

# Oracle I/O: Supply and Demand

*Bob Sneed (Bob.Sneed@Sun.Com)*

SMI Performance & Availability Engineering (PAE)

SUPerG @ Amsterdam, October, 2001

Rev. 1.1

## Abstract

Storage performance topics can be meaningfully abstracted in terms of the economic notions of "supply" and "demand" factors. The analogy can be extended into characterizing I/O problems as "shortages", and to the notion of addressing shortages with both supply-side and demand-side strategies.

While there is often much that can be done with SQL tuning and schema design to get a given task done with the minimal amount of I/O, this paper focuses primarily on platform-level configuration and tuning factors. A wide variety of such factors are discussed, including disk layouts, filesystem options, and key Oracle instance parameters.

Practical advice is offered on configuration strategies, tuning techniques, and measurement challenges.

## Introduction

The field of Economics offers a wealth of opportunities for phrasing metaphors and analogies relevant to Oracle database I/O. While it is easy to get carried away with excessive creativity along these lines, the basic concept of distinguishing supply issues from demand issues can actually be quite useful for explaining many database I/O options and tradeoffs. This frame of reference is no substitute for proven analysis and tuning methodologies, but it is hoped that it will prove useful in the various processes of architecting, configuring, and tuning Oracle to the Solaris™

Operating Environment (OE) and underlying storage equipment.

While Economics has created a wide variety of modeling strategies and computational techniques, the intention here is to steer away from formalism and merely borrow some economic terminology at an abstract conceptual level. When I/O is involved in a problem at all, it may be possible to improve results with strategies and tactics of either decreasing I/O demand or increasing I/O supply.

After a few introductory metaphors, we will present several key Oracle demand topics, then survey many supply factors in the I/O stack of the Solaris OE – from the bottom up.

## 1. Economic Metaphors

Much of the language of Economics is easily adaptable to discussion of I/O topics.

### – *Equilibrium*

In Economics, the intersection of supply and demand curves indicates a market equilibrium point. With I/O, statistics observed at the disk driver level represent an empirical equilibrium of sorts based on the interaction of numerous supply and demand factors. While a great deal of constructive tuning can be accomplished by studying `iostat` data, these numbers do not really illuminate demand characteristics much at all past the point of observing channel or disk utilization or saturation.

When business throughput is meeting requirements and expectations, equilibrium

measurements are interesting for capacity planning and for spotting operational anomalies. However, when business throughput is not satisfactory, equilibrium measurements may not help clearly identify options for improvement.

### – *Measuring Supply & Demand*

In Economics, demand is said to vary with price, and it is usually very difficult to characterize without conducting experiments. In this discussion, the notion of demand is being used absent the notion of economic cost, but the difficulty of characterization is similar to the economic case.

With I/O, the term *I/O operations Per Second* (IOPS) is frequently employed to characterize both empirical and theoretical throughput levels. This statistic is often of limited utility, as theoretical and marketing IOPS numbers are often difficult to match in practice, and the circumstances under which they occur is not cited with any consistency.

Oracle STATSPACK<sup>1</sup> reports are most useful for determining actual I/O statistics per file since its measurements incorporate queuing delays and throttles from the filesystem and volume manager layers, as well as possible read hits from the OS page cache. In contrast, `iostat` or Trace Normal Form<sup>2</sup> (TNF) data will only show what passes to the physical I/O layer of the OS.

Oracle's measurements are per file, while `iostat` measurements are per device. Untangling the mapping of files to volumes, channels, and devices can be a laborious task. It may add to the confusion that an Oracle PHYSICAL I/O is a LOGICAL I/O to the OS, and thus may reflect cache hits in the OS page cache. Indeed, to the degree that Oracle's I/O statistics do not correlate with OS I/O statistics, the OS page cache provides the obvious explanation.

Veritas offer statistics at the Volume Manager level for various objects, using its own means of data capture and presentation. With the Veritas Filesystem's Quick I/O (QIO) option, a means is provided to observe per file OS page

cache hits, once again using a unique mechanism.

There exists no comprehensive instrumentation which spans all layers of the I/O stack. Some interesting statistics, such as range of variance at various levels, are not generally observable. There is certainly room for progress in instrumenting the I/O stack to get a clearer picture of what's going on.

Oracle's wait event statistics are the principal tool for observing what Oracle is waiting for, and how much stands to be gained by improvements in specific areas. Guidance in interpreting these statistics is offered in several texts, including [YAPP Method, 1999] and [ORA\_9i\_PER]. While these tools and techniques undoubtedly provide the best holistic approach to Oracle I/O optimization, close cooperation between DBA's, System Administrators and Storage Administrators is required. It is hoped that perspectives offered here will be useful in the furtherance of such collaborations.

### – *Quotas & Tariffs*

The I/O stack contains a wide assortment of throttle points and fixed limits. Knowing where these are is the key to working with them or working around them, just as everyone looks forward to shopping at duty-free stores.

### – *I/O Surplus*

In Economics, surpluses are associated with lower prices and modern phenomena like inventory write offs. With I/O, it would appear that the main consequence of excess supply is wasted capital.

In practical terms, an excess I/O supply may have tangible advantages, such as allowing certain application optimizations, or providing the comfort of 'headroom' from a configuration management perspective. Certainly, surplus supply is better than a shortage, and properly configured systems should possess some degree of surplus for the sake of headroom alone.

### – *I/O Shortage*

In Economics, the consequences of shortages include higher prices, longer lines, and maybe even famine or despair. With I/O, the simple analogue to higher prices is increased latency

---

<sup>1</sup>The ancestor of STATSPACK prior to Oracle 8 is BSTAT/ESTAT reporting.

<sup>2</sup>See `prex(1)` et al.

due to queuing, and famine and despair might be terms with which a user could relate when system response is especially poor.

The *only* effect one normally expects from an I/O shortage is degraded response and longer run times. However, worse things are possible. With Oracle, for example, when using Asynchronous I/O (AIO), any request noted to take longer than 10 minutes usually<sup>3</sup> results in a fatal ORA-27062 incident. Failure to update a control file can cause a fatal error after 15 minutes.

It is rare that timeouts like these occur purely due to I/O shortages. They are normally expected only in the event of hardware failures. However, it would appear that the vast majority of Oracle timeout events actually result with no hardware error present, and usually at load levels where one would not predict an I/O could possibly take 10 minutes.

#### – *Supply Chain Failures*

The I/O stack is like a supply chain, and failures at lower levels cause a domino effect of problems through the higher layers. Here again, there is room for improvement in instrumenting the I/O stack to simplify isolation of problem areas.

#### – *Death and Taxes*

Hardware failures and error recovery can seriously skew performance, or even lead to ORA-27062 timeout incidents. Measures taken to insure against such failures, like RAID data protection or Oracle log file multiplexing, can each take their toll – much like a tax or life insurance premium. Hardware failures and compromises to insure against them are just as certain as death and taxes.

Certain bugs have been associated with ORA-27062 incidents, but perhaps the most common cause arises from AIO to filesystem files due to an issue recently identified with the AIO library and the timesharing (TS) scheduler class<sup>4</sup> by Sun BugID 4468181. In the interest of minimizing deaths from ORA-27062 incidents,

---

<sup>3</sup>As of Oracle 8.1.7.2, or with patches to selected prior releases, the message "aiowait timed out" may occur 100 times before becoming fatal condition.

<sup>4</sup>See `priocntl(1)` and `priocntl(2)`.

some subsequent points of discussion will include their bearing on ORA-27062 exposure.

## 2. Oracle I/O Demand

Many design and tuning techniques are aimed at reducing I/O demands. Conventional wisdom says that 80% of the gains possible for an application lie in these areas. This conventional wisdom often leads to neglect of many important factors at the Oracle and platform levels. The discussion offered here largely avoids tuning topics found in many books, such as database schema design, application coding techniques, data indexing, and SQL tuning.

### Oracle Demand Characteristics

Different Oracle operations have different I/O demand characteristics, and understanding these is important to formulating useful strategies and tactics for tuning. Some vocabulary is introduced here for use in subsequent discussion of controlling Oracle I/O demand.

#### – *Synchronous Writes*

All Oracle writes to data files are *data synchronous*<sup>5</sup>. That is, regardless of filesystem buffering, writes must be fully committed to persistent storage before Oracle considers them complete. This synchronous completion criteria is quite distinct and separate from whether or not writes are managed as AIO requests.

#### – *Latency, Bandwidth & IOPS*

These three common and closely related measures of I/O performance each have their place.

The most latency sensitive I/O from an Oracle transaction perspective are physical I/O operations on the critical path to transaction completion. This includes all necessary read operations and writes to rollback and log areas. In contrast, checkpoint writes are not latency sensitive per se, but merely need to keep pace with the aggregate rate of Oracle demand.

---

<sup>5</sup>This is due to Oracle's use of the `O_DSYNC` flag in its `open(2)` operations, but it is inherent with RAW disk access.

Decision Support Systems (DSS) and Data Warehouse (DW) loads typically depend a great deal on pure bandwidth maximization. Also, operations like large sorts can depend heavily on effective bandwidth exploitation. In contrast, Online Transaction Processing (OLTP) workload performance frequently hinges more on random IOPS.

#### – *Aggregate Demand*

Different programming and tuning factors can significantly impact the total amount of I/O demanded to execute a given set of business tasks. The sum total of I/O required can significantly impact run times and equipment utilization.

#### – *Temporal Dependency*

Some things have to happen before other things can happen. This simple notion of time ordered dependencies is all that *temporal dependency* means. For example, log file writes must complete before an INSERT or UPDATE transaction can complete, and before the corresponding checkpoint writes can be issued.

#### – *Temporal Distribution*

In this context, the term *temporal distribution* means "distribution of demand over time". The terms "bursty" and "sustained" are common characterizations of temporal distribution. In some cases, tuning to distribute "bursty" demand over time will result in improved service times and throughput.

As an analogy, consider the arrival of a bus at a fast food store. The average service time for each customer will be seriously impacted by the length of the line. If the same customers strolled in individually over a period of time, they would likely enjoy a better average service time. With demand evenly distributed over time, servers might keep more consistently busy delivering good average service times, rather than idly waiting for the next bus.

#### – *Physical Distribution*

The mapping of Oracle data files across available channels and storage can be extremely important to database performance. The diagnosis and correction of "hot spots" or "skew" is a common activity for DBA's, system administrators, and storage administrators.

The art of database layout includes application of principles such as I/O segregation, I/O interleaving, and radial optimization. Layout decisions can be significantly constrained by available hardware and project budgets.

Disoptimal layout can decrease I/O supply by causing disk accesses to compete excessively among themselves; by disoptimal use of available cache; or by placing bottlenecks in the critical path of temporally dependent operations.

#### – *Localities of Reference*

The concept of *localities of reference* is that certain areas of a larger set of data may be the focus of frequent accesses. For example, certain indexes might be traversed quite frequently, yet represent a small subset of the total data.

Access within good localities of reference stands to gain from optimal cache retention strategies. Cache efficiency will vary depending on how data retention strategies align with actual localities of reference. Opportunities for caching occur in the Oracle SGA, the OS page cache, and external storage caches. Each offers different controls over retention strategies, and these may be configurable to achieve compounded beneficial effects.

## **Controlling Oracle I/O Demand**

We'll skip the obvious topics of proper database design and index utilization, and focus on the areas typically controlled by an Oracle DBA. Topics discussed here represent several of the I/O issues customers commonly wrestle with Oracle, and a few of the "big knobs" used to control these factors. Some of these topics pertain to the "Top Ten Mistakes Made by Oracle Users" presented in [ORA\_9i\_TUN].

#### – *The Critical Path*

As mentioned earlier, some I/O operations, such as log writes and rollback I/O, are on the critical path to transaction completion. Others, such as log archive writes and checkpoint writes, are not, but must keep pace with sustained demands. This difference should be kept in mind during physical database design.

For example, the most demanding INSERT/UPDATE workloads may require dedicated disks or low latency cached

subsystems to allow the online logs to keep pace with transactional demand.

### – *Reduce, Reuse, Recycle*

This modern ecological mantra succinctly encapsulates a great deal of the philosophy of I/O tuning and cache management.

- Reduce – the best I/O is one that never occurs. Many programming and tuning techniques have this as the underlying strategy.
- Reuse – this includes a broad variety of topics including pinning objects with `DBMS_SHARED_POOL.KEEP`, using a separate `KEEP` pool, and exploiting filesystem buffering options.
- Recycle – this is the common concept underlying `priority_paging` (prior to OE 8) and use of a separate `RECYCLE` pool in Oracle.

### – *System Global Area (SGA) Size*

Historically, 32-bit Oracle has allowed SGA sizes up to about 3.5 GB in the Solaris OE. 64-bit Oracle brings the possibility of extremely large SGA's. Indeed, expanded SGA capability is the single distinguishing feature of 64-bit Oracle.

Enlarging `db_block_buffers` is the principle means of exploiting large system memory. Proper sizing and use of the second largest component of the SGA, the shared pool, is key to performance for many workloads.

At any rate, SGA size must not exceed a reasonable proportion of system memory in relation to other uses. There are tradeoffs to be made regarding the comparative advantages of using memory for SGA versus OS page cache, especially in consolidation scenarios. Regardless of how employed, system memory provides the potential for reducing physical I/O demand by servicing logical reads from memory.

Incidentally, Oracle 9i now allows dynamic online resizing of the SGA in conjunction with the new Dynamic Intimate Shared Memory (DISM) feature of Solaris OE 8.

### – *Concurrency*

I/O concurrency with Oracle can arise from several sources, including:

- The client population.
- Use of `PARALLEL` features.
- Use of log writer or DB writer I/O slaves .
- Use of AIO.
- Multiple DB writer processes.

Of these, the most common factors impacting demand for write concurrency are AIO and the use of multiple DB writers. Simply put, AIO allows greater I/O demand creation by a smaller number of Oracle processes with the least overhead. Alternate schemes of achieving concurrency, such as using Oracle I/O slaves, are in a different league from the I/O demand facilitated by AIO.

Prior to Oracle 8, AIO and multiple DB writers were mutually exclusive. With earlier versions of Oracle 8, AIO was not used. There have been various issues with AIO over the years which have led many users to disable AIO, often without firm reasoning. Over time, many users have increased DB writers to its maximum setting of 10 to achieve write concurrency without AIO. Some problems have arisen from users re-enabling AIO with maximal DB writer settings already in place.

In theory, each DB writer can issue up to 4096 I/O requests. In practice, we often see DB writer processes queuing about 200 requests each. In either case, these are large numbers relative to the ability of most systems to concurrently service requests. Pushing an I/O backlog from Oracle to the OS increases queuing and correlates with increased rates of ORA-27062 incidents.

Therefore, in short, when using AIO, `db_writer_processes` should not be increased beyond one (1) without demonstrable business benefit, and tuning upwards from one should be incremental. If AIO is switched off for any reason, increasing DB writers is usually required to attain adequate write concurrency.

### – *Checkpoints & Log File Sizing*

In the event of an abnormal shutdown, recovery time will vary in proportion to the number of dirty database blocks in the SGA. The longer a data block is retained in the SGA prior to

checkpointing, the more chance it has of absorbing INSERT or UPDATE data, and the greater the efficiency of the checkpoint write when it finally occurs. Deferral of checkpoint writes using large logs leads to bursty checkpoint activity, which in turn makes log archiving a relatively more bursty process, and further implies that log shipping strategies will be very bursty.

Thus, both the temporal distribution of checkpoint writes and aggregate writes over time depend on the size of the logs and the setting of various parameters which control checkpointing. Tuning these factors represents tradeoff between performance and recovery time.

This has been an area of significant development in Oracle. Historically, Oracle parameters `log_checkpoint_interval` and `log_checkpoint_timeout` have impacted this balance. Oracle 8i variously offered `db_block_max_dirty_target` and `fast_start_io_target` in this arena. Oracle 9i goes a step further in offering `fast_start_mttr_target` and related parameters to explicitly control the time expected for recovery. Regardless of the method of control, the tradeoffs remain essentially the same.

One can observe checkpoint activity in the Oracle alert file by setting `log_checkpoints_to_alert=TRUE`, but interpretation of the resulting messages varies between Oracle releases.

### – *Sorting and Hash Joins*

Disk sorts are best avoided or optimized. Hash joins have a great deal in common with sorts, but here we will comment only on sorting.

The obvious and traditional means of reducing disk sorts is to set large values for `sort_area_size`. Many users are reticent to do this, however, fearing that aggregate memory demand from clients will be excessive. However, only clients that do large sorts will demand a maximal `sort_area_size` from the OS, so this prospect should not be feared as much as it seems to be.

It is worth noting here that since client shadow processes can write directly into TEMP areas, there is some potential for client-side ORA-

27062 incidents arising from intense write contention for these areas. Also, recommendations regarding use of the TEMPORARY tablespace attribute for TEMP areas may vary between Oracle releases.

There are many other parameters pertaining to sort and hash join optimization, and [Alomari, 2001] gives a good discussion of these.

### – *Large I/O*

The Solaris OE default maximum low-level I/O size is 128 KB<sup>6</sup>, but this can easily be tuned upwards to 1 MB in OE 2.6, or as high as 8 MB in OE 8. Regardless of kernel settings, any process can issue large I/O requests, but large requests may be broken up into smaller requests depending upon kernel settings and volume management parameters. Regardless of whether I/O requests are broken up at the OS level, using fewer larger operations for sustained sequential I/O is typically advantageous due solely to decreased overhead from process context switches.

Factors effecting the supply aspects of Large I/O do not fit neatly in the subsequent discussion of I/O supply topics, so we will mention them briefly here. In short, supply-side optimization of I/O requests involves many factors, including disk layout, volume management parameters, and the kernel's `maxphys` setting. Filesystem factors such as clustering, the UFS `maxcontig` setting, or data prefetching logic can also have significant impact. The topic of bandwidth optimization is the focus of many I/O whitepapers.

In the case of sustained reading, large reads might not always represent an ideal solution. Factors such as pre-fetching at the storage subsystem and filesystem levels, competition for resources, and 'think time' per unit data may result in optimal throughput with intermediate read sizes. In other words, effective pipelining can conceivably reduce 'dead time' that can occur while waiting for large read requests to complete. In general, however, use of large I/O is often key to attaining maximum sequential throughput.

Most I/O demand from Oracle occurs in `db_block_size` units, and large I/O is used

---

<sup>6</sup>Controlled by `maxphys` in `/etc/system`, which defaults to 131072.

only in limited circumstances. Parameters effecting large I/O have been the subject of much revision between Oracle releases, but some examples include:

- Full table or index scan operations – reads are (`db_file_multiblock_read_count * db_block_size`) bytes.
- Disk sorts – `sort_write_buffer_size` bytes in Oracle 7, dynamically sized in Oracle 8i.
- Datafile creation – writes are issued in `ccf_io_size` bytes in Oracle 7, or `db_file_direct_io_count` (DB blocks) in Oracle 8.

Some Oracle versions will revert the setting of large I/O parameters to 128 KB defaults without warning. The best way to confirm the actual I/O size being used by an Oracle process is to use the Solaris `truss(1)` command.

Setting large values of Oracle's `db_file_multiblock_read_count` has the side effect of biasing the optimizer's preference towards using full scans. Because of this, there may be tradeoffs between different queries. One workaround to this is to implement large reads at the session level (ie: `ALTER SESSION`) so that the impact is not global.

Memory constrained systems may suffer from having excessive amounts of memory locked down due to active large I/O operations. For these reasons, the desirability of exploiting large I/O will vary.

As a guideline, DSS and DW systems will tend to enjoy large reads, while OLTP systems may not. Using large reads in conjunction with wide disk striping is a common formula for optimizing sequential performance. Tuning read size to exploit filesystem and hardware pre-fetch characteristics is another means of improving realized bandwidth.

#### – *Data Checksumming*

Oracle allows round-trip data integrity validation by generating checksum data for each data block written, and validating the checksum each time data is read. While the algorithm used is extremely efficient, it does impose some additional latency on every read and write, and therefore has a slight throttling effect on demand. Also, for each read satisfied

from the OS page cache, the data must be re-validated, thus creating some feature bias in favor of a larger SGA and unbuffered filesystem I/O.

While this feature was default OFF in earlier Oracle releases, `db_block_checksum` is TRUE by default in Oracle 9i.

#### – *Backup Schemes*

Time spent in HOT BACKUP impacts aggregate demand from log and log archive writes, because entire DB blocks must be logged for tablespaces that are in BACKUP mode.

Data snapshot technologies, such as Sun StorEdge™ Instant Image, Veritas Volume Replicator (VVR), or Sun StorEdge™ 9900 ShadowImage may be directed at reducing this impact.

### 3. I/O Supply Topics

This section is an annotated survey of various I/O supply factors in the I/O stack from the bottom-up. High impact topics spring up at all layers of the stack. Much as a chain is only as strong as its weakest link, I/O supply can only be as good as the worst bottleneck will allow.

#### Supply Characteristics

Some additional terminology is useful for discussing I/O supply factors.

##### – *Latency vs. Bandwidth*

Defined simply, *latency* is the time required for a single operation to occur, while *bandwidth* characterizes the realized rate of data transfer. Both latency and bandwidth will vary in part with the size of the requested operations.

In latency terms, there are three main approaches to I/O performance issues:

1. Reduce baseline latency, such as by adding cache, improving cache retention strategies, or by a baseline technology upgrade.
2. Improve work per operation, such as by using a larger I/O size.
3. Increase concurrency, such as by adding channels or disks, spreading demand across available resources, or placing more demands upon existing resources.

Opportunities for these manipulations may be presented at various layers of the I/O stack.

#### – *Variance*

In the case of a single simple disk, variance in service times is explained primarily by mechanical factors, such as rotational speed and seek time. In the absence of hardware failure, one could say that with such a simple system, the variance is actually bounded by these well-known mechanical factors. In real systems, sources of variance are diverse, and include:

- Process prioritization (both for LWP threads and simple UNIX processes).
- Geometry-based wait queue sorting.
- Cross-platform competition in SANs.
- Cache utilization.
- Bugs in the I/O stack.

Variance is infrequently measured and characterized, but no advanced math or statistics is needed to characterize the most basic measure of variance – *range*. It is one thing to observe a range of I/O service times in the range of, say, 5 to 500 milliseconds, but quite another to observe variance on the range of 5 milliseconds to 5 minutes!

Concern over variance might be summarized like this:

1. Sources of variance should be identified and scrutinized.
2. Sources of unbounded variance are bad and beg to be eliminated.
3. Sources of bounded variance beg to be purposefully controlled.

I/O statistics are most often cited as averages and sums, and most folks expect variance to be 'reasonable' about the mean. Certainly, ORA-27062 incidents are categorically examples of when this fails to occur.

#### – *Competition*

I/O supply to Oracle is net of any competition. In other words, one can say that the supply of I/O to Oracle is diminished by competing I/O demands involving the same resources. Possible sources of competition are numerous, and some are easily overlooked. Consider these:

- Backup and recovery operations

- Volume reconstruction, mirror re-silvering
- Physical data reorganization
- Transient utility operations
- Other hosts in a SAN
- Data services (eg: volume copies)
- Committal of previously cached write data

Even when these activities occur 'off host', they can compete for head movements and decrease the net I/O supply to Oracle, both in terms of IOPS and latencies.

#### – *Throttles*

Anything that extends the code path for an I/O implicitly adds to I/O latency. Compared to other factors, marginal code path costs are relatively minor in practice. Some components in the I/O stack explicitly enforce certain limitations.

## Supply Factors – Bottom Up

Each layer in the I/O supply chain can contribute significant complexity. This survey of the I/O stack discusses a limited set of issues at each layer.

#### – *The Disk Spindle*

The ultimate limiting factors to sustainable IOPS in a complex system are the number of disks and their operational characteristics. Effective cache utilization at the system and storage subsystem layers may increase the supply of logical IOPS, but when caches and buses are saturated, it is the humble disk spindle that sets the speed limit.

The primary disk characteristics are rotational speed and seek time. Disks vary in many other ways though, including number and type of access paths; quantity of onboard cache and intelligence of onboard cache management; use of tagged commands; and sophistication of error recovery logic. From time to time, significant disk performance issues may be discovered in disk firmware.

#### – *Storage Subsystems*

The range of features available from storage subsystems has mushroomed in recent years. The spectrum of subsystems ranges from the

basic JBOD<sup>7</sup> array to sophisticated RAID subsystems and Storage Area Networks (SANs). Even humble JBOD arrays have developed sophisticated features such as environmental monitoring and fault isolation capabilities.

Intelligent arrays can take complexity off the host and facilitate new data management paradigms, such as off-host backups, contingency copies, and sophisticated disaster recovery strategies, in addition to providing basic RAID features. For this discussion of I/O supply, only a small subset of these features will be covered, including, RAID, caching, and data services.

### – *Storage Subsystems: RAID*

When implemented in an integrated subsystem, RAID features are variously called 'hardware RAID', 'box-based RAID', or 'external RAID'.

In reviewing various RAID levels, an essential point to note is that 'striping' across disks is essential to achieving sustainable bandwidth greater than that of a single spindle. RAID 0 is merely striping across disks with no data protection features. RAID 5 and similar schemes also involve striping across disks.

In the case of RAID 5, since parity must be generated and saved to disk, write activity will have the effect of causing additional mechanical activity that may compete with read requests. Sophisticated integrated storage subsystems like the Sun StorEdge™ T3 or the Sun StorEdge™ 9900 series arrays, are designed to minimize this impact by intelligent I/O scheduling.

In the event of a complete disk failure in a RAID 5 set, all disks in the set must be read to reconstruct lost data, and that process can decrease the net supply of IOPS quite seriously. Once a replacement disk is installed, the process of rebuilding a failed spindle involves reading all of the surviving disks completely, and that can produce significant competition with user demand. RAID 5 implementations typically feature some means of tuning the rebuild rate to avoid unacceptable competition with live workloads. Some more sophisticated RAID 5 implementations, such as in the Sun StorEdge™

---

<sup>7</sup>JBOD means 'Just a Bunch Of Disks', and has come to be common terminology in the industry.

9900 series arrays, may predict the likely failure of a particular disk, and evacuate its data far more efficiently than by RAID 5 reconstruction.

It is sometimes overlooked that in the absence of writes, RAID 5 read performance should be expected to be comparable with RAID 0 performance, since RAID 5 parity data is usually never referenced unless a disk fails<sup>8</sup>.

### – *Storage Subsystems: Caching*

Common to hardware RAID implementations is the use of persistent cache memory. Cache management in external RAID offers the possibility of enhancing I/O supply to attached systems variously by caching writes or by facilitating pre-fetching or other intelligent data retention schemes.

When external cache is disabled or saturated, subsystem performance as seen by attached hosts will degrade. External cache will typically cease to be used for caching writes if its battery backup becomes dubious, but read caching is typically not impacted when cache persistence becomes questionable. Control over caching policies, such as pre-fetching or random read data retention, varies tremendously between different subsystems, and are often controllable on a per-LUN<sup>9</sup> basis.

One simple fact that is often overlooked in evaluating empirical I/O statistics is that writes which appear very low-latency to a host are not truly complete until they are flushed from the external cache. This flushing activity can compete with new I/O requests for head movement and therefore increase latency, especially for new reads. Correlation of current I/O activity with recent writes may be difficult with host-based tools, but one should keep in mind that "*what goes up, must come down*".

It is widely considered that RAID 5 performs less well on writes than alternatives such as RAID 1. While this may be true for host-based RAID 5, it is not categorically true for hardware-based RAID solutions. So long as write demand remains at levels that do not saturate the cache, all RAID levels look

---

<sup>8</sup>An exception would be 'parity checking' operations implemented in some subsystems.

<sup>9</sup>A LUN, or Logical Unit Number, is the term most often used to designate a virtual disk managed by an intelligent storage subsystem.

essentially the same to attached hosts so far as writes are concerned. That is, writes to cache will be very low latency until and unless the cache becomes saturated.

#### – *Storage Subsystems: Data Services*

Persistent cache in external arrays can also be leveraged in implementing efficient box-based data services, such as Sun StorEdge™ 9900 ShadowImage and Sun StorEdge™ 9900 TrueCopy. As with host-based data services, these can contribute some competition and variance to the I/O stack. Any synchronous data forwarding product can inherit tremendous variance from the Quality of Service (QOS) of the underlying data link.

#### – *Layout Strategies*

Whether or not an intelligent storage subsystem is involved, placement of data across available disks can have significant impact on the actual supply of I/O. Regardless of how it comes to be, if two 'hot spots' come to reside at opposite ends of the same disk spindle, performance will suffer. With modern SANs, competition for head movement may even originate from different hosts.

The trend toward higher disk densities and lower IOPS per disk gigabyte has led to substantially different layout strategies from just a few short years ago when use of many smaller spindles was the norm. Right or wrong, the most popular strategy a few years ago was to segregate I/O by class, whereas now the more popular strategy is far more bias towards interleaving I/O classes.

Oracle has promoted a "Stripe and Mirror Everything" or "SAME" approach, while other writers have presented similar concepts variously as "Wide-Thin Striping" or "Plaid" layout strategies. The common underlying notion is to distribute all (or most) Oracle I/O uniformly across a large spindle population, thereby distributing demand across available spindles and minimizing the likelihood of excessive queuing for any single disk. These strategies are often implemented using a combination of hardware-based RAID and host-based RAID.

Another consideration commonly made in layout strategies is the fact that the outer

cylinders of disks contain more data<sup>10</sup>, and therefore provide slightly better performance than the innermost cylinders. Exploitation of this characteristic is sometimes called 'radial optimization'.

In practice, disk layout is something of an art, though its underpinnings are all scientific. It can be helpful, if not essential, to construct drawings of available disks, channels, and their usage to be able to comprehend any given disk layout. Especially when a great deal of the complexity may be housed in an intelligent storage subsystem, effective visualization of the issues may otherwise be impossible.

#### – *Disk Interconnect*

The most conspicuous characteristics of a disk interconnect are its signaling rate and its protocol overhead. 100 MB/sec (1 gigabit per second) Fiber Channel Arbitrated Loop (FC/AL) interconnects are now the commonplace building blocks for modern SAN architectures. Speeds of 2 Gbit/sec and beyond are on the horizon, as are alternate protocols such as iSCSI and Infiniband.

Besides the obvious bandwidth limitations imposed by the native signaling rate, there are two aspects of FC/AL that deserve special mention. First, as a loop-based protocol, some scaling issues can arise if too many targets share the same loop. Second, due to the 'arbitrated loop' aspect of FC/AL, its performance can degrade rapidly with distance. For this reason, SAN switches employing more efficient Inter-Switch Link (ISL) protocols may be required to effectively extend FC/AL over distances.

Alternate interconnect paths can variously serve to add resilience to a system or be exploited for performance. Management of alternate paths is implemented in several software products above the interconnect layer. Multiplexed I/O (MPXIO) provides OS level awareness of multiple paths as an option for Solaris OE 8. Veritas Volume Manager provides a Dynamic Multipathing (DMP) capability. Alternate paths to a Sun StorEdge™ 9900 series array can be managed with Sun StorEdge™ 9900 Dynamic Link Manager. Each of these products has its

---

<sup>10</sup>This is due to variable formatting known as Zone Bit Recording, or 'ZBR'.

own feature set and capabilities with respect to I/O supply and error handling.

### – *Interface Drivers*

A step above the host bus adaptor (HBA) hardware is a device driver software stack. This stack begins with HBA-specific drivers (eg: `esp`, `isp`, `fas`, `glm`, `pln`, `soc`, `socal`), which are integrated using the Device Driver Interface (DDI) framework. Higher level functions are provided by target drivers like `sd` (for SCSI devices) and `ssd` (for FC/AL devices).

Some key quotas and controls for disk I/O are implemented in the `sd` and `ssd` drivers. Despite the close relationship between the `sd` and `ssd` drivers, their controls are distinct. Generally speaking, variables discussed here for the `sd` driver are the same in the `ssd` driver, but are named with `ssd` as the initial letters.

Error recovery parameters, such as `sd_retry_count` and `sd_io_time`, control the time required for the driver stack to return control to higher layers of the I/O stack in the event of hardware failures. By default, these allow some failures to take up to five minutes to be reported to upper layers. Varying these controls can be tricky business, as they also pertain to the resilience of systems during power up operations. This is a topic that begs for further development.

The maximum number of concurrent requests that can be passed to the HBA driver is controlled by `sd_max_throttle`. While certain HBA hardware or attached storage may be limited with respect to its capacity for concurrency, this setting has systemwide impact which can have adverse performance implications. By default, this value is 256, which maximally exploits SCSI *tagged command queuing* capabilities.

Several systemwide SCSI features can be controlled by the bitmask variable `scsi_options`, including *tagged command queuing* and control over the maximum transfer rate which can be negotiated for SCSI transfers. Control over specific interfaces and targets may be available at the HBA driver level, such as described in `glm(7D)`. It would appear that the `scsi_options` parameter is frequently manipulated by third party storage vendors, perhaps often when it should not be. These

controls should be manipulated only with a full understanding of the desired objective and the possibility of unintended consequences.

One aspect of the `sd` layer that is presently not adjustable is logic that continuously sorts the disk wait queue according to the geometry of the underlying disk. While this is quite useful with direct attach JBOD disks, the geometry of external disk subsystems presented to the OS is usually purely fictional. Write caching and I/O scheduling in external RAID subsystems make this `sd` feature of dubious value, and it could conceivably introduce undesirable variance to response times. The ability to control this may appear in a future release of the Solaris OE.

### – *In-host Write Caching*

The Sun StorEdge™ Fast Write Cache (FWC) (see [Sneed, 2001]) provides an in-host redundant and persistent write cache which can greatly reduce the write latency seen by Oracle. While FWC imposes a throttle on bandwidth and is incompatible with cluster environments, its reduction in latency can be quite beneficial to processes which depend on serially dependent writes on the critical path to transaction completion, such as Oracle's log writer.

Another means of in-host write caching is use of cached RAID interface cards, such as the Sun StorEdge SRC/P Intelligent SCSI RAID Controller™ System.

### – *Host-based Data Services*

A wide variety of host-based data services are available, including Sun StorEdge™ Instant Image (II), Sun StorEdge™ Network Data Replicator (SNDR), Veritas Volume Replicator (VVR), and an assortment of filesystem snapshot facilities. These categorically impact aggregate physical I/O demand, and generally introduce some variance, throttling, and temporal dependencies in maintaining data structures such as bitmaps.

As with their externally-based counterparts, synchronous remote replication products can introduce significant or pathological throttling and variance depending on the Quality of Service (QOS) of the network interconnects.

Nevertheless, as a class, these products effectively facilitate a wide range of cost effective business solutions. System designs

should incorporate appropriate considerations to assure success with these products. For example, cached and/or dedicated storage might be allocated for bitmap data structures.

### – *Volume Management*

Volume managers, such as the Veritas Volume Manager (VxVM) or Solstice DiskSuite™ (SDS) provide myriad techniques for manipulating I/O supply. The code path overhead of Volume Management is usually considered to be so slight as to be insignificant, but several topics in volume management can have extreme significance. Volume Managers provide host-based RAID capabilities, which is where many layout decisions are typically implemented.

Host-based RAID 0, RAID 1, and hybrids of the two actually impose very little overhead relative to the native capabilities of the attached storage. Host-based RAID 5, which is implemented at the Volume Manager layer, is entirely another matter. Since host memory is not persistent like the caches of hardware RAID subsystems, RAID 5 parity data must be flushed along with each synchronous write in realtime, and this is typically disastrous relative to database write requirements. On the other hand, some very large DSS databases might rationally employ economical host-based RAID 5 for predominantly read-oriented tasks where the write penalty is not too severe with respect to the rate of data updating. Host-based RAID 5 exhibits very good read bandwidth.

Due to the popularity of VxVM on Sun™ systems, VxVM issues tend to have a high rate of impact when they occur. Therefore, a few details specific to VxVM are offered here.

With mirrored volumes implemented in VxVM, recovery time after an abnormal system shutdown can be greatly reduced by implementing Dirty Region Logging (DRL's). It is generally recommended to group all DRL areas for a system on a few disks dedicated to that purpose. When properly implemented, DRL maintenance overhead is often characterized as reducing write supply by about 10%. Without DRL's, recovery after a crash can require complete resilvering of disrupted mirrors.

One VxVM issue to which ORA-27062 incidents were explicitly tied pertained to systems using Dirty Region Logging (DRL's) with mirrored volumes<sup>11</sup> under heavy load. An I/O could get 'hung' awaiting a DRL update, thus leading to extreme variance.

Another recent issue was noted in VxVM 3.0.1 where the default read policy for mirrored data was changed from a prior default that tended to favor one side of the mirror for a series of sustained reads<sup>12</sup>. Under the prior default with JBOD arrays, sustained reading would benefit from revisiting the spindle cache of the chosen disk. This issue surfaced in terms of reduced supply of sequential performance.

### – *Filesystem Cache*

Simply put, filesystem caching is bad for synchronous writes, but good for reads that are satisfied from it. That being said, it may not be simple to determine the optimal filesystem caching scheme for a given Oracle workload. Filesystem cache performance and its contribution to Oracle performance can vary greatly depending upon caching options used and the amount of memory available. With 32-bit Oracle, use of the filesystem cache is a principal means of exploiting large server memories.

In the Solaris OE, the filesystem cache is not a fixed size area, but rather lives in the OS page cache, so the terms are used interchangeably in this discussion. See [Mauro & McDougall, 2001] for more information on Solaris memory management.

The 'extra copy penalty' often cited against using filesystem cache is not highly significant in the overall scheme of things. Copying data in memory is an operation at which modern Sun™ systems are particularly adept.

### – *Filesystem Block Size*

Since Oracle I/O is typically in `db_block_size` increments and 8 KB blocks or larger are becoming the norm, it is usually advised to use a VxFS filesystem block size of 8 KB. Smaller block sizes will decrease supply and cost in terms of CPU usage when used on

---

<sup>11</sup>See Bug Report #4333530, fixed at VxVM 3.1 and by current patch to version 3.0.4.

<sup>12</sup>See Bug Report #4255085, fixed at VxVM 3.1.

Oracle data filesystems that never see I/O requests smaller than 8 KB.

By default, VxFS selects the filesystem block size at the time of filesystem creation based on filesystem size, without regard for its intended use. Therefore, one must explicitly specify a block size option to `mkfs13` to assure consistent results.

On UFS, 8 KB is the only blocksize permitted on modern Sun™ systems.

### – *Filesystem Throttles*

UFS has tunable thresholds at which it will suspend and resume processes with uncommitted deferred write data, and thus prevent a single process from overrunning the page cache or using an inequitable share. A 'high-water mark' (UFS:UFS\_HW) and 'low-water mark' (UFS:UFS\_LW) implement this throttle.

A similar throttle is introduced with VxFS Version 3.4. Absent such a throttle, the importance of aggressive Virtual Memory Management with deferred buffered I/O was magnified (see [Sneed, 2000]).

The mere presence of a filesystem code path has some I/O throttling effect relative to RAW disk, but this turns out to be a very minor factor in most cases.

### – *Filesystem Single-Writer Lock*

Perhaps the single most significant factor that limits Oracle write throughput with filesystems is the "single writer lock" constraint required by POSIX standards to assure proper ordering of synchronous writes. Increased write concurrency at the Oracle level is for naught when the filesystem serializes the writes. The impact of the single writer lock is reduced with externally cached storage, which provides low write latency and correspondingly lowered lock dwell times.

Since Oracle handles its own write ordering considerations, the single writer lock common to standards-conforming filesystems is both unnecessary and injurious to Oracle I/O performance. There are several ways to avoid the single writer lock altogether, including:

- Using RAW disk volumes, LUNs, or slices.

<sup>13</sup>See `mkfs_vxfs(1M)`.

- Using VxFS Quick I/O (QIO)
- Using the UFS concurrent forced direct I/O feature (`forcedirectio`) available in Solaris OE 8, Update 3 or later.
- Using the Sun™ QFS filesystem Q-Write feature.

For users on filesystems, implementing VxFS QIO, UFS `forcedirectio`, or Sun™ QFS Q-Write requires no movement of data. In contrast, conversion from filesystem to RAW would require completely offloading and reloading the data, as would conversion between filesystems<sup>14</sup>.

Other points to ponder about these options include:

- QIO and RAW share the advantage of exploiting the Kernel Asynchronous I/O (KAIO) code path, which is capable of supplying more IOPS than the LWP AIO code path.
- QIO offers the ability of per-file control of filesystem caching ('Cached QIO') and the ability to observe read hits rates in the OS page cache, while UFS `forcedirectio` categorically disables OS buffering.
- QIO introduces some operational complexity relative to UFS `forcedirectio`.
- QIO is a separately licensed feature, while UFS `forcedirectio` comes with the Solaris OE.

Avoidance of the single writer lock and use of the KAIO code path are known techniques for reducing the probability of ORA-27062 incidents. Each option for doing this has its own pros and cons.

### – *Asynchronous I/O (AIO)*

The Solaris OE offers two alternate AIO Application Programming Interfaces (API's). One conforms to the POSIX Real Time (RT) specifications (eg: `aio_read(3RT)`, `aio_suspend(3RT)`, et al) implemented in `librt(3LIB)`, and the other conforms to SUNW private specifications (eg: `aioread(3aio)`, `aiowait(3aio)`, et al) implemented in `libaio(3LIB)`.

Oracle uses the SUNW API.

<sup>14</sup>In some cases, tools may exist to allow in-situ conversion of filesystems.

The SUNW implementation in `libaio` includes a hard-coded throttle called `_max_workers` which limits the number of active requests per process. This value is set to 256<sup>15</sup>. A process can queue requests past this limit subject to available memory, but queuing far past this limit offers no tangible benefit.

For more information on SUNW AIO implementation and usage, see [Mauro & McDougall, 2001] and [McDougall & Gummuluru, 2001].

Kernel Asynchronous I/O (KAIO), which is used on RAW and VxFS QIO files, is often cited as yielding 10–12% gains on extremely write-intensive benchmark workloads. However, for many workloads its impact may be negligible, especially considering that many database workloads are either extremely read biased or more constrained by bandwidth than write concurrency.

As mentioned earlier, it is not advisable to use too many DB writer processes when AIO is used.

### – *Process Scheduling*

Last but not least, Oracle processes cannot generate I/O demand unless they are in memory, ready to run, and not deprived of needed CPU and memory resources. Aside from the topics of having the CPU and memory appropriately sized for the workload, this relates to several system management topics including Virtual Memory Management (VMM) configuration<sup>16</sup> and manipulation of process priorities.

Process priority manipulation is a rather complex topic in the Solaris OE, and a wide assortment of tools and techniques are addressed at this topic. `dispadm(1M)` and related topics from the 'SEE ALSO' section of the `dispadm` man page are the main tools for manipulating priority schemes. Other facilities include the venerable `nice(1)` command; `processors sets (psrset(1M))`; and the separately licensed Solaris Resource Manager™.

It is noteworthy that each AIO request to filesystem files involves a lightweight process

<sup>15</sup>This was 50 prior to Solaris OE 2.6.

<sup>16</sup>See [Sneed, 2000] for more information on VMM tuning and Intimate Shared Memory (ISM) usage with Oracle.

(also called a 'LWP' or 'thread'), and that these threads have their priorities shuffled in the same manner as other threads in the TS scheduler class. This can give rise to variance in response times seen by Oracle, and with extreme loads, this variance can be pathological.

Applied incorrectly, priority manipulations can lead to seriously adverse outcomes.

## Conclusion

Hopefully, some of the perspective and details offered here will be found to be useful in applied terms.

Each layer of the I/O stack can be meaningfully discussed in terms of its impact on the supply of I/O to the layers above it. I/O demand results in an equilibrium that depends on numerous supply factors. This perspective can be useful in comprehending the complexity of the I/O stack, and in understanding I/O options and tradeoffs.

## References

Oracle publications are generally available online via the Oracle Technology Network (OTN) Web site at <http://otn.oracle.com>. Membership is free.

[ORA\_9i\_PER] – "Oracle 9i Database Performance Guide and Reference", Oracle Corporation Part No. A87503–02, 2001.

[ORA\_9i\_TUN] – "Oracle 9i Database Performance Methods", Oracle Corporation Part No. A87504–02, 2001.

[Alomari, 2000] – "Oracle 8i and UNIX Performance Tuning", Ahmed Alomari, Prentice Hall, September 2000, ISBN 0130187062.

[Mauro & McDougall, 2001] – "Solaris Internals", Jim Mauro & Richard McDougall, Prentice Hall, 2001, ISBN 0–13–022496–0.

[YAPP Method, 1999] – "Yet Another Performance Profiling Method (YAPP Method)", a whitepaper by Anjo Kolk, Shari Tamaguchi & Jim Viscusi of Oracle Corporation, 1999.

[McDougall & Gummuluru, 2001] – "Oracle Filesystem Integration and Performance" – a

whitepaper by Richard McDougall and Sriram Gummuluru or Sun Microsystems, January 2001.

[Sneed, 2001] – "**Sun StorEdge™ Fast Write Cache Application Notes**" – a whitepaper presented at the April, 2001 Sun Users and Performance Group (SUPeR) conference.

[Sneed, 2000] – "**Sun/Oracle Best Practices**" – a Sun Blueprints™ Online article available at <http://www.sun.com/blueprints/0101/SunOracle.pdf>.

## **Acknowledgments**

Special thanks to Geetha Rao (Veritas), Vance Ray (Veritas), Yuriy Granat (Oracle), and Jamie Vuong (Oracle) from the Veritas/Oracle/Sun Joint Escalation Center (VOS JEC) for all their contributions to the VOS JEC. Thanks to Jim Viscusi (Oracle), Rey Perez (Sun), and Elizabeth

Purcell (Sun) for their review and feedback. Thanks also to Janet, my wife and volunteer editor. This document was created using StarOffice™ software.

© 2001 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, Solaris, Solaris Resource Manager, StarOffice, Sun Blueprints, Sun QFS, Sun StorEdge, Sun StorEdge Instant Image, Sun StorEdge Network Data Replicator, Sun StorEdge T3, Sun StorEdge 9900, Sun StorEdge 9900 ShadowImage, Sun StorEdge 9900 TrueCopy, Sun StorEdge Fast Write Cache, Sun StorEdge SRC/P Intelligent SCSI RAID Controller System, and Solstice DiskSuite are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States of America. All SPARC trademarks are used under license and are trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.