Optimizing MySQL performance with ZFS

Neelakanth Nadgir
Allan Packer
Sun Microsystems

Who are we?

Allan Packer
Principal Engineer, Performance
http://blogs.sun.com/allanp

Neelakanth Nadgir
Senior Engineer, Performance
http://blogs.sun.com/realneel

What do we do?

- Work in performance organization at Sun Microsystems
- Work in the MySQL performance virtual team
- Check out a video about our group
  - Google for “mysql optimization lab”

Agenda

- ZFS Introduction
- ZFS Performance features
- ZFS and MySQL
  - MySQL IO model
  - Best practices
  - Performance Results
- ZFS FAQ

ZFS – The last word in Filesystems

- Developed at Sun circa 2004
- Opensource (CDDL)
  - 47 patents have been donated to CDDL Patents Common
- Supported Platforms
  - Officially supported on Solaris
  - Default filesystem on OpenSolaris
  - Read-Only access in MacOS 10.5
  - Experimental feature on FreeBSD 7.1
  - FUSE on Linux

ZFS – Core Features

- Data Integrity
  - > Everything is checksummed
- Immense capacity
  - > 128 bit filesystem
  - > Max size of a file is 2^{64} bytes
  - > Each directory can hold 2^{48} files
- Simple administration
  - > zpool & zfs are the only two commands you need to know
- Performance

ZFS – Design Principals

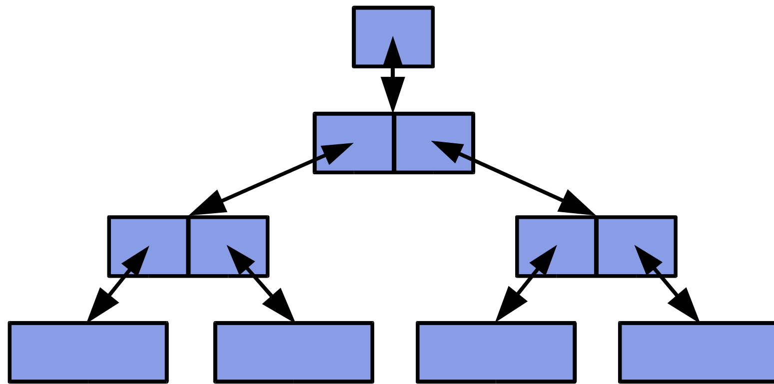
- Pooled Storage
  - > Common pool from which filesystems are allocated
- End-to-end data integrity
  - > Historically thought to be expensive, not really
  - > Alternative is unacceptable
- Everything is transactional
  - > Always consistent on disk
  - > Removes almost all constraints on IO order
  - > Think database transactions

ZFS – Design Principals

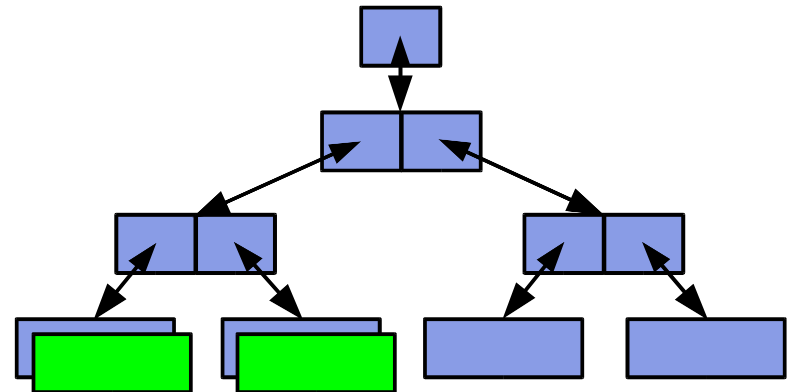
- Copy on Write
  - Never overwrite live data
  - On-disk state is always consistent.
    - No fsck
- Entire storage pool is a tree of blocks rooted at "uberblock"
  - Transactions are COW of the tree
  - Transaction group is committed when uberblock is rewritten to point to new tree
  - All levels of the tree are checksummed
  - Checksum stored in parent node, separate from data

Copy-On-Write Transactions

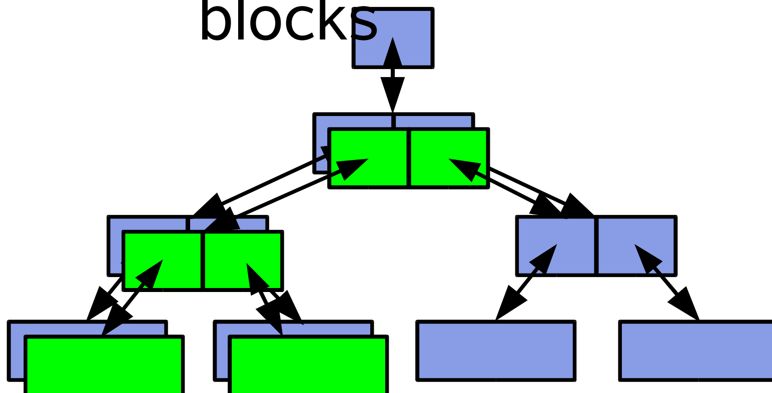
1. Initial block tree



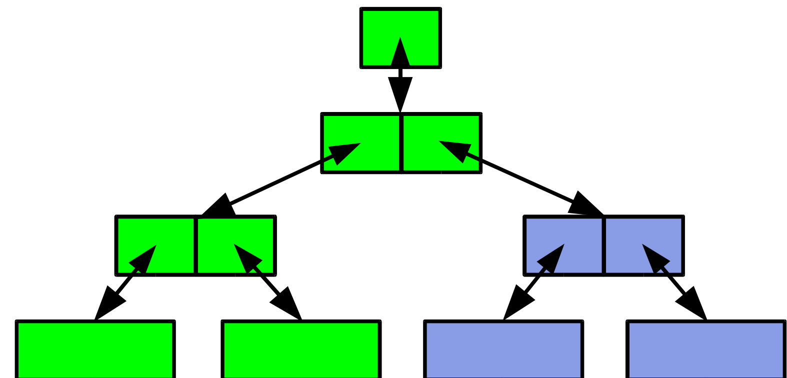
2. COW some blocks



3. COW indirect blocks

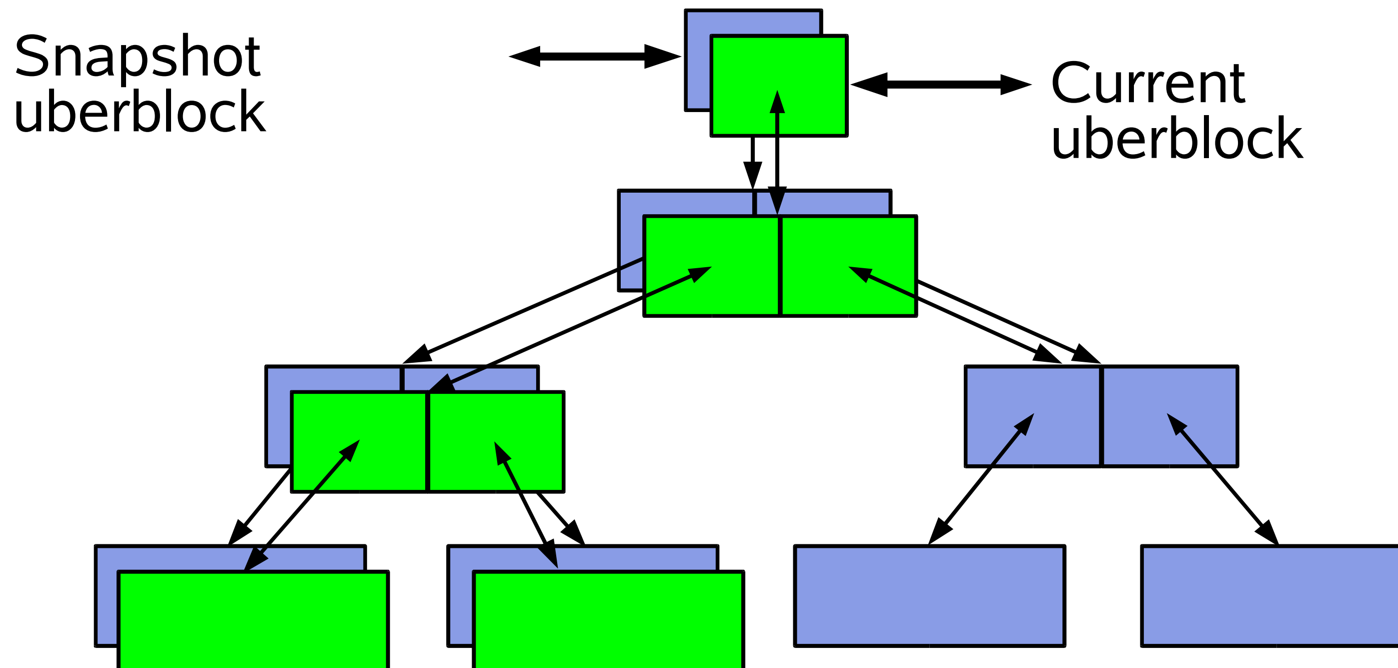


4. Rewrite uberblock (atomic)



Constant-Time Snapshots

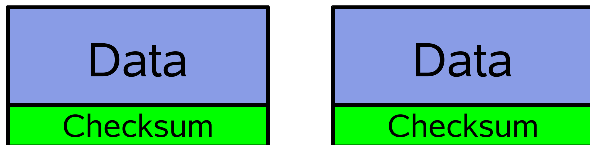
- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



End-to-End Checksums

Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't even detect stray writes
- Inherent FS/volume interface limitation

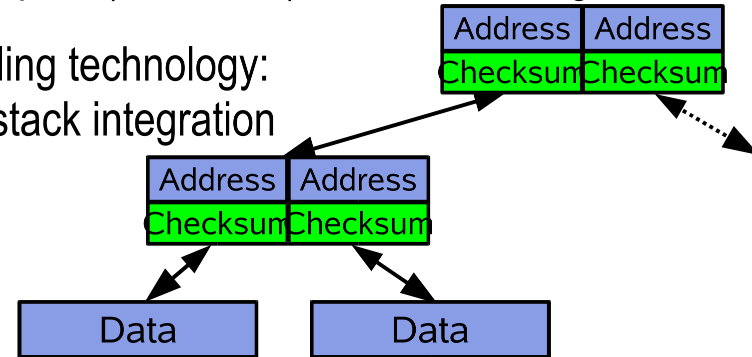


Only validates the media

- ✔ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS Checksum Trees

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire pool (block tree) is self-validating
- Enabling technology: ZFS stack integration



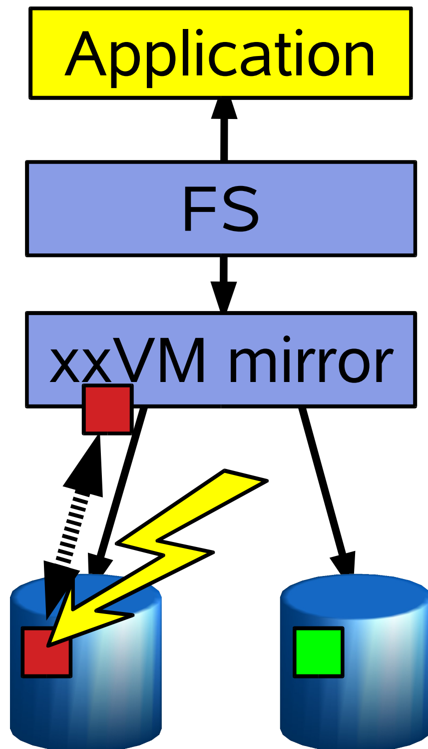
Validates the entire I/O path

- ✔ Bit rot
- ✔ Phantom writes
- ✔ Misdirected reads and writes
- ✔ DMA parity errors
- ✔ Driver bugs
- ✔ Accidental overwrite

Traditional Mirroring

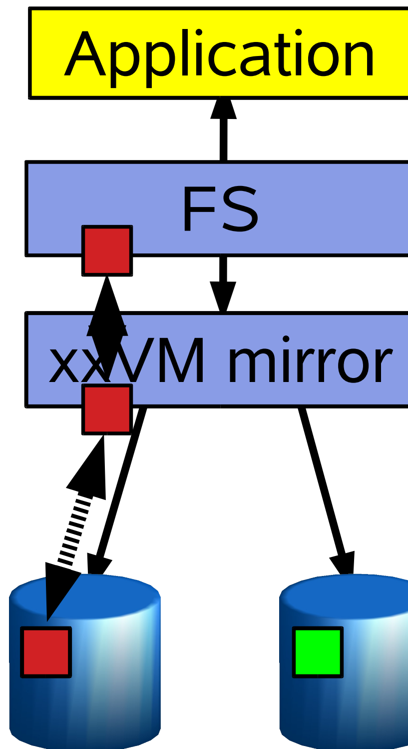
1. Application issues a read. Mirror reads the first disk, which has a corrupt block.

It can't tell.

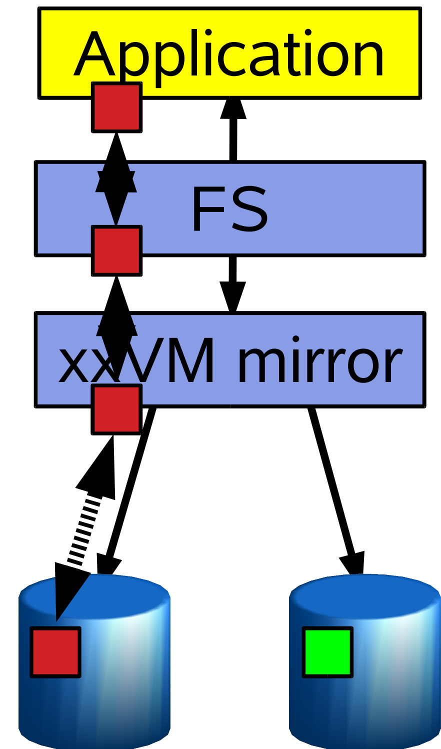


2. Volume manager passes bad block up to filesystem.

If it's a metadata block, the filesystem panics. If not...

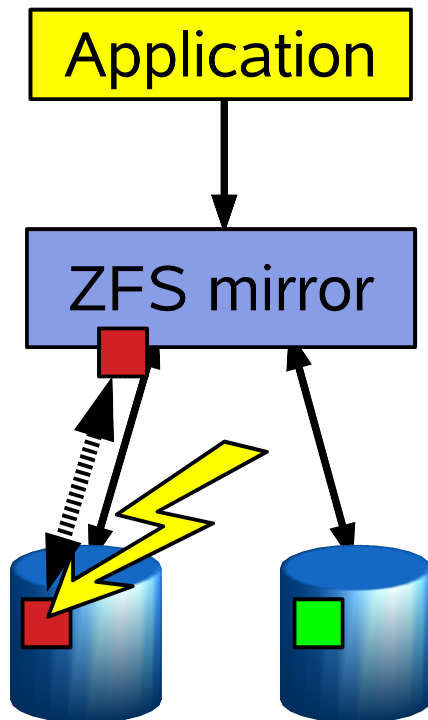


3. Filesystem returns bad data to the application. If the data is modified, both good & bad mirror copies will then be corrupted.

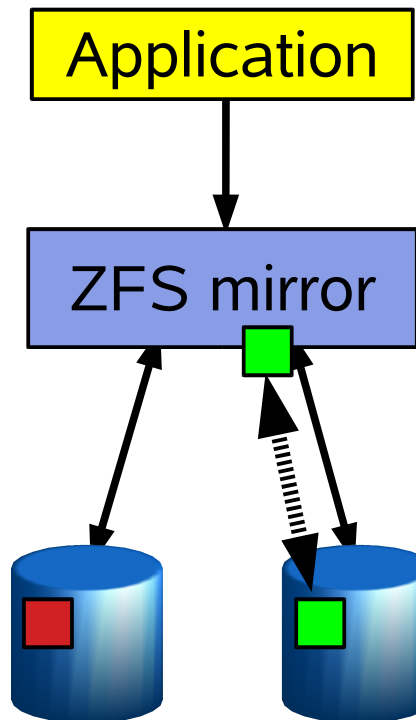


Self-Healing Data in ZFS

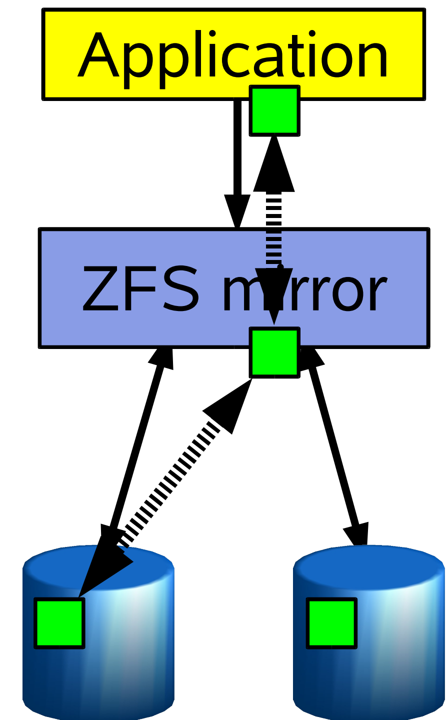
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



2. ZFS tries the second disk. Checksum indicates that the block is good.



3. ZFS returns good data to the application and repairs the damaged block.

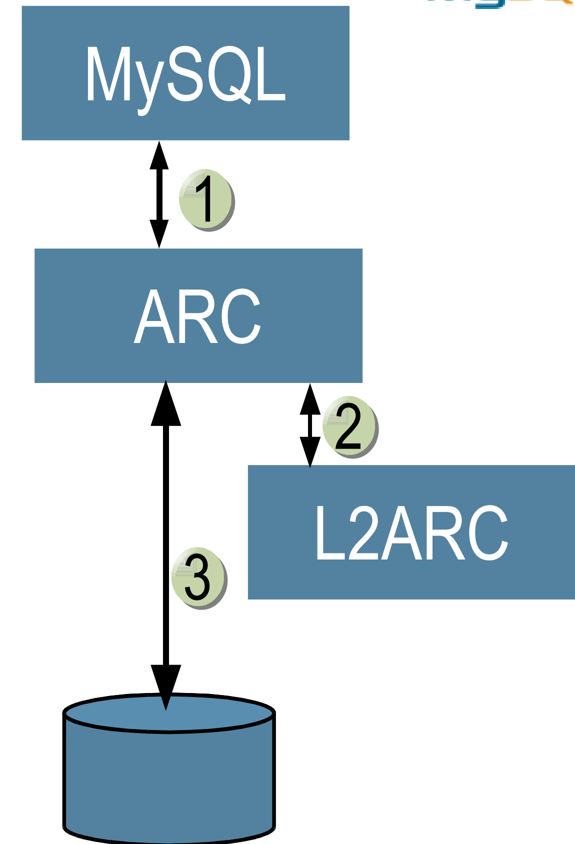


ZFS Performance features

- Dynamic striping across all devices maximizes throughput
- Copy on write makes most writes sequential
- Intelligent prefetch
- Multiple block sizes
- O(1) directory operations
- Explicit IO priority with deadline scheduling
- Globally optimal IO sorting and aggregation
- Concurrent writes
- Safely use write cache

ZFS read(2) code path

- ZFS Primary cache – The ARC
  - primarycache=[all|metadata|none]
- ZFS Second level cache - L2ARC
- When primarycache is used, data is buffered in ARC
- If L2ARC is used, it is checked before going to disk
- Prefetch is trigged if needed
- Reads have higher priority than regular writes



ZFS write(2) code path

- Regular writes are buffered in memory
- Periodically they are flushed to disk
  - Usually a sequential write to disk
- Synchronous writes are written to the ZFS Intent Log
  - After periodic write, the ZIL is cleaned up
  - ZIL aggregates IO from multiple writers
  - Can use a separate disk (or SSD) for the ZIL
  - ZIL can be disabled (Don't)
- ZFS employs byte-range locking to allow maximum concurrency. i.e No Single Writer Lock

ZFS ARC (Filesystem buffer)

- Adaptive Replacement Cache
- Dynamically switches between MRU/MFU
- Caches data from all pools
- Dynamically shrinks or grows based on memory pressure
- Survives full table scan
- Limitations
  - Works better with 64bit kernel
  - Works better with swap configured

MySQL IO Model

- Dependent on Storage engine
- Dependent on Workload
- Replication
  > One thread reading and applying the binlog to the datafiles (sequential reads, random writes)
  > One thread updating the binlog (sequential writes)
- MyISAM
  > Relies on filesystem to buffer data
  > Index is buffered in the key cache

InnoDB IO Model

- InnoDB
  - Reads are issued by user connection threads (N)
  - Writes are done by asynchronous threads
    - 1 for log and 1 for data files
    - Configurable with Performance version
  - Writes are either
    - Synchronous writes
    - Writes followed by a fsync()
  - Doublewrite buffer

MySQL and ZFS Best practices

Best practices - Caching

- Prefer to cache inside MySQL/Innodb rather than ARC
  - > Benchmark shows 7-200% improvement
  - > Same block is buffered inside Innodb as well as ARC
- Limit ARC Size
  - > Even though ARC is dynamic, more efficient to just limit it
- Cache only metadata for Innodb
  - > `zfs set primarycache=metadata tank/db`

Best practices – Record size

- Match recordsize to block size
  - > `zfs set recordsize=16k tank/db`
  - > Can be changed dynamically, but do this before creating the database
- Prevents read-modify-write
- Read only data that you want and nothing more
- Innodb
  - > 16k recordsize for data
  - > 128k recordsize for log and binlog

Best practices – Prefetch

- ZFS has two kinds of prefetch
  > File level prefetch AKA zfetch
  > Low level prefetch AKA vdev prefetch
- Turn off file level prefetch
  > `set zfs:zfs_prefetch_disable = 1`
- Low level prefetch is not trigged when recordsize is set (i.e not 128k)
- Innodb prefetch assumes file is laid out in order of primary key.
  > Not true for ZFS
  > Not configurable right now, but should be easy to fix

Best practices – IO

- ZFS IO scheduler prioritizes reads over regular writes
  > ZFS Log writes are still higher priority
  > If IO queue is full, have to wait for empty slot
  > Bug 6471212: will be fixed soon using reserved slots
- Prefer Raid0 or Mirroring over RaidZ
  > RaidZ is not suitable for random IO
- Use L2ARC to reduce penalty of missing buffer cache
  > `zpool add tank cache c2t0d0 c2t1d0`

Best practices – Separate Intent Log

- ZFS log writes can use the Separate Intent log
  - Usually NVRAM card or SSD
  - Can be done dynamically.
  - Match reliability of the pool
  - Seen 10-20% improvement for certain workloads
- Use slog to get low latency writes
  - `zpool add tank log c2t0d0`
  - Watch out – Cannot remove a slog. Fix in progress

Best practices – Cache flush

- ZFS issues a cache flush after every transaction group sync and synchronous writes
- Some vendors flush every time even if they have a battery backed cache
  - set zfs:zfs_nocacheflush = 1
- Be fair when you are comparing ZFS with other filesystems which do not flush caches.

Best practices – Compression

- ZFS supports a pluggable compression
  - > Gzip and other algorithms
  - > Data is not compressed if less than 12.5% compression
- Scalable, asynchronous compression
  - > No need for query to wait for compression to complete
- CPU cost
  - > Compression is not free, but many algorithms to choose from
- IO reduction
  - > CPU cost sometimes offset by IO reduction

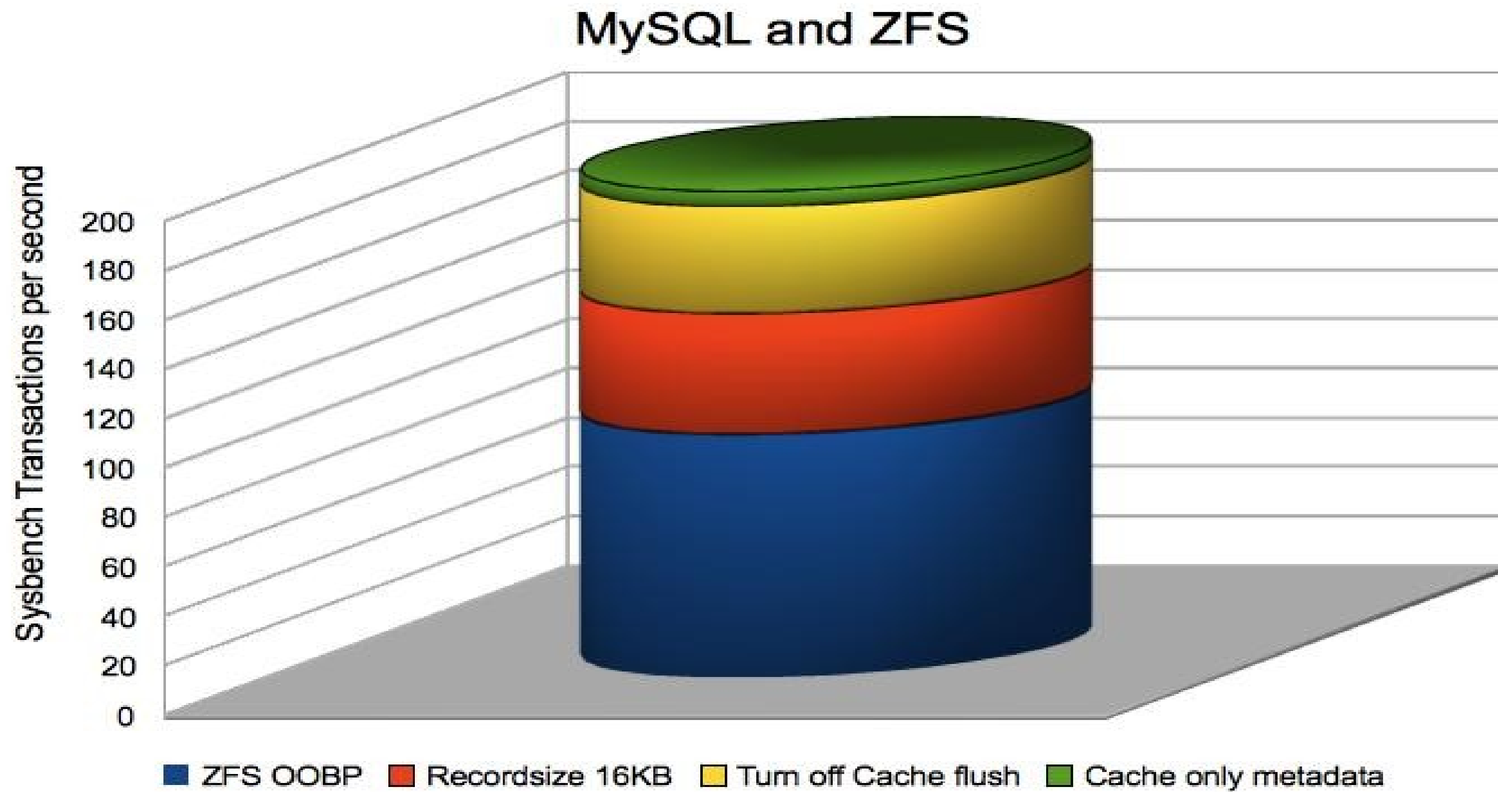
Best practices – Innodb

- Follow general Innodb IO tunings
- Innodb provides checksum and compression
  - So does ZFS
  - But ZFS "self heals" instead of crashing :-)
- Disable innodb_doublewritebuffer to remove redundant writes
  - `skip-innodb_doublewrite`

Best practices – Backup/Restore

- ZFS snapshots are very cheap
- Need to quiesce the database before snapshot
  - Flush tables with read lock
  - Get a snapshot
  - Unlock tables
- Zmanda Recovery Manager supports backup and recovery using ZFS.
- ZFS clone is a read-write snapshot
  - Useful in replication in a shared storage scenario

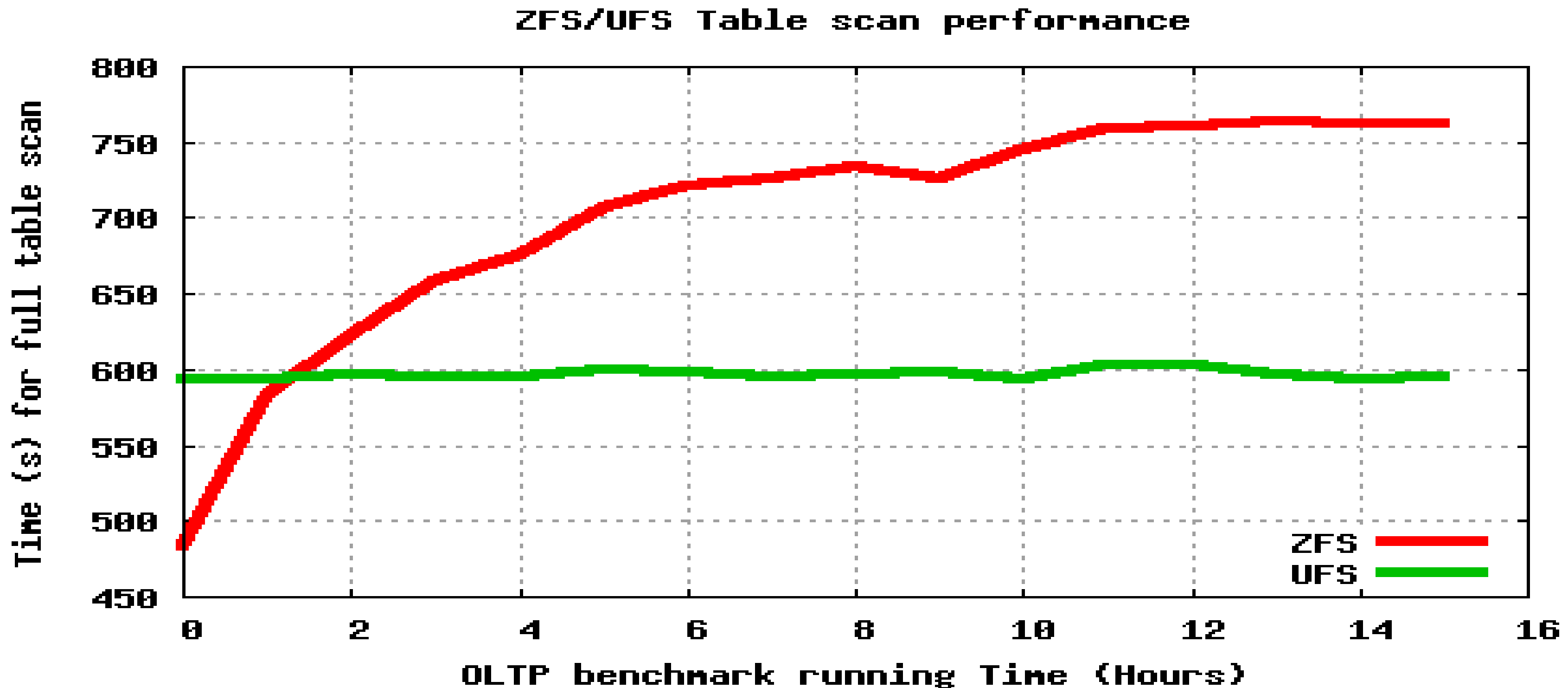
Applying best practices



ZFS COW penalty for table scans

- Since ZFS is Copy-on-write, Sequential scans will be slower when compared to 'in-place' modification
- We ran sysbench read write for a week to study this impact
  - > One hour sysbench read-write tes
  - > Followed by select count(*) from sbtest
  - > Repeat

ZFS COW penalty for table scans



Around 25% penalty after few hours

ZFS COW penalty for table scans

- Copying file over reorders the file in the optimal way.
- Idea for 'in-place' editing for db workloads
  - Bug#6699230
- SSDs nullify this penalty

ZFS FAQ (for database people)

- ZFS needs more RAM – Not true
  - > Need one byte to cache one byte
  - > Metadata is usually 1% (and can be compressed)
- ZFS needs more CPU – Somewhat true
  - > Feature set provided is much stronger than for other FS
  - > Performance bugs are actively being fixed
  - > Checksumming is not free
  - > Some people want more CPU-hungry features (ex gzip)
  - > Industry benchmarks that run at 100% CPU utilization will be at an disadvantage; however most customers rarely run at 100% utilization

ZFS FAQ - DirectIO

- ZFS and Directio
  - > DirectIO is an overloaded term to mean several things
  - > No double caching – ZFS primarycache property
  - > Concurrent writes – ZFS Range locks
  - > Direct copy to application buffer – Not supported in ZFS
  - > No inflated IO – ZFS supports multiple recordsizes
  - > ZFS does not yet support the `directio`() hint.

More information

- ZFS Best Practices Guide
  - http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide
- ZFS Evil Tuning Guide
  - http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide
- Blogs
  - http://blogs.sun.com/realneel
  - http://blogs.sun.com/roch
- Mail to zfs-discuss@opensolaris.org

Questions?

Neelakanth.Nadgir@sun.com

Allan.Packer@sun.com