

Oracle File System Integration and Performance

Richard Mc Dougall, Sriram Gummuluru

January 2001

In this paper, we discuss file system features and performance tuning. We explore the performance overheads and configuration options of Oracle with the Solaris UFS and Veritas file systems, and general tuning guidelines.

1.0 Introduction

The Solaris Operating Environment provides a feature rich platform for the Oracle database. A combination volume management and file system products provide the features which allow us to configure for the require levels of performance, availability and manageability. We begin by discussing the layers of the Solaris storage stack, and then discuss Oracle performance in more detail.

2.0 Storage Management

A database administrator typically strives to achieve service levels set down by users and customers. These service levels are often contain goals for performance and availability, and one of the more significant challenges we face is how to best configure the systems's persistent storage to meet these goals. In the Solaris operating environment, storage is managed by volume managers and file systems - the volume manager manages the lower level aggregation of disks into virtual volumes, and the file system provides us with our familiar Unix administration model based on files.

A vast range of options and products are available, which leaves us with having to make decisions about which options to use when. In this paper, we focus on the specific storage requirements of Oracle in a Solaris environment, and

provide some answers to the questions posed when making these decisions.

2.1 The Volume Manager

The volume manager provides a mechanism of aggregating physical storage into virtual pools. In an Oracle environment, we typically use the volume manager to:

- Combine physical disk drives into fewer, larger volumes. The volumes are then used as the storage administration unit for table spaces within the database. Volumes may be configured with RAID levels such as striping (RAID-0), concatenation, mirroring (RAID-1) and parity (RAID-5).
- Provide transparent access to the storage via multiple redundant paths. Modern storage interconnects, such as fibre-channel allow multiple physical paths to the storage, which provides the ability for load sharing across the interconnects, and to provide continuous access to storage in the event that one of the paths fail. The volume manager presents a single device to the application, and transparently manages the mapping of I/Os across the underlying paths.
- Manage the ownership of storage devices between systems when part of a cluster. In a SunCluster environment, the volume manager protects storage so that only an active node can perform I/O's to it's storage, and then in the

event of a fail over, the volume manager coordinates hand-over of the storage to the new node taking over the workload. In a parallel database environment such as Oracle OPS, the volume manager provides concurrent access to storage, and provides a unified system-wide view of the disks and volumes.

The three most common forms of volume management in a Solaris environment are:

- Solaris Volume Manager (also known as DiskSuite). The Solaris volume manager is shipped with Solaris, and has recently been integrated into Solaris 8. It provides basic volume management capabilities, but at the time of writing does not provide the ability to manage multi-pathed storage.
- Veritas Volume Manager. The Veritas volume manager is currently the most popular volume management mechanism for Solaris systems. It provides features which address the above requirements.
- Hardware RAID Volume Managers, such as that found in the A3500 and T3 storage devices. Hardware RAID devices implement volume management within their controllers, and present volumes as devices to the host.

Figure 1. contrasts software and hardware volume management.

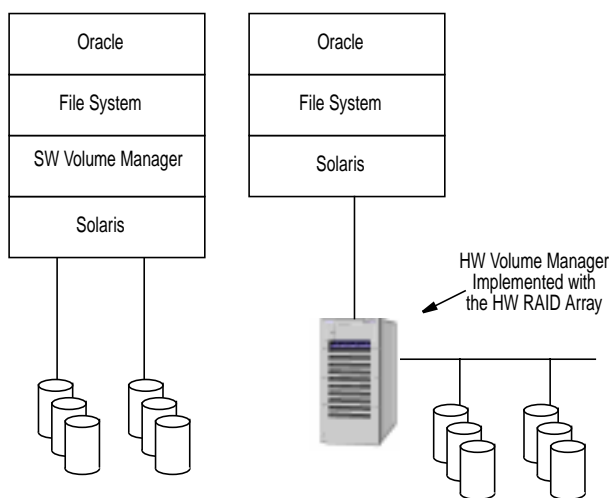


Figure 1. Typical Solaris Storage Stacks

Software volume management allows mirroring (RAID-1) to be configured at the host level, providing protection for practically all single points of storage failure, including disk, controller and interconnect failure. Software RAID-5 implementations however, come with a large write performance penalty, due to the inherent read/modify/write overhead, and should be avoided if possible.

Hardware volume management unloads the host of the CPU overhead of managing RAID, and provides efficient RAID-5 implementations.

2.1.1 Simplifying Storage Management

We may use striping and/or RAID-5 to aggregate storage, and eliminate the need for individual disk management. Typically, the ideal volume size is a tradeoff between the ease of manageability (as large as possible), and the availability risks (minimizing the mean time to recover an entire volume or file system).

2.2 The File System

The file system manages storage allocation at the system wide level, by dividing volumes into files. There may be multiple file systems within a system, each which may contain many files. The file system presents the database administrator with a flexible administration model, allowing a wide range of on-line capabilities. It does however often come with an associated performance penalty, which we will explain and quantify further.

Ideally, we would always want to use a file system for its administration model, but the raw disk path (bypassing the file system) is often used to overcome some of the inherent file system performance overheads. Third party file systems provide features to overcome some of the traditional file system overheads, while retaining the file system administration model.

With these options in mind, what is the best strategy for databases on file systems? How do we pick the correct mix of database, operating system and third party options for a given system?

2.2.1 Administration Capabilities

The file system provides the most flexible administration model for database administrators. Com-

bined with the volume manager, it allows us to manage storage for each table space within the database. The volume manager manages the coarse grained aggregation of storage devices, and the file system allows us to subdivide that aggregated storage for use by table spaces.

The administration features of the file system can be summarized as the following:

- It allows multiple oracle tables to reside within a single volume. Oracle *tablespaces* are contained in Oracle *datafiles*, which reside within the file system. The maximum size of each datafile is limited to 2^{22} blocks. This limits the maximum size of a datafile to 8GBytes for 2KByte block database. The file system allows us to create multiple tables within a large device, obviating the need to create one raw volume for each datafile.
- Grow table space on-line. The file system allows simple growth of datafiles - simply expanding the datafile via the Oracle server manager results in growth of the underlying Unix file. With raw devices, co-ordination with the volume manager is required, typically resulting in manual volume manager operation each time a datafile is grown.
- Administration tools - mount, df, ls etc work. The regular Unix system administration tools can be used to manage the storage space. With raw disk access, these tools do not work, leading to clumsy and sometimes erroneous management of each storage volume.
- Optional use of memory for caching. The file system uses the hosts system memory for caching. In certain cases where tables are not cached by Oracle, this can be beneficial. For example, a 32 bit Oracle instance can use a maximum of 3.9GB of memory for caching within the SGA, while the file system can use all of the available system memory. This allows Oracle to use up to 60GBytes of memory for caching within a Solaris system.
- Snapshot capabilities. Some file systems support snap-shot capabilities, which allows the file system to be frozen and a backup to be performed while the database is on-line.

2.3 Availability

More so than ever, systems are required to meet ever increasing high levels of availability. Key to meeting these levels is reliability and recoverability of the underlying storage. If incorrectly configured, the file system can have a negative impact on the overall systems availability. Two key areas of attention are warranted: minimizing restart times, and using simple, well tested configurations.

Re-start times are affected by the file systems ability to recover to a known state after an unexpected outage.

The original implementations of Unix file systems did not allow fast restart after a system outage, since they left the file system in an unknown state in the event of a system crash or power outage, and often took a very long time (3+ hours for a 50-Gbyte file system) for consistency checking at boot time.

To overcome these issues, logging (sometimes referred to as journaling) was introduced to prevent the file system structure from becoming corrupted during a power outage or a system failure. The term journaling is used to describe a file system that logs changes to on-disk data in a separate sequential rolling log. The primary reason for using this procedure is to maintain an accurate picture of file system state, so that in the event of a power outage or system crash, the state of the file system is known. Then, rather than doing a lengthy scan of the entire file system (via `fsck`), we can check the file system log and correct the last few updates as necessary. A logging file system can mean the difference between mounting a heavily populated file system in 20 seconds versus 3+ hours without a log.

Starting with Solaris 7, the Solaris UFS file system has an option to enable logging. You should consider this feature mandatory for any system which is required to meet higher levels of availability. In fact, because of the small amount of file creates, deletes and size changes, there is little performance overhead for file system logging when used for Oracle table spaces. For this reason, it makes sense to enable logging with all file systems used for Oracle.

2.4 Performance

As we mentioned in the introduction, file systems are the preferred mechanism for database storage management, because of their flexible administration model. However, the file system can impose a performance overhead on the database, in certain circumstances. The raw disk path can be used to avoid these performance overheads, but at the expense of losing the file system administration model. In the following text, we will break down the major contributors that affect file system performance, and compare them to the performance we would achieve if we were to use the raw disk path.

2.4.1 Direct I/O

The Oracle database caches tables in its own cache held within the database shared global area. This is known as the database block buffer cache, and sized at database start-up according to the `db_block_buffers` `init.ora` parameter. Database reads and writes are cached in block buffer cache so that subsequent accesses for the same blocks do not need to re-read data from the operating system.

The Solaris operating system also buffers reads through its global file system cache, which means by default we are caching each read potentially twice. In addition to double caching, we are also paying extra CPU overhead for the code which manages the operating system file system cache. A more efficient way for Oracle to perform I/O is to have the operating system bypass its file system cache, and read blocks from disk straight into the Oracle block buffer cache. We can do this by using a file system feature known as *Direct I/O*.

Direct I/O uses a very similar code path to that when the raw disk path is used, but rather than bypassing the file system completely, we simply bypass the file system cache. This has the advantage of retaining the regular file system administration model, with the advantage of providing an efficient code path similar to that of raw disk.

Solaris, since version 2.6 has provided an option for Direct I/O access for the UFS file system. The Direct I/O feature is enabled by mounting the file system with the `forcedirectio` option:

```
# mount -o forcedirectio /dev/dsk/c0t1d0s2 /db1
```

The Direct I/O path provides a substantial performance improvement for three reasons:

- It eliminates double buffering
- It provides a small, efficient code path for reads and writes
- It removes memory pressure from the operating environment. Solaris uses the virtual memory system to implement file system caching, and as a result, many page-in operations result when reading data through the file system cache. With direct I/O enabled, database reads do not need to involve the virtual memory system.

When using direct I/O we no longer use the file system cache, it is also important to correctly size the database block buffer cache. The Oracle `bstat/estat` statistics facility can be used to monitor the cache hit rates within the database to aid sizing the internal cache.

2.4.2 Log File I/O

The database transaction log is written synchronously during update transactions, and holds the data updated during each transaction. In the event of a failure, the transaction can be rolled back, leaving the data in the tables in its initial state before the transaction began. Thus, each transaction involves writing to the transaction log, and if improperly configured, the transaction log can easily become a bottleneck.

The performance of the I/O subsystem servicing the database transaction log requires special attention. The I/O performance of the transaction log file has a direct impact on the update performance of the entire database instance.

Oracle batches multiple transactions together, so many small updates can be rolled into fewer large writes. The writes are completed in a synchronous mode, so the database can be sure that transaction log information has reached non-volatile storage. Batching writes together can provide a significant performance improvement, since the cost of performing a large I/O is often not much greater than a small I/O.

By default, the Solaris file system breaks synchronous writes into 8k units - a single 64Kbyte write will be performed as 8 back to back 8Kbyte synchronous writes. Thus, regardless of the size of the Oracle I/O, the log writes will be performed as individual 8Kbyte writes, indirectly limiting the log throughput to the number of synchronous I/O's the underlying device can perform. For non-cached storage, this is often around 150 I/Os per second. For OLTP style workloads, this limits the entire database throughput to around 4200 transactions per second, regardless of the number of CPUs and disks in the configuration.

The Direct I/O feature eliminates this behavior, allowing large writes to complete as a single I/O. Enabling Direct I/O for the transaction log allows the Oracle log writer to efficiently batch log file writes together, eliminating the log file as a bottleneck, and allowing scalable throughput.

Figure 2. shows the impact of enabling Direct I/O on an OLTP workload. The increase in performance reflects both the log file performance increase, and the previous-mentioned Direct I/O advantages.

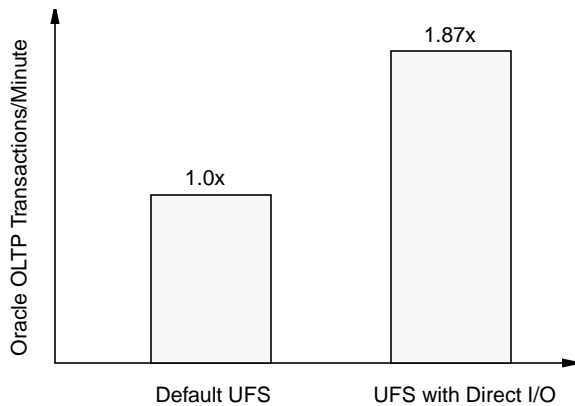


Figure 2. Direct I/O Impact on Throughput

2.4.3 Asynchronous I/O

Asynchronous I/O is a mechanism to achieve I/O concurrency to for greater performance on multi-disk storage. The Oracle database issues writes though central database writer processes. The database writers optionally use the Solaris asynchronous I/O mechanism for issuing writes to the operating system, allowing multiple outstanding writes. The alternative mechanism used when

asynchronous I/O is disabled, issues individual writes using the pwrite() system call, at the expense of limiting the concurrency of database table writes.

To achieve maximum performance, a high degree of concurrency is required, so that multiple overlapping write I/Os can be queued. The pwrite() database writer alternative can be configured to allow a small amount of concurrency by using multiple processes to issue writes - up to 10 database writer processes can be started. Although this provides a small amount of concurrency, it does not provide the same levels of write/update performance as the full asynchronous I/O implementation.

The asynchronous I/O implementation is enabled by setting the Oracle init.ora parameter disk_async_io. When set to true, the database writer issues writes using the aio_write() system calls, and will overlap up to 3072 writes. The concurrent writes are handled by operating system threads, allowing a single Oracle database writer process to issue asynchronous writes. Figure 3. shows a comparison of a OLTP Oracle benchmark with asynchronous I/O disabled and enabled. Such a marked difference is apparent due to the relatively high write component of the OLTP benchmark.

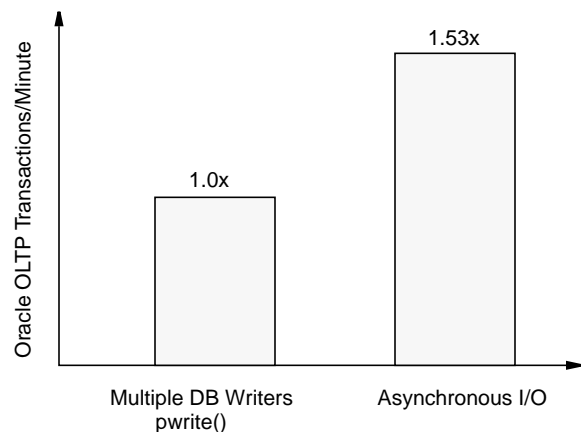


Figure 3. OLTP Example comparing the affect of pwrite() with asynchronous I/O.

Oracle enables asynchronous I/O by default for raw devices. When Oracle is used on file systems, asynchronous I/O is enabled by default on Oracle versions up to 8.1.4, and disabled for 8.1.5/8.1.6. There is currently no mechanism to enable asynchronous I/O for Oracle 8.1.5 and 8.1.6. Asyn-

chronous I/O was re-enabled by default for file systems with the introduction of Oracle 8.1.7.

2.4.4 Kernel Asynchronous I/O

Another minor optimization for asynchronous I/O is available when using the raw disk path. When using raw disk, the Solaris operating system uses kernel threads to manage the asynchronous I/O requests, lowering the amount of time spent context switching and lowering the number of system calls required. This optimization can sometimes provide a few extra percent of performance for the raw disk path.

2.4.5 Single Writer Lock

Solaris implements a per-file reader/writer lock in accordance with POSIX semantics. When writes are performed synchronously (writes are issued with the O_SYNC flag so that the write will not return control to the application until the physical write to storage is complete), the reader/writer lock ensures consistent write ordering, that is writes are issued to disk in the order that they were issued from the application.

Solaris guarantees ordering for synchronous writes by allowing only one writer to a single file at any time. For the duration of a write, no other reads or writes may occur. This can have a significant impact on Oracle database performance, since Oracle uses synchronous I/O for table writes. The overhead of the single writer lock varies - for a system with little or no write/update component, there is little overhead. For an update intensive workloads, there can be quite a significant overhead.

Oracle has been engineered to work correctly with raw disk, which means that it already manages the ordering of reads and writes before handing them to the operating system, and hence does not need the operating system to do so - which allows further optimizations within the operating system which we will discuss further.

Three methods to overcome the single writer lock are:

- Use raw disk. Raw disk does not have the single writer lock limitation

- Use cached storage with non-volatile RAM. The A3500 and T3 storage devices utilize a non-volatile RAM to cache synchronous writes within the hardware storage controller. This helps overcome the single writer lock, since the writes can be cached and overlapped within the storage device, regaining some of the lost concurrency. Cached storage does not completely overcome the single writer lock overhead, but it does provide a significant improvement over non-cached storage.
- Solaris 8 Concurrent Direct I/O - A new implementation of Direct I/O which eliminates the single writer lock will be available in Solaris 8 update 3. This allows the UFS file system to perform at near raw disk performance when used with the Oracle database.

Figure 4. shows the difference between UFS with Direct I/O, and the new Concurrent Direct I/O implementation in Solaris 8 update 3. Results are measured with our OLTP benchmark, which contains a high proportion of update activity. (Workloads without a heavy update component may achieve near-raw disk performance with the regular Solaris Direct I/O)

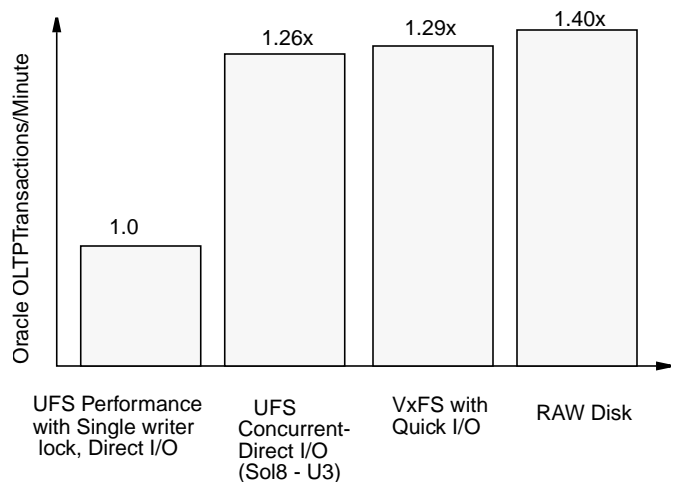


Figure 4. Single Writer Lock Overhead

2.5 Product Versions/Features

Oracle 8.1.7 and Solaris 8 update 3 provide the necessary integration to provide optimum performance in typical system configurations. TABLE 1. shows the Oracle and Solaris versions, together with the respective recommended options.

TABLE 1. Oracle and Solaris Versions

Oracle Versions	Comments
7.x,8.0	Mount fs with forcedirectio
Solaris 8 Update 3	
8.1.5	No Async I/O available
8.1.7	Enable direct I/O by setting
Solaris 8 Update 3	_filesystemio_options=set all init.ora flag