ORACLE

SOLARIS

An Oracle Technical White Paper
January 2014

Configuring Oracle Solaris ZFS for an Oracle Database

ORACLE

Introduction

This document provides planning information and step-by-step instructions for configuring and maintaining Oracle Solaris ZFS file systems for an Oracle database.

A large number of enterprise customers today run an Oracle relational database on top of storage arrays with different sophistication levels and many customers would like to take advantage of the ZFS file system. Using the Oracle Solaris ZFS file system can reduce the cost and complexity of managing data and storage devices for hosting your Oracle database. Since storage product capabilities often overlap ZFS features, a wide range of options are offered for deployment. This document provides guidance on how to economically deploy an Oracle database on ZFS to maximize data integrity and performance. We start with a brief planning section and follow with step-by-step instructions for configuring and maintaining ZFS file systems for an Oracle database.

Planning for an Oracle Database on ZFS Configuration

The following sections provide specific planning information that is needed before you configure ZFS file systems for an Oracle database.

- Oracle Solaris release information - Use the latest Oracle Solaris 11 release or the latest Oracle Solaris 10 release, such as Oracle Solaris 10 1/13 and apply the latest kernel patch, if possible. The minimum Solaris release required is Oracle Solaris 10 10/09.
- Storage array considerations - Confirm with your array vendor that the disk array is not flushing its non-volatile cache after write cache flush requests issued by ZFS.

ZFS Storage Pool Considerations

If you have traditional hardware RAID protection and you are comfortable with that level of protection, then use it. Otherwise, we recommend using ZFS redundancy for the highest level of protection, such as a mirrored storage pool. With a mirrored storage pool, ZFS can automatically detect and repair data storage corruption. We do not recommend RAIDZ redundancy when performance and IOPS are the main criteria for selecting a storage configuration for your database. For more information about creating redundant ZFS storage pools, see the *Oracle Solaris ZFS Administration Guide*.

To ready your database for a ZFS environment, review the following steps:

- Create LUNs or requests LUNs from your SAN storage to create ZFS storage pools
- Create ZFS file systems within the pools
- On each file system, set properties for optimal performance

Two ZFS storage pool reference configurations are covered in this paper:

- A single ZFS pool with specialized file systems for each database.
- Two ZFS pools are used in an optimized way with one pool specifically for the redo log and one pool for the rest of the data. Isolating the redo log onto it's own pool helps the system deliver more consistent latency. The extra administrative burden of creating more pools should only be required in highly critical performance environment.

Single-Pool Model

In the single pool model, most LUNs are striped together in a pool and one or more LUNs are configured as ZFS log devices. The size of the pool or the sum of the LUN sizes configured in the pool should be made to accommodate all the files, while leaving room for growth and free space. Leaving a certain amount of free space of 10-20% usually helps to achieve the highest level of performance. In all cases, always operate with at least 5% of free space. In addition to the data LUNs, the pool is configured with a pool log device that is used to accelerate synchronous operations. This pool log device should be minimally sized according to the rate of data writes going to the pool. The size of the pool log device must cover 15 seconds of peak modification rate along with a security factor

of 2. There is no downside to over-sizing the pool log device, but the penalty for under-sizing is important. For example, to accommodate 10 MB/sec of redo log modification rhythm, a minimum size of 300 MB should be used for the log LUN.[1]

Optimized Two-Pool Model

In this model, we create 2 pools, one for holding most of the database (`dbpool`) and one pool specifically for the redo log. The `dbpool` is setup as a stripe (or mirror) of one of more LUNs, but does not require a log device. The redo log pool consists of one of more LUNs sized to cover only redo log files and has a pool log device sized to cover 15 seconds of redo log peak activity along with a security factor of 2. So, to cover 10 MB/sec of peak redo log file activity, then a 300 MB LUN is necessary as a pool log device. The two-pool model should only be considered when performance is highly dependent on small and steady redo log latency.

Sizing ZFS Storage Pool LUNS

The preferred way to deploy either pool model is to request LUNs that are sized roughly the same as the underlying disk technology as well as a properly sized LUN for the pool log device. If the storage procurement model involves a large number of small LUNs, ZFS accommodates this model quite well. If the model imposes very large LUNs (or even a single one), you can use the `format` utility to carve out a properly sized slice for the pool log device, leaving the rest of the LUN as a slice for the main storage. When creating slices with the `format` utility, be aware that alignment of the slices can have important consequences on performance of the storage array. Create slices that align with the internal block size used in the array. For simplicity, aligning slices on a 1 MB boundary should work in most circumstances.

Some storage provisioning models involve dynamic (on-demand) allocation of space in the storage backend. When used in conjunction with ZFS, it should be noted that the amount of space required to store ZFS pool data is greater than that amount of data consumed at the ZFS level due to the copy-on-write nature of ZFS.

Managing Storage Space

In an environment with a high rate of data update (or churn), it is advisable to maintain a certain amount of free space. ZFS is a copy-on-write file system and relocates all writes to free disk space. Keeping a certain amount of free blocks allows ZFS to easily find space in big chunks, allows ZFS to aggregate writes and reduce the write IOPS demand on hard disks. Streaming aggregated writes can be 50-100 times less demanding on hard drives than doing small random writes. Therefore, the incentive for keeping free space is high, especially in a high churn environment, such as an active OLTP database.

---

[1] ZFS enforces a minimum of 64 MB for a separate log device.

- For smaller pools up to 1TB, we advise an initial free space target value of 20%.
- For pools larger than 5TB, the amount of free space can be relaxed to 10% but never less than 5%.

In a data warehouse environment, data is inserted in the DB and normally stays present for long periods with little churn. This use case accommodates a much higher level of occupancy on the storage pool.

The preceding numbers are given as initial guidelines, your workload experience is the final judge of what is acceptable in your environment.

As a way to enforce this free space, create a special free, unused file system with a ZFS file system reservation. This reservation prevents other file systems in the system from growing beyond the desired point. As you reach the limit, ZFS does not accept the creation of new reservations. As that happens, you may decide to reduce the free, unused reservation, reclaim reservations from other file systems, redirect projects to other storage, or simply add storage space.

Using Oracle's File Auto Extent

Regarding space consumption, you can use the Oracle database file `auto extent` feature to instruct the Oracle database not to preallocate the entire data file size, but grow it in controlled increments.[2] Using this method leaves plenty of free space to ZFS.

In addition to the above reservation, data is located in a smaller range of hard disk addresses leading to greater efficiency. When using this feature, you specify an upper limit to the size of each data file at the Oracle configuration level. At the ZFS level, it is also possible to set up a corresponding ZFS file system reservation, which *guarantees* never running out of space. This space can be reclaimed at any time if it turns out the sum of the data file maximum size was over provisioned.

So while this feature allows you to run with more free space in the pool and brings some flexibility in operations, it does come with a downside, which is that is not always compatible with every database application. When the database needs to run through an extension, it causes some disruption to operations, but a small next auto extent, typically of 1MB allows a very short allocation time and limits the impact of that disruption for the application. While a ZFS reservation helps you deal with space concerns, your final experience is the key in determining if the feature is desirable in your environment.

For more information about using ZFS quotas and reservation, see the *Oracle Solaris ZFS Administration Guide*.

ZFS File System Considerations

Match ZFS Record Size to Oracle Database Block size

[2] For more information, see http://www.dba-oracle.com/t_alter_autoextend_on.htm.

The number one rule for setting up an Oracle database on ZFS is to set ZFS `recordsize` equal to the database block size for the file systems that contain the Oracle data files.

```
recordsize = db_block_size
```

Important Note: Keep in mind that modifying the ZFS file system `recordsize` parameter affects only the files created after the change. To change the size that is used by an existing data file, you must first change the `recordsize` property of the file system used to store the file, and then copy the files back into the file system.

For additional performance in the case of a large database, you may use different `db_block_size` values for different database components. In these cases, you should use one ZFS file system per `db_block_size` and match file system's record size to `db_block_size`.

Improving Database Writing and Caching Performance

On systems running current Oracle Solaris 10 or 11 releases, you can use the following properties to improve performance:

- Set the `logbias` property per file system
- Use the `primarycache` property to control what is stored in main memory cache

Tuning Synchronous Write Activity

ZFS manages synchronous writes internally with the ZFS intent log (ZIL). By default, synchronous writes are stored in the ZIL at low latency and shortly thereafter (within approximately 5 seconds), ZFS does a bulk update of the pool state by committing a transaction group.[3]

Another way exists for the ZIL to handle synchronous write activity, which is of slightly higher latency, but also involves almost half as much traffic on the channel to the storage array. This mode is triggered by the `logbias=throughput` value. This mode is recommended for file systems holding Oracle data files since writes to those files are highly threaded background operations from the database perspective, which does not warrant the utmost latency requirements. Using `logbias=throughput` for data files frees up the storage resource and as a consequence, allows the redolog to be handled with lower latency. The redo log itself should generally not be tuned for this property.

Setting ZFS File System Properties

[3] Note that data for a transaction group is kept in memory and not read from the ZIL, which is only used in case of a system failure.

The following table identifies the file system type and the matching record size value, `logbias` value, `primarycache` value, and compression value for a generic OLTP database (`db_block_size` is commonly set to 8 KB).

TABLE 1. ZFS FILE SYSTEM PROPERTIES

FILE SYSTEM	RECORD SIZE	LOGBIAS VALUE	PRIMARYCACHE VALUE	COMPRESSION
Tables	`db_block_size`	`throughput`	`all` (data and metadata)	off (default)
Redo	128 KB or 1 MB (Solaris 11.1)	`latency` (default) [4]	`all` (data and metadata)	off (default)
Index	`db_block_size`	`throughput`	`all` (data and metadata)	off (default)
Undo	128 KB or 1 MB (Solaris 11.1)	`throughput`	`all` (data and metadata)	off (default)
Temp	128 KB or 1 MB (Solaris 11.1)	`throughput`	`all` (data and metadata)	off (default)
Archive	128 KB or 1 MB (Solaris 11.1)	`throughput`	`all` (data and metadata)	On

Since the initial release of this paper, the `primarycache` value for the undo and archive file systems have been changed from caching just metadata to caching data and metadata, which is the default. The reason is that setting `primarycache` of metadata disables ZFS level prefetching and the risk of a performance impact does not sufficiently balance the advantages of the reduced cache footprint. Only in special cases when undo and archive are very large, should you consider setting a `primarycache=metadata` value for them.

Note: Data for a transaction group is kept in memory and is not read from the ZIL, which is only used in case of a system failure.

Since Oracle Solaris 11.1, the maximum `recordsize` supported by ZFS has grown from 128 KB to 1 MB. For files used in nearly exclusive streaming mode, replacing 128 KB with 1 MB in the above table is slightly beneficial.

ZFS File System Property Settings for Data Warehouse Workloads

Some databases hold a mix of volatile and historical data. The distinction is that historical data is subject to a reduced rate of modification compared to the volatile nature of evolving OLTP data. The historical data has effectively become frozen. Such data can still be kept in the main database for the purpose of reporting and trend analysis.

[4] A special case exists when redo traffic saturates the channel bandwidth to the storage. In that special case, setting redo log file shares to `logbias=throughput` has been observed to help, but this rarely happens.

When the amount of historical data is large, consider storing it on less expensive drives, such as larger disks with smaller RPM[5]. Such static data that is read in database scan operations should benefit from being stored on a large ZFS `recordsize`, and if necessary compressed by ZFS, to reduce the on-disk footprint. Separately, index data might benefit from using a larger `db_block_size` (16 or 32 KB)[6]. For historical data, we also advocate setting the `logbias` file system property to throughput. Note also, that contrasted to volatile data, file systems holding historical data on ZFS interact well with storage based dynamic provisioning, the amount of data stored after compression is closer to the amount provisioned by the storage. The data warehouse workloads usually have creation and modification during integration phase, but have nearly no modification on integrated data. This modification pattern allows data warehouse databases to push up the fill percentage of a ZFS pool to the 90-95% range without adverse effect.

TABLE 2. ZFS FILE SYSTEM PROPERTY SETTINGS FOR DATA WAREHOUSE WORKLOADS

FILE SYSTEM	RECORD SIZE	LOGBIAS VALUE	PRIMARYCACHE VALUE	COMPRESSION
Tables data	128 KB or 1 MB (Solaris 11.1)	`throughput`	`all` (data and metadata)	on
Index data	`db_block_size`	`throughput`	`all` (data and metadata)	default

Oracle Large Object (LOB) Considerations

The Oracle LOB, CLOB, NCLOB or BLOB data types are somewhat less common, but when present can benefit from special handling. Because this type of data leads to large read and write I/Os, such as a full table scan, the set up of these objects can be done by using the same guidelines as the ones given for data warehouses.

For these applications, we can use the Oracle SQL syntax to set apart the LOB columns in specific tablespaces. The LOB are frequently large, many MB, and are split over many Oracle data blocks. Consider creating a separate file system and specify a larger record size for the LOB tablespace. [7] You may use a ZFS record size of 128 KB or even 1MB with Oracle Solaris 11.1, but we advise to use a ZFS `recordsize` no greater that the average size of individual LOBs.

TABLE 3. ZFS FILE SYSTEM PROPERTY SETTINGS FOR LOB APPLICATIONS

FILE SYSTEM	RECORD SIZE	LOGBIAS VALUE	PRIMARYCACHE VALUE	COMPRESSION
LOB	128 KB or 1 MB (Solaris 11.1)	`throughput`	`all` (data and metadata)	on

[5] Even when using a single pool, migrating the data to different file systems in the main database pool still allows you to use specific tuning described in Table 2.

[6] For more information, see Configuring Multiple Block Sizes for an Oracle Database.

[7] For more information, see Configuring Multiple Block Sizes for an Oracle Database.

Using ZFS Secondary Cache (L2ARC)

You can add LUNs as secondary cache (L2ARC) devices in a storage pool starting in Oracle Solaris 10 10/09. These devices store a cached copy of disk content for quick access and serve to increase the total random read throughput of the storage. Using a low latency and efficient $/IOPS device, such as an SSD or other flash based device, is recommended.

Setting the `secondarycache` property determines which file systems will use the secondary cache.

Using a secondary cache is beneficial for read-latency sensitive workloads, for both index and database blocks. As usual, a cache layer is most effective when cache hit rates are high, implying that the total size of cache devices should be sized according to the application's warm working set. Moreover, in an environment where performance peaks when physical disks are saturated with random reads, the caching devices offer a venue for additional read IOPS.

- Caching devices do not retain data between system reboots.
- Allow for hours of operation to gauge their effectiveness in your environment.
- Enabling the secondary cache for redo logs is not recommended.

Using ZFS Snapshots with the Oracle Database

Creating ZFS snapshots and clones of your Oracle database can reduce the time spent on data management.

A key ZFS feature is creating a snapshot of a given file system or sets of file systems. Creating a snapshot is nearly instantaneous and allows you to operate on a read-only point in time representation of a ZFS file system. A snapshot can be used to keep previous images of the database at a very modest space cost. Multiple snapshots can be generated during the day, each representing a potential recovery point. Generally, a snapshot and the file system from which it originated actually point to the same physical blocks. Therefore, the cost of taking snapshots of disk blocks is equal to only data that has been modified in the database since the last snapshot was taken.

Starting with Oracle Solaris 11.1, blocks that are referenced from either the snapshot or from the database image actually share ZFS cache buffers for a reduced memory footprint.

Taking a snapshot of a database can involve placing the database in hot-backup mode just for the instant necessary to capture the ZFS snapshot. The database can then be returned to normal operation.

Snapshots can also be used to replicate the database to a remote host by using the `zfs send` and `zfs receive` commands. When creating a series of snapshots, only data that was updated between the two snapshots needs to be transferred over the network. If done regularly, the *incremental* data to be transferred is either in the cache or on the disk with blocks that have physical proximity to one another for more efficient operation.

For backups, either the snapshot captured on the primary host or the replicated snapshot on a remote host can serve as the source of a backup. The remote host replicates the full snapshot while the primary host transfers the incremental snapshot changes. It is also possible to clone and mount a copy of the database on a replicated site and share the largest part of the replicated volume. The replicated

database is some hours, days, behind the production database and can be used for business intelligence queries or as a pre-production environment.

Sending and receiving ZFS snapshot streams is an alternative to the commonly used storage based replication. With storage based replication of an ZFS storage pool, be aware you **must** configure all LUNs belonging to a ZFS storage pool to be in the *same storage coherency group* in the SAN array. Failure to do so prevents the replicated LUNs from being imported by ZFS on the replication site. On the other hand, a successful receive operation is immediately accessible on the replication site. In addition, during reception, previously replicated file systems snapshots are still accessible and ready for use. For more information, see the *Oracle Solaris ZFS Administration Guide*.

ZFS Cloning for Test and Development

While snapshots are read-only file systems, it is also possible to create a clone of any ZFS snapshot. A *clone* is a full read/write accessible fork of the original database. Therefore, by using either the primary or a replicated storage system, you can cheaply create hundreds of copies of your *up-to-date* database to run test and development workloads under real data conditions. Again, starting with Oracle Solaris 11.1, clones, snapshots, and the original file systems running in the same storage, share cached blocks between each other.

For more information about sending and receiving snapshots, see the *Oracle Solaris ZFS Administration Guide*.

Additional System Configuration

Managing Storage Array I/O Queues

The `zfs_vdev_max_pending` parameter defines the maximum number of I/Os that ZFS can send to any storage pool device.[8] ZFS aggregates I/Os that are physically adjacent. Therefore, one ZFS I/O can represent a number of database I/O operations. In a legacy storage environment, the `ssd_max_throttle` and `sd_max_throttle` parameters define the maximum number of concurrent I/Os that the driver can send to the storage. By setting the `zfs_vdev_max_pending` default value equal to the value of the `[s]sd_max_throttle` parameter, we prevent ZFS from queuing I/O beyond what the SD layer will accept.

If you have `ssd:ssd_max_throttle` or `sd:sd_max_throttle` in the `/etc/system` file in your existing environment, then set `zfs:zfs_vdev_max_pending` to the same value.

For example, if the storage array administrator asked for the following setting:

[8] ZFS aggregates I/Os that are physically adjacent. Therefore, one ZFS I/O can represent a number of database I/O operations.

```
set ssd:ssd_max_throttle=20
```

Then, also set these parameters as follows:

```
set ssd:ssd_max_throttle=20
set zfs:zfs_vdev_max_pending=20
```

Setting these parameters allow ZFS to control each LUN queue. This means that the total number of pending I/Os in the storage can grow as follows:

```
number of LUNs * ZFS_VDEV_MAX_PENDING
```

Therefore, adding LUNs to a storage pool is a way to increase the total I/O count that a pool may have pending in a storage array.

Memory and Swap Considerations

The ZFS cache grows to consume most of unused memory and shrinks when applications generate memory demand. But while ZFS can shrink its cache quickly, it does take time for the free memory list to be restored. When applications demand memory, usually at application startup, memory is consumed faster than the pace with which free memory is generated. Have enough swap configured to allow applications to start up smoothly. The swap space acts as a buffer zone for the application's allocation of pages for itself while the Oracle Solaris OS is converting freed buffers into free pages. Having sufficient swap space prevents standard application from aborting due to `ENOMEM` errors.

Configuring an amount of swap equivalent to *all* of system memory is always enough (in fact excessively so) for this purpose. This swap space is not expected to be used, but is needed as a reservation area. Using a much smaller amount of swap is normally sufficient to cover memory demand bursts from regular applications.

For information about increasing swap space, see the *Oracle Solaris ZFS Administration Guide*.

ISM or DISM Shared Memory Considerations

Applications which use ISM or DISM shared memory (`shmget(2)`, `shmat(2)`) should be set up to retry when getting errors. If the amount requested is very large, then the timeout for such operations needs to be scaled appropriately. It takes minutes to convert hundreds of GB of freed buffers into free memory.

Another way to deal with bursts of memory demand is to cap the maximum ZFS cache by setting `zfs:zfs_arc_max` in `/etc/system`. This is especially appropriate if the memory consumption of applications is known in advance. ZFS is then instructed to never grow beyond a desired size. See Tuning the ZFS ARC (`zfs_arc_max`).

Another benefit of limiting the ZFS cache is to preserve the set of large memory pages that can then be used by the database's ISM/DISM segments. When those pages are available, some high intensity

databases are observed to perform better. Note that the preferred way to insure availability of large pages for a critical database is to startup the instance shortly after boot.

Here are few considerations of trying to balance memory use between the Oracle database and ZFS.

- A database SGA is a great place to cache data. When a system is devoted to running a single database instance, it usually makes sense to configure a maximum amount of memory to the SGA. However, many large scale systems run a number of independent database and it is not practical to maximize the SGA for each one. In these cases, it makes sense to size each of the SGA a minimum, and then allow ZFS to do the unified system caching in front of the disk subsystem.
- In cases where the SGA is maximized and is the primary location of caching information for a database, it is possible to limit the ZFS cache size. When the cache size is set very low, it is possible to adjust the cache by setting `primarycache=metadata` on some ZFS file systems. This file system property reclaims more I/O on the storage. However, file systems with `primarycache` set to `metadata` negate the benefit of prefetching, which can be a problem during any long read as full scan, backup, restore (`archivelog`). Therefore, we recommend undoing this setting during a restore operation because it relies on prefetching for performance. Also consider that a file system holding an index file normally benefits significantly from file system level caching. Generally, those file systems should not have the `primarycache` property set.
- Set `primarycache=metadata` on the `archivelog` file system, but it can slow down the copy of the `archivelog` file system to external storage. So, set `primarycache=all` on the `archivelog` file system for a restore operation.

Configuring Multiple Block Sizes for an Oracle Database

You can define a specific tablespace for the LOB using a 32 KB block size tablespace, which also requires some 32 KB cache in the SGA (System Global Area), as we do for a data warehouse environment, by using syntax similar to the following:

```
zfs create -o recordsize=8k,mountpoint=/my_db_path/data_8k dbpool/data_8k
zfs create -o recordsize=32k,mountpoint=/my_db_path/data_32k dbpool/data_32k
zfs create -o recordsize=128k,mountpoint=/my_db_path/data_128k dbpool/data_128k
```

In `init.ora`, you would find numbers similar to the following example:

```
DB_BLOCK_SIZE=8192
SGA_TARGET=20G
DB_32K_CACHE_SIZE=400M
```

In SQL, you would find numbers as follows:

```
CREATE TABLESPACE data_32k_example
BLOCKSIZE 32K
DATAFILE '/my_db_path/data_32k/image_lob01.dbf' SIZE 1G
AUTOEXTEND ON NEXT 10M MAXSIZE 20G;
```

Tuning the ZFS ARC (`zfs_arc_max`)

The `zfs_arc_max` parameter is in bytes and accepts decimal or hexadecimal values. It is highly recommended to put a date and justification as comments alongside the tuning.

The following text shows how to set this parameter to 16 GB, as an example:

```
* Jan 1 2013; JB.
* Cap the ARC to 16GB reserving 500GB for applications
* set zfs:zfs_arc_max=0x400000000
```

Add this text with appropriate values to `/etc/system` and then reboot the system.

Display the current memory size in bytes that is used as ZFS cache:

```
# kstat zfs::arcstats:size
```

Monitor ZFS cache sizes with the preceding command and re-adjust the `zfs_arc_max` parameter when needed. If the `vmstat` command shows high level of free memory, consider increasing the value of `zfs_arc_max` to allow for better caching.

Note: For ZFS 8 KB records, metadata consumes about 1.5% of the active portion of the database. Reducing the ZFS cache size to a level where metadata is not cached will lead to an inflation of disk IOPS normally detrimental to acceptable performance.

How to Configure ZFS File Systems for an Oracle Database

After you have reviewed the preceding planning section, use the steps below to configure ZFS for an Oracle database.

1. **Verify your Oracle Solaris release information.**

   Confirm that you are running the Oracle Solaris 11 release or the latest Oracle Solaris 10 release. The minimum Oracle Solaris 10 release is Oracle Solaris 10 10/09.

```
# cat /etc/release
           Oracle Solaris 10 1/13 s10s_u11wos_24a SPARC
  Copyright (c) 1983, 2013, Oracle and/or its affiliates. All rights reserved.
                             Assembled 17 January 2013

# cat /etc/release
                           Oracle Solaris 11.1 SPARC
  Copyright (c) 1983, 2012, Oracle and/or its affiliates.  All rights reserved.
                          Assembled 19 September 2012
```

2. **Create a storage pool for data files for tables, index, undo and temp data.**

   a. Create a storage pool for the database files.

      ```
      # zpool create dbpool c0t5000C500335DC60Fd0
      ```

      Consider creating a mirrored storage pool to provide a higher level of data redundancy. For example:

```
# zpool create dbpool mirror c0t5000C500335DC60Fd0 c0t5000C500335F907Fd0 mirror
c0t5000C500335BD117d0 c0t5000C500335F95E3d0
# zpool status dbpool
pool: dbpool
state: ONLINE
scan: none requested
config:

        NAME                       STATE     READ WRITE CKSUM
        dbpool                     ONLINE       0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335DC60Fd0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0

errors: No known data errors
```

b. Create a storage pool for redo logs with a separate log device.

In this example, the disk volume is partitioned into two slices, a small slice, s0, in the 64 to 150 MB range, for the separate log device. The s1 slice contains the remaining disk space for the redo log.

```
# zpool create redopool c0t5000C500335FC3E7d0s1 log c0t5000C500335FC3E7d0s0
# zpool status redopool
pool: redopool
state: ONLINE
scan: none requested
config:

        NAME                       STATE     READ WRITE CKSUM
        redopool                   ONLINE       0     0     0
          c0t5000C500335FC3E7d0s1  ONLINE       0     0     0
        logs
          c0t5000C500335FC3E7d0s0  ONLINE       0     0     0

errors: No known data errors
```

For databases with high redo log activity, such as a typical OLTP database with many commits, use a separate log device LUN.

c. Create a storage pool for the archivelog.

A system's internal disk can handle this type of load. The `archivelog` file system can also be a file system in the `dbpool`.

```
# zpool create archivepool c3t0d0
```

d. Create the ZFS file systems and set the specific file system properties by using the following guidelines:

Create separate file systems for redo, archive, undo, and temp database components by using the largest record size available of 128 KB or 1M. The general rule is to set the file system `recordsize = db_block_size` for the file systems that contains Oracle data files. For table data and index components, create a file system with an 8 KB record size. For more information about ZFS file system properties, see the *Oracle Solaris ZFS Administration Guide*.

i. Create file systems for the table data files and index data files with an 8 KB `recordsize`. Use the default value for `primarycache`.

```
# zfs create -o recordsize=8k -o mountpoint=/my_db_path/data dbpool/data
# zfs set logbias=throughput dbpool/data
# zfs get primarycache,recordsize,logbias dbpool/data
NAME            PROPERTY      VALUE         SOURCE
dbpool/data     primarycache  all           default
dbpool/data     recordsize    8K            local
dbpool/data     logbias       throughput    local


# zfs create -o recordsize=8k -o mountpoint=/my_db_path/index dbpool/index
# zfs set logbias=throughput dbpool/index
# zfs get primarycache,recordsize,logbias dbpool/index
NAME            PROPERTY      VALUE         SOURCE
dbpool/index    primarycache  all           default
dbpool/index    recordsize    8K            local
dbpool/index    logbias       throughput    local
```

ii. Create file systems for temporary and undo table spaces, using the default `recordsize` and `primarycache` values.

```
# zfs create -o mountpoint=/my_db_path/temp dbpool/temp
# zfs set logbias=throughput dbpool/temp
# zfs create -o mountpoint=/my_db_path/undo dbpool/undo
# zfs set logbias=throughput dbpool/undo
```

Starting in Oracle Solaris 11.1, adjust `recordsize`:

```
# zfs create -o recordsize=1m -o mountpoint=/my_db_path/temp dbpool/temp
# zfs set logbias=throughput dbpool/temp
# zfs create -o recordsize=1m -o mountpoint=/my_db_path/undo dbpool/undo
# zfs set logbias=throughput dbpool/undo
```

iii. Create a file system for redo logs in the redo pool. Use default file system values for `primarycache` and largest available `recordsize`.

```
# zfs create -o mountpoint=/my_db_path/redo redopool/redo
# zfs set logbias=latency redopool/redo
```

Starting in Oracle Solaris 11.1, adjust `recordsize`:

```
# zfs set recordsize=1M redopool/redo
```

iv. Create a file system for the `archivelog` files in the archive pool, enabling compression, use the default value for `recordsize`. In case of low ZFS cache, adjust the `primarycache` to `metadata`, then for Oracle Solaris 11.1 with a large enough ZFS cache.

```
# zfs create -o compression=on -o mountpoint=/my_db_admin_path/archive
archivepool/archive
# zfs set primarycache=metadata archivepool/archive
# zfs get primarycache,recordsize,compressratio,compression,available,
used,quota archivepool/archive
NAME                PROPERTY       VALUE         SOURCE
archivepool/archive  primarycache   metadata      local
archivepool/archive  recordsize     128K          default
archivepool/archive  compressratio  1.32x         -
archivepool/archive  compression    on            local
archivepool/archive  available      40.0G         -
```

```
archivepool/archive  used           10.0G         -
archivepool/archive  quota          50G           local
```

Starting in Oracle Solaris 11.1:

```
# zfs create -o compression=on –o recordsize=1M -o mountpoint=/my_db_admin_path/archive
archivepool/archive
# zfs get primarycache,recordsize,compressratio,compression,available,
used,quota archivepool/archive
NAME                 PROPERTY       VALUE         SOURCE
archivepool/archive  primarycache   all           local
archivepool/archive  recordsize     1M            local
archivepool/archive  compressratio  1.32x         -
archivepool/archive  compression    on            local
archivepool/archive  available      40.0G         -
archivepool/archive  used           10.0G         -
archivepool/archive  quota          50G           local
```

e. Consider the following steps to help maintain disk space and to create snapshots.

i. Set a reservation on the main database file system to reserve the required database file system space.

```
# zfs set reservation=200gb dbpool/data
```

ii. Set a reservation on a dummy file system to reserve 10-20% of pool space to maintain pool performance.

```
# zfs set reservation=20gb dbpool/freespace
```

iii. Consider setting a quota on the archive file system if other data is also archived in the archive pool.

```
# zfs set quota=40g archivepool/archive
```

iv. Create a recursive snapshot of database file systems and send to a backup pool.

```
# zfs snapshot -r dbpool@snap1
# zfs send -vp dbpool@snap1 | zfs receive -e bkpool
sending from @ to bkpool@snap1
```

For More Information

For more information on Oracle Solaris 11 and ZFS technology, see the following references.

TABLE 4. REFERENCES FOR MORE INFORMATION ABOUT ORACLE SOLARIS 11 AND ZFS TECHNOLOGY

Oracle Solaris 11	http://www.oracle.com/solaris
Oracle Solaris 11 technical content on OTN	http://www.oracle.com/technetwork/server-storage/solaris11/overview/index.html
Oracle Solaris ZFS technology	http://www.oracle.com/technetwork/server-storage/solaris11/technologies/zfs-338092.html
Oracle Solaris 11 documentation Oracle Solaris ZFS Administration Guide	http://www.oracle.com/technetwork/server-storage/solaris11/documentation/index.html http://docs.oracle.com/cd/E26502_01/html/E29007index.html
Oracle Solaris 11 Articles	http://www.oracle.com/technetwork/server-storage/solaris11/documentation/how-to-517481.html

Configuring Oracle Solaris ZFS for an Oracle Database
January 2014
Authors: Cindy Swearingen, Roch Bourbonnais, Alain Chéreau

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together