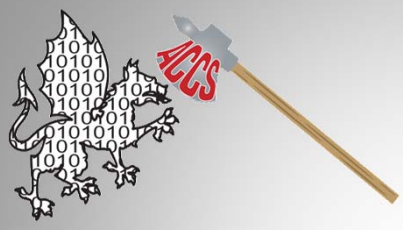


ZFS
Reliability AND Performance

Peter Ashford
Ashford Computer Consulting Service
5/22/2014

What We'll Cover

This presentation is a "deep dive" into tuning the ZFS file-system, as implemented under Solaris 11. Other versions of ZFS are likely to be similar, but I have not used them extensively, and, therefore, do not know.

If you wish to try some of the parameter changes discussed here, please do your research.

ZFS Description

ZFS is a combined file-system and logical volume manager designed and implemented by Sun Microsystems, and currently supported by Oracle.

ZFS is targeted at the Enterprise space, as can be seen in many of its capabilities and limitations.

ZFS is a Next-Generation file-system, primarily due to its end-to-end checksums and its self-healing ability.

ZFS History

ZFS development was started by Sun in 2001, with a first release in 2005 (Solaris 10).

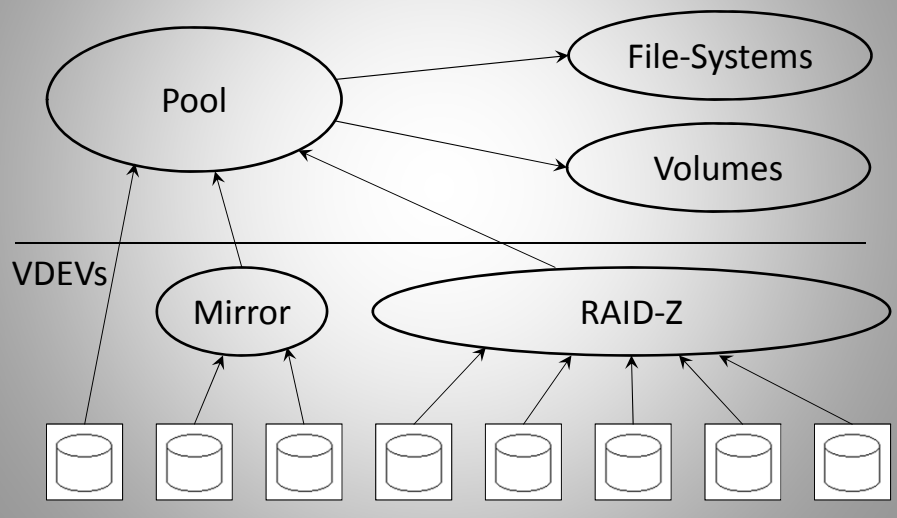
The ZFS source code was released to the open-source community by Sun under the CDDL. Oracle ceased enhancing the open-source code when they purchased Sun. This has caused a fork in the code development.

ZFS is integrated into the FreeBSD kernel and is available as a package for many Linux distributions.

ZFS Goals

- High Capacity & Scalability
- High Reliability – Identify bit-rot and bad writes through end-to-end checksums and correct them with validated redundant data
- High Availability – Maintain data availability through redundant disks and data structures
- High Performance – Highly parallel implementation – delivers the most from today's multi-core processors

ZFS Architecture



ZFS Features

- 128-bit file-system
- Copy-On-Write (COW) transactional object model
- Adaptive block size
- Fast snapshots with clones and rollback
- End-to-End checksums with scrubbing and self-healing
- Integrated NFS/CIFS/SMB/iSCSI management
- Integrated compression and dedup (deduplication)
- Integrated encryption (new in Solaris 11)
- Integrated Lustre interface
- Clustered ZFS available from 3rd parties

ZFS Features - Checksums

- Checksum options are None (not advised), Fletcher-2, Fletcher-4 (default), SHA-256 and SHA-256+MAC (encrypted only)
- In ZFS with mirrors or RAID-Z, the checksums protect against:
  - Bit rot
  - Misdirected writes
  - Torn writes
  - Data path corruption
  - Parity pollution

ZFS Advantages

- Large pool/file-system support (2^{78} byte limit)
- Large file support (2^{64} byte limit)
- Large numbers of disks in a pool (2^{64} max)
- Large directory support (2^{48} entries)
- Unicode name support
- Add space to a pool by adding VDEVs
- On-disk structure is always consistent
  - no need for a file-system check program

ZFS Disadvantages

- Large memory requirement, due to ARC (Adaptive Replacement Cache)
- VERY large memory requirements when dedup is enabled (~5GB memory/TB disk used)
- Not possible to add a disk to an array
- Not possible to change the structure of an array
- Not possible to remove a VDEV from a pool
- Complex on-disk format
- Relatively high CPU requirement vs. previous-generation file-systems

ZFS Backups

ZFS file-systems can easily be large enough to make normal backup methods, such as tape, difficult to use. Usually, the best way to back up large ZFS file-systems is to send the data to another file server, using snapshots, ZFS send and ZFS receive.

ZFS Performance

Hardware Tuning

ZFS Performance – CPU

- A fast clock is more important than lots of cores
  - Checksums require significant CPU power
  - Compression and encryption (when enabled) each require more CPU power and memory bandwidth than Checksums
- This is a single-thread latency issue, not a throughput issue
- A large CPU cache will help ZFS performance

ZFS Performance – Memory

- It is strongly advised to use ECC memory and an ECC-enabled CPU with ZFS, to maximize the potential of the self-healing file-system
- Use the highest bandwidth memory supported by the CPU & Motherboard

ZFS Performance – Memory

- The ARC likes to have a LOT of memory
- L2ARC metadata is all stored in the ARC, reducing the ARC's effectiveness
- Dedup likes to keep all the checksums in the ARC, requiring a HUGE amount of memory
- Any action that releases a large amount of memory from the ARC will have a significant impact on system performance

ZFS Performance – Disks

- Disks become a bottleneck with heavy write loads, or when there are a lot of cache misses
- In general, more small disks will deliver better performance than fewer large disks
- Best RAID-Z performance is when there are 2, 4 or 8 disks, plus the parity disks
- When ZFS was created, SUN found that performance dropped when there were >10 disks in a VDEV

ZFS Performance – Disks

- Number of disk channels doesn't usually matter, but each array should be spread over as many as possible
  - I/O to a single VDEV should not saturate a channel
  - Loss of a channel should not cause loss of data
- Better balance of I/O load is achieved when all VDEVS in a pool have the same capacity
- It may take a few weeks to rebalance the I/O load after a new VDEV is added to a pool

ZFS Performance – SSDs

- SSDs are an excellent way to improve performance of a ZFS pool
- Primary SSD uses for ZFS are ZIL and L2ARC, both of which can have high write rates
- When using an SSD as either ZIL or L2ARC, only "Enterprise" SSDs should be used, due to their ability to survive longer with sustained high write rates

ZFS Performance – ZIL

- ZIL – ZFS Intent Log – Write cache for critical metadata and synchronous writes
- Can be disabled – Not good for reliability
- Can be placed on an SSD
  - HUGE synchronous (NFS and metadata) write performance improvement
  - NFS read performance improvement – similar to 'noatime'
  - zpool add (pool) log (disk|mirror)

ZFS Performance – L2ARC

- ZFS can back the ARC with an L2ARC (Level 2 Adaptive Replacement Cache)
- Useful if hit rate of Ghost MFU or Ghost MRU is over 10-20%
- Use an SSD or fast disk as an L2ARC device if memory can't be expanded enough
  - zpool add (pool) cache (disk)

ZFS Performance – Disk Timeouts

- SCSI device timeout can be adjusted in /kernel/drv/sd.conf
  - sd:sd_io_time = <Seconds for a disk timeout>
- Most disks use the SCSI driver (sd)

ZFS Performance – AF Disks

- Most modern disks use 4KB physical sectors and 512B logical sectors (Advanced Format)
- AF disks have a 7-11% density increase, and improved ECC
- Small (i.e. 512B) writes to an AF disk trigger a RMW (Read/Modify/Write) cycle in the disk
- Replacing a 512B sector drive with a 4KB sector drive (when they become available) will probably fail, due to the sector size change

ZFS Performance – AF Disks

- It is possible to override the physical block size of a disk on a model-by-model basis, in /kernel/drv/sd.conf:

```
sd-config-list =
    “SEAGATE ST3300657SS”, “physical-block-size:4096”,
    “DGC     RAID”, ”, “physical-block-size:4096”,
    “NETAPP  LUN”. ”, “physical-block-size:4096”;
```

Partial list:

http://wiki.illumos.org/display/illumos/List+of+sd-config-list+entries+for+Advanced-Format+drives

ZFS Performance – 10GbE

- Increase buffers
  - ipadm set_prop –p recv_buf 1048276tcp
  - ipadm set_prop –p send_buf 1048276tcp
  - ipadm set_prop –p max_buf 2097152 tcp
  - ipadm set_prop –p _cwnd_max 2097152 tcp
- The above changes are persistent across boots

ZFS Performance – 10GbE

- Improve interrupt processing
  - ip:ip_squeue_fanout = 1
  - ip:ip_squeue_bind = 0
  - ip:ip_squeue_wput = 1
- In /kernel/drv/ixgbe.conf, set intr_throttling to 0

ZFS Performance

NFS Tuning

ZFS Performance – NFS

- Increase number of threads
  - nfs:nfs3_max_threads = <1024>
  - nfs:nfs4_max_threads = <1024>
- Increase read-ahead count
  - nfs:nfs3_nra = <32>
  - nfs:nfs4_nra = <32>
- Increase number of lock servers
  - Add “LOCKD_SERVERS=16384” to /etc/dafault/nfs

ZFS Performance – NFS

- Increase maximum transfer size
  - Defaults to 1MB in NFS3 and 32KB in NFS4
  - Limited to <64KB for UDP mounts
  - nfs:nfs3_max_transfer_size = <1048576>
  - nfs:nfs4_max_transfer_size = <1048576>
- Increase logical block size
  - Defaults to 32KB (holdover from NFS over UDP)
  - nfs:nfs3_bsize = <1048576>
  - nfs:nfs4_bsize = <1048576>

ZFS Performance

Application Tuning

ZFS Performance – Applications

We should verify that locally created applications aren't doing anything that's likely to cause a performance problem.

- Use large and well formed I/O
  - Multiple of 4KB (bigger is better)
  - Starts on a 4KB boundary (bigger power-of-2 is better)

ZFS Performance – Applications

- No huge directories
  - ZFS uses “scalable hash algorithms” for directory searches, allowing them to be fast
  - When one thread is searching a directory, the directory inode is locked
  - Other threads wait for the lock to be released
  - For heavily used directories, it’s better to use nested directories
    - For example, “/tmp/asdfjkl2d148e” becomes “/tmp/projectA/48e/2d1/asdfjkl2d148e”

ZFS Performance

OS Tuning

ZFS Performance

Principal ZFS performance enhancement document – “ZFS Evil Tuning Guide”

http://solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide

Tuning ZFS is “evil”, but sometimes it’s a necessary evil.

ZFS Performance – Parameters

These commands can be used to retrieve most of the current ZFS parameters:

- ZFS – echo “::zfs_params” | mdb -k
- ARC – kstat -n arcstats
- VDEV tree – echo “::spa -c” | mdb -k
- Kernel memory – echo “::kmastat” | mdb -k

ZFS Performance – Pools

- Highest overall ZFS performance will be obtained when there is only one data pool on the system, and all data file-systems are in that pool, as this provides the maximum number of disks to all of the file-systems
- OS should be in a separate pool
- Allocate entire disks to ZFS whenever possible
- Don't mix ZFS and other file-systems on disks
- Don't mix ZIL and L2ARC on disks

ZFS Performance – Free space

- ZFS performance used to drop quickly below 20% free space
- At 70% usage, ZFS would switch from a "first fit" allocation policy to a "best fit" allocation policy
- This problem has been **resolved**
- The change in behavior was moved to about 4% free space

ZFS Performance – Checksums

- ZFS checksums data and metadata by default
- ZFS can use different checksums, or none (not recommended)
- Checksums run in multiple threads for high throughput and low latency
- Checksums under ZFS are used to verify data & metadata, and to perform dedup
- The checksum algorithm is set on a per-file-system basis by the “zfs” command

ZFS Performance – Compression, and Encryption

- Compression and encryption are CPU-intensive
  - Affects single I/O latency to the application
  - Improved only with faster CPUs
- These are set on a per-file-system basis by the “zfs” command

ZFS Performance – Dedup

- Dedup is memory intensive, and can be disk I/O intensive
  - Dedup likes HUGE memory
  - If dedup is set for no data verify on synonym, use a complex checksum like SHA-256 (adds CPU load)
  - If dedup is set for data verify on synonym (can be disk I/O intensive), use a simple checksum like Fletcher-2
- Set on a per-file-system basis by the “zfs” command

ZFS Performance – Blocksize

- ZFS automatically adjusts blocksize
- ZFS blocksize can be tuned on a per-file-system basis (the “recordsize” property)
- Tuning the blocksize will not help with sequential workloads, due to prefetch
- Tuning the blocksize is most helpful with applications perform mostly random I/O
- Tuning the blocksize may affect the amount of metadata in a pool
- If in doubt, don’t tune the blocksize

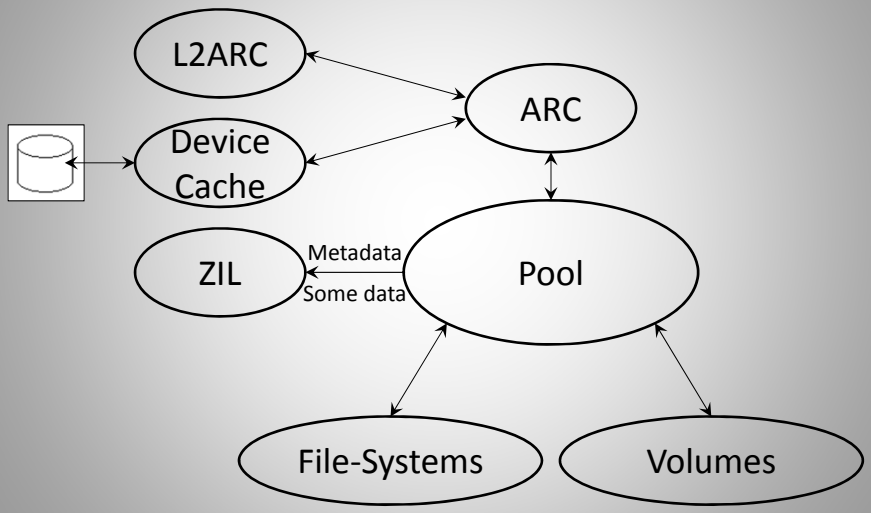
ZFS Performance – I/O Queue size

- ZFS maximum queue depth defaults to 10 per disk
  - Optimum for pools with ZIL, using local disks
  - Too small for pools backed by RAID or SAN
  - May be too large for pools without a ZIL
  - zfs:zfs_vdev_max_pending = <Max Queue Depth for device>
  - Should be the smaller of sd:sd_max_throttle and ssd:ssd_max_throttle (these may need to be adjusted)

ZFS Performance – Name Cache

- ZFS caches names of recently used files and directories
- Oracle suggests sizing for a hit rate of at least 90%
- The size of the cache can be adjusted
  - ncsize=129728

ZFS Performance – Caching



ZFS Performance – Device Cache

- ZFS can perform device-level caching
- Reads expanded to 64KB
  - zfs:zfs_vdev_cache_max = 16384
  - zfs:zfs_vdev_cache_bshift = 16
- Defaults to 0MB per VDEV of MRU cache
  - zfs:zfs_vdev_cache_size = 0

ZFS Performance – File Prefetch

- ZFS performs file-level prefetch by default
  - On by default – should be disabled for workloads that have high percentages of random I/O
  - If prefetch hits are less than 10%, disable prefetch
  - zfs:zfs_prefetch_disable = <0/1>
  - Prefetch hit counters in ARC statistics
    - prefetch_data_hits, prefetch_metadata_hits
    - prefetch_data_misses, prefetch_metadata_misses

ZFS Performance – Read Cache

- ZFS uses an ARC (Adaptive Replacement Cache)
- Uses larger of 3/4 physical memory or 1GB less than physical memory for the ARC – Too large on most systems
- Change ARC size with:
  - zfs:zfs_arc_max = <Max bytes in ARC>
  - zfs:zfs_arc_min = <Min bytes in ARC>
- Setting the maximum size is critical in Linux

ZFS Performance – Read Cache

- This is a debugging hint
- ZFS shrinks ARC for unknown reasons on a dedicated file server
  - Set zfs:zfs_arc_min to 3/4 or more of zfs_arc_max
  - This is a kludge, but it should help to determine where the memory pressure is coming from, without destroying ZFS performance

ZFS Performance – Read Cache

- ZFS shrinks ARC to free up space
  - zfs:arc_shrink_shift – defaults to 7 – 1/128 memory
  - On large memory systems (>64GB), this should be tuned to be 128MB-512MB

ZFS Performance – L2ARC

- Allow simultaneous read & write to L2ARC
  - Large performance increase with SSD cache
  - Significant performance hit with spinning disks
  - zfs:l2arc_norw = 0
- Increase L2ARC write buffers from default of 2
  - Useful if multiple L2ARC devices are used
  - Useful if the L2ARC devices are VERY fast
  - zfs:l2arc_headroom = <Number of L2ARC write buffers>

ZFS Performance – L2ARC

- L2ARC write bandwidth defaults to 8MB/S
  - zfs:l2arc_write_max = <Size of L2ARC write buffer>
  - Should be under 1/8 total L2ARC disk/SSD write rate
- Faster initialization of L2ARC
  - zfs:l2arc_write_boost = <Size of L2ARC initial fill write buffer>
  - Should be under 1/4 total L2ARC disk/SSD write rate

ZFS Performance – Cache flush

- ZFS originally designed to communicate directly with disks
- ZFS issues a “Cache Flush” command to the disk every few seconds
- Some RAID and SAN back-end devices can have performance issues when cache flush requests are received
  - zfs:zfs_nocacheflush = <0/1>

ZFS Performance – Write flush

- Write flushes default to every 30 seconds
  - zfs:zfs_txg_timeout = <Seconds between flushes>
- Write flush duration defaults to 5 seconds
  - zfs:zfs_txg_synctime_ms = <Duration of flush in ms>
- Long write flush duration can slow reads
  - Set the write flush duration to 125-500ms

ZFS Performance – Write Throttling

- ZFS can throttle write speed
- Sometimes used to limit impact of large writes on read cache
  - zfs:zfs_write_limit_min = <lowest value to be used for max bytes/S limit>
  - zfs:zfs_write_limit_max = <highest value to be used for max bytes/S limit>
- Write cache throttling can be disabled
  - zfs:zfs_no_write_throttle = <0/1>

Other Tuning Helpers

- Dtrace Toolkit
  - http://www.solarisinternals.com/wiki/index.php/DTraceToolkit
- Dtrace Scripts from Kirill Davydychev of Nexenta (may not work correctly on Solaris 11)
  - https://github.com/kdavyd/dtrace

Sources

- ZFS Best Practices Guide
- ZFS Evil Tuning Guide
- ZFS in the Trenches by Ben Rockwood
- ZFS Tune One-Liners by Rigel-Major
- Illumos WIKI
- http://www.zfsbuild.com/2012/03/05/when-is-enough-memory-too-much-part-2/
- http://zfsonlinux.org/
- https://maczfs.googlecode.com/files/ZFSOnDiskFormat.pdf

Future Enhancements

The following enhancements are planned for this presentation at some time in the future:

- Add information on FreeBSD and Linux
- Convert to a technical paper, instead of a presentation

Feedback

My goal is to improve this presentation. To that end, I would appreciate feedback to:

- ashford@whisperpc.com
- www.linkedin.com/in/peterashford