ZFS Allocation Performance

Matt Ahrens
mahrens@delphix.com

Overview

- Write? Must allocate place to write it.
- How does ZFS allocate space on disk?
- How *should* ZFS allocate space for best performance?
  - To combat non-homogenous device perf
  - To minimize metadata writes

Why is that so hard?

- ZFS is Copy-on-Write: all writes allocate
- Variable block size
- 512 Bytes to 16MB
- Compression
  - every multiple of 512 is possible
- 32,768 allocation (and free!) sizes
- (default max 128KB on 4K device => 32 sizes)

How does ZFS allocate space on disk? (open ctx)

- Write syscall (or via NFS or iSCSI)
  - record dirty data in memory
- Periodically write all dirty data to disk
  - ~1-10 seconds
  - spa_sync() aka TXG flush

How does ZFS allocate space on disk? (vdev)

- in spa_sync():
- Select: 1. vdev 2. metaslab 3. offset
- Select vdev
  - Assume vdevs have independent perf
  - Round robin among vdevs
  - Allocate same amount (512KB) to each
    - exception: less free space -> less allocation

How does ZFS allocate space on disk? (metaslab)

- Select metaslab in vdev
  - Assume sequential writes are better
    - Try to stick with same metaslab(s)
  - Start with metaslab with most fragmentation-weighted free space
  - Stick with it until it's >70% frag
    - (average free chunk size <32KB)

Calculate % frag & frag-weighted free space

- Spacemap has on-disk free chunk size histogram

FREE CHUNK SIZE	NUM CHUNKS	FREE SPACE	FRAG PCT	FWFS	NUM_CHUNKS_HISTOGRAM
512B	1226	1MB	100%	0MB	****
1KB	2347	2MB	100%	0MB	*******
<=2KB	4195	8MB	98%	0MB	**********
<=4KB	7507	29MB	95%	1MB	**********
<=8KB	10429	81MB	90%	8MB	**********
<=16KB	13454	210MB	80%	42MB	**********
<=32KB	11910	372MB	70%	111MB	**********
<=64KB	7426	475MB	60%	190MB	**********
<=128KB	1570	200MB	50%	100MB	*****
<=256KB	50	12MB	40%	7MB	*
		1390MB		**459MB**	

FWFS = CHUNK_SIZE * NUM_CHUNKS * (1-PCT)

frag% = 1 - FWFS / TOTAL_FREE = 66%

How does ZFS allocate space on disk? (metaslab?)

- Why 70% frag?
- What if all metaslabs >70% frag?

New metaslab selection

- Start with metaslab with most, largest, free chunks
- Work on it until another metaslab has a free chunk 4x larger than our largest
- (by George Wilson - needs to be upstreamed)

metaslab=106 FREE=3.08G FRAG=62%

```
FREE CHUNK    NUM
     SIZE CHUNKS NUM_CHUNKS_HISTOGRAM
========== ====== ========================================
      512B   1322 ****
       1KB   2396 *******
     <=2KB   4483 ************
     <=4KB   9096 ************************
     <=8KB  10935 *****************************
    <=16KB  14561 ****************************************
    <=32KB  15655 ****************************************
    <=64KB  13874 *************************************
   <=128KB   4166 ***********
   <=256KB    521 **
   <=512KB     63 *
  <=1024KB      2 *
```

metaslab=108 FREE=5.59G FRAG=60%

```
FREE CHUNK    NUM
     SIZE CHUNKS NUM_CHUNKS_HISTOGRAM
========== ====== ========================================
      512B   1496 ***
       1KB   2903 *****
     <=2KB   5977 **********
     <=4KB  11393 ******************
     <=8KB  14993 ************************
    <=16KB  21416 **********************************
    <=32KB  26419 ****************************************
    <=64KB  25383 ****************************************
   <=128KB   9183 **************
   <=256KB   1078 **
   <=512KB    116 *
  <=1024KB      1 *
```

metaslab=109 FREE=2.68G FRAG=62%

```
FREE CHUNK    NUM
     SIZE CHUNKS NUM_CHUNKS_HISTOGRAM
========== ====== ========================================
      512B   1427 *****
       1KB   2485 *******
     <=2KB   4624 *************
     <=4KB   8582 ************************
     <=8KB  11100 *******************************
    <=16KB  14264 ****************************************
    <=32KB  14203 ****************************************
    <=64KB  11845 *********************************
   <=128KB   3656 ***********
   <=256KB    365 **
   <=512KB     31 *
  <=1024KB      0
```

metaslab=107 FREE=2.39G FRAG=63%

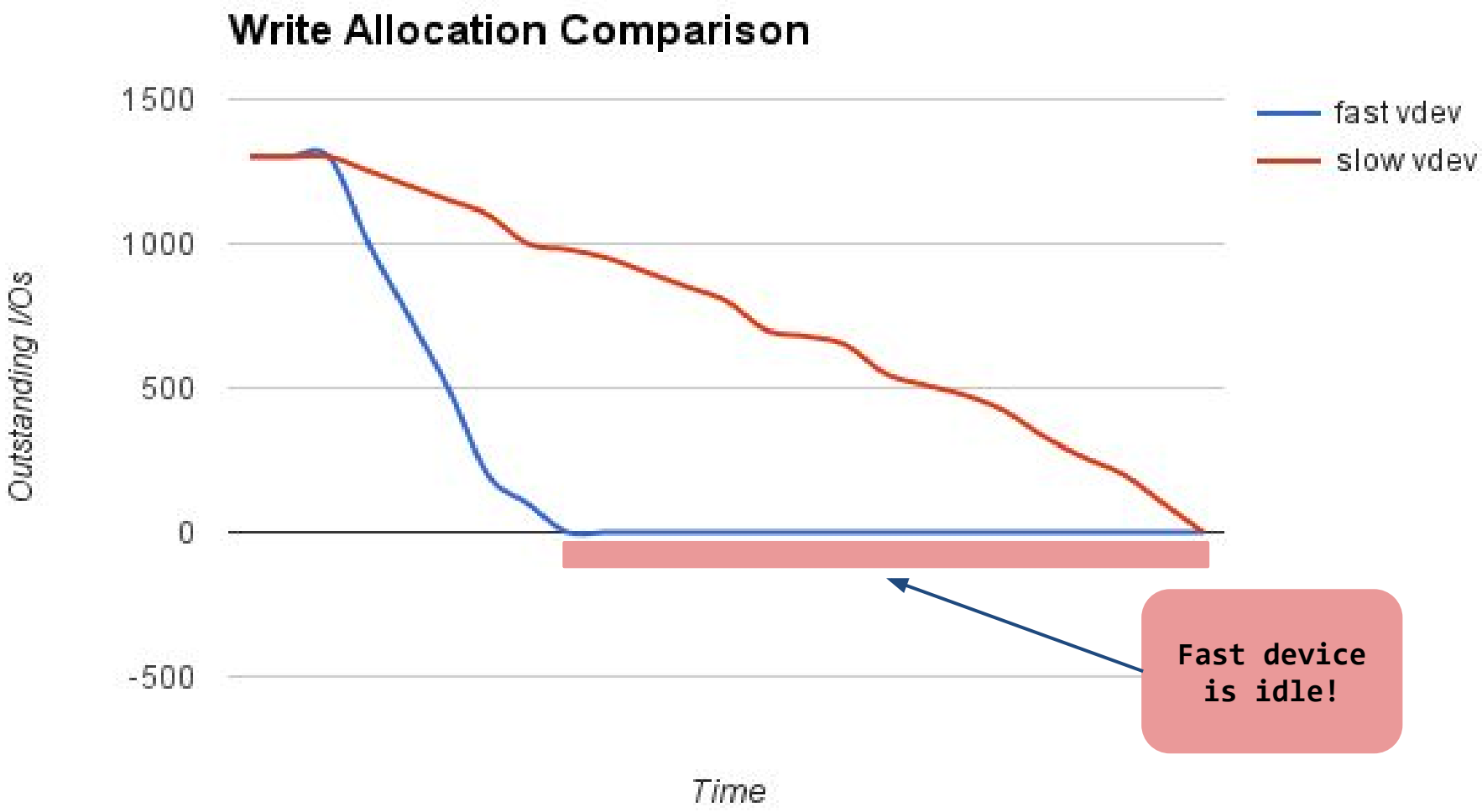
```
FREE CHUNK    NUM
     SIZE CHUNKS NUM_CHUNKS_HISTOGRAM
========== ====== ========================================
      512B   1380 ****
       1KB   2451 ********
     <=2KB   4497 *************
     <=4KB   8300 ************************
     <=8KB  10442 ******************************
    <=16KB  13846 ****************************************
    <=32KB  13499 ****************************************
    <=64KB  10499 ******************************
   <=128KB   2809 *********
   <=256KB    343 *
   <=512KB     25 *
  <=1024KB      0
```

Problem:
Devices in pool have differing (non-homogenous) write performance

Why do devices have different write perf?

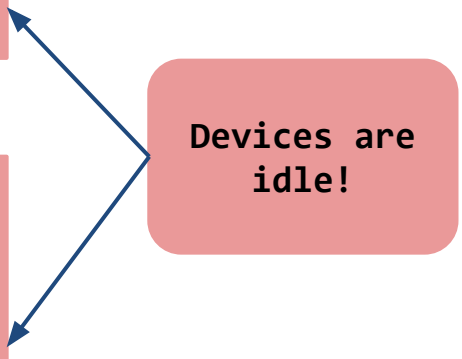
NAME	SIZE	ALLOC	FREE	EXPANDSZ	FRAG	CAP	DEDUP	HEALTH	ALTROOT
dcenter	12.2T	8.73T	3.46T	-	62%	71%	1.00x	ONLINE	-
mirror-0	556G	458G	98.3G	-	66%	82%			
mirror-1	556G	457G	99.4G	-	67%	82%			
mirror-2	556G	455G	101G	-	66%	81%			
mirror-3	556G	456G	100G	-	66%	81%			
mirror-4	556G	455G	101G	-	66%	81%			
mirror-5	556G	455G	101G	-	66%	81%			
mirror-6	1016G	690G	326G	-	59%	67%			
mirror-7	1016G	642G	374G	-	58%	63%			
mirror-8	1016G	674G	342G	-	59%	66%			
mirror-9	1016G	672G	344G	-	59%	66%			
mirror-10	1016G	670G	346G	-	59%	65%			
mirror-11	1016G	716G	300G	-	64%	70%			
mirror-12	1016G	713G	303G	-	64%	70%			
mirror-13	1016G	714G	302G	-	64%	70%			
mirror-14	1016G	712G	304G	-	63%	70%			

I/O completion rate



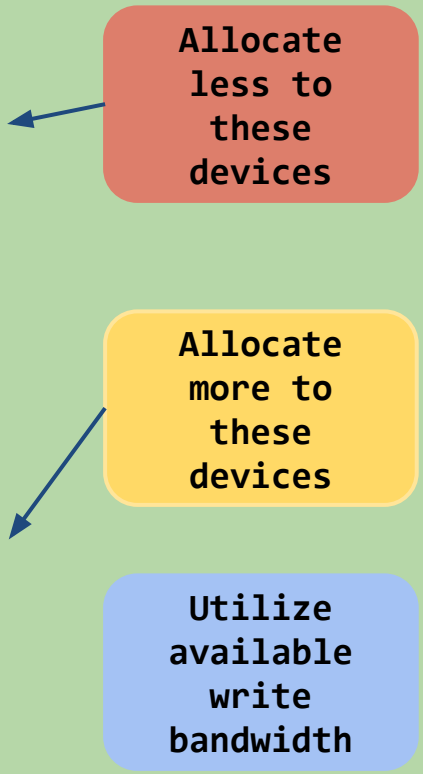
Impact of different write performance

```
                capacity  bandwidth
pool                 free      write
-----------         -----      -----
dcenter             2.85T      6.05M
  mirror-0          53.3G       224K
  mirror-1          56.3G          0
  mirror-2          57.1G          0
  mirror-3          57.6G          0
  mirror-4          58.3G      2.65M
  mirror-5          58.6G      3.18M
  mirror-6           296G          0
  mirror-7           326G          0
  mirror-8           346G          0
  mirror-9           346G          0
  mirror-10          352G          0
  mirror-11          218G          0
  mirror-12          230G          0
  mirror-13          228G          0
  mirror-14          232G          0
```



Proposed Solution

NAME	SIZE	ALLOC	FREE	EXPANDSZ	FRAG	CAP	DEDUP	HEALTH	ALTROOT
dcenter	12.2T	8.73T	3.46T	-	62%	71%	1.00x	ONLINE	-
mirror-0	556G	458G	98.3G	-	66%	82%			
mirror-1	556G	457G	99.4G	-	67%	82%			
mirror-2	556G	455G	101G	-	66%	81%			
mirror-3	556G	456G	100G	-	66%	81%			
mirror-4	556G	455G	101G	-	66%	81%			
mirror-5	556G	455G	101G	-	66%	81%			
mirror-6	1016G	690G	326G	-	59%	67%			
mirror-7	1016G	642G	374G	-	58%	63%			
mirror-8	1016G	674G	342G	-	59%	66%			
mirror-9	1016G	672G	344G	-	59%	66%			
mirror-10	1016G	670G	346G	-	59%	65%			
mirror-11	1016G	716G	300G	-	64%	70%			
mirror-12	1016G	713G	303G	-	64%	70%			
mirror-13	1016G	714G	302G	-	64%	70%			
mirror-14	1016G	712G	304G	-	63%	70%			



Solution: Allocation Throttle

- Allocate incrementally throughout spa_sync()
- Keep 100 allocations outstanding on each vdev
  - When write completes, allocate another write on same vdev
- End results:
  - Allocates more space from faster devices
  - Keeps all devices working
  - Up to 2x better write throughput!
- https://github.com/openzfs/openzfs/pull/130
- (by George Wilson)

On-disk structures for tracking space allocation

On-disk structures

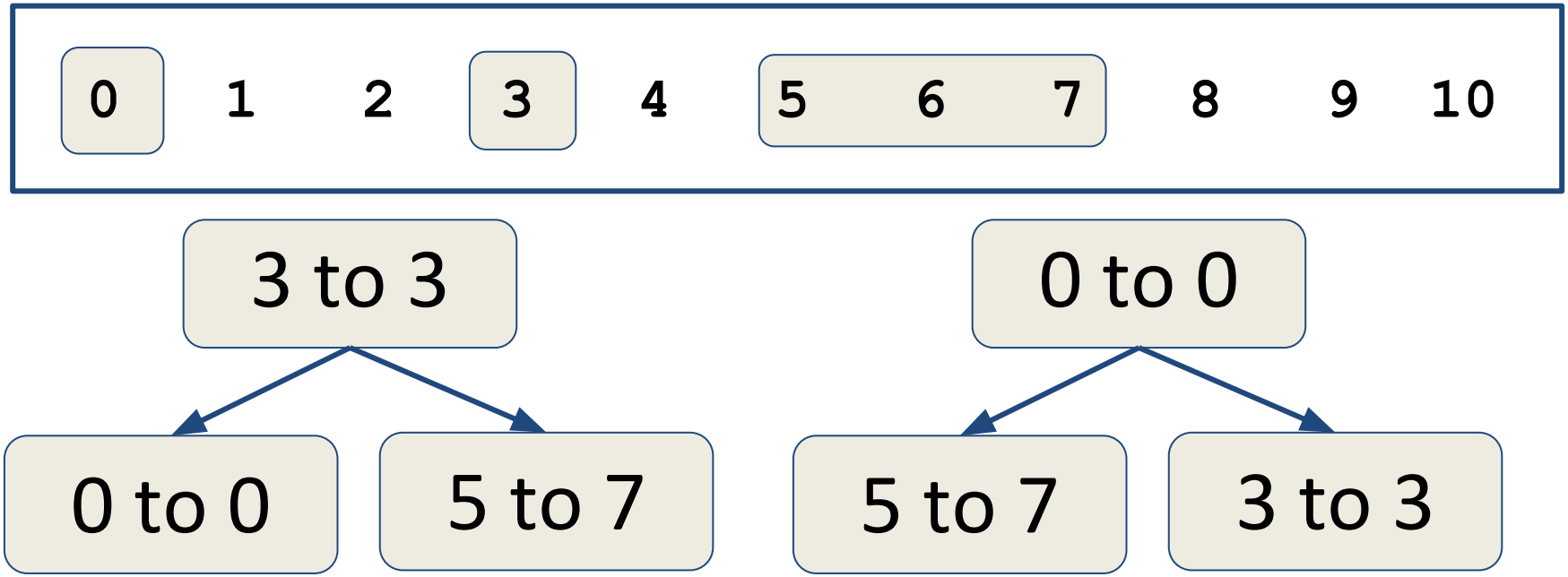
- Each vdev divided into ~200 metaslabs
  - Each metaslab tracks free space in on-disk spacemap
- Spacemap is on-disk log of allocations & frees

Alloc 4 to 7	Alloc 1 to 2	Free 5 to 7	Alloc 8 to 10	unused

- Each spacemap stored in object in MOS
- Periodically “condense” each spacemap
  - write out as all ALLOC records

Recording allocations on-disk

- To allocate, must know exactly what is allocatable
- Must load spacemap from disk into range tree
- range tree is in-memory structure
  - balanced binary tree of free segments, sorted by offset
  - 2nd tree sorted by length



Writing spacemaps

- Each spa_sync() (i.e. TXG) each metaslab tracks
  - allocations (in a range_tree)
  - frees (in a range_tree)
- At end of TXG
  - append alloc & free range_trees to space_map
  - clear range_trees
- Dynamic behavior
  - Usually freeing blocks in most metaslabs
  - Usually appending to most metaslabs
  - ~600 i/os per vdev per txg! (1 data + 2 indirect)

Problem: Reading spacemap is slow

- Because **`space_map_blksz`**=4KB
- Because we append to every spacemap every txg
  - **`space_map_blksz`**=128KB => write 25MB/vdev/txg
  - 10 vdevs => at least 6% bandwidth overhead
- Because frees are scattered across all metaslabs
- 1TB vdev => spacemap up to 5MB (1,280 blocks)

Problem: loaded spacemap uses lots memory

- Metaslab covers huge range (50GB on 10TB vdev)
- range tree uses up to 320MB RAM (on 10TB vdev)
  - 5 million segments per metaslab

Solutions?

- Different # metaslabs per vdev?
  - 200.
  - **more => less memory; faster loading**
  - less => less i/o (append to fewer spacemaps)
  - Fixed metaslab size (e.g. metaslab=1GB?)
- Different space map blocksize?
  - 4KB.
  - **bigger => faster loading**
  - smaller => less bandwidth

- Problem: Append to most spacemaps most TXGs
  - ~600 i/o's per vdev per txg! (1 data + 2 indirect)

Solution.

- Don’t flush every modified metaslab every txg
- Keep changes in memory until flushed
  - range tree of unflushed allocs
  - range tree of unflushed frees
  - non-overlapping! limits size and don’t need order info
  - to load metaslab
    - 1. read spacemap from disk
    - 2. apply unflushed allocs & frees
- Flush ~1 metaslab per TXG
  - It will have lots of changes (many TXG’s worth)
  - Efficiency: each TXG append lots of data to few spacemaps
- What if we crash?
  - Can’t lose unflushed allocs/frees

Solution.

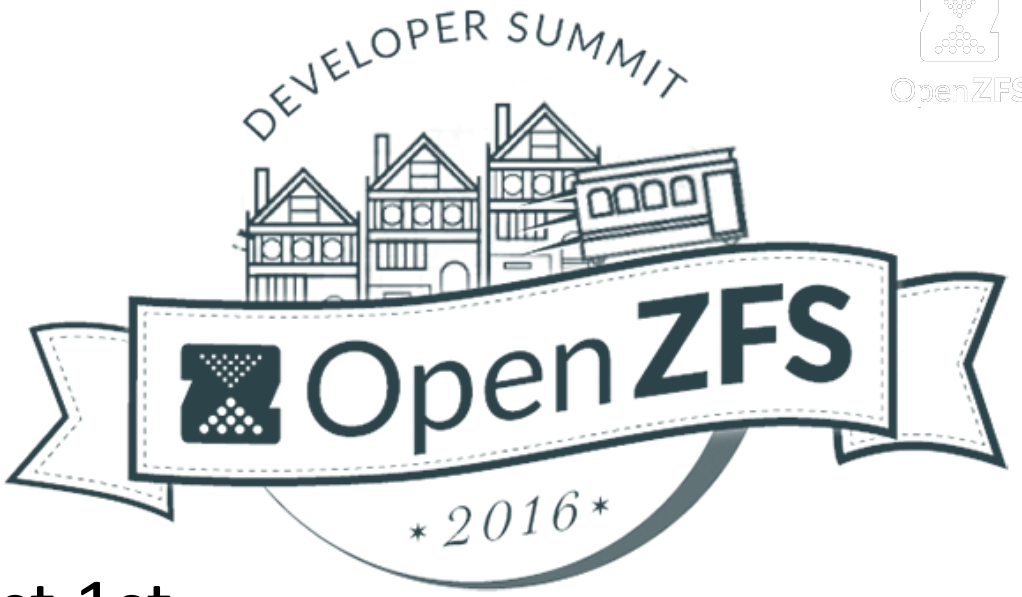
- What if we crash?
  - Can't lose unflushed allocs/frees
- Write all metaslabs' changes in one spacemap
  - (per vdev, per txg)
- After unclean shutdown, read vdev's spacemap
  - reconstruct each metaslab's unflushed allocs/frees

Solution.

- How many metaslabs to flush each TXG?
- At least 1
- Limit reconstruction time
  - Limit size of vdev's spacemap
- Limit memory
  - Limit # segments in unflushed allocs/frees
- Flush more metaslabs until under limits

Results.

- Increase # metaslabs to ~1/GB
  - loaded metaslabs use 1/50th RAM
  - loading metaslab read 1/50th as many blocks
    - (w/10 TB vdev)
- Increase space_map_blksz to 128KB
  - loading metaslab reads 1/32nd as many blocks
- Append to 2 spacemaps per txg
  - 1/100th as many i/os
  - 1/10th as much bandwidth



- September 26-27th
- San Francisco
- Talks; Hackathon
- http://open-zfs.org/
- Submit talks by August 1st
- Registration now open!
- Sponsorship opportunities
- Thanks to our early-bird Platinum sponsors:

DELPHIX

intel®
Software