Storage Pool Checkpoint

25 October 2017

Serapheim Dimitropoulos
Delphix

Timeline

- Started with Dan Kimmel at last year's hackathon
- Redesigned from scratch later that year
- Pushed in product last spring and soaking since then

Can only be upstreamed after Device Removal. Help us get there :-)

Motivation

Motivation

Upgrading a Delphix Engine

- Reboot to new OS version
- Run several upgrade scripts

Each upgrade script

- Manipulates ZFS datasets and their properties
- Has a respective rollback script in case it fails

Motivation

Problems with rollback scripts

- Tedious & time-consuming to write
- but most importantly, error-prone

The main problem with our rollback

- Upgrade scripts manipulate datasets (not files!)
- Thus, taking a regular snapshot and rolling back is not possible

Storage Pool Checkpoint

Storage Pool Checkpoint

A *"pool-wide"* snapshot!

- Remembers the entire state of the pool at a point in time
- Users can rewind back to it later or discard it

Like a snapshot

- creating a checkpoint is almost instantenuous
- its space comnsumption consists only of references to old data

Delphix Use Case

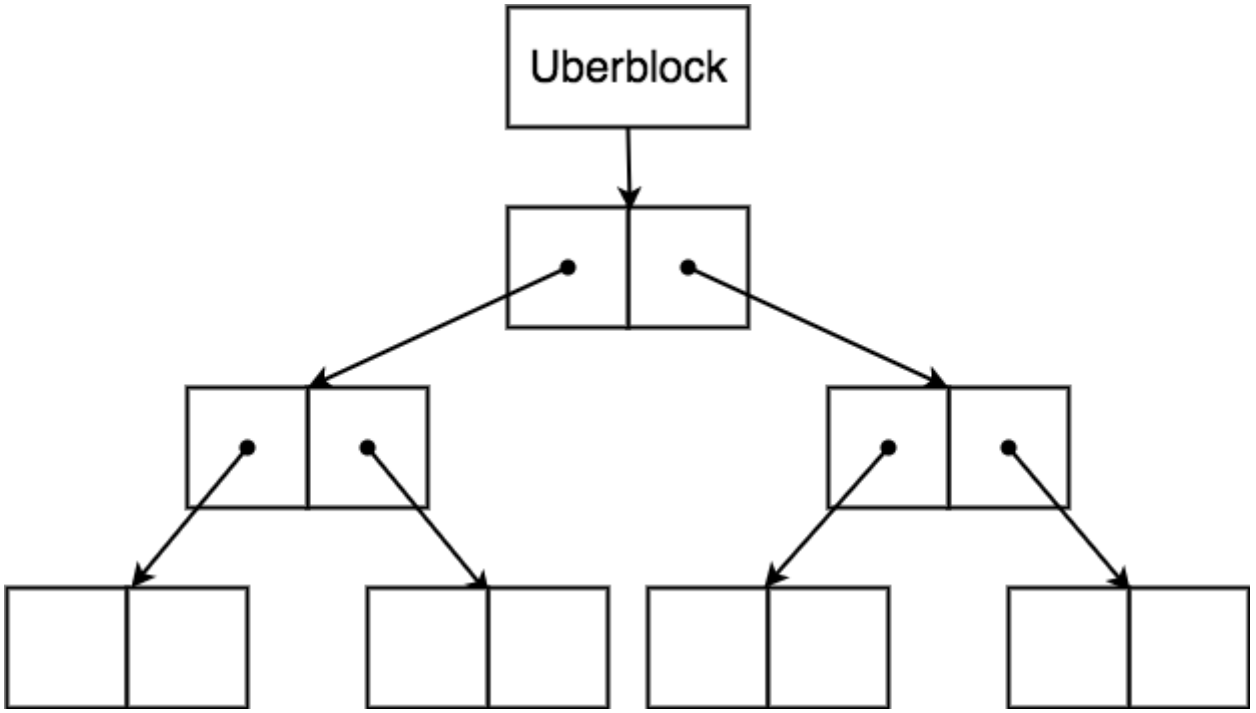
New workflow:

- Take a checkpoint
- Run upgrade scripts
- If a script fails rewind to checkpoint
- Else discard it when upgrade is done

No need for rollback scripts!

High-Level Internals

High-Level Internals



Uberblock - The block that references all the state of a pool

High-Level Internals

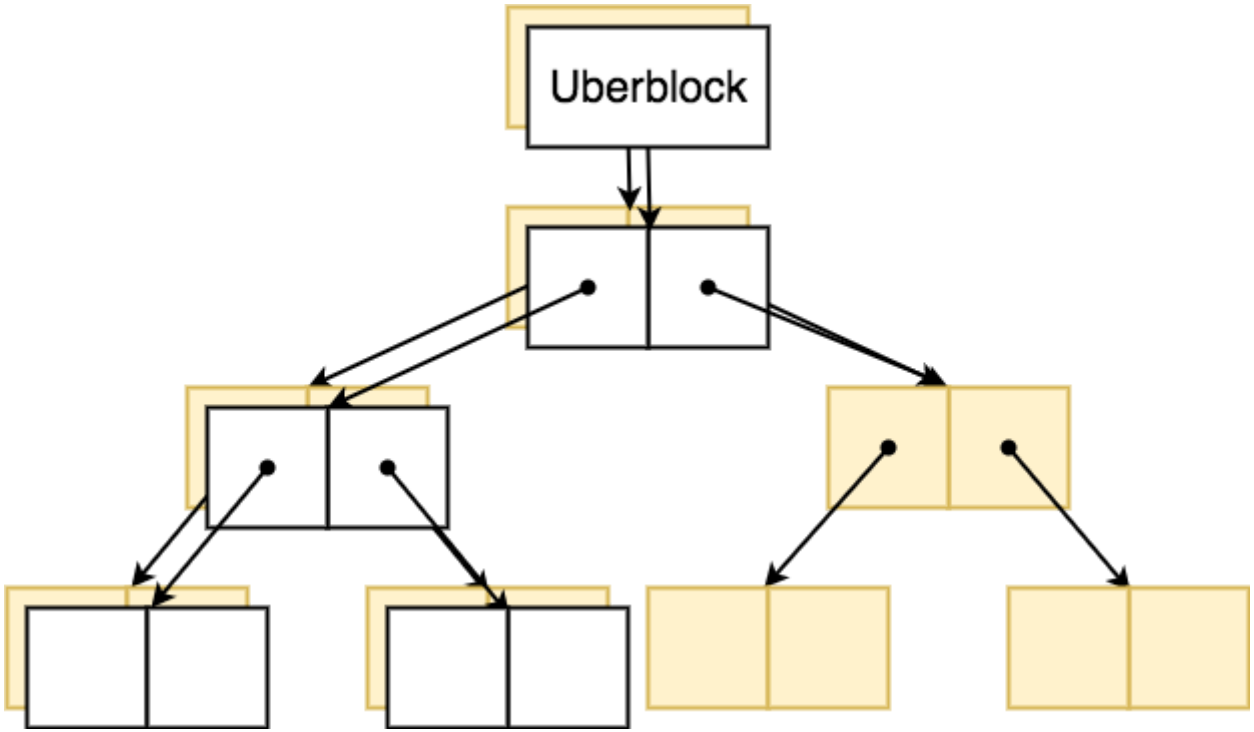
ZFS uses logical timestamps to represent time called **Transaction Groups** (aka **TXGs**)

In each TXG:

- We accumulate changes
- We write a new uberblock that references those changes

All blocks in ZFS have a **birth time** equal to the TXG that they were created

High-Level Internals



ZFS CoW Nature

New uberblock references new blocks plus blocks from older uberblocks

As time passes and we are writing to disk, blocks that are not referenced are reused

High-Level Internals

Checkpoint Pool
Save the current (at the time) uberblock

Rewind to Checkpoint
Place saved uberblock back as the current uberblock

Discard Checkpoint
Get rid of the saved uberblock

High-Level Internals

Problem

Blocks that are referenced by the checkpointed uberblock but not the current one.
They may be marked for reuse and get overwritten!

Solution

Look at block's birth TXG:

- if it was born after the checkpoint mark it for reuse.
- else keep the block marked as allocated and save its location on a log on-disk.

This way:

- checkpointed blocks don't get reused
- when the checkpoint is discarded we go through our on-disk log marking everything as free

Usage

Cheatsheet

```
# Take checkpoint
$ zpool checkpoint <pool>

# Preview checkpointed state
$ zpool import -o readonly=on --rewind-to-checkpoint <pool>

# Rewird to the checkpoint
$ zpool import --rewind-to-checkpoint <pool>

# Discard the checkpoint
$ zpool checkpoint --discard <pool>

# Check when the checkpoint was taken
$ zpool status

# Check the space usage of the checkpoint
$ zpool list
```

Demo

Caveats

When there is a checkpoint:

1] Operations changing the vdev configuration (e.g. device removal) are not allowed

2] Reservations may be unenforceable

3] `zpool scrub` may not scrub your checkpointed data (yet?)

As a general rule, always be sure to know exactly what you are reverting to

Resources & Updates

- Introductory Blogpost (https://sdimitro.github.io/post/zpool-checkpoint/)
- Slack #OpenZFS (https://openzfs.slack.com/)
- Tweet @OpenZFS (https://twitter.com/OpenZFS)

And once upstreamed, good old `man zpool` :-)

Thank you

Serapheim Dimitropoulos
Delphix
serapheim@delphix.com (mailto:serapheim@delphix.com)
https://sdimitro.github.io/ (https://sdimitro.github.io/)
@AmazingDim (http://twitter.com/AmazingDim)