Zero performance overhead OpenZFS dedup

Matt Ahrens

Dedup performance sucks...

- Assume DDT fits in RAM (ARC cache)
- One logical write
  - write (3 copies of) one random DDT block + (maybe) write data
  - ~4x write inflation (ignoring indirect blocks)
- One logical free
  - write (3 copies of) one random DDT block

… because on-disk hashtables suck

Dedup performance sucks...

- What if DDT doesn’t fit in RAM?
- One logical write
  - Read one random DDT blocks
  - write (3 copies of) one random DDT block + (maybe) write data
  - ~5x i/o inflation (ignoring indirect blocks)
- One logical free
  - Read one random DDT blocks
  - write (3 copies of) one random DDT block

… because reading from disk sucks

Dedup doesn’t have to suck!

Limit DDT to fit in RAM

- When too big, evict refcount=1 entries
- Gives new data a chance to dedup
- Eviction policy: random (for now)
- Teach zio_free() that it’s OK if we can’t find entry in DDT
  - (it was evicted)
- Note: still possible to fill RAM with refcount>1 entries
  - Then can’t add new entries to DDT

DDT on-disk: hashtable vs log

Checksum	DVA (location on disk)	Refcount
0x12345678	vdev=1 offset=765382	1
0x98765432	vdev=0 offset=827358	5
0x12345678	vdev=1 offset=765382	0
0x98765432	vdev=0 offset=827358	6
next	Next entry goes here	next

DDT on-disk: hashtable vs log

- 1024x logical writes or frees -> write one DDT log block
- Open pool -> read log, reconstruct in-memory DDT
- When log is ~75% obsolete entries, write in-memory DDT to new log

Theoretical performance gains

- One logical write
  - Write 1/1024th DDT log block + (maybe) write data
  - Old: 4-5 i/os; new: 1.003 i/os
- One logical free
  - Write 1/1024th DDT log block
  - Old: 1-3 i/os; new: 0.003 i/os

Make dedup perform well by default!

- Proof of concept implemented
  - https://github.com/ahrens/illumos/tree/dedup
- Delphix doesn’t use dedup
  - (snapshots + clones is much more efficient way to share blocks, when possible)
- Who wants to make this real?

TODO

- Property to control behavior
  - `dedup_memory=auto | <size> | <percent>% | legacy`
  - Auto: use 25% of RAM (and DDT-log)
  - Legacy: use DDT-ZAP

TODO

- Background DDT condensing
  - Currently condensed in one TXG

TODO

- What if not enough RAM to hold DDT?
  - Check this when loading DDT at pool open
  - New blocks won’t be dedup’d
  - When free dedup block, we can still log it
    - Keeps on-disk format consistent for when we add RAM

TODO

- Better in-memory representation
  - Use hashtable (currently AVL tree)
  - Compact entry (currently 192 bytes, could be ~64B)

TODO

- Better on-disk representation
  - Compact entry (currently 168 bytes, could be ~64B)
  - Need “decrement” type entry (don’t know absolute refcount when insufficient RAM to load table)

TODO: optional

- Observability: how much has been evicted?
- Investigate better eviction policy (LRU?)
- Allow “in-memory” hashtable to span RAM + 3D Xpoint / NVMe / SSD

TODO

- Better on-disk representation
  - Compact entry (currently 168 bytes, could be ~64B)
  - Need “decrement” type entry (don’t know absolute refcount when insufficient RAM to load table)

Dedup doesn’t have to suck!

Let’s make it better