	63%	65%	47%	46%	65%	24
68%	63%	65%	68%	67%	67%	67%	68%	68%	68%	67%	68%	36
66%	65%	68%	68%	72%	71%	68%	69%	73%	69%	73%	72%	48
71%	72%	72%	72%	71%	74%	67%	71%	75%	71%	73%	71%	60
73%	72%	77%	70%	58%	69%	74%	63%	74%	74%	78%	83%	72
77%	76%	83%	76%	68%	73%	70%	75%	78%	80%	79%	77%	84
76%	75%	72%	82%	72%	75%	83%	82%	76%	76%	76%	73%	96
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	108
100%	100%	100%	100%	100%	100%	100%	100%					120

Color indicates fullness;
Green = empty
Red = full

Percentage is remaining free space.

Credit for the idea: https://www.delphix.com/blog/delphix-engineering/zfs-write-performance-impact-fragmentation

Visualizing Metaslabs

vdev0												mslab
23%	51%	23%	78%	22%	13%	12%	10%	11%	11%	16%	14%	12
32%	24%	18%	23%	20%	22%	20%	15%	15%	27%	14%	20%	24
18%	26%	33%	32%	40%	26%	25%	35%	27%	25%	27%	45%	36
35%	43%	46%	38%	50%	51%	54%	46%	32%	34%	22%	4%	48
4%	4%	85%	100%	100%	100%	100%	100%	100%	100%	100%	100%	60
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	99%	100%	72
99%	100%	100%	100%	100%	100%	100%	99%	100%	99%	100%	100%	84
100%	100%	100%	92%	100%	100%	100%	100%	100%	100%	100%	100%	96
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	108
100%	100%	100%	100%	100%	100%	100%	100%					120

After data rewrite, allocations are tighter.

Performance problem is gone.

Did we just create "ZFS Defrag"?

Spacemap Detail

From one slab, on one vdev, using `zdb -mmm`:

```
Metaslabs:
        vdev          0
        metaslabs   116   offset                spacemap          free
        --------------   -------------------   --------------   -------------
        metaslab     39   offset   9c00000000   spacemap    357   free    13.2G
                          segments    1310739   maxsize    168K   freepct   82%
            [     0] ALLOC: txg 7900863, pass 1
            [     1]    A  range: 9c00000000-9c00938800  size: 938800
            [     2]    A  range: 9c00939000-9c0093d200  size: 004200
            ...
            [4041229] FREE: txg 7974611, pass 1
            [4041230]    F  range: 9d79d10600-9d79d10800  size: 000200
            [4041231]    F  range: 9e84efe400-9e84efe600  size: 000200
            [4041232] FREE: txg 7974612, pass 1
            [4041233]    F  range: 9e72ba4600-9e72ba5400  size: 000e00
```

90th %ile freed size: ~9K

31% of frees are 512b-1K

>4M records in one spacemap...
Costly to load, only to discover you can't allocate from it!
Probably contributes to those long metaslab_alloc() times.

OpenZFS Improvements

We weren't alone!

Since our initial foray into this issue, new features have come out:

- Spacemap histograms
  - Visible via zdb (-mm) and mdb (::spa -mh)
- Metaslab fragmentation metrics
- Allocator changes to account for fragmentation
- New tuning knobs* for write throttle

** http://dtrace.org/blogs/ahl/2014/08/31/openzfs-tuning/*

Spacemap Histogram

```
metaslab      2   offset    400000000   spacemap    227   free    7.99G
On-disk histogram:              fragmentation 90
                  9: 632678 ****************************************
                 10: 198275 ************
                 11: 342565 *********************
                 12: 460625 *****************************
                 13: 213397 *************
                 14:  82860 ******
                 15:   9774 *
                 16:    137 *
                 17:      1 *
```

Key is power-of-2 range size

Fragmentation metric is based on this distribution

What We Learned

"Doctor, it hurts when I do <this>..."
"Then don't do that."

Our workload is (sometimes) pathologically bad at scale

- Performance is fine until some percentage of metaslabs are "spoiled", even though overall pool used space is low.
- Once in this state, only solution is bulk data rewrite.
- Happens sooner if you have fewer/smaller slabs.
- Happens sooner if you increase ingestion rate.

What We're Doing About It

Follow doctor's orders

Avoid those single-column updates

- Use in-memory DB to accumulate incoming data
- Batch-update columnar files with large, sequential writes
- Eventually replace columnar files with some other DB format

Thanks for listening!

Questions? @eirescot

Eric Sproul, Circonus

ZFS User Conference
March 2017

CIRCONUS