

Solid State Disks in an Oracle Environment

The New Rules of Deployment

James A Morle, Scale Abilities Ltd

1.0 Introduction

This is yet another paper about solid-state disks (SSDs). So, why should you read such a paper? The reason is simple: One should always reassess the suitability of technology over time. For example, a 1930s Theremin audio synthesizer probably didn't fit too well, either physically or musically, into orchestras of the time, and yet the entire 80s music genre was dominated by music of the synthesized variety! Things change, and none more quickly than the industry of computing.

2.0 Background

For this paper, I used an SSD from Imperial Technology. Imperial believe strongly in SSDs, and they've been making them longer than anybody else. After reading my Sane SAN whitepaper [Morle 2001], Imperial graciously lent me some of their equipment (en route to a trade show), namely a MegaRAM 1000 with 3GB of storage, in order for me to ascertain the place for SSDs in modern Oracle deployments.

I say 'modern' deployments, because when SSDs first hit the streets, a number of factors were very different to today:

- there were no cached disk arrays like we have today
- system memory was expensive and restricted in terms of addressability
- CPUs were significantly lower power
- SANs were not yet conceived
- Limited storage interface performance
- Low disk density

So, in the early days of SSDs, there were not many methods available for accelerating the storage layer of the database, other than using a carefully tuned Oracle buffer cache, or some kind of SSD. However, the relatively low density of data on disk ensured that a database of reasonable size was composed of many physical disk spindles.

A number of attributes have radically changed since then:

- DRAM pricing dropped significantly

- CPU power dramatically increased
- Storage and server consolidation trend
- Memory addressability increased by at least two orders of magnitude
- Gigabit+ storage interfaces
- SANs
- High disk density

The attribute which has not changed a great deal is the performance of physical disk [Morle 2001]. Combined with the increase in speed of CPUs, this fact makes the performance gulf between *processing* data and *storing* data wider than it has ever been. To address this issue, there are two alternatives: caching, and SSDs.

The most common approach with modern hardware is to adopt a large cache in the storage tier. Ranging in size from a few MB to 32GB+, this cache is designed to accelerate both writes (by deferring the physical writes), and reads (by servicing reads from memory). However, there are a few downsides to this:

1. Writes can only occupy a (typically small) percentage of the total cache before they must be flushed to disk to make space for subsequent writes. As soon as this happens, write performance is comparable to physical disk performance.
2. Read caching relies on *temporal* and *spatial* locality of reference:
 - a. Temporal locality is where accesses to the same data are made within a short period.
 - b. Spatial locality is where accesses are made to data that is *close to* recently read data.
3. In a multi-cache hierarchy [Morle 1999] such as Oracle database servers, most of the useful temporal and spatial cache hits are already 'stolen' by the Oracle buffer cache.

Clearly, there are cases where caching does not provide the required speedup, so let's look into whether there is a place for SSDs in this modern world.

3.0 Assessing Solid-State Disk

First of all, let's quickly review the major attributes of both standard disks, and solid-state disks:

- Attributes of Standard Disks
 - Relatively cheap **per MB**
 - Non-volatile by default
 - Ubiquitous in nature

- Bound by physical motion of head
- Unpredictable response times
- Relatively high latency
- Relatively low bandwidth
- Attributes of SSD
 - Relatively expensive **per MB**
 - Volatile by default: needs some kind of battery backup and/or physical disk staging (the Imperial product had both built-in)
 - Virtually nil contention between requests, due to the lack of physical head movement
 - Ultra low latency servicing requests
 - Predictable response times
 - Bandwidth bound only by the speed of the interface port¹

Notice the boldface type for ‘per MB’ in both lists of attributes. The fact is, for many database uses, the MB is just the wrong measure of value in the storage infrastructure. It is like choosing the biggest car without checking it has an engine: For any performance sensitive areas of the database, it is the number of I/O operations per second that can be sustained that is the most important attribute. SSDs are the clear winners here, with a single unit able to sustain many thousand I/Os/s, and virtually zero latency (the majority of the latency is in the Operating System, and intermediate interfacing). However, it goes a little deeper than that...

The latency provided by a physical disk drive is directly proportional to the distance the head must move before servicing the request, plus the time for the disk to rotate into the correct position. This means that, if there is contention on the disk that causes the head to move, the resulting contention *dramatically* reduces the number of I/O operations the drive can fulfil per second.

Finally, an SSD can provide a good deal more bandwidth than a physical disk. Although, from looking at the datasheets, a physical disk appears to offer incredible bandwidth, the reality is somewhat different. If the disk is even mildly contended for while the read progresses, the band width from the disk reduces significantly. This is not the case with an SSD, which can service multiple high-bandwidth consumers concurrently with servicing many low bandwidth consumers. I used the word ‘concurrent’ in the last sentence. This is

¹ The SSD has a *total aggregate bandwidth* that cannot be exceeded. However, this is often close to the maximum capability of all Fibre Channel interfaces combined, and is rarely reached even under high load.

a highly desirable attribute of a database server, and one which SSD lends itself to very well.

4.0 Uses for Solid-State Disk

All this talk is very well and good, but what does this make them useful for in this modern Oracle environment? Here's a few examples, in order of generalized impact, where the pros of SSD can be maximized, whilst limiting the cons. All configurations discussed do not include redundancy, for the sake of clarity.

4.1 Redo Logs!

Big surprise, exactly as they have always been used. However, it's worth looking at how they help save money and increase performance in a modern environment. In a high-performance database server, I always advocate the use of at least two physical disk drives (plus mirrors), 100% dedicated for use as redo logs [Morle 1999]. The idea behind this is that one log should be written sequentially by the LGWR process, and then switch to the alternate drive on each log switch. This leaves the previous drive free (until the next switch) for the ARCx processes to read for the purposes of archive logs. This technique maximizes the available bandwidth of the drives, and minimizes head movement, thus optimizing write latency and improving commit performance to the end user. Often, this involves doing a very unpopular thing: wasting significant capacity on the drives. There are not many sites that require 146GB+ (2x 73GB drives) of online redo log, and even if they did, it's likely they would need more bandwidth and less latency than a single physical drive (remember, we only use one at a time) could provide. A more likely scenario is that perhaps 10GB of each drive is used.

I am also a big advocate of RAID 1+0 (mirrored then striped) disk layouts for general high performance use. The problem is, allocating these two disks for redo logs often upsets the appcart on the nice, uniform disk layout.

This is where SSD can help out. By deploying a *single* SSD, all redo logs can be located away from the RAID 1+0 array, whilst providing low latency writes and high bandwidth reads (for archiving). There is no contention between the log writer and the archiver in this scenario, regardless of whether, for example, operational difficulties have prevented archiving for a while. It gets better...

Server consolidation is becoming popular. Put all your storage into one big SAN, and run your databases off domains of a single machine. Perhaps you run lots of Linux or Windows servers? It's likely that a SAN is still the centre of the storage universe, even though you operate many small servers. In any of these cases, it's likely that *the same single SSD* can support the redo logs of every database. That's the beauty of zero contention and high bandwidth.

There is another scenario where multiple threads of redo are used and high performance is needed: Real Application Clusters. If the promise of capacity-on-demand for RAC ever becomes the norm, there's an awful lot of redo threads to find homes for. Not only that, but any one of these redo logs can each become a limiting factor in the performance of the *whole* cluster. Therefore, a ten-node RAC cluster would need a minimum of twenty physical spindles for redo alone (excluding protection), of which a potentially very small amount of the capacity would be used. Using SSD, a single (small) SSD could provide the service for all nodes, and at a price point that compares well to the standard disk!

As a final word on redo logs: The cache management of many cached disk arrays are optimized to be 'fair share' – that is, not allow any one system or disk object to dominate the shared cache resource. In reality, however, certain database objects, namely redo logs, would benefit from having an unfair share of this cache, used only as a write cache. Achieving the same effect on a cached disk array can be a complex and non-intuitive affair!

4.2 Non-Data Tablespaces

SSD can be an effective tool in the 'support' tablespace types:

- TEMPORARY
- UNDO (or Rollback Tablespaces prior to 9i)

Both of these tablespace types contain special types of data, for the purpose of supporting Oracle in the provision of its primary function. The TEMPORARY-type of tablespace is used as a staging area during large sorts, and the UNDO-type of tablespace is used to store the undo, or rollback, information for DML operations. They have the following attributes which make them strong SSD candidates:

- TEMPORARY
 - Direct reads and writes: Oracle shadow processes (the per-user process on the database server) can directly read and write to the TEMPORARY tablespace, bypassing the Oracle buffer cache and DBWR
 - Concurrently used by multiple sessions
- UNDO
 - All database modifications result in writes to the UNDO tablespace. If not using the 9i automatic undo segment feature, you will have standard tablespaces with *rollback segments* created within them; the principle is exactly the same.
 - Concurrently used by multiple sessions

I consider these tablespace types to be in order of merit, though this will vary between applications. TEMPORARY tablespaces stand to benefit the most, because of the direct

write pathology, whilst UNDO are somewhat buffered by the Oracle buffer cache. However, if you are running a pure OLTP system that never sorts on disk, the UNDO tablespace would likely be the best candidate. Checking your I/O statistics in `v$filestat` and `v$tempstat` will quickly demonstrate the area demanding the most attention!

A significant attribute of *both* tablespace types is that different sessions could be using them at different offsets in the file. This would impose an increased seek overhead which, once again, hurts the performance of standard disk. SSD would not be affected by this issue.

4.3 Data Tablespaces

The natural temptation would be to build the entire Oracle database out of SSD. However, it's not clear that this represents good value for money, and I will explain why this is. Oracle is designed to run from disk, and therefore includes a relatively high proportion of architecture and codepath specifically to speed up this activity. If the whole database were running from SSD, there would be enormous pieces of unnecessary work going on, such as:

- Management of the buffer cache
- Context switches into kernel mode to perform I/O
- Conversion of the request into SCSI/Fibre Channel
- Transmission across the SAN
- And all the way back again

In comparison to disk I/O, this whole process is stunningly fast. In comparison to just reading the data straight from user space memory, however, it is incredibly slow! Why make that comparison? Because when you spend enough money on SSD to put your whole database in memory, it is very expensive – possibly as much or more than a proprietary in-memory database engine which does exactly the second scenario.

Time for the big 'HOWEVER' ...

HOWEVER, the reality of most commercial (as opposed to scientific) databases is that there is a very definite working set of data that is much smaller than the total size of the database. If this set can be identified, and operationally staged on the SSD, then 80% of the gain will be realized with significantly lower cost than putting the whole database onto SSD. For example, consider the following potential SSD targets:

- The 'current/active' partitions of large OLTP systems. When a partition becomes less current (old reservations, for example), they could be copied (using Transportable Tablespaces) to a standard disk area, freeing the SSD for a newer set of data.

- Specific datafiles involved in large batch runs. By using a level 8 10046 trace file,² and looking at the 'db file %' wait events, it can be easily determined how much speedup certain files (the p1 value) would contribute if placed on SSD.

5.0 Testing

The proof of the pudding, as they say, is in the eating. Thanks to the loan of the Imperial Technology MegaRam, I was able to try out a few simple tests in order to establish whether or not decent speedups were achievable. All the tests were run on a relatively modest hardware setup:

- Dual Intel Pentium III 1.2GHz processors
- 1 GB memory
- Qlogic Fibre Channel HBA
- 14 x 18GB 10k rpm FC drives (Seagate Cheetahs)

I ran three tests in total:

4. Order Entry Benchmark, 10 users, very aggressive transaction rates
5. Same benchmark, with two redo log *members* on the *same drive* to simulate seeks on the redo disk
6. A large join operation

5.1 Test 1 – Order Entry Benchmark

The order entry benchmark was tuned specifically to bottleneck on the redo logs. Originally written as a PL/SQL-based benchmark, it was modified in the following ways to put more strain on the redo logs:

- Re-written in Java. PL/SQL contains a specific optimization to the commit mechanism which prevented sufficient pressure being put on the redo logs
- Any index range scans removed to ensure that the workload between commits was kept to a minimum

Like many packaged applications (to remain nameless), this application now spent a ridiculous amount of time committing, which was perfect for this test. The test was run twice, once with the redo logs on dedicated (zero seek) Cheetah drives, and once with the logs located on the Imperial MegaRam. Both tests were run with just 10 users, which was sufficient to fully load the test platform due to the aggressive nature of the benchmark, and an equal number of transactions. The results follow:

² SQL> alter session set events '10046 trace name context forever, level 8';

Redo Placement	Completion time	% Time spent in 'log file sync'	% Time spent in 'SQL*Net message from client'
Cheetah Drives	42.18s	40.5%	17.43%
Imperial MegaRam	34.22s	7.07%	28.90%

The MegaRam was 20% faster, but the improvement was limited somewhat by the available CPU on the server. This is demonstrated by the dramatic increase in 'SQL*Net message from client' event: The benchmark had no sleeps in it at all, and so if available resource were available, there should be negligible waits on this event. If the total reduction in 'log file sync' were applied without an increase in CPU utilization, the MegaRam would have completed in a time of 29.95s, or approximately 30% faster. The maximum possible would have been 40.5%, as per the 'log file sync' waits in the first test, so 30% is pretty respectable.

5.2 Test 2 – Seeks on Redo Drive

This test was designed to simulate the effect of seeks on the latency of writes to the redo logs. To do this, I decided to place two members of the same redo log group onto the same drive. This means that the LGWR process has to write to both files concurrently, seeking between them each time. The same benchmark configuration was used as the previous test, to ensure a measurable baseline. The results follow:

Redo Placement	Completion time	% Time spent in 'log file sync'	% Time spent in 'SQL*Net message from client'
Cheetah Drives	69.2s	77.6%	8.35%
Imperial MegaRam	32.4s	13%%	35.03%

The first thing to note is that the MegaRam time was +/- 5% identical to the test without the multiple log members. This is a clear indication of the concurrent scalability of the MegaRam, where it is not affected by contention like a physical disk. The Cheetah drives produced a result that was over twice as slow as the MegaRam, and a good 65% slower than the same Cheetah configuration with no contention.

In terms of I/O rates during the test, the Cheetah was unable to perform greater than 100 operations/s due to the seeks across the disk, whereas the MegaRam was still returning good response times at greater than 300 operations/s.

5.3 Test 3 – Large Table Joins

This was a very simple little test. I created a query that would require extensive disk sorting to perform the request SORT MERGE JOIN, and executed it with the TEMPORARY tablespace located on either a 12 drive disk stripe, or on the Imperial MegaRam. This test turned out to be quite a CPU intensive operation, and so the results are not as extreme as they could have been. However, there is still quite a marked improvement simply by using the MegaRam as the TEMPORARY tablespace. The striped disk managed to return the query result in 379s, whilst the MegaRam returned it in just 281s. This represents an improvement of 25%, even on such a compute-bound query.

6.0 Summary

The deployment of Solid-State Disk can be an extremely cost-effective solution to gaining faster, more predictable performance from a SAN. It would be my preference to implement SSD in conjunction with an existing disk-based SAN, using the guidelines in this paper to maximize the benefits and value.

7.0 About The Author

James Morle is the founder of Scale Abilities Ltd, a specialist consulting company offering both high-end consulting, and unique software products. With over 10 years experience in architecting some of the world's largest Oracle systems, James is a well respected member of the Oracle community. James is also the author of the critically acclaimed book, *Scaling Oracle8i: Building Highly Scalable OLTP System Architectures*.

8.0 References

[Morle 1999]

Scaling Oracle8i: Building Highly Scalable OLTP System Architectures; J.A. Morle 1999
Addison-Wesley ISBN 0201325748

[Morle 2001]

Sane SAN; J.A. Morle 2001
<http://www.scaleabilities.com/whitepapers.shtml>