OpenZFS

Shared Log Pool
Introduction

- Paul Dagnelie (he/him)
- Delphix
- 10th Dev Summit!
- Past Talks
  - Redacted Send/Receive
  - Metaslab Performance
Background: The ZIL

- Transaction Groups (TXGs)
  - Efficient, but infrequent
- Synchronous writes
  - Low latency
  - High frequency in some workloads
- Anti-synergy
Background: The ZIL

- **Transaction Groups (TXGs)**
  - Efficient, but infrequent

- **Synchronous writes**
  - Low latency
  - High frequency in some workloads

- **Solution: ZFS Intent Log (ZIL)**
  - Per-dataset
  - Chain of blocks
  - Not long-term storage
  - Only read on recovery
Background: ZIL Alloc/Write

- Sync write comes in
- Allocate block
- Can’t wait for allocation to sync
- Solution: Chain blocks together
Sync write comes in
Allocate block
Can’t wait for allocation to sync
Solution: Chain blocks together

When TXG syncs, advance head
  Data is in order, no losses
Background: ZIL Claim/Replay

- System crash/power event
- Need to find all ZIL blocks before we start allocating
- ZIL Claim:
  - For each dataset:
    - Iterate over ZIL chain:
      - Mark each block as allocated
- ZIL Replay:
  - On mount, iterate over ZIL chain:
    - Apply each record in each block
Background: SLOGs

- Where do ZIL writes go?
- Embedded SLOG
  - Easy Administration
  - Complex performance
- SLOG devices
  - Harder administration
  - Better Performance
  - Expensive
**Context: Multiple Pools**

- Moving data
  - FibreChannel/SAN
  - Shift pools from server to server
  - Load balancing
Context: Multiple Pools

- Moving data
  - FibreChannel/SAN
  - Shift pools from server to server
  - Load balancing
Context: Multiple Pools

- Fault isolation
  - Caching data
  - Don’t want to lose other data if one disk dies
  - Some software already handles this
Context: Multiple Pools

- Moving data
  - FibreChannel/SAN
  - Poor networking
- Fault isolation
- Varying redundancy/performance requirements
The Problem

- Per-pool SLOG devices
- Capacity planning?
- Load balancing?
- Adding or removing pools?
The Problem

- Per-pool SLOG devices
- Capacity planning?
- Load balancing?
- Adding or removing pools?

Insight: This is what zpools were built to solve!
The Solution

- Pool SLOG devices
- Multiple clients, one provider
- Performance near-parity
- Simple administration
Shared Log Pool

- Normal zpool except:
  - No filesystems
  - Config flag
  - New data structure: the Chain Map
    ■ Details later!

```bash
$ zpool create -L shared_log sdb sdc sdd
$ zpool list -v -o name,size
NAME       SIZE
shared_log  240G
  sdb       80G
  sdc       80G
  sdd       80G
rpool      69.5G
  sda1      70.0G
```
Client Pool

- Normal zpool except:
  - No physical SLOG
  - Depends on shared log pool
  - ZIL blocks stored in shared log pool

- Create or import

```
$ zpool create -l shared_log client sde sdf sdg
$ zpool list -v -o name,size client
NAME          SIZE
client         6T
  sdb          2T
  sdc          2T
  sdd          2T
  shared log   -
    shared_log 240G
```
Client Pool

● Normal zpool except:
  ○ No physical SLOG
  ○ Depends on shared log pool
  ○ ZIL blocks stored in shared log pool

● Create or import

$ zpool import -m -l shared_log client
$ zpool list -v -o name,size client

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>6T</td>
</tr>
<tr>
<td>sdb</td>
<td>2T</td>
</tr>
<tr>
<td>sdc</td>
<td>2T</td>
</tr>
<tr>
<td>sdd</td>
<td>2T</td>
</tr>
<tr>
<td>shared log</td>
<td>-</td>
</tr>
<tr>
<td>shared_log</td>
<td>240G</td>
</tr>
</tbody>
</table>
But Wait!

- ZIL header points to shared log pool
- Cross pool blkptrs?
- ZIL Claim
- Need a better way
The Chain Map

- Map from objset to ZIL chain
  - In-memory representation
  - On-disk format
ZIL Use: New ZIL

- New filesystem created in client
ZIL Use: New ZIL

- New filesystem created in client
- Create chain map entry
- Allocate first block
ZIL Use: New ZIL Block

- Sync write comes in
- Allocate in shared log pool
ZIL Use: New ZIL Block

- Sync write comes in
- Allocate in shared log pool
- Append to chain
  - No new logic
ZIL Use: TXG sync

- Client pool syncing TXG
- Need to move chain head forwards
ZIL Use: TXG sync

- Client pool syncing TXG
- Need to move chain head forwards
- After TXG syncs, update chain map
  - spa_zil_map
- Free old ZIL Blocks
ZIL Use: Removing ZIL

- Deleting filesystem
- Need to clean up chain
ZIL Use: Removing ZIL

- Deleting filesystem
- Need to clean up chain
- After deletion syncs in client, update chain map
  - spa_zil_deletes
- Free all blocks in chain
ZIL Claim

- Crash/power outage
- On shared log pool import
  - Iterate over each client in chain map
    - Iterate over each filesystem
      - Mark each ZIL block as allocated
ZIL Replay

- Once client pool imports
  - For each filesystem, get chain from map
    - Replay all records in chain
Client Import Cleanup

- Deleting ZIL
- Crash before shared log pool syncs
- Leaked space?
Client Import Cleanup

- Deleting ZIL
- Crash before shared log pool syncs
- Leaked space?
- Backup solution:
  - On client import
    - Iterate over chain map
      - Any entries that don’t have a real filesystem, clean up
Deregistration

- Removing a client pool
- Need to clean up chain map entries
Deregistration

- Removing a client pool
- Need to clean up chain map entries
  - For each filesystem in client
    - Free each block in chain
- Remove dependency
Missed Deregistration?

- Accidents happen
- Pools are moved/destroyed
- GC as backup

```
$ zpool list -o name,guid client
NAME       GUID
client  7505453946292746732
$ zpool export client
$ zpool recycle -n shared_log
Cleaned up (dry run): [7505453946292746732]
```
Shared Log Deletion

- Forbidden if any clients currently using
- Deletes all chains, frees all blocks
- All clients need to discard logs

```
$ zpool list -o name
NAME
client1
client2
shared_log
$ zpool destroy client1
$ zpool destroy client2
$ zpool destroy shared_log

$ zpool list -o name
NAME
client1
client2
shared_log
$ zpool export client1
$ zpool destroy client2
$ zpool destroy shared_log
$ zpool import -m client1
```
Performance results

- For non-shared-log pools, no difference
- < 2% normally
- ~7% for workloads with many filesystems
  - Further improvements are possible
Caveats

- No reguiding
- No checkpoints
  - Meaningless for shared log
  - Doable for client, but not in MVP
Current Status

- **PR 14520**
- Reviews & comments welcome!
- Find me after the talk!
Questions?
Thank you!
Bonus Slides
High-Level Workflow

- Creation
- Registration
- ZIL creation
- ZIL use
- Unregistration
- Deletion
The Chain Map

- Map from objset to ZIL chain
  - In-memory representation
  - On-disk format
- Pass \texttt{-L} to \texttt{zpool create}
- Marked with key in pool config
- No new filesystems
- No receives
- No mounting
- Chain map created
  - Details later!
Pass `-l` to `zpool create/import`

- Key added to config marking dependance
- Metaslab log class becomes “virtual”
  - No mixing with regular SLOG
- ZILs point to blocks in shared log pool
Details: ZIL Creation

- ZIL creation proceeds mostly as normal
- New chain map entry
- Allocation in shared log pool
● ZIL updates proceed as normal
  ○ Allocations from shared log pool
● Every client TXG, chain map is updated
  ○ spa_zil_map
  ○ spa_zil_deletes
● Claim
● Replay
● Client import cleanup
Details: Unregistration

- Iterate over chain map entries
  - Free blocks in chain
  - Delete entry
- Remove from list of registered clients
- Remove marker in client
- GC as backup