ZFS Best Practices Guide

From Siwiki

Jump to: navigation, search

Contents

[hide]

[edit] ZFS Administration Considerations

[edit] ZFS Storage Pools Recommendations

This section describes recommendations for setting up ZFS storage pools.

[edit] Systems

- Run ZFS on a system that runs a 64-bit kernel

[edit] Memory and Swap Space

- One Gbyte or more of memory is recommended
- Approximately 64 Kbytes of memory is consumed per mounted ZFS file system. On systems with 1,000s of ZFS file systems, we suggest that you provision 1 Gbyte of extra memory for every 10,000 mounted file systems including snapshots. Be prepared for longer boot times on these systems as well.
- Because ZFS caches data in kernel addressable memory, the kernel sizes will likely be larger than with other file systems. You may wish to configure additional disk-based swap to account for this difference for systems with limited RAM. You can use the size of physical memory as an upper bound to the extra amount of swap space that might be required. In any case, you should monitor the swap space usage to determine if swapping is occurring.
- If possible, do not use slices on the same disk for both swap space and ZFS file systems. Keep the swap areas separate from the ZFS file systems. The best policy is to have enough RAM that your system will not normally use the swap devices.

For additional memory considerations, see
#Memory_and_Dynamic_Reconfiguration_Recommendations

[edit] Storage Pools

- Set up one storage pool using whole disks per system, if possible.
- For production systems, consider using whole disks for storage pools rather than slices for the following reasons:

```
 * Allow ZFS to enable the disk's write cache, for those disks that have write
caches. If you are using a RAID array
   with a non-volatile write cache, then this is less of an issue and slices as
vdevs should still gain the benefit of
   the array's write cache.
 * Recovery process of replacing a failed disk is more complex when disks contain
both ZFS and UFS file systems on
   slices.
 * ZFS pools (and underlying disks) that also contain UFS file systems on slices
cannot be easily migrated to other
   systems by using zpool import and export features.
 * In general, maintaining slices increases administration time and cost. Lower
your administration costs by
   simplifying your storage pool configuration model.
```

- For all production environments, set up a redundant ZFS storage pool, such as a raidz, raidz2, or a mirrored configuration, regardless of the RAID level implemented on the underlying storage device.
- Do not create a raidz, raidz2, or a mirrored configuration with one logical device of 48 devices. See the sections below for examples of redundant configurations.
- In a replicated pool configuration, leverage multiple controllers to reduce hardware failures and improve performance. For example:

```
# zpool create tank mirror c1t0d0 c2t0d0
```

- Set up hot spares to speed up healing in the face of hardware failures. Spares are critical for high mean time to data loss (MTTDL) environments. For example:

```
# zpool create tank mirror c1t0d0 c2t0d0 spare c1t1d0 c2t1d0
```

- Run zpool scrub on a regular basis, to identify data integrity problems. If you have consumer-quality drives, consider a weekly scrubbing schedule. If you have datacenter-quality drives, consider a monthly scrubbing schedule.
- ZFS works well with solid-state storage devices that emulate disk drives (SSDs). You might wish to enable compression on storage pools that contain such devices because of their relatively high cost per byte.

[edit] Multiple Storage Pools on the Same System

- The pooling of resources into one ZFS storage pool allows different file systems to get the benefit from all resources at different times. This strategy can greatly increase the performance seen by any one file system.
- If some workloads require more predictable performance characteristics, then it can be interesting to separate loads into different pools.
- For instance, Oracle log writer performance is critically dependent on I/O latency and we expect best performance to be achieved by keeping that load on a separate small pool that has lowest possible latency.

[edit] Root Pool Recommendations

If you are using a ZFS root file system, keep the root pool (that is, the pool with the dataset that is allocated for the root file system) separate from pool(s) that are used for data.

Several reasons exist for this strategy:

- Some limitations on root pools exist that you would not want to place on data pools. Mirrored pools and pools with one disk will be supported. However, no RAID-Z or unreplicated pools with more than one disk will be supported.
- Data pools can be architecture-neutral. It might make sense to move a data pool between SPARC and Intel. Root pools are pretty much tied to a particular architecture.
- In general, we think it's a good idea to separate the "personality" of a system from its data. Then, you can change one without having to change the other.

Setting up different pools for what we currently allocate as separate file systems (root, /usr and /var, for example) makes no sense at all. It probably won't even be a supported configuration. Separate datasets

for those directories will be possible, but only in the same pool.

For info about ZFS and a SVM mirrored root scenario, see #UFS/SVM.

[edit] Storage Pool Considerations

[edit] General Storage Pool Performance Considerations

- For better performance, use individual disks or at least LUNs made up of just a few disks. By providing ZFS with more visibility into the LUNs setup, ZFS will be able to make better I/O scheduling decisions.
- Depending on workloads, the current ZFS implementation can, at times, cause much more I/O to be requested than other page-based file systems. If the throughput flowing toward the storage, as observed by iostat, nears the capacity of the channel linking the storage and the host, tuning down the zfs recordsize should improve performance. This tuning is dynamic, but only impacts new file creations. Existing files keep their old recordsize.
- Tuning recordsize does not help sequential type loads. Tuning recordsize is aimed at improving workloads that intensively manage large files using small random reads and writes.
- See also #ZFS_and_Database_Recommendations.
- Currently, pool performance can degrade when a pool is very full and file systems are updated frequently, such as on a busy mail server. Under these circumstances, keep pool space under 80% utilization to maintain pool performance.

[edit] Memory and Dynamic Reconfiguration Recommendations

The ZFS adaptive replacement cache (ARC) tries to use most of a system's available memory to cache file system data. The default is to use all of physical memory except 1 Gbyte. As memory pressure increases, the ARC relinquishes memory.

Consider limiting the maxmium ARC memory footprint in the following situations:

- When a known amount of memory is always required by an application. Databases often fall into this category.
- On platforms that support dynamic reconfiguration of memory boards, to prevent ZFS from growing the kernel cage onto all boards.
- A system that requires large memory pages might also benefit from limiting the ZFS cache, which tends to breakdown large pages into base pages.
- Finally, if the system is running another non-ZFS file system, in addition to ZFS, it is advisable to leave some free memory to host that other file system's caches.

The trade off is that limiting this memory footprint means that the ARC is unable to cache as much file system data, and this limit could impact performance.

In general, limiting the ARC is wasteful if the memory that now goes unused by ZFS is also unused by other system components. Note that non-ZFS file systems typically manage to cache data in what is nevertheless reported as free memory by the system.

In the current Solaris Nevada release, you can use the zfs_arc_max system variable to set the maximum ARC size. For example, if an application needs 5 Gbytes of memory on a system with 36-Gbytes of memory, you could set the arc maximum to 30 Gbytes (0x780000000). Add:

set zfs:zfs_arc_max = 0x780000000

To the /etc/system file and reboot.

In current Solaris 10 releases, you can only change the ARC maximum size by using the mdb command. Because the system is already booted, the ARC init routine has already executed and other ARC size parameters have already been set based on the default c_max size. Therefore, you should tune the arc.c and arc.p values, along with arc.c_max, using the formula:

arc.c = arc.c_max

arc.p = arc.c / 2

For example, to the set the ARC parameters to small values, such as arc_c_max to 512MB, and complying with the formula above (arc.c_max to 512MB, and arc.p to 256MB), use the following syntax:

```
# mdb -kw
 > arc::print -a p c c_max
ffffffffc00b3260 p = 0xb75e46ff
ffffffffc00b3268 c = 0x11f51f570
ffffffffc00b3278 c_max = 0x3bb708000

 > ffffffffc00b3260/Z 0x10000000
ffffffffc00b3260:  0xb75e46ff        = 0x10000000
 > ffffffffc00b3268/Z 0x20000000
ffffffffc00b3268:  0x11f51f570        = 0x20000000
 > ffffffffc00b3278/Z 0x20000000
ffffffffc00b3278:  0x11f51f570        = 0x20000000
```

You should verify the values have been set correctly by examining them again in mdb (using the same print command in the example). You can also monitor the actual size of the ARC to ensure it has not exceeded:

```
#echo "arc::print -d size" | mdb -k
```

This will display the current ARC size in decimal.

[edit] RAID-Z Configuration Requirements and Recommendations

A RAID-Z configuration with N disks of size X with P parity disks can hold approximately (N-P)*X bytes and can withstand one device failing before data integrity is compromised.

- Start a single-parity RAID-Z (raidz) configuration at 3 disks (2+1)
- Start a double-parity RAID-Z (raidz2) configuration at 5 disks (3+2)
- (N+P) with P = 1 (raidz) or 2 (raidz2) and N equals 2, 4, or 8
- The recommended number of disks per group is between 3 and 9; if you have more disks, use multiple groups.

[edit] Mirrored Configuration Recommendations

- No currently reachable limits exist on the number of devices
- On a Sun Fire X4500 server, do not create a single mirror with 48 devices. Consider creating 24 2-device mirrors. This configuration reduces the disk capacity by 1/2, but up to 24 disks or 1 disk in each mirror could be lost without a failure.
- If you need better data protection, a 3-way mirror has a significantly greater MTTDL than a 2-way mirror. Going to a 4-way (or greater) mirror may offer only marginal improvements in data protection. Concentrate on other methods of data protection if 3-way mirror is insufficient.

[edit] Should I Configure a RAID-Z, RAID-Z2, or a Mirrored Storage Pool?

A general consideration is whether your goal is to maximum disk space or maximum performance.

- A RAID-Z configuration maximizes disk space and generally performs well when data is written and read in large chunks (128K or more).
- A RAID-Z2 configuration offers excellent data availability, and performs similarly to RAID-Z. RAID-Z2 has significantly better mean time to data loss (MTTDL) than either RAID-Z or 2-way mirrors.
- A mirrored configuration consumes more disk space but generally performs better with small random reads.
- If your I/Os are large, sequential, or write-mostly, then ZFS's I/O scheduler aggregates them in such a way that you'll get very efficient use of the disks regardless of the data replication model.

For better performance, a mirrored configuration is strongly favored over a RAID-Z configuration particularly for large, uncacheable, random read loads.

For more information about RAIDZ considerations, see this blog:

WHEN TO (AND NOT TO) USE RAID-Z[[1]]

[edit] RAID-Z Configuration Examples

For RAID-Z configuration on a Thumper, mirror c3t0 and c3t4 (disks 0 and 1) as your root pool, with the remaining 46 disks available for user data. The following raidz2 configurations illustrate how to set up the remaining 46 disks.

```
* 5x(7+2), 1 hot spare, 17.5 TB
* 4x(9+2), 2 hot spares, 18.0 TB
* 6x(5+2), 4 hot spares, 15.0 TB
```

[edit] ZFS Migration Considerations

[edit] UFS/SVM

Currently, you can't use ZFS as a root file system in the Solaris 10 6/06 release. If you want a mirrored root file system, you need to use SVM to mirror the slices that contain the system software (root, /usr, /var, and so on). All other storage can be managed with ZFS. Do not overlap storage between ZFS and SVM. For example, you can use a disk or slice as either part of an SVM volume or as part of a ZFS

storge pool. Do not use the same disk or slice in both an SVM and ZFS configuration.

Consider the following practices when migrating data from UFS file systems to ZFS file systems:

- Unshare the existing UFS file systems
- Unmount the existing UFS file systems from the previous mount points
- Mount the UFS file systems to temporary unshared mount points
- Migrate the UFS data with parallel instances of rsync running to the new ZFS file systems
- Set the mount points and the sharenfs properties on the new ZFS file systems

[edit] UFS/SVM Interaction

ZFS works best without any additional volume management software.

If you need an extra level of volume management, ZFS expects that 1 to 4 Mbytes of consecutive logical block map to consecutive physical blocks. By keeping this rule, we should allow ZFS to drive the volume with the efficiency we want.

[edit] VxVM/FS

[edit] General ZFS Administration Information

- ZFS administration is performed while the data is online.
- For information about setting up pools, see #ZFS_Storage_Pools_Recommendations.
- ZFS file systems are mounted automatically when created.
- ZFS file systems do not have to be mounted by modifying the /etc/vfstab file.
- Currently, ZFS doesn't provide a comprehensive backup/restore utility like ufsdump and ufsrestore commands. However, you can use the zfs send and zfs receive commands to capture ZFS data streams. You can also use the ufsrestore command to restore UFS data into a ZFS file system.
- For most ZFS administration tasks, see the zfs.1m and zpool.1m man pages. For more detailed documentation, see the ZFS Administration Guide. To ask ZFS questions, join the opensolaris/zfs discussion group.

[edit] Using ZFS for Application Servers Considerations

[edit] ZFS NFS Server Practices

Consider the following lessons learned from a UFS to ZFS migration experience:

- Existing user home directories were renamed but they were not unmounted. NFS continued to serve the older home directories when the new home directories were also shared.
- Do not mix UFS directories and ZFS file systems in the same file system hierarchy because this model is confusing to administer and maintain.
- Do not mix NFS legacy shared ZFS file systems and ZFS NFS shared file systems. Go with ZFS NFS shared file systems.

- ZFS file systems are shared with the sharenfs file system property and zfs share command. For example:

```
# zfs set sharenfs=on export/home
```

- This syntax shares the file system automatically. If ZFS file systems need to be shared, use the zfs share command. For example:

```
# zfs share export/home
```

For information about ZFS-over-NFS performance, see #ZFS_and_NFS_Server_Performance.

[edit] ZFS NFS Server Benefits

- NFSv4-style ACLs are available with ZFS file systems and ACL information is automatically available over NFS.
- ZFS snapshots are available over NFSv4 so NFS mounted-home directories can access their .snapshot directories.

[edit] ZFS Home Directory Server Practices

Consider the following practices when planning your ZFS home directories:

- Set up one file system per user
- Use quotas and reservations to manage user disk space
- Use snapshots to back up users' home directories

Consider the following practices when migrating data from UFS file systems to ZFS file systems:

- Unshare the existing UFS file systems
- Unmount the existing UFS file systems from the previous mount points
- Mount the UFS file systems to temporary unshared mount points
- Migrate the UFS data with parallel instances of rsync running to the new ZFS file systems
- Set the mount points and the sharenfs properties on the new ZFS file systems

See the ZFS NFS Server Practices section for additional tips on sharing ZFS home directories over NFS.

[edit] ZFS Home Directory Server Benefits

- ZFS can handle many small files and many users because of its high capacity architecture.
- Additional space for user home directories is easily expanded by adding more devices to the storage pool.
- ZFS quotas are an easy way to manage home directory space.
- Use ZFS property inheritance to apply properties to many file systems.

[edit] ZFS Mail/News Server

[edit] ZFS Software Development Server

[edit] ZFS Backup / Restore Recommendations

- Use ZFS snapshots as a quick and easy way to backup user home directories. For example, the following syntax creates recursive snapshots of all home directories in the tank/home file system.

```
# zfs snapshot -r tank/home@monday
```

- You can use the zfs send and zfs receive commands to archive snapshots to more permanent storage.
- You can create an incremental snapshot stream (see "zfs send -i" syntax)
- However, zfs send and receive commands are not enterprise-backup solutions.
- The Sun StorEdge Enterprise Backup Software (Legato Networker 7.3.2) product can fully backup and restore ZFS files including ACLs.
- The Veritas NetBackup product can be used to back up ZFS files, and this configuration is supported. However, this product does not currently support backing up or restoring NFSv4-style ACL information from ZFS files. Traditional permission bits and other file attributes are correctly backed up and restored.
- ZFS snapshots are accessible in the .zfs directories of the file system that was snapshot. Configure your backup product to skip these directories.

For the latest information about enterprise-backup solution issues with ZFS, see the Solaris 10 6/06 release notes.

[edit] ZFS and Database Recommendations

For information about on-going ZFS and database performance testing, see zfs_and_databases

- General remarks
  - If the database uses a fixed disk block or record size for I/O, set the ZFS *recordsize* property to match. You can do this on a per-file system basis, even though multiple file systems can share a single pool.
  - ZFS checksums every block stored on disk. This alleviates the need for the database layer to checksum data an additional time. If checksums are computed by ZFS instead of at the database layer, any discrepancy can be caught and fixed before the data is returned to the application.
  - ZFS performance on databases is a very fast moving target. Keeping up-to-date with Solaris releases is very important.
  - As of November 2006, ZFS does 2 things that can impact database performance:
    - ZFS does some low-level prefetches of up to 64K for each input block, which can cause saturation of storage channels.

        - ZFS issues up to 35 concurrent I/Os to each top-level device and this can lead to inflated service time in a latency sensitive context.
    - Using 8K of prefetches and between 5 and 10 concurrent I/O was shown to help some database loads. For help tuning these values see ztune The need for such tuning is expected to go away in future releases.

- Oracle Considerations

1. Match the ZFS recordsize to the Oracle db_block_size.
2. Use the default 128k record size for Oracle logs.
3. If possible, use a separate storage pool for the Oracle logs. This is especially true if your workload has a high-write component to it.

- Postgres
- MySql

[edit] ZFS and Complex Storage Considerations

- Certain storage subsystems stage data through persistent memory devices, such as NVRAM on a storage array, allowing them to respond to writes with very low latency. These memory devices are commonly considered as stable storage, in the sense that they are likely to survive a power outage and other types of breakdown. At critical times, ZFS is unaware of the persistent nature of storage memory, and asks for that memory to be flushed to disk. If indeed the memory devices are considered stable, the storage system should be configured to ignore those requests from ZFS.

[edit] Driver issues

- SATA Andy Bowers

[edit] ZFS Management Tools / Observability

[edit] Virtualization Considerations

[edit] ZFS and Virtual Tape Libraries (VTLs)

VTL solutions are hardware and software combinations that are used to emulate tapes, tape drives, and tape libraries. VTLs are used in backup/archive systems, with a focus on reducing hardware and software costs.

VTLs are big disk space eaters and we believe ZFS will allow them to more efficiently and securely manage the massive, online disk space.

- Falconstor VTL - This VTL requires the disk block device to work. It doesn't use an external file system. Therefore, it is impossible to use a combination of ZFS and Falconstor's VTL, at the time of this writing.
- Copan VTL - Same as above (Copan uses Falconstor VTL).

- NetVault from BakBone - This backup solution includes a VTL feature that has been tested on Thumper running ZFS.

[edit] ZFS and VMware

- Joost

[edit] ZFS Performance Considerations

See the following sections for basic system, memory, pool, and replication recommendations:

- #ZFS_Storage_Pools_Recommendations
- #Should_I_Configure_a_RAID-Z.2C_RAID-Z2.2C_or_a_Mirrored_Storage_Pool.3F
- Roch/Jim/Neel

[edit] ZFS and Application Considerations

[edit] ZFS and NFS Server Performance

ZFS is deployed over NFS in many different places with no reports of obvious deficiency. Many have reported disappointing performance, but those scenarios more typically relate to comparing ZFS-over-NFS performance with local file system performance. It is well known that serving NFS leads to significant slowdown compared to local or directly-attached file systems, especially for workloads that have low thread parallelism. A dangerous way to create better ZFS-over-NFS performance at the expense of data integrity is to set the kernel variable, zil_disable. Setting this parameter is not recommended.

See also #ZFS_and_Database_Recommendations.

For more detailed information about ZFS-over-NFS performance, see zfs and nfs

- Web Server
- Streaming workloads

[edit] Misconceptions about file systems and their behaviour (Roch?)

[edit] DTrace profile to classify the application type

[edit] Performance Dynamics

[edit] Tuning and Policy Settings

[edit] ZFS Overhead Considerations

- Compression
- Checksum - checksumming has been measured to consume very roughly 1 Ghz of CPU to checksum 500MB/sec of data.
- RAID-Z
- As of the Solaris 10 11/06 release, checksum, compression, and RAID-Z parity computation all occur in the context of the thread synching pool data. Under load, that thread may be a performance choke point. It is expected that in future releases all these computations will be done in concurrent threads.