OpenZFS

ZFS Capacity Usage Simulator

Kody Kantor - Joyent

Tough questions

OpenZFS

- How much space does it take to store a file?
- How much capacity overhead does ZFS introduce?

Tough questions

OpenZFS

- How much space does it take to store a file?
- How much capacity overhead does ZFS introduce?
- How do recordsize, stripe width, compression, etc. impact capacity used by the data we have in production?

Impetus: recordsize

OpenZFS

- A surprisingly tricky setting to understand!
- Lots of questions internally and in the community
- We know it impacts performance, but can it impact capacity?

Option A - Unlimited time and hardware

OpenZFS

- Write production data to a production-sized machine
- Observe capacity usage
- Rinse and repeat

Option A - Unlimited time and hardware

OpenZFS

- Write production data to a production-sized machine
- Observe capacity usage
- Rinse and repeat

Problems:

- Hundreds of TiB is expensive
- Access to production data
  - Is production data the same on every machine? Does it change over time?
- Boredom

Option B - No time or hardware

OpenZFS

- Gather stats about production data
- Write a script to simulate and account ZFS allocation
- Send production data stats through the script on your laptop

Option B - No time or hardware

OpenZFS

- Gather stats about production data
- Write a script to simulate and account ZFS allocation
- Send production data stats through the script on your laptop

Problems:

- That's what this talk is for!

- Theory: Fewer records means less parity which means less capacity used by the same data!
- Example 1:
  - 1M file, RAIDZ2, 4K disk sectors, recordsize=1M
  - 1x1M record + 2x4K parity sectors = 1M+8K
  - Parity overhead is **1/128th** of the file size!
- Example 2:
  - 1M file, RAIDZ2, 4K disk sectors, recordsize=128K
  - 8x128K records + 16x4K parity sectors = 1M+64K
  - Parity overhead is **1/16th** of the file size!
- Turns out this assumption is incorrect

Verifying understanding with ZDB...

OpenZFS

```
[root@coke /zones/testfs]# zfs set recordsize=1M zones/testfs
[root@coke /zones/testfs]# dd if=/dev/urandom of=/zones/testfs/uncompressed_file bs=1048576 count=1
1+0 records in
1+0 records out
1048576 bytes transferred in 0.028027 secs (35.7MB/sec)
[root@coke /zones/testfs]# zdb -vvvvvO zones/testfs uncompressed_file
obj=4 dataset=zones/testfs path=/uncompressed_file type=19 bonustype=44

    Object  lvl   iblk   dblk  dsize  dnsize  lsize   %full  type
         4    2   128K     1M  1.00M     512     1M  100.00  ZFS plain file (K=inherit) (Z=inherit)
                                          168   bonus  System attributes
...
Indirect blocks:
               0 L1  0:83253d400:400 20000L/400P F=1 B=393087/393087
               0  L0 0:92288f600:100000 100000L/100000P F=1 B=393087/393087
          100000  L0 0:0:0 20000L B=393087

            segment [0000000000000000, 0000000000100000) size    1M
```

First problem: Last record is big

OpenZFS

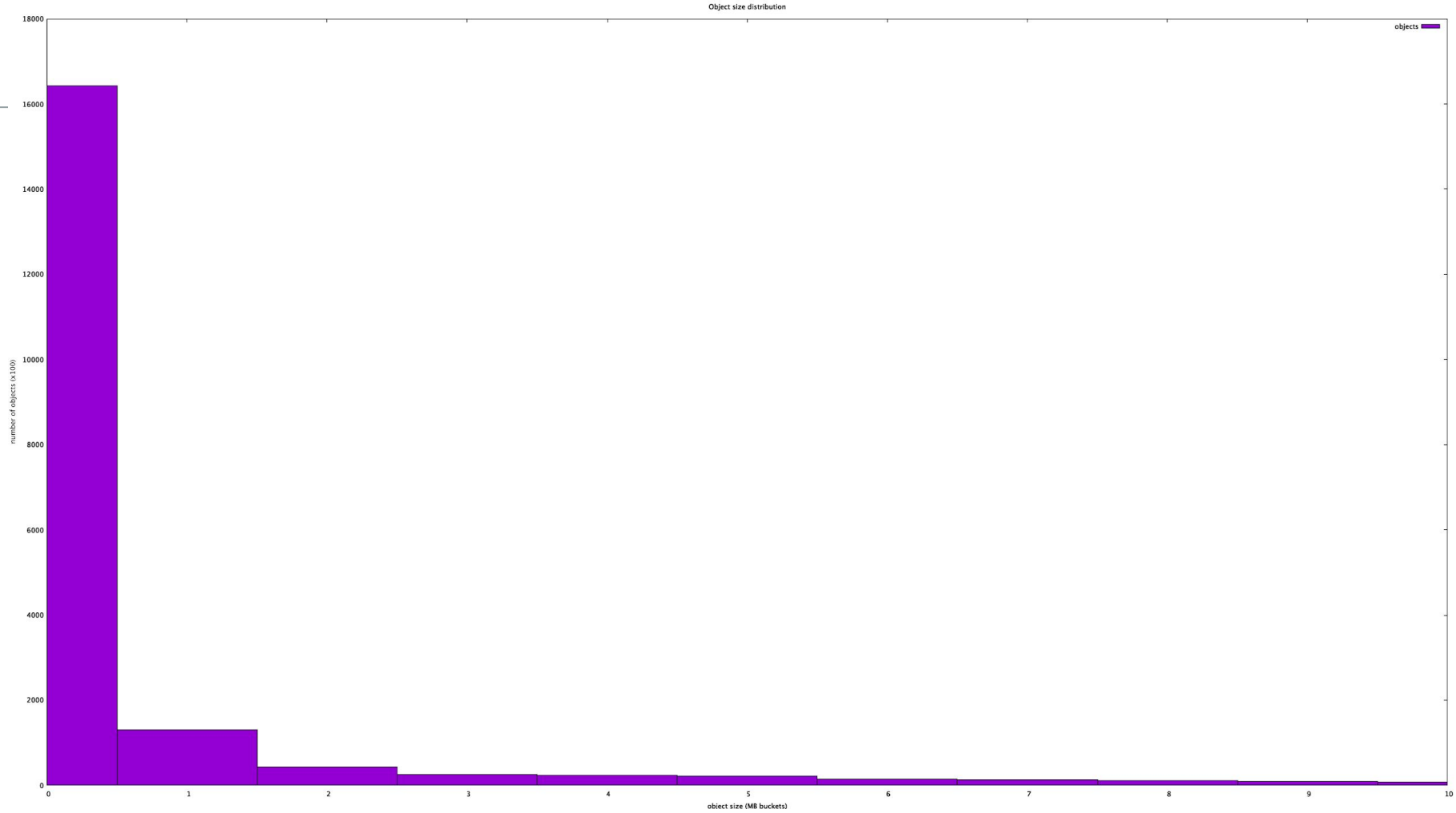
```
[root@coke /zones/testfs]# dd if=/dev/urandom of=/zones/testfs/uncompressed_file bs=1048577 count=1
1+0 records in
1+0 records out
1048577 bytes transferred in 0.027602 secs (36.2MB/sec)
[root@coke /zones/testfs]# zdb -vvvvvO zones/testfs uncompressed_file
obj=4 dataset=zones/testfs path=/uncompressed_file type=19 bonustype=44

    Object  lvl   iblk   dblk  dsize  dnsize  lsize   %full  type
         4    2   128K     1M  2.00M     512     2M  100.00  ZFS plain file (K=inherit) (Z=inherit)
                                        168   bonus  System attributes
...
Indirect blocks:
               0 L1  0:8325a1000:400 20000L/400P F=2 B=393124/393124
               0  L0 0:922c0ce00:100000 100000L/100000P F=1 B=393124/393124
          100000  L0 0:922d0ce00:100000 100000L/100000P F=1 B=393124/393124

            segment [0000000000000000, 0000000000200000) size    2M
```

First problem: Last record is big

OpenZFS

- The last record is not truncated
- Example 1: recordsize=1M, 1M file, ~**1M used**
- Example 2: recordsize=1M, 1,048,577 byte file (1M + 1 byte), **~2M used!**

- Not so trivial anymore. Changes in object size could lead to massive fluctuations in efficiency.


Object size distribution
objects
number of objects (x100)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
8
9
10
object size (MB buckets)

Second problem: Data doesn't fit in a spreadsheet

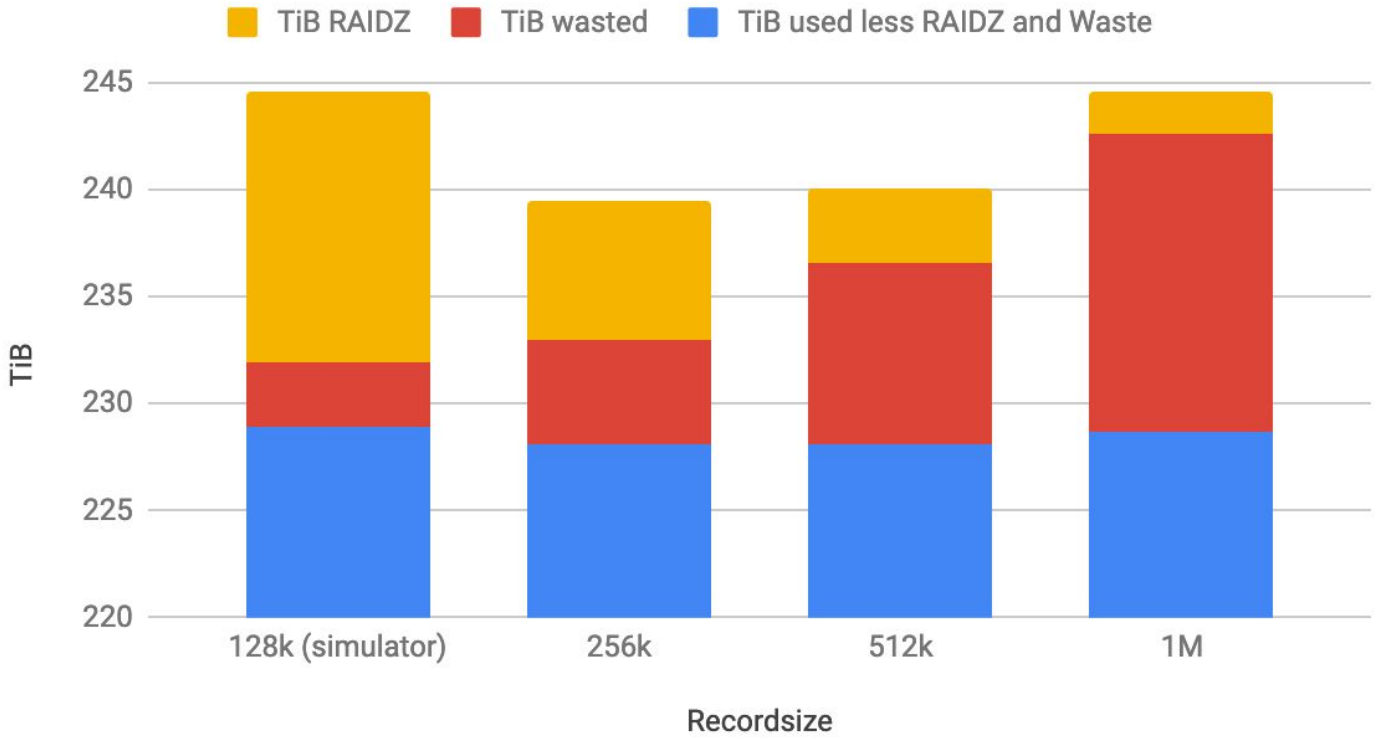
OpenZFS

- Millions of files on a pool
- A simulator is born!

```
$ head ./listing
user0 2801
user0 920
user0 2801
user2 13
user2 13
user2 13
user2 13
user2 13
user2 13
user2 13
```

```
$ ./zfs_usage_simulator.sh ./listing
Simulating using account *, parity 2, and recordsize
524288
=== REPORT ===
3297791040796           Bytes Used
121377069548            Wasted Bytes
6658153                 Records
13316306                RAIDZ sectors
3.00                    TiB Used
0.11                    TiB wasted
0.05                    TiB RAIDZ
```

TiB Used


TiB RAIDZ
TiB wasted
TiB used less RAIDZ and Waste
245
240
235
230
225
220
TiB
128k (simulator)
256k
512k
1M
Recordsize

Wed, Jun 26, 2019

David Pacheco: Just read the write-up on the recordsize simulator. Pretty cool! I'm even more convinced that it'd make a great talk or blog post.

kkantor: yeah, it's an interesting experiment. Thanks for taking a look

It occurred to me after we spoke yesterday that I never checked if using compression avoids the wasted space problem.

OpenZFS

```
[root@coke /zones/testfs]# zfs set compression=lz4 zones/testfs
[root@coke /zones/testfs]# dd if=/dev/urandom of=/zones/testfs/compressed_file bs=1048577 count=1
1+0 records in
1+0 records out
1048577 bytes transferred in 0.030710 secs (32.6MB/sec)
[root@coke /zones/testfs]# zdb -vvvvvO zones/testfs compressed_file
obj=6 dataset=zones/testfs path=/compressed_file type=19 bonustype=44

    Object  lvl   iblk   dblk  dsize  dnsize  lsize   %full  type
         6    2   128K     1M  1.01M     512     2M  100.00  ZFS plain file (K=inherit) (Z=inherit)
                                          168   bonus  System attributes
...
Indirect blocks:
               0 L1  0:6e2a16400:400 20000L/400P F=2 B=406248/406248
               0  L0 0:83253ec00:100000 100000L/100000P F=1 B=406248/406248
          100000  L0 0:8324ff800:1200 100000L/1200P F=1 B=406248/406248 <----- 0x1200 = 4608 bytes

             segment [0000000000000000, 0000000000200000) size    2M
```

Now the simulator doesn't match reality...

OpenZFS

- Read blog posts
- RAIDZ is fundamentally different than original understanding
- What I thought was all about recordsize is all about… everything!
- Need to take into account
  - Parity complexities (padding, stripe width)
  - Blockpointers
  - Minuscule allocations
  - Compression

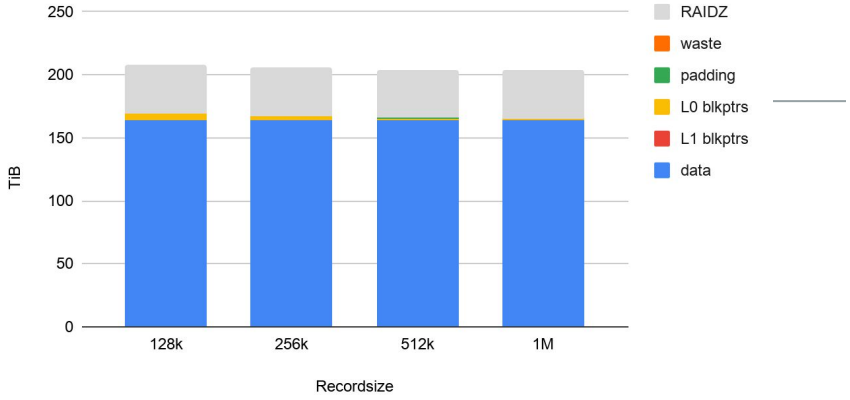
Now the simulator doesn't match reality...

OpenZFS

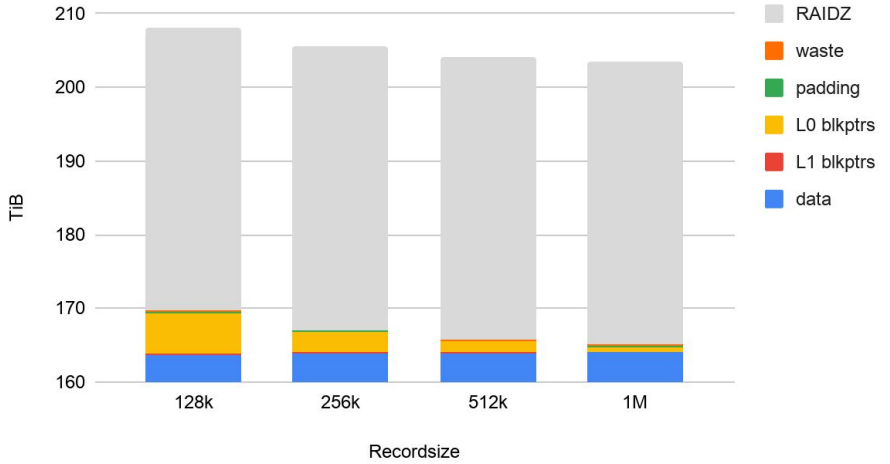
- Read blog posts
- RAIDZ is fundamentally different than original understanding
- What I thought was all about recordsize is all about… everything!
- Need to take into account
  - Parity complexities (padding, stripe width)
  - Blockpointers
  - Minuscule allocations
  - Compression

```
Simulating... account *, parity 2, recordsize 131072, raidz width 12, sector size 4096
=== REPORT ===
1443825068306             Records
208714706.91              GiB Used
183.85                    GiB wasted
35249253.92               GiB RAIDZ
166.06                    GiB Padding
5507755.54                GiB for blkptrs
60697.73                  GiB for ind blocks
```

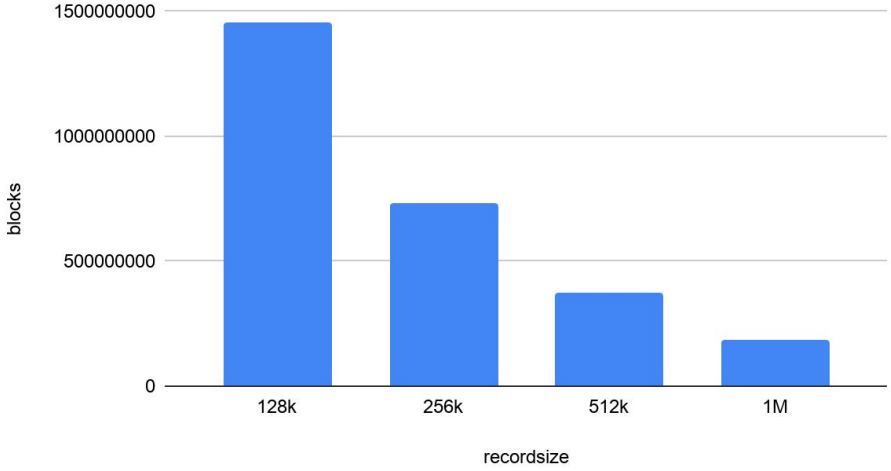
TiB Used



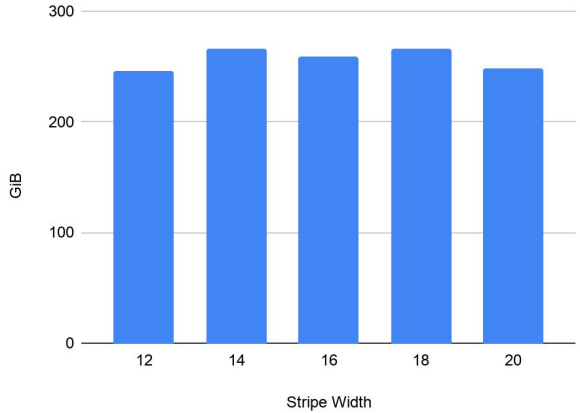
TiB Used (zoomed)



blocks vs. recordsize



Padding - RAIDZ3 128k records



Future ideas

OpenZFS

- Improve simulator
  - Pool usage (per-vdev)
  - MOS, ZAP
  - Other things!
- ‘null’ pool
  - A special zpool that accounts for all data, but does not store data
  - Pool still stores metadata, or the minimal data required to make the pool ‘work’

Helpful resources

OpenZFS

ZFS RAIDZ stripe width, or: How I Learned to Stop Worrying and Love RAIDZ

https://www.delphix.com/blog/delphix-engineering/zfs-raidz-stripe-width-or-how-i-learned-stop-worrying-and-love-raidz

Bruning Questions: ZFS Record Size

https://www.joyent.com/blog/bruning-questions-zfs-record-size

Simulator (need to find a better home)

https://github.com/KodyKantor/kodyops/commits/master/illumos/misc/zfs_usage_simulator.sh

zdb

https://github.com/illumos/illumos-gate/blob/master/usr/src/cmd/zdb/zdb.c
https://github.com/zfsonlinux/zfs/blob/master/cmd/zdb/zdb.c

Option C - Some hardware, some time

OpenZFS

- Create a ‘null’ pool
- Only stores the minimum necessary data to make the pool work

Benefits:

- Don’t need to know how ZFS works
- Should be more accurate than an awk script
- Possibly faster than writing data to disk

Problems:

- Need to write code
- Probably still really slow
- Requires at least some hardware

What about objects < recordsize?

OpenZFS

```
[root@coke /zones/testfs]# dd if=/dev/urandom of=/zones/testfs/small_uncompressed_file bs=4096 count=1
1+0 records in
1+0 records out
4096 bytes transferred in 0.000138 secs (28.4MB/sec)
[root@coke /zones/testfs]# zdb -vvvvvO zones/testfs small_uncompressed_file
obj=5 dataset=zones/testfs path=/small_uncompressed_file type=19 bonustype=44

    Object  lvl   iblk   dblk  dsize  dnsize  lsize   %full  type
         5    1   128K     4K     4K     512     4K  100.00  ZFS plain file (K=inherit) (Z=inherit)
                                          168   bonus  System attributes
...
Indirect blocks:
               0 L0 0:6e057d000:1000 1000L/1000P F=1 B=405847/405847

              segment [0000000000000000, 0000000000001000) size    4K
```