Sane SAN 2010

James Morle, Scale Abilities Ltd

1. Introduction

It has been ten years since I wrote my original whitepaper, Sane SAN. I'll refer to that paper as Sane SAN 2000 from now onwards to keep the terminology clear. Whilst a great deal has changed since that whitepaper, many things remain pretty much the same as they were at the start of the millennium. That situation is changing with emerging technologies, and this will mark revolutionary changes in the way we use persistent storage with our databases.

2. Think Differently About Storage

I have a single goal with this paper: To get you to think differently about storage. Storage has increasingly become a 'black box'; the domain of a different team, an independent piece of hardware that should be revered. The storage array is so expensive and so mission critical that it has become the Emperor's New Clothes: Nobody will say anything bad about it for fear of looking foolish themselves. Some database administrators would even admit to being a little scared about mentioning anything negative about this intimidating monolith. To those DBAs I would say: You are not alone!

I want to change some of this right now by sharing a significant observation that I have repeatedly made over the last ten years of working on storage platforms of every possible shape and size:

Storage Performance Has Been Terrible For Years

It would appear that I am somewhat late with this revelation:

"I/O certainly has been lagging in the last decade" - Seymour Cray, 1976

There are many reasons why it is so poor. The fact that the magnetic disk is based on physical moving parts is certainly a big factor, but when that is combined with over-stretched CPUs in the storage array and colliding workloads across disparate systems then it is no wonder that things don't quite perform as well as they might. The storage arrays that start off running with reasonable performance also generally degrade very quickly.

So why is this happening - why do storage arrays always end up running so slowly? The answer is simple: Storage arrays are complex. When things become complex, they become difficult to understand, particularly for those not working with them on a daily basis. When things become difficult to understand, it's human nature to look for a shortcut, and that shortcut is available in force with storage: "The Best Practice".

"Best Practice" is really where it all goes wrong with storage, and it is because the arrays get so complex that best practices are applied with such fervour. How do you know when something is really and truly the "best" practice rather than just "standard" practice?

Some of these practices make perfect sense. The Stripe and Mirror Everything (SAME 2000) methodology as used by default in Oracle's ASM makes a lot of sense[1], for example. It's not perfect for everything, but it is certainly good enough for most requirements. Crucially, SAME dramatically reduces the complexity of the storage layout.

[1] Technically it is better to use MASE - Mirror and Stripe Everything, in that order. I guess it didn't sound so snappy as SAME, though.

The trick is to build something that is as simple as possible, and then apply the “Right Practices” for the job in hand. And if you don’t like the rules, cheat! If a “Best Practice” does not work for your system, you know best - no Best Practice is mandatory.

3. Sane SAN 2000 - A Review

Let's review the main observations and points of Sane SAN 2000 ([SANE 2000]) and go on to see if any of them are still valid in 2010 and whether any of them might be candidates for Right Practices moving forwards.

When [SANE 2000] was written, SAN and NAS devices were relatively new. In fact, the paper even had to explain what a NAS device actually is. In the paper I suggest that SAN-based storage arrays had a slight edge on NAS devices in terms of mission critical performance. Performance of both devices has dramatically increased since then, so maybe the balance of power has changed. Let's see later on in the paper.

[SANE 2000] also pointed out that drives were getting progressively slower over time when measuring IOPs per gigabyte. Capacity was increasing quickly, but performance was not increasing at the same rate. We will recalculate the relative performance of storage within this paper and evaluate the current state of play.

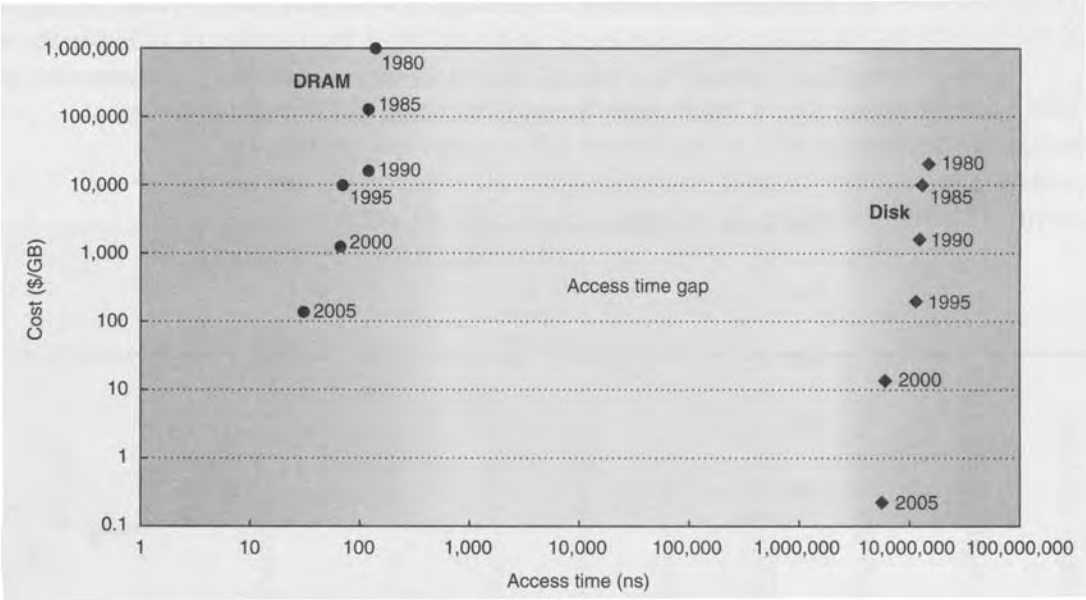
The main point of [SANE 2000] was to recommend the adoption of MicroSANs. The proposal was to cordon off discrete sets of channels and drives in the storage array to ensure sufficient dedicated spindles for the IOPs requirement of the database. The emphasis with the MicroSAN was to provide a consistent and guaranteed minimum level of service, regardless of other workloads in the array. The counter argument to this is that it is better to spread everything as wide as possible to maximise resources. Again, we'll evaluate the wisdom of the MicroSAN later in this paper.

4. So What's New?

Ten years have passed since the writing of [SANE 2000] and much has changed. Or has it? Until very recently, the only changes to mainstream storage arrays have been:

- More/faster processors in the array
- Better backplanes
- Uprated connectivity bandwidth
- Larger capacity drives
- Support for slow SATA drives
- Moderately faster drives

Looking at that list, it seems to me that the storage industry has been focused on incremental improvement - nobody in the storage array market has been coming up with anything revolutionary. Storage arrays are basically the same as in 2000, but a bit quicker. The following chart [HENN 2007] shows the 'access time gap' between DRAM and disk storage between 1980 and 2005:



This chart shows that, when viewed with a logarithmic scale, neither the performance of DRAM nor disk have actually changed much at all over time. It's not surprising - there are limitations of physics involved here that will prevent a performance improvement of multiple orders of magnitude. The fact is that magnetic storage can never close this gap between DRAM speed and storage speed.

Thanks to the huge increase in storage capacity, and some dubious advertising about efficient sharing of information by storing it in one array, we have now gone from having perhaps two or three databases in a ten shelf storage array to being able to store every bit and byte of data in the enterprise with the same number of shelves. Things have gotten worse!

In very recent times the situation has started to get much more interesting, and it all started with the iPod.

Semiconductor-based Storage

Semiconductor-based storage, or Solid State Disk, has been around for many, many years - there is nothing new about it at all. The part that is new, and the part that was driven by the iPod, is the commoditisation of

flash memory components, and the subsequent shift in affordability. There are various types of SSD, the two main ones being flash memory and DRAM memory as they are the mainstream technologies of today. There are other technologies in the wings, though, and what everybody refers to today as 'SSD' (which is flash memory) can only really be thought of as "version 1.0" of solid-state storage. There has been much written about the advantages of SSD by myself and others so I won't go into all that now, except to highlight the two main features, both of which are a result of eliminating the moving parts of a hard drive:

- It is much more resistant to contention issues
- It is much better at random I/O (no seek time)

I don't want to suggest that SSD is contention-free — flash memory has all manner of blocking operations, particularly when writes are taking place. DRAM is pretty much contention-free, though, and can shoulder a good deal of the potential contention burden when used in conjunction with flash memory.

In a recent blog post [BLOG 001] I made the somewhat bold prediction that SSD, specifically flash memory for now, will hit price parity with magnetic disk within 3-4 years. This is a really big deal, because it means that the barriers to entry are completely eliminated for wide-spread adoption of SSD. If the hardware/software interface can make the best of this change, it will be a major turning point in database performance. It also means that the storage vendors will have to, for the first time, be slaves to Moore's Law, which is something that their current lengthy product development cycles are not used to.

10 Gigabit Ethernet, Good NFS Clients and Direct NFS

One of the age-old objections to deployment of storage using the NFS protocol has been reliability and performance when compared to Fibre Channel. This is something that has not only gone away, but has actually reversed - Ethernet-based storage arrays now have much greater bandwidth interconnects than their Fibre Channel cousins. The old hangovers of unreliable NFS have now been eliminated by decent NFS client software including Direct NFS, and through the deployment of multipath I/O for IP networks.

The biggest advantage of all with NFS deployments is the simplicity. Although an Ethernet-based storage network needs to be thought out just as much as a Fibre Channel SAN, once it is in place it is much less interdependent on other components in the stack to work properly. Anybody that has tried to keep Operating System, HBA drivers, multipath drivers, switch firmware and storage array firmware at compatible levels through an upgrade will understand this complexity very well.

"Fast Enough" Arrays

The incremental improvement in storage arrays hasn't been entirely useless. The capability of even a mid-range storage array is now more than adequate for a great many database requirements, particularly when deployed in conjunction with a generous amount of host memory for caching at the server level. Combined with the simplicity of an NFS-based deployment, these kind of arrays really make sense for a lot of architectural requirements. In the same way that you don't use finest Sherry for cooking, it doesn't make sense to implement a massively complex storage array if your needs can be serviced using a non-complex NFS-based mid-range storage array. Like object-oriented programmers rely upon faster and faster CPUs to counterbalance increasingly abstract code, we can use the improved performance of simple technology to reduce complexity for our requirements.

Speed of Disk Calculations 2010

Over the years I occasionally recalculate an important number for determining the relative performance of hard drives. The number that I calculate is the number of IOPs per 100GB, as this shows the performance improvement over time relative to the capacity increase. This is an important number because relative performance is the single most difficult concept to explain to the people that control the budget. They can

understand capacity, but they can't really get a firm grasp on the performance metrics. IOPs per 100GB give that performance metric an index number.

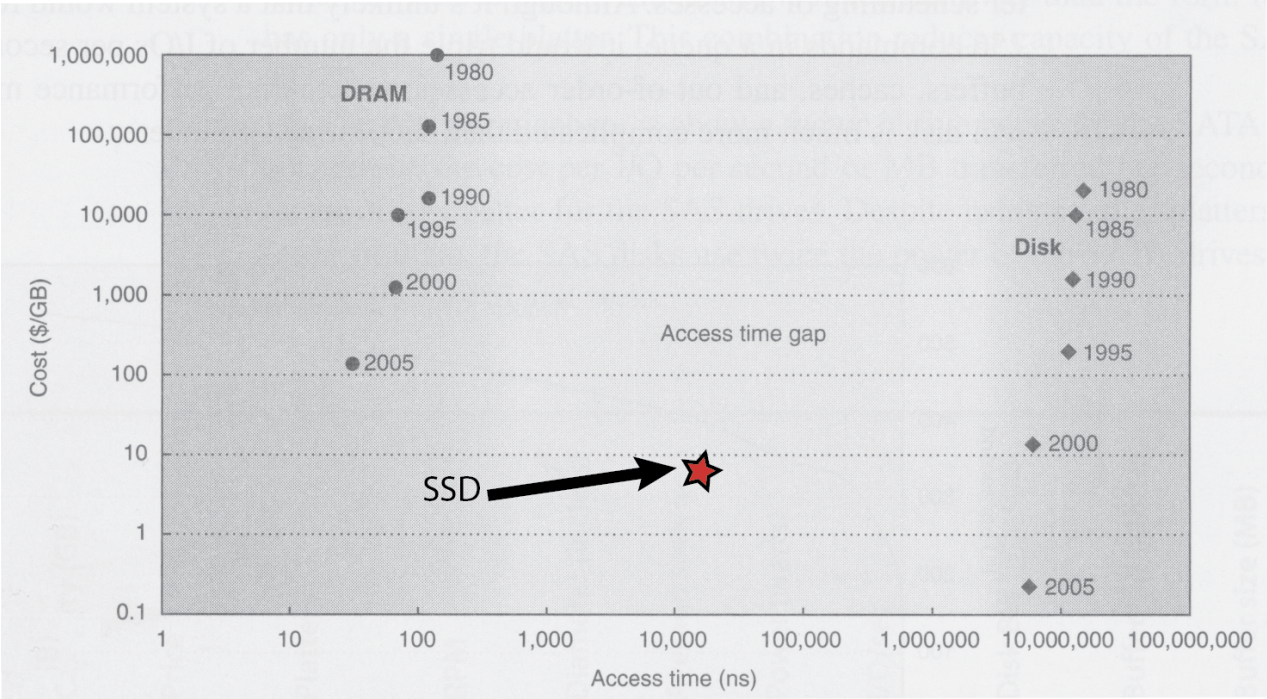
For this paper I have taken my previous results and combined them with that of the STEC Zeus^{IOPS} SSD drive, which is the drive type used in the EMC DMX4 range. This is not the lowest cost semiconductor-based storage out there, and is a fusion of DRAM and flash-based storage, but the 3.5" hard disk-like packaging makes it easy to compare against the magnetic disk drives.

	Seagate Barracuda 4LP (1994)	**Seagate Cheetah 73LP (2001)**	**Seagate Cheetah 15K.4 73GB (2005)**	**STEC Zeus^{IOPS} Enterprise SSD (2010)**
Capacity	2.16GB	73.4GB	73.4 GB	800GB
Rotation Speed	7200rpm	10000rpm	15000rpm	N/A
Rotational Delay (avg, ½ rotation)	4ms	3ms	2ms	N/A
Time to read 32KB (avg, based upon avg bytes/track)	3ms	1ms	0ms	N/A
Seek Time (avg)	9ms	5ms	4ms	N/A
Total time for single 32KB I/O	16ms	8ms	6ms	0.025ms
I/Os per second (conservative)	62	119	172	40,000
I/Os per 100GB	**2,870**	**162**	**234**	**5,000**

There are a couple of things to note in this chart. First, the relative performance of drives has decreased between 1994 and 2001 by a factor of 17.7. This fact nicely summarises the battles that many DBAs have had with their management during that period. Second, the 73GB drive from 2005 was chosen deliberately to be the same size as the 73GB drive from 2001 so that it is possible to see that there has indeed been a performance improvement over that period — approximately 40%. This is a decent enough change in itself, but it's still very much within the realms of evolutionary improvement rather than the revolution required to significantly close the 'access time gap'.

When we introduce the STEC drive, the situation changes considerably. I have conservatively used the **write** IOPs number for for this comparison, and I have ignored the fact that, even with the large DRAM buffer, this drive will probably suffer to some degree from the infamous spikes in write latency associated with flash memory devices (known as the "write cliff"). I have also taken the largest capacity drive available from STEC, which somewhat reduces the number of IOPs per 100GB just because it is 10x larger than the largest hard drive in the comparison. Despite all this conservatism, the STEC still delivers 5,000 IOPs per 100GB, which is twice as good as the scenario modelled on the old 1994 Barracuda drive, and with significantly lower latency per I/O.

Let's plot the performance of the STEC on the chart from earlier:



The star representing the STEC SSD is placed on the chart by hand, approximately in the correct location in terms of access time. I have not paid much attention to the cost axis, but it is very roughly correct as far as I can determine. This shows that the current generation of SSD devices have very successfully closed the access time gap. As the pricing drops further, the cost/GB will eventually become lower than even magnetic media — this is a very significant event in database processing.

And finally: There is one attribute that I did not include in the IOPs calculation, mostly because it was insignificant until now. That attribute is the latency of the interface between drive and host — all figures are based purely upon the performance of the physical drive. This is now a significant factor, as we shall see later in this paper, and so future charts will have this aspect included!

Infiniband Suddenly Makes Sense

Inifiniband (IB) has been around for quite a while now. It was always positioned primarily as a way of interconnecting cluster nodes, and has gained a large degree of acceptance in the High Performance Computing (HPC) cluster[2] market. It also makes a decent Oracle RAC interconnect, but most RAC users don't really need the latency uplift on the interconnect or the bandwidth offered by IB. So the uptake in database computing has not been huge, but now it is about to become mainstream.

This doesn't quite qualify as mainstream yet, but to my knowledge the adoption of IB in the Exadata platform is the first major use of the technology in an easily available commercial database platform. Notably, the IB deployment in Exadata is for both the RAC interconnect *and* the storage network, and as this is a storage paper that's the part we are interested in.

Why is IB so important now, considering that I suggested earlier that Ethernet is a great solution for storage networking? The answer all lies in latency, specifically now that we are on the verge of a "latency revolution" as SSD becomes mainstream. Consider the following latency numbers:

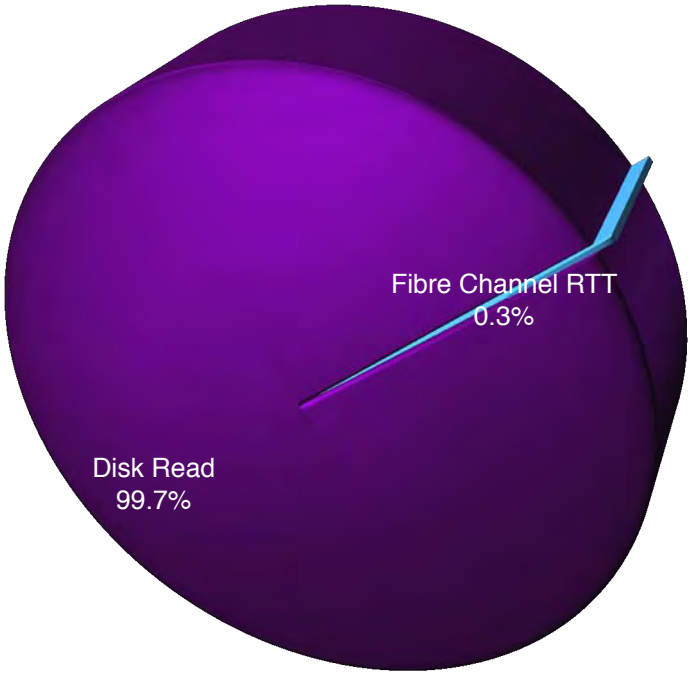
- Disk Read Latency: 6ms (6000µs)
- SSD Read Latency: 25µs

[2] HPC clusters are the very large compute-only clusters used for primarily for scientific research

- Fibre Channel Round-trip Latency: 10-20µs
- Infiniband Round-trip Latency: 1µs

If we plot those numbers onto a few pie charts, it becomes clear what the problem is. First of all, let's look at the latencies for a typical present-day storage array:

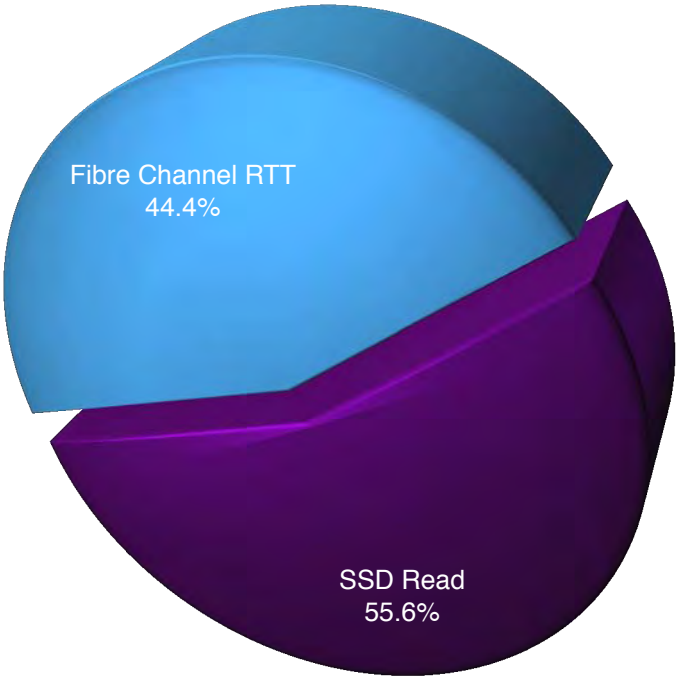
Disk Read Latencies Over Fibre Channel



There is no problem evident when using Fibre Channel in this configuration; the disk latencies are so high that the Fibre Channel latency is minimal. There would be no benefit at all seen from using IB in this configuration from a latency standpoint, though there could still be one for reasons of bandwidth.

Now let's look at some I/O to an SSD drive over Fibre Channel, and see what happens to the relative proportions of latencies:

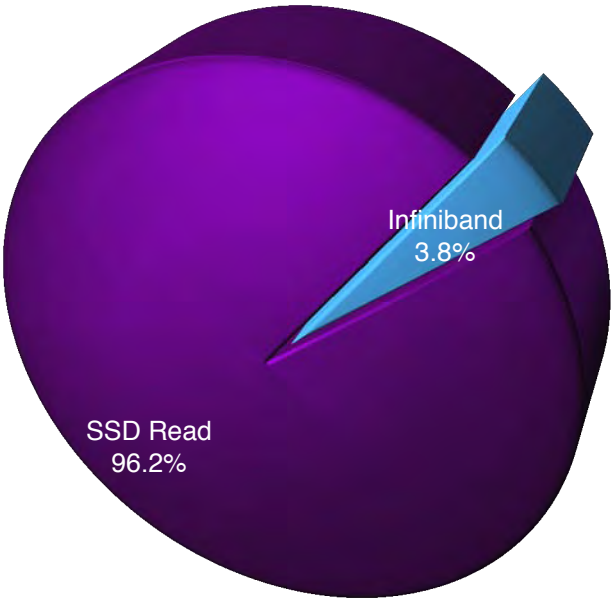
SSD Read Latencies Over Fibre Channel



As you can clearly see, the overhead of the Fibre Channel latency is now a significant factor in the total latency of the read operation. Please note that the Fibre Channel time has not *increased*, but rather the proportion of total I/O time has increased because the I/O time has reduced so much.

That overhead is probably not acceptable, so let's look at I/O to the SSDs via IB:

SSD Read Latencies Over Infiniband



The application of Infiniband makes sense for this reason: It returns the overhead of the interconnect back to the relative levels we are used to with Fibre Channel and magnetic disk.

Host-based Mirroring is Back

Host-based mirroring has made a re-appearance in the form of Oracle's Automatic Storage Management (ASM). Host-based mirroring very much went out of fashion in the late 1990s with the availability of hardware mirroring inside the storage array, but now needs re-evaluating. It became unfashionable largely because of a perceived overhead of host-based mirroring and because of some poor implementations of mirroring software leaving a sour taste in the mouth.

The complex support matrix also caused some headaches and, all-in-all, I guess people just felt it was easier to do it inside the shiny new storage array. I believe all of this has changed with ASM, and there are quite a few reasons to reconsider host-based mirroring in the form of ASM:

- It removes layers in the software stack, reducing complexity and improving reliability and diagnostics
- It gives the DBA direct visibility of the storage and its protection
- There is no performance penalty (writes are performed asynchronously in parallel)
- It enables the use of heterogenous storage on either side of the mirror
- It's free

5. Right Practice: Evaluate Your Storage Needs

In this part of the paper I will discuss various topics to help you evaluate the profile of your workload in a bid for you to understand which aspects of storage are important for your system.

Am I Latency Sensitive or Bandwidth Sensitive? (or both?)

Database applications have varying levels of sensitivity to both latency and bandwidth. It's important to understand where your system sits in this respect, because without this understanding it is impossible to implement the right storage platform for your needs. Data warehouse workloads tend to be *bandwidth-sensitive*, in that the query response times are adversely affected by a drop in bandwidth. Transactional workloads tend to be *latency-sensitive*, in that user response times are adversely affected by increased latency. Why does this matter? Let's see.

For a bandwidth-sensitive application, I/O requests are made in big chunks of, say, 1MB in size or greater. The amount of time it takes that data to be returned to the database is largely a function of the bandwidth of the array and the storage network. The actual service time might be relatively low for the request but, if there is insufficient bandwidth available, the request may queue. For example, let's imagine 100 concurrent requests for 1MB: If each request takes 100ms to service by the array, and the array can service 100 requests concurrently, we can perform 100 x 1MB=100MB every 100ms, or 1000MB per second. However, if there is only a single 2Gbps pipe between the server and the storage we can only physically transport approximately 200MB/s down the cable. This restriction in bandwidth means that the requests are effectively taking 500ms each and the query will take longer to complete [3]. If we add further 2Gbps adapters to this system we will see a continual improvement in query response time until there are a total of five adapters, at which point the system can deliver the full 1000MB/s. Conversely, if we improve the I/O *response time* to 50ms per I/O while still running off one 2Gbps adapter we will see **nil** effect on performance because we remain bottlenecked on bandwidth.

For a latency-sensitive application I/O requests tend to be small. For example, a request for a single leaf block of an index that was not found in the cache. This request might be only 4KB in size, but we might need a couple of hundred of these serviced sequentially during the query execution in order to return the response to the user. If these 200 sequential I/O requests take an average of 4ms each we will complete the query in 4ms x 200=**800ms**. If the requests take take longer, say 10ms, we are no longer providing sub-second response times to the user (10ms x 200=**2000ms**). This is a latency-sensitive application, it is all about how quickly small requests can be serviced. Let's calculate the bandwidth of this application to make the distinction clear: If we were to execute 4KB reads sequentially in a tight loop, each returning in 5ms, we could perform 200 reads per second. 200 reads per second is only 800KB/s bandwidth, very different to the previous example. If we add further HBAs to this system there will be **nil** effect on performance, because we are only using a fraction of the current bandwidth provision. However, if we reduce the latency of our reads to 2ms we will see an increase in IOPs to 500 reads per second.

Of course, many systems are a composite of both types of workload, but it is important to know how each type of request is affected by the latency and the bandwidth of the system.

What is Busy?

With reference back to "Fast Enough" Arrays, it's worth having a reality check on what actually constitutes a 'busy' storage system these days. Many users are convinced that they have a busy I/O system when the reality is that they have quite a modest I/O requirement that does not require all the esoteric complexity of a high-end storage solution. Let's lay out some ballpark numbers:

---

[3] I have ignored the effects of queuing to keep this example simplistic.

- Busy in IOPs terms: over 10,000 I/Os per second
- Busy in bandwidth terms: over 500MB/s

If your I/O requirement is less than both of these numbers, you probably need to have a careful think about the storage architecture that best fits your purpose. Why not make it simpler, and lower cost?

How Fast is 'Fast'?

It is useful to have a bunch of 'must have' numbers in the back of your mind when talking about storage performance. We've all been told by storage administrators that "15ms is an acceptable average read time", have we not? Well **it isn't**, not by anyone's imagination, and running a database on storage that poor will present all sorts of challenges.

OK, so what are acceptable latency numbers for today's hardware? I like to use the following:

- *Latency Numbers*
  - 6-8ms or less single block random reads
  - 1-2ms or less small writes, such as log writes

- *Bandwidth Numbers*
  - 40MB/s or greater full table scan (serial, 1MB reads)
  - 40MB/s or greater redo log write bandwidth

If I encounter an unfamiliar storage implementation, and the basic statistics are very far off these, then there is probably something that needs looking into. It's possible, of course, that the bandwidth numbers are not being met because the workload is not requesting that much throughput, so it's important to be context sensitive when reviewing the bandwidth numbers.

6. The Software/Hardware Interface

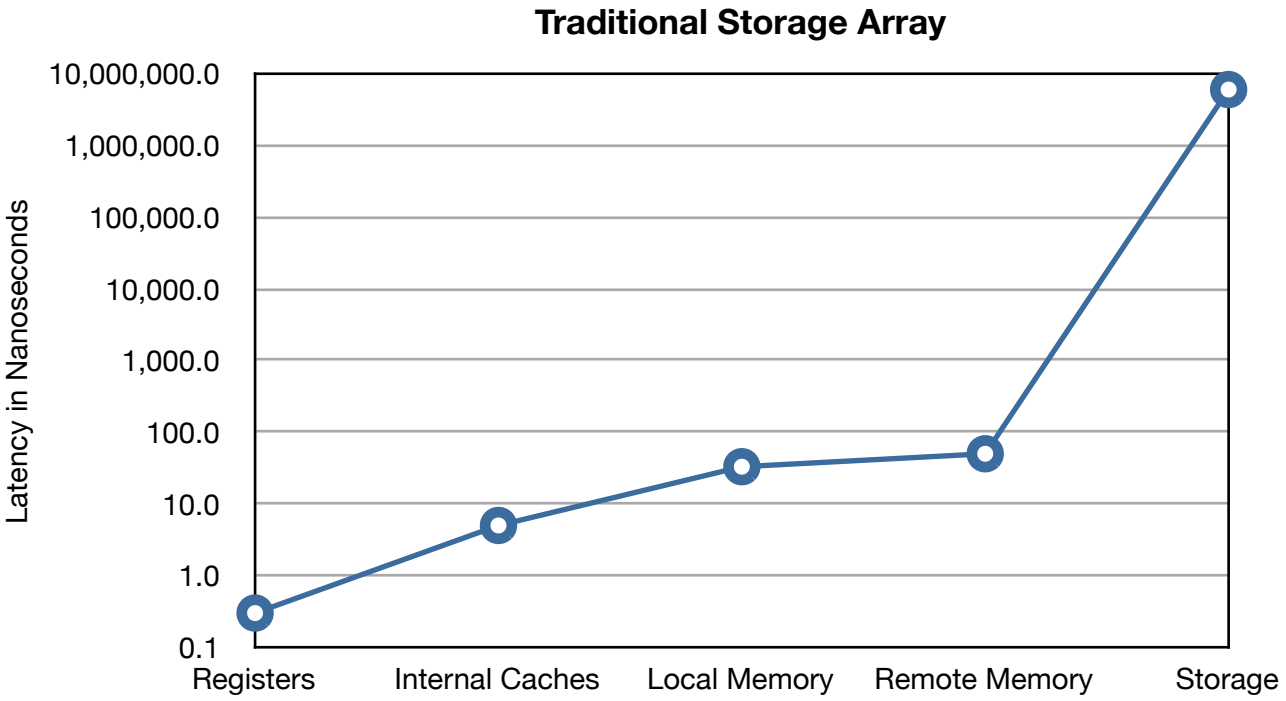
With the ‘latency revolution’ upon us, it is time to re-evaluate how we store and access data. Software is primarily written with a specific hardware configuration in mind. If this hardware evolves over time, the software can normally be tweaked to work well with this new evolution. When *revolutionary* changes take place, however, it normally implies some kind of fundamental alteration to the software to optimally take advantage of the hardware change. For example, moving from a uniprocessor to a multiprocessor architecture implies significant change to an operating system kernel because data can be modified by more than one engine at a time. The assumptions about locking made for the uniprocessor simply don’t work any more. The same is true when the ratios between key layers of the memory hierarchy takes place— Non-Uniform Memory Access (NUMA) was the last time that happened, where different portions of system memory have different access latencies dependent upon physical location in the system. Operating System and database software are still being modified to make the best of this new architecture which is the default for practically all servers these days. Now the latency ratios have changed again with the adoption of SSD.

Reviewing the Memory Hierarchy

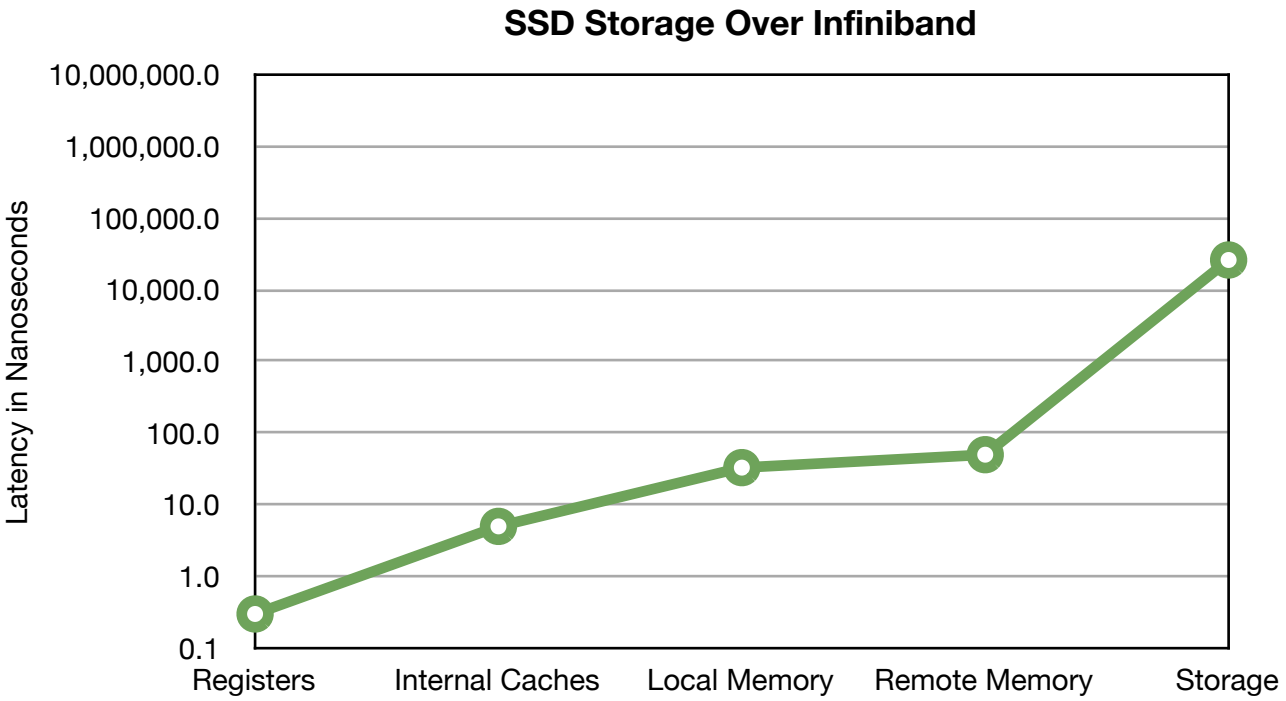
Memory hierarchy is the name given to the relationships in latency between different layers of the server memory architecture, including ‘persistent memory,’ also know as storage. The relative ratios in the following table are calculated for a recent Intel Nehalem processor, as follows:

Component	Latency (ns)
Registers	0.30
Internal Caches	5.00
Local Memory	33.00
Remote Memory	50.00
IB RDMA	1,000.00
Fibre Channel	20,000.00
SSD Read	25,000.00
Disk Read	6,000,000.00

When plotting these on a logarithmic chart, a very clear jump in latency is evident at the storage tier, which is defined as the sum of storage network latency and disk read latency:



Conversely, when plotting the SSD storage (using IB as the interconnect), the trend is almost linear (albeit on a logarithmic chart), with no sharp rise for the storage latency:



This is really just another way at looking at the ‘access time gap’ defined in [HENN 2007], but applied to the whole memory hierarchy. It shows that the memory hierarchy has moved from a span of eight orders of magnitude to only five, even when looking all the way back to the registers.

Effects On Software Architecture

This three orders of magnitude reduction in latency has an implication on the software architecture, specifically in terms of when it is algorithmically optimal to move data between memory and storage. Let's consider caching, for example. In current disk caching algorithms, the break point is simple: Going to disk should be avoided at all costs. The latency to go to disk is so incredibly high (a factor of 160,000 compared to even remote memory access) that it makes sense to keep as much in memory as possible. Moving to SSD brings this cost down to a relatively tiny 500x. Though it is still clearly an advantage to get a memory-based cache hit, the advantage is substantially reduced compared to magnetic storage. It is reduced sufficiently that it might make more sense to have, for example, a proportion of free buffers reserved in the cache at any one time, to save the cost of creating space for urgently required buffers. Or to speculatively fill more of the cache with buffer that *might* be needed, with the knowledge that the cache management cost of 'getting it wrong' is not terribly high. Or, to really push it out there, what if it now makes sense to only cache index blocks and always go to SSD for the data blocks?

The ability to maximise the gain of low-latency storage can only be maximised through a hardware **and** software solution. The winners in the near future will be those that successfully integrate the two and implement algorithms that consider the reduced latency.

7. Things To Think About

Fibre Channel is History

As shown in the section on Infiniband, Fibre Channel imposes an unacceptable overhead on low-latency storage, and also implies significant complexity. Granted, there is a great deal of Fibre Channel out there, and it forms the backbone of most datacentres, but that does not mean that it is still a current product. At one stage datacentres used to be full of serial TTY lines and 10base-5 Ethernet, but they are not anymore. The same is true of Fibre Channel. I advocate a two tier storage network for the future: Infiniband for latency-sensitive applications, and 10Gbps Ethernet for other requirements, including the general inter-host fabric. This pattern also trickles down into my thoughts on Storage Arrays...

"Storage Arrays" Are History

Recent developments make me believe that the conventional storage array will soon be a thing of the past. Storage arrays are essentially expensive high-latency devices that are made unmanageably complex through aggressive asset sweating and extreme cohabitation by too many applications.

The cohabitation problem is a strange one: Why do we put so many systems' storage onto a single storage array? What's the benefit here? The storage vendors will explain it thus:

1. All applications can access the aggregate bandwidth of all other applications
2. More efficient use of storage and other resources
3. Simple management because it's one box
4. Enhanced sharing of information

Let's start at the bottom: Sharing of information requires hosts, potentially even multiple tiers of hosts. If hosts are required, there is no advantage having the information on the same storage array. Next: Simple management is not the domain of a single array, it is a software problem. There is no reason that multiple arrays should be more difficult to manage than a single device with thousands of attached drives. Items (1) and (2) are interesting. On the one hand I can see that advantage, but on the other hand I have also seen the myriad problems associated with doing exactly that. If all applications have access to all the resources of the other applications, isn't that an unmanageable, anarchic mess? When an advantage becomes a disadvantage, is it really worth having?

The "asset sweating" issue needs to be taken in hand with the "expensive" attribute. The only reason that storage arrays are so aggressively filled to the brim is that they are so expensive in the first place. Would we really do that if the array were cheap?

The latency issue is a big one. Storage arrays are built from the ground up to allow access to slow magnetic disk. Now that we have low latency storage devices, the current crop of storage arrays just serve as another bottleneck between the host and the storage (in addition to their Fibre Channel interfaces). In the same way that software needs to be redesigned to work with SSDs, storage arrays need to be redesigned from scratch if they are to work optimally with these devices. Not only that, but the development cycles of the storage vendors will now have to run at the rate determined by Moore's Law instead of the more sedate pace they are used to with magnetic storage.

Even if the array were redesigned, it may not be enough. General-purpose storage arrays will still be trying to be all things to all men. That philosophy may not work too well in the light of specialist products such as Exadata which are successfully able to combine low-latency storage devices and high bandwidth data scanning directly into the appropriate tier of the database kernel.

Did I mention that storage arrays are expensive? This means they are also not the right solution for 'web scale' architectures, which are designed for failure from the outset and use cheap, commodity parts.

At least they are reliable though, right? Not really — storage arrays are still a single-box solution and they do fail. The impact of these failures, particularly if there is wide-scale data loss, is just very large.

Another way to look at those last two points (expensive and unreliable) is to address the underlying requirement a different way. Instead of one very expensive, fully-redundant storage array that still fails, why not replace it with *two* or more cheaper storage platforms in conjunction with host-based mirroring via ASM? Admittedly ASM only reads from the 'preferred' side of the mirror, but how about making that primary side an SSD (for performance) and the other side traditional disk (for redundancy)? More redundancy and more performance for less money than a current storage array.

I think that the storage array will always be with us in some form — we must have external storage to protect against host failure — but I think the ground has moved and the array must move with it.

The Return of the MicroSAN

In [SANE 2000] I advocated the use of a MicroSAN, a private area within the SAN, to ensure performance and reliability for each application/database. I still think that's a good idea, but now I am retreating from the recommendation to have it residing inside a big storage array — I just don't see the advantage of *having* the big storage array in the first place. If you don't gain advantage of having all the data in one array, you should not subject yourself to the associated challenges. Why not evaluate each of the requirements and decide where they should reside?

A large number of database requirements could just be easily fulfilled using NAS devices, using NFS, over 10Gbps Ethernet (or even 1Gbps Ethernet for small requirements). I think that NAS makes a great deal more sense than a Fibre Channel storage array, because the administration is so much easier and the performance is more than adequate for a large percentage of database needs. NAS devices can be easily deployed and scaled by having a number of smaller units rather than a very large single device.

If you have a workload that is either high-bandwidth or requires low latency, then the argument for a MicroSAN becomes even stronger. It makes no sense to share resources when the criticality of the system is so high. In the past you may have dedicated a whole storage array for such a purpose, but it's not the right way to do it anymore, for the many reasons mentioned in this paper. Perhaps Exadata is the right MicroSAN for your requirement — it is even straightforward to implement multiple MicroSANs in one Database Machine system after all. Or perhaps the most appropriate solution is to build something with Infiniband-attached SSDs, even if it is only as one side of an ASM mirror as mentioned earlier?

Conclusion

Hopefully this paper has highlighted some of the more interesting topics in the world of database storage. Storage has been too complex for quite a while now, but now we have a chance to strip back some of that complexity and to build much higher performance storage solutions than we are used to. There is certain to be an amazing amount of change in this area of the market over the next few years, particularly when SSD technology approaches comparable costs to magnetic storage. Now is the time to start thinking about strategies for how future databases should be deployed.

About The Author

James Morle is the founder of Scale Abilities Ltd, a full-stack Oracle consulting company based in the UK. He has over 20 years experience working on some of the world's largest and most complex Oracle-based systems. He is the author of Scaling Oracle8i, co-author of Oracle Insights, co-founder of the OakTable Network and an Oracle ACE Director. He can be reached by email on James.Morle@scaleabilities.co.uk

8. Bibliography

[SAME 2000] - http://www.oracle.com/technology/deploy/availability/pdf/oow2000_same.pdf

[SANE 2000] - http://www.scaleabilities.co.uk/whitepapers/Sane_SAN_WP_Morle.pdf

[BLOG 0001] - http://jamesmorle.wordpress.com/2010/05/29/flash-storage-will-be-cheap-the-end-of-the-world-is-nigh/

[HENN 2007] - Page 359, Computer Architecture: A Quantitive Approach, 4th Edition, John L Hennessy and David A Patterson. First published 1990. Copyright Elsevier Press 2007 ISBN-13 978-0-12-370490-0