ZFS DIRECTORY SCALING

Look up and create files fast
WHO AM I?

• FreeBSD committer since 2018 (0mp@)
• FreeBSD Core Team member
• Most days I work on cool stuff with folks @ Klara Inc.

• In general, I poke around things like:
  • Documentation
  • Ports
  • rc(8) scripts (script all the things!)
  • Tracing
  • ZFS
OUTLINE

ZFS
Directories
Scaling
Numbers
ZFS ...
ZFS
What is it?

• Copy-on-write file system

History
• Developed at Sun in 2001
• Imported into FreeBSD in 2008
• As of 2023, FreeBSD uses OpenZFS
  • Works on both Linux and FreeBSD
ZFS
What does it do?

• 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
ZFS

What else does it do?

• Snapshots
• Efficient remote replication
• Compression
• Encryption
• Deduplication
• RAIDZ
• Quotas
• Boot environments
• …
ZFS

How does it work?

EDITOR=vi
PAGER=less
ZFS

Write New Blocks

```
EDITOR=vi
EDITOR=ed
PAGER=less
```
ZFS
Update Indirect Blocks
ZFS

Checkpoint!

uber  uber  uber  uber

... .profile ...

... EDITOR=ed ...

... PAGER=less ...

... ...

... ...

... ...

... ...

... ...
ZFS

What is a file?

- What is a file?
  - File is an object
- What is an object?
  - Group of blocks
  - Organized by a dnode
- Almost everything is an object
  - Files, directories, datasets, …
- Let’s take a closer look!
### ZFS

**Inspecting a File with zdb(8)**

```sh
# dd bs=512M count=1 if=/dev/random of=/tank/ds/bigfile && zdb -ddddd tank/ds "$\text{(stat -f %i /tank/ds/bigfile)}"
```

Dataset tank/ds [ZPL], ID 67, cr_txg 7, 512M, 9 objects, rootbp DVA[0]=<0:1000c800:200>, DVA[1]=<0:1800c600:200>, [L0 DMU objset]

**fletcher4** | **lz4** unencrypted LE contiguous unique double size=1000L/200P birth=78L/78P fill=9 cksum=00000000c7863135c:

<table>
<thead>
<tr>
<th>Object</th>
<th>lvl</th>
<th>iblk</th>
<th>dblk</th>
<th>dsize</th>
<th>dnsize</th>
<th>lsize</th>
<th>%full</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>3</td>
<td>128K</td>
<td>128K</td>
<td>512M</td>
<td>512</td>
<td>512M</td>
<td>100.00</td>
<td>ZFS plain file (K=inherit) (Z=inherit=lz4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>168</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bonus System attributes</td>
</tr>
</tbody>
</table>

... path /bigfile

... size 536870912

[parent 34]

... Indirect blocks:

|   | L2   | 0:20025a00:400 0:28025a00:400 20000L/400P F=4096 B=78/78 cksum=0000000042cf061fb:
|   | L1   | 0:4fcea00:a400 0:8027e00:a400 20000L/a400P F=1024 B=72/72 cksum=000000db03b46137:
| 0 | L0   | 0:5a00bc00:20000 20000L/20000P F=1 B=71/71 cksum=00003f43129f47:
|   | lfe0000 | 0:6202aa00:20000 20000L/20000P F=1 B=78/78 cksum=00003f435b2a0425:

 segment [0000000000000000, 0000000020000000] size 512M
ZFS

zdb(8) Example: /usr/share/misc/flowers (1/2)

```bash
# ls -lah /usr/share/misc/flowers
-r--r--r--  1 root wheel  1.4K Sep  1 08:21 /usr/share/misc/flowers

# zfs set recordsize=512B tank/ds

# cat /usr/share/misc/flowers > /tank/ds/flowers

# zdb -ddddd tank/ds $(stat -f %i /tank/ds/flowers)

segment [0000000000000000, 0000000000000000] size 1.50K

# zdb -ddddd tank/ds $(stat -f %i /tank/ds/flowers) | awk '/  L0 /{print $3; exit}'
0:28032c00:200
```

[Output]

```
[...]
0 L1 0:3800f800:400 20000L/400P F=3 B=1007/1007 cksum=00000008b56b724f9:
[...]
0 L0 0:28032c00:200 200L/200P F=1 B=1007/1007 cksum=0000002d1f3a62b8:
[...]
200 L0 0:28032e00:200 200L/200P F=1 B=1007/1007 cksum=0000002e5c3c058:
[...]
400 L0 0:28033000:200 200L/200P F=1 B=1007/1007 cksum=00000020e7272cab:
[...]
```
ZFS

zdb(8) Example: /usr/share/misc/flowers (2/2)

# zdb -R tank 0:28032c00:200 | tail -n 3
0001d0: 0a486f6e65797375 636b6c653a426f6e .Honeysuckle:Bons
ds of love..Ivy:
0001e0: 6473206f66206c6f 76652e0a4976793a d of love..Ivy:
0001f0: 467269656e647368 69732c2066696465 Friendship, fide

# head -c 512 /usr/share/misc/flowers | tail -n 2
Honeysuckle:Bonds of love.
Ivy:Friendship, fide
... Directory ...
DIRECTORIES

Definition

• From dir(5):
  • Directories provide a convenient hierarchical method of grouping files while obscuring the underlying details of the storage medium.
  • It consists of records (directory entries) each of which contains information about a file and a pointer to the file itself.
Hierarchical File System Operations

- From *The Design and Implementation of The FreeBSD Operating System, Table 9.1*:
  - pathname searching (e.g., lookup)
  - name creation (e.g., create)
  - name change/deletion (e.g., rename)
  - attribute manipulation (e.g., getattr)
  - object interpretation (e.g., open)
  - process control (e.g., ioctl)
  - object management (e.g., lock)
ZFS DIRECTORIES

What Is a Directory in ZFS?

• An object
  • Just like a file
• Contains a tree of ZAP blocks
  • Instead of data blocks (as file objects do)
ZFS DIRECTORIES

ZFS Attribute Processor (ZAP)

- Key-value store
- Keys can be strings and values can be, e.g., strings, numbers, or arrays of numbers
- Primarily used for directories
- Extensible hash table
- Scales nicely up to hundreds of millions of entries in a single directory

- Also called a FatZAP
ZFS DIRECTORIES

MicroZAP

- Single directory block
- Stores the mapping of file names to objects directly in the directory block
- Max 2047 entries
  - Keys: 50 bytes (due to max MicroZAP size of 128 KiB)
  - Values limited to integers.
- Automatically promoted to FatZAP
  - A FatZAP never shrinks back into a MicroZAP
    - OpenZFS#8420
    - OpenZFS#14088
... Scaling
ZFS TUNING

Different Ways of Tuning ZFS

• zfs set atime=off tank
• zfs create -o recordsize=16m tank/bigfiles
• sysctl vfs.zfs.zap_micro_max_size="1048576"
• echo 'vfs.zfs.zio.taskq_batch_pct="20"' >> /boot/loader.conf
ZFS TUNING

Popular Tunables

• ZFS properties:
  • atime (e.g., off)
    • Usually not necessary
  • recordsize (e.g., 16m)
    • Good when aligns with the workload
  • primarycache (e.g., metadata)
    • Beneficial, if applications do their own caching
  • compression (e.g., lz4)
    • Different algorithms offer different tradeoffs
ZFS DIRECTORY TUNING

Maximum size of a MicroZAP

- Control the maximum size of a MicroZAP with `vfs.zfs.zap_mircro_max_size`
  - Changes the moment of the switch to a FatZAP
  - Default: 128 KiB (2047 files)
    - 1 MiB (16383 files)
  - Introduced in OpenZFS#14292 (2023-01-10)

- Advantages and disadvantages of a larger MicroZAP:
  - Directory object has less indirect blocks
  - More bytes to process during write operations
ZFS DIRECTORY TUNING

Size of Indirect Blocks

• Control the size of indirect blocks with `vfs.zfs.default_ibs`
  • Default: 17 ($2^{17}$, 128 KiB)
  • Available on FreeBSD for a long time (since 2023-01-11 also on Linux, OpenZFS#14293)

• Advantages and disadvantages of a smaller indirect block size:
  • Less bytes per block to process when reading or writing
  • More blocks to traverse
Numbers
NUMBERS

Overview

- Two benchmarks
  - Lookup (read-only)
  - Create (read & write)
- Benchmarking harness
  - hyperfine
- Tunables
  - `vfs.zfs.zap_micro_max_size`
  - `vfs.zfs.default_ibs`
BENCHMARK 1

Lookup
BENCHMARK 1: LOOKUP

Overview

• Measuring:
  • Time to list files of all subdirectories (with ftw(3))

• Parameters:
  • Files per subdirectory
  • Maximum MicroZAP size:
    • 131072 (128 KiB, default)
    • 1048576 (1 MiB)
  • Indirect block size:
    • 15 ($2^{15}$, 32 KiB)
    • 17 ($2^{17}$, 128 KiB, default)
BENCHMARK 1: LOOKUP

16000 Files (Exceeds 128-KiB MicroZAP & Fits 1-MiB MicroZAP)

<table>
<thead>
<tr>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
</tr>
</thead>
<tbody>
<tr>
<td>'lookup 16000 1048576 15' ran</td>
<td>1.00 ± 0.01 times faster than 'lookup 16000 1048576 17'</td>
<td>1.06 ± 0.01 times faster than 'lookup 16000 131072 15'</td>
</tr>
<tr>
<td>1.06 ± 0.01 times faster than 'lookup 16000 131072 17'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Larger MicroZAPs increase performance
- Indirect block size does not matter that much
BENCHMARK 1: LOOKUP

64000 Files (Exceeds 128-KiB & 1-MiB MicroZAPs)

<table>
<thead>
<tr>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
</tr>
</thead>
<tbody>
<tr>
<td>'lookup 64000 131072 15' ran</td>
<td>1.00 ± 0.01 times faster than 'lookup 64000 131072 17'</td>
<td></td>
</tr>
</tbody>
</table>

- Once FatZAPs kick in, the benefits of MicroZAPs disappear
## BENCHMARK 1: LOOKUP

16500 Files (Exceeds 128-KiB & 1-MiB MicroZAPs) & primarycache=none

<table>
<thead>
<tr>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
</tr>
</thead>
<tbody>
<tr>
<td>'lookup 16500 1048576 15' ran</td>
<td>1.01 ± 0.00 times faster than 'lookup 16500 131072 15'</td>
<td>1.42 ± 0.00 times faster than 'lookup 16500 1048576 17'</td>
</tr>
</tbody>
</table>

- When blocks read from disk...
  - 👁 MicroZAP size does not matter
  - 👉 indirect block size matters
BENCHMARK 2
Create
NUMBERS
Benchmark 2: Create

• Measuring:
  • Time to create files in a directory (with open(2))

• Parameters:
  • Files per subdirectory
  • Indirect block size:
    • 15 (2^{15}, 32 KiB)
    • 17 (2^{17}, 128 KiB, default)
  • Maximum MicroZAP size:
    • 131072 (128 KiB, default)
    • 1048576 (1 MiB)
**BENCHMARK 2: CREATE**

2047 Files (Fits 128-KiB & 1-MiB MicroZAPs)

<table>
<thead>
<tr>
<th></th>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
</tr>
</thead>
<tbody>
<tr>
<td>'create 2047 131072 17' ran</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00 ± 0.04 times faster than 'create 2047 1048576 15'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00 ± 0.03 times faster than 'create 2047 1048576 17'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.02 ± 0.05 times faster than 'create 2047 131072 15'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- No observable difference for small directories
## BENCHMARK 2: CREATE

16838 Files (Exceeds 128-KiB MicroZAP & Fits 1-MiB MicroZAP)

<table>
<thead>
<tr>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
</tr>
</thead>
<tbody>
<tr>
<td>'create 2047 131072 17' ran</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.20 ± 0.21 times faster than 'create 16838 131072 15'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.23 ± 0.20 times faster than 'create 16838 131072 17'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.98 ± 0.41 times faster than 'create 16838 1048576 17'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.03 ± 0.42 times faster than 'create 16838 1048576 15'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Larger MicroZAPs are more expensive when writing
- 128-KiB MicroZAPs are x2 faster than 1-MiB MicroZAPs
BENCHMARK 2: CREATE
64000 Files (Exceeds 128-KiB & 1-MiB MicroZAPs)

<table>
<thead>
<tr>
<th>fpd</th>
<th>microzap</th>
<th>ibs</th>
<th>'create 2047 131072 17' ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.82 ± 0.76 times faster than 'create 64000 131072 15'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.88 ± 0.76 times faster than 'create 64000 131072 17'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.53 ± 0.98 times faster than 'create 64000 1048576 15'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.57 ± 0.97 times faster than 'create 64000 1048576 17'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Larger MicroZAPs are more expensive when writing
- Smaller performance gap between 128-KiB MicroZAPs and 1-MiB MicroZAPs
Summary
SUMMARY

• Two important tunables for directory scaling:
  • Maximum MicroZAP size: `vfs.zfs.zap_micro_max_size`
  • Indirect block size: `vfs.zfs.default_ibs`

• Tuning takeaways:
  • **Reads** are faster with larger MicroZAPs and smaller indirect blocks
  • **Writes** may slow down when using larger MicroZAPs
  • Tuning depends on the system. Measure the system before tuning it.
Questions?
Thanks!