The White Papers

Implementing RAID on Oracle

By Gaja Krishna Vaidyanatha

Contents

Implementing Raid On Oracle Systems

Gaja Krishna Vaidyanatha, Quest Software Inc.

Overview

RAID, the final frontier…these are the voyages of an Oracle DBA, who boldly dealt with mass storage vendors and their claims, and achieved optimal I-O performance by application of common sense. Many misconceptions surround RAID technology. This paper will define what RAID is and what it is not. It will also explain how RAID works including the differences between each RAID level.

Further, it will provide you with implementation recommendations for each of the RAID types, by sharing real-life examples that illustrate Input-Output (I-O) optimization on Oracle's Very Large Databases (VLDBs). The primary goal of this paper is to elucidate the details of successfully solving I-O problems in a consistent fashion with the appropriate RAID configurations. Even though this paper is not intended to make you a "*RAID Expert*", it will provide you with enough information to have an intelligent dialogue with your System/Storage Administrator.

It is my intention to dispel the myths surrounding the marriage of Oracle and RAID and allow you to get on with more important activities in life, e.g. having a beer or two. All the configurations in this paper have been proven against Oracle databases from 200 GB – ~1 TB, but these principles are applicable to all Oracle databases.

What Is RAID?

Besides being a leading brand insecticide, RAID is the technology for expanding the capacity of the I-O system and providing the capability for data redundancy. Depending on whom you ask, it stands for *Redundant Array of* ***Inexpensive*** *Disks* or *Redundant Array of* ***Independent*** *Disks*. But since you are asking me, I am going with *Inexpensive,* because that makes my accounting friends feel better when I ask for more disks. And I will.

Conceptually, RAID is the use of 2 or more physical disks, to create 1 logical disk, where the physical disks operate in tandem to provide greater size and more bandwidth. RAID has become an indispensable part of the I-O fabric of any system today and is the foundation for storage technologies, supported by many mass storage vendors. The use of RAID technology has re-defined the design methods used for building storage systems that support Oracle databases.

What RAID Is Not?

RAID is NOT the cure to all I-O issues. It is not a panacea for all of your I-O problems. One myth about RAID is that it can eliminate the need for the database or system administrator to concern themselves with I-O issues. This is particularly the case with proprietary RAID based systems. Like me, you have probably been in at least one sales presentation, where the sales person promised that, if you bought their brand of disk arrays and ran their "*easy to use*" configuration program, you could just walk away and never have to consider any I-O issues again. They'll tell you that now you can do more important things like tuning the application and they will take on all of your I-O issues. And, you never see them again!

True, the benefits of tuning the application cannot be compared to tuning anything else. However, you will be rudely awakened to the fact that some of the system-wide issues performance issues that you are trying to solve, got worse, because of the "*blind faith*" you placed on storage vendors. Remember, they are mortals, like you and me and they definitely do not possess magical powers over their own disk drives! A disk cannot spin faster than its design specification, cannot service data-transfers that exceed its I-O bandwidth and cannot support a zillion concurrent I-O operations. This is true regardless who your storage vendor is.

One such example of falling prey to blind faith, is the story that storage vendors tell you that "*you do not have to worry about where you physically place your tables and indexes*" and that it could be in the same set of drives. They will further claim that their storage system and RAID will take care of all of your administrative pains and relieve you of tablespace placement headaches. Wrong! In my experience, and I have had the opportunity to prove this time and again to many of my esteemed customers - physical independence of tables and indexes is required for applications that utilize the tables and indexes in tandem. There are no two ways about it.

Batch environments are typical of such use where the application performs index range scans on large tables followed by corresponding table look-ups. In this case, physically storing tables and indexes on the same set of disks will significantly hurt performance. This is true, even if the disk architecture supports storing the "*entire track's data*" in its track buffer cache, to alleviate the cost of expensive seeks. RAID can and will provide excellent I-O performance, when implemented with the same care that database administrators have historically taken in designing simple disk solutions, i.e., separate tables from their corresponding indexes, if they are accessed in tandem.

On the other hand, it can wreak havoc when implemented in a haphazard fashion. Some things in life do not change…this is one of them. It must also be noted here that RAID does not possess "*the divinity*" to protect your system from failures related to disk bus, host adaptor, interface, disk controller, cooling system, power system, host, application and human beings. Thought you should know!

Why Should You Care About RAID?

Though, one often sees database or system administrators choosing RAID to be "*technically in vogue*" or to impress their bosses, let's focus on the two main technical reasons for making the jump to RAID. These are **scalability** and **high availability** in the context of I-O and system performance. As discussed earlier, scalability and availability are the secrets to job security in our positions as database administrators or consultants.

No database administrator was ever fired (I hope not) because the database was always "up" or because the system continued to provide fast responses regardless of the load. Over the past few years, the size of Oracle databases has grown dramatically. In fact, Oracle now supports databases in the petabyte range. With this growth, has arrived increased complexity, and with this increase in size and complexity, has come the challenges of managing the I-O required by these mammoth systems. If we were to ignore all other factors in tuning, the size alone requires new techniques to scale I-O performance to the growing demands of applications.

Figure 1 illustrates the point driven in the aforementioned paragraph, by comparing the size ceilings between two major releases of Oracle and why RAID is becoming indispensable in an Oracle system:

Item	**Oracle7**	**Oracle8 and Above**
Database Size	32 terabytes	The product of (65533 files * Size of the largest file supported on your operating system platform)
Tablespaces	1022	65536
Data files per database	1022	65533
Columns per table	254	1000
Columns per index	16	32
Extents per table	Unlimited**	Unlimited
LOB columns per table	1 LONG/LONGRAW	1000 LOBS
Maximum LOB size	2 gigabytes	4 gigabytes
CHAR column	255 bytes	2000 bytes
VARCHAR2 column	2000 bytes	4000 bytes

Figure 1

* Operating System dependent

** Available in Oracle 7.3 and above

"*The scalability of an Oracle System is a direct function of the scalability of the storage system*"

Add to the increased size and complexity, terms such as "*mission critical*", *"e-business"* and *"24x7xforever*" and we find more benefits to using RAID. Corporate system availability needs have increased to all time highs with more data being incorporated into the "*enterprise data stores*" and more users accessing that data.

With the growth in data and expansion of business use of that data, enterprises have become very dependent on their applications and databases. An availability requirement of 99.999% (the infamous five nines) or higher for mission-critical systems, is not unusual.

These requirements mean that disk failures cannot become "*show stoppers.*" Failures at the disk level resulting in database or application downtime translate into revenue losses and "*ill will.*" Because of intense market competition and regulatory issues, many businesses may not be able to sustain these failures, even for a short time. The point made here is very simple – storage systems need to scale I-O performance commensurate to the increase in data volume without any loss of availability. RAID offers exactly that.

The 3 Main Concepts In RAID

When you talk RAID, 3 terms are important and relevant. One is *striping,* another is *mirroring* and the third is *parity*.

What Is Striping?

Striping is the process of breaking down data into pieces and distributing it across multiple disks that support a logical volume – "*Divide, Conquer & Rule*". This often results in a "*logical volume*" that is larger and has greater I-O bandwidth than a single disk. It is purely based on the linear power of incrementally adding disks to a volume to increase the size and I-O bandwidth of the logical volume. The increase in bandwidth is a result of how read/write operations are done on a striped volume.

Imagine that you are in a grocery store. With you are about two hundred of your closest friends and neighbors all shopping for the week's groceries. Now consider what it's like when you get to the checkout area and find that only one checkout line is open. That poor clerk can only deal with a limited number of customers per hour. The line starts to grow progressively. The same is true of your I-O sub-system. A given disk can process a specific number of I-O operations per second. Anything more than that and the requests start to queue up. Now stop and think about how great it feels when you get to the front of the store and find that all 20 lines are open. You find your way to the shortest line and your headed out the door in no time.

Striping has a similar effect to your I-O system. By creating a single volume from pieces of data on several disks, we can increase the capacity to handle I-O requests in a linear fashion, by combining each disk's I-O bandwidth. Now, when multiple I-O requests for a file on a striped volume is processed, they can be serviced by multiple drives in the volume, as the requests are sub-divided across several disks. This way all drives in the striped volume can engage and service multiple I/O requests in a more efficient manner. This "*cohesive and independent*" functioning of all the drives in a logical volume is relevant for both read and write operations. It must be noted that striping by itself, does not reduce "*response time*" for servicing I-O requests. However, it does provide predictable response times and facilitates the notion of better performance, by balancing I-O requests across multiple drives in the striped volume.

Figure 2 depicts a 4-way striped volume (v1) with 4 disks (1-4). A given stripe of data (Data1) in a file on v1 will be split/striped across the 4 disks, into 4 pieces (Data11-Data14).

Disk1	**Disk2**	**Disk3**	**Disk4**
Data11	Data12	Data13	Data14
Data21	Data22	Data23	Data24

Figure 2

What Is Mirroring?

Mirroring is the process of writing the same data, to another "member" of the same volume simultaneously. Mirroring provides protection for data by writing exactly the same information to every member in the volume. Additionally, mirroring can provide enhanced read operations because the read requests can be serviced from either "*member*" of the volume. If you have ever made a photocopy of a document before mailing the original then you have mirrored data. One of the common myths with mirroring, is that it takes "*twice as long*" to write. But in many performance measurements and benchmarks, the overhead has been observed to be around 15-20%.

Figure 3 illustrates a 4-way striped mirrored volume (v1) with 8 Disks (1-8). A given stripe of data (Data1) in a file on v1 will be split/striped across the Disks (1-4) and then mirrored across Disks (5-8). Disks (1-4) and (5-8) are called "*Mirror Members*" of the volume v1.

Disk1	**Disk2**	**Disk3**	**Disk4**	**Disk5**	**Disk6**	**Disk7**	**Disk8**
Data11	Data12	Data13	Data14	Data11	Data12	Data13	Data14
Data21	Data22	Data23	Data24	Data21	Data22	Data23	Data24

Figure 3

What Is Parity?

Parity is the term for error checking. Some levels of RAID, perform calculations when reading and writing data. The calculations are primarily done on write operations. However, if one or more disks in a volume are unavailable, then depending on the level of RAID, even read operations would require parity operations to rebuild the pieces on the failed disks. Parity is used to determine the write location and validity of each stripe that is written in a striped volume. Parity is implemented on those levels of RAID that do not support mirroring.

Parity algorithms contain Error Correction Code (ECC) capabilities, which calculates parity for a given '*stripe or chunk*' of data within a RAID volume. The size of a chunk is operating system (O-S) and hardware specific. The codes generated by the parity algorithm are used to recreate data in the event of disk failure(s). Because the algorithm can reverse this parity calculation, it can rebuild data, lost as a result of disk failures. It's just like solving a math problem what you know the answer (checksum) and one part of the question e.g. 2+X =5, what is X? Of course, X=3.

Figure 4 depicts a 4-way striped RAID 3 volume with parity – v1 with 5 Disks (1-5). A given stripe of data (Data1) in a file on v1 will be split/striped across the Disks (1-4) and the parity for Data1 will be stored on Disk 5. There are other levels of RAID that store parity differently and those will be covered in the following sections.

Disk1	**Disk2**	**Disk3**	**Disk4**	**Disk5**
Data11	Data12	Data13	Data14	Parity1
Data21	Data22	Data23	Data24	Parity2

Figure 4

Putting It All Together

Striping yields better I-O performance, mirroring provides protection and parity (when applicable) is a way to check the work. With these 3 aspects of RAID, we can achieve scalable, protected, highly available I-O performance.

The Types Of RAID

RAID can be implemented as software-based, where the control software is usually either bundled with the O-S or in the form of an add-on, like a "*Volume Manager*" (e.g., Veritas Volume Manager). This type of RAID is also known as host-based RAID. This type of implementation does impose a small overhead, as it consumes Memory, I-O bandwidth and CPU on the host where it is implemented. Normally, this overhead is something that is not alarming, but should be factored into the resource capacity plans of the host.

RAID implemented by hardware in the form of micro-code present in dedicated disk controller modules that connect to the host. These controllers are internal to the host where RAID is implemented. This type of RAID is also known as embedded controller-based RAID.

RAID can also be implemented using controllers that are external to the host where it is implemented. This implementation is "bridge-based" implementations and is not preferred, as they incur longer "service times" for I-O requests. This is due to the longer I-O paths from the disks to the host. This type of implementation is usually typical of I-O sub-systems that are "half-Fiber" and "half-SCSI". It is also common to see this implementation on storage systems that support multiple hosts running multiple operating systems. The "bridges" also have a tendency to become saturated, when the system is busy with I-O requests. Hardware-based RAID should be preferred over software-based or host-based RAID over bridge-based RAID.

The Levels Of RAID

Initially, RAID was a very simple method of logically joining of two or more disks, but like all things in our industry more choices were needed to meet different requirements. Today, RAID levels usually range from 0 to 7, and because of the peculiar way that we count in our world it gives us more than 8 choices. The differences between the various levels, is based on varying I-O patterns across the disks. These I-O patterns by their inherent nature offer different levels and types of protection and performance characteristics.

RAID 0: This level of RAID is a 'normal' filesystem with striping, in which data loss is imminent with any disk failure(s). Simply put, it is data striped across a bunch of disks. This level provides good read/write performance but no recoverability. Figure 1 illustrates the concept of striping in a RAID 0 volume.

RAID 1: In very simple terms this level of RAID provides mirroring and thus full data redundancy. This is often called a 'mirrored disk'. In most cases, the volume that the operating system sees is made up of two or more disks. However, this is presented to an application or a database as a single volume. As the system writes to this '*volume*', it writes an exact copy of the data to all members in the volume. This level of RAID requires twice the amount disk storage as compared to RAID 0. Additionally, some performance gains can be reaped from parallel reading of the two mirror members. RAID 1 doubles the capacity of processing read requests from the volume when compared to not having mirrored members. There are no parity calculations involved in this level of RAID. Figure 2 illustrates the concept of mirroring in a RAID 1 volume.

RAID 0+1: "*Stripe First, Then Mirror What You Just Striped*". This level of RAID combines levels 0 and 1 (striping and mirroring). It also provides good write and read performance and redundancy without the overhead of parity calculations. On disk failure(s), no reconstruction of data is required, as the data is read from the surviving mirror. This level of RAID is the most common implementation for write-intensive applications and is very widely used. The most common complaint is the cost, since it requires twice as much space. To justify this cost, you will have to spend some time understanding the performance requirements and availability needs of your systems. Figure 3 illustrates the concept of striping and mirroring in a RAID 0+1 volume. Notice however, that if one of the pieces (say Data11 on Disk 1) becomes unavailable due to a disk failure on Disk 1, the entire mirror member (located on Disk 1-4) becomes unavailable. This is a very important consideration as the loss of an entire mirror member, reduces the I-O servicing capacity of the volume by 50%.

RAID 1+ 0: "*Mirror First, Then Stripe Over What You Just Mirrored*". This level of RAID has the same functionality as RAID 0+1, but is better suited for high availability. This is because on the loss of 1 disk in a mirror member, the entire member of a mirrored volume does not become unavailable. It must be noted here that, the loss of 1 disk of a mirrored member, does not reduce the I-O servicing capacity of the volume by 50%. This should be preferred method for configurations that combine striping and mirroring, subject to hardware limitations.

Figure 5 illustrates the concept of RAID 1+0. Notice here that if one of the pieces (say Data11 on Disk 1) becomes unavailable due to a disk failure on Disk 1, the other disks in the member (Disk 2-4) do not become unavailable. Only Disk 1 is unavailable and Disks 2-4 are still available to service I-O requests. It must be noted here that RAID 10 is a derivative of RAID 1+0 with similar performance and availability characteristics, but with some implementation differences.

Disk1	**Disk2**	**Disk3**	**Disk4**	**Disk5**	**Disk6**	**Disk7**	**Disk8**
Data11	Data11	Data12	Data12	Data13	Data13	Data14	Data14
Data21	Data21	Data22	Data22	Data23	Data23	Data24	Data24

Figure 5

RAID 2: This level of RAID incorporates striping, and the redundancy/protection is provided through parity. This method requires less disk space compared to RAID 1, but the need to calculate and write parity, will make writes slower. This level of RAID was one of the early implementations of "*striping with parity*" using the famous hamming code technique, but was later replaced by RAID 3, 5 and 7. This level of RAID is very rarely implemented.

RAID 3: In this level of RAID, the ECC algorithm calculates parity to provide data redundancy as in RAID 2, but all of the parity is stored on 1 disk. The parity for this level of RAID is stored at the bit/byte-level as opposed to the block/chunk level. RAID 3 is slowly gaining popularity but is still not very widely used. It is best suited for data mart/data warehouse applications that support a few users but require sequential bulk I-O performance (data-transfer intensive). When full table scans and/or index range scans are the norm for a given application and the user population is small, RAID 3 may be just the ticket. Figure 3 illustrates the physical layout and characteristics of a RAID 3 volume.

RAID 4: This level of RAID is the same as RAID 3 but with block level parity and is very rarely implemented.

RAID 5: This is by far one of the most common RAID implementations today. In this level of RAID, data redundancy is provided via parity calculations like in RAID 2, 3, 4, and 7, but the parity is stored along with the data. Hence, the parity is distributed across the number of drives configured in the volume. For many environments RAID 5 is very attractive, because it results in minimal loss of disk space to parity values, and it provide good performance on random read operations and light write operations. RAID 5 caters better to Input Output Per Second (IOPS) with its support for concurrently servicing many I-O requests. It should not be implemented for write-intensive applications, since the continuous process of reading a stripe, calculating the new parity and writing the stripe back to disk (with the new parity), will make writes significantly slower.

An exception to this rule that requires consideration is when the I-O sub-system has significant amounts of "*write cache*" and the additional overhead imposed by the ECC algorithms is measured and confirmed by analysis to be minimal. The definition of "*significant*" is left to the discretion of the reader, but in general a write cache sized in many gigabytes can be considered significant.

On many systems, however, the performance penalty for write operations can be expensive even with a "*significant write cache*" depending on the number of writes and the size of each write. RAID 5 is best suited to read-only applications. Like RAID 3, it is best suited for data mart/data warehouse applications, but it can support many application users performing random I-O instead of sequential I-O.

Figure 6 is a 4-way striped RAID 5 volume where data and parity is distributed. The following illustrates the physical placement of stripes (Data1 - Data4) with their corresponding parities distributed across the 5 disks in the volume.

Disk1	**Disk2**	**Disk3**	**Disk4**	**Disk5**
Data11	Data12	Data13	Data14	Parity1
Data21	Data22	Data23	Parity2	Data24
Data31	Data32	Parity3	Data33	Data34
Data41	Parity4	Data42	Data43	Data44

Figure 6

RAID 6: In this level of RAID, parity is calculated using a more complex algorithm and redundancy is provided using an advanced multi-dimensional parity method. RAID 6 stores 2 sets of parity for each block of data and thus makes writes even slower than RAID 5. However, on disk failures, RAID 6 facilitates quicker availability of the drives in the volume (after a disk failure), without incurring the negative performance impact of re-syncing the drives in the volume. This level of RAID is very rarely implemented.

RAID 7: This is a better implementation of RAID 3. Since read and write operations on RAID 3 are performed in a synchronous fashion, the parity disk can bottleneck during writes (it depends!), RAID 7 allows asynchronous reads and writes, which inherently improves overall I-O performance. RAID 7 has the same characteristics as RAID 3, where all of the parity is stored on a dedicated drive. RAID 7 is relatively new in the market and has great potential to be a great candidate for implementations that historically have chosen RAID 3. With RAID 7, you can "*have the cake and eat it too*". One can reap the data-transfer benefits of RAID 3 and not lose the transactional I-O features that RAID 7 offers.

RAID-S: If you are using EMC Storage arrays, then this is your version of RAID 3/5. It is well suited to data mart/data warehouse applications. This level of RAID should be avoided for "*write intensive*" or "*high-volume transactional*" applications for the same reasons as any RAID 5 implementation. EMC storage solutions are usually configured with large write caches, but generally speaking, these write caches are not large enough to overcome the additional overhead of the parity calculations during writes.

Auto RAID: With Auto RAID (implemented by HP), the controller along with the intelligence built within the I-O sub-system, dynamically modifies the level of RAID on a given "*disk block*" to either RAID 0+1 or RAID 5, depending on the near historical nature of the I-O requests on that block. The recent history of I-O patterns on the disk block is maintained using the concept of a "*working set*" (which is a set of disk blocks). For obvious reasons, there is one working set each for reads and writes, and blocks keep migrating back and forth between the two sets, based on the type activity. A disk block in this context is 64K in size.

Said in a different way, a RAID 5 block can be dynamically converted into a RAID 0+1 block, if the "*intelligence*" determines and predicts that the block will be primarily being accessed for writes. The controller can also perform the converse of the previous operation, namely converting a RAID 0+1 block into a RAID 5 block, if it determines and predicts, that the block will be primarily accessed for reads. To support this configuration, all the drives in the array are used for all RAID volumes that are configured on that array. This means that physical drive-independence across volumes cannot be achieved.

While this implementation of RAID relieves a significant amount of work and maintenance for the System Administrator and the Oracle Database Administrator, care should be exercised while implementing this on hybrid Oracle systems which can be "*write and read intensive*", at the same time.

If your system becomes suddenly and seriously "*write intensive*" after a period of "*read intensive*" activity, the conversion process may not occur immediately and your blocks may get stuck in RAID 5 (from the read phase) even though you are in the write phase. This happens when the system load is high, and the conversion process defers to a quieter period. This behavior may be prevalent on busy hybrid Oracle systems.

If you implement this RAID technology on heavy-duty Oracle systems, be prepared for unpredictable changes in I-O performance, unless your system is normally "*write intensive*" or normally "*read intensive*" with occasional changes like those nightly batch jobs. So, when implementing Auto RAID, every effort should be taken to segregate write-intensive and read-intensive components on separate arrays.

Figure 7 is a summary of the various levels of RAID:

Level of RAID	Functionality
RAID 0	Striping, No recoverability, Require read/write performance without recoverability
RAID 1	Mirroring, Recoverability, Require write performance
RAID 0+1/1+0	Combination of 0 and 1, Recoverability, Require read and write performance, Very widely used, 1+0 is better than 0+1 for availability
RAID 2	Early implementation of striping with parity, Uses the Hamming Code Technique for parity calculations, Was replaced by RAID 3, RAID 5, and RAID 7, Very rarely implemented
RAID 3	Striping with bit/byte-level parity, Dedicated parity disk, Recoverability, Require read performance for bulk sequential reads, Require data-transfer over IOPS, Not widely used but gaining popularity
RAID 4	Striping with block-level parity, Dedicated parity disk, Recoverability, Very rarely implemented
RAID 5	Striping with block-level parity, Distributed parity across the number of disks in the volume, Recoverability, Require read performance for random reads that are small in nature, Require IOPS over data-transfer, Very widely used
RAID 6	Striping with block-level multi-dimensional parity, Recoverability, Slower writes than RAID 5,Very rarely implemented
RAID 7	Same as RAID 3, but with better asynchronous capability for reads and writes, Significantly better overall I-O performance when compared to RAID 3, Significantly more expensive than RAID 3
RAID-S	EMC's implementation of RAID 3/5
Auto RAID	Hewlett Packard's (HP) automatic RAID technology that auto configures the I-O system based on the nature and type of I-O performed on the disk blocks within the RAID Array

Figure 7

Oracle & RAID

Because RAID is either implemented as host-based (software-based) or hardware-based (best), it is transparent to the Oracle RDBMS. All RAID-specific operations are handled "*under the covers*" by the hardware, the O-S or the volume management control software. The manner in which I-O operations are performed and managed, contributes tremendously to how well Oracle performs. Each component of an Oracle database is best suited to a specific RAID implementation. This means that, "one size" definitely does not fit all. In fact, it also means that the RAID level you start with, may not be appropriate at a later date depending on the changing nature of data access patterns.

RAID 1: This level of RAID is best suited for both online and archived redo logs, as they are accessed in a sequential fashion and performance is enhanced by having the write head of the disk near the location of the last write. One important point to keep in focus is that as LGWR is writing to one redo-log, ARCH may be reading from the previous log to create the archived log. This means that more that one RAID 1 volume is required for optimal performance and elimination of contention between LGWR and ARCH. If you are strapped for disks, consider 1 volume each for the even and odd numbered redo-log groups, along with a separate volume for the archived logs.

RAID 0+1 Or 1+0: As a rule of thumb, write-intensive applications should be implemented using RAID 0+1/1+0 and read-intensive applications should be implemented using RAID 3/5. This works best, because RAID 0+1/1+0 does not pose the performance penalty on the I-O sub-system, during writes in the form of pesky parity calculations. Parity calculations are incurred on RAID 3, 5 or 7, but the access patterns on those volumes should mostly be reads. However, to optimally configure a RAID 0+1/1+0 volume, one requires more disks, when compared to a RAID 3/5/7 volume. As noted before, RAID 1+0 should be preferred as it provides better availability when compared to RAID 0+1.

RAID 3 vs. RAID 5: So how do you choose between RAID 3 and RAID 5? The answer goes back to the DBA mantra, "know your application". If the application performs sequential I-O (full or range table/index scan), then the degree of striping has a direct impact on I-O performance. RAID 3 implementations should be preferred in read-intensive application environments where the read patterns are sequential and bulky in nature. This is because the spindles in a RAID 3 volume work in a *synchronized* fashion, i.e. all the disks in a given volume will service an I-O request from the same disk location (track, cylinder, sector).

On the other hand, RAID 5 implementations should be preferred where the read patterns are random and are not very bulky in nature. This is because, the spindles in a RAID 5 volume work in an "*independent*" fashion i.e. all the disks in a given volume can potentially service multiple I-O requests from different disk locations.

Thus the architecture of RAID 3 is best suited for decision support systems (DSS) and data warehousing (DW) applications with small user populations and sustained bulk read patterns, as it better caters to data-transfer and hence should be preferred. In contrast the underlying architecture of RAID 5 better caters to IOPS and hence should be preferred for DSS/DW applications with large user populations and small-sized random read patterns.

So which do you have? You can get a good idea by determining the number of "*long and short table scans*" on your system. This information is available in the dynamic performance views and in the report.txt generated by utlbstat/utlestat. If you find that your application environment is characterized by many large table/index scans and you have a relatively with a small user population, RAID 3 should be your choice. One production system benchmark on RAID 3 showed a data-transfer rate of approximately 13 MB/s, compared to a similar RAID 5 volume, which showed 9 MB/s, for the same type of activity. Additionally please note that contrary to popular belief, the dedicated parity drive on a RAID 3 device does not pose as a write bottleneck, if the controller supporting the volume is not overloaded and the size of the writes equals the size of the stripe-width on the volume (covered in more detail in later sections). This is definitely true with "*full fiber-channel*" disk systems.

RAID 7: If your system requires the data-transfer capacity of RAID 3, without hindering smaller transaction-based I-O requests, then RAID 7 is your answer. But, you better re-work your storage system budget, as RAID7 will cost you a significant chunk of change.

Important:

It is important to note here that the RAID 3 vs. RAID 5 discussion on data-transfer bandwidth vs. IOPS is done solely in the context of read-intensive applications. It is preferred and recommended that write-intensive applications be always configured using RAID 0+1/1+0 volumes. If you have made the decision to use RAID 3, based on your application and user population's characteristics, you may want to investigate the support of RAID 7 on your storage system.

If RAID 7 is supported (and you can afford it), then comparable tests should be done to validate its cost-benefit analysis performance against RAID 3. The ability to perform asynchronous reads and writes will increase the overall throughput of the storage system, and your tests should corroborate the cost and the selection of RAID 7 over RAID 3. A test is definitely worth a 1000 speculations.

Auto RAID: As mentioned in the previous section, configuring and using Hewlett Packard's Auto RAID requires care, as the automatic disk block conversion process, constantly converts disk blocks from RAID 0+1 segments to RAID 5 and vice versa, based on its determination and prediction of I-O on those blocks. The key exposure areas for Oracle are the Rollback Segment (RBS) and Temporary (TEMP) tablespaces, and are adversely affected by the conversion process.

Since the I-O patterns for these tablespaces often alternate between extensive reads and writes, performance may vary dramatically. The alternation between intense writes followed by intense reads, can cause serious system performance degradation because the RAID controller may attempt to compensate for this by changing the RAID type too frequently and yet not fast enough. It has been observed that the conversion often doesn't get done "in time" to support the future nature of the operations requested.

The problem mentioned here can occur even on all other components of the database, if there are periods of lull, followed by varying operations (reads followed by writes or vice versa) that cause the disk blocks to be converted back and forth.

Further, as mentioned before, the lack of control over drive allocation for various volumes can cause serious disk contention problems, if the application performs significant index-range scans followed by table lookups.

In a benchmark, it was observed that if the RBS tablespace was not written to for a period of time, but was read from (as part of building a read-consistent image for a long running query), the disk blocks housing the rollback segments of the database were converted to RAID 5. Then when a slew of write activity was launched at the database, the disk blocks remained as RAID

5, and this degraded write performance significantly as parity had to be calculated and written for those blocks. Later when the I-O sub-system got a breather, these blocks were re-converted to RAID 0+1. A similar phenomenon was also noticed on the TEMP tablespace.

The following table (Figure 8) outlines the usage of different RAID levels and the components of an Oracle database:

Level of RAID	When and where to use it?
RAID 0	Not suitable for any critical component of an Oracle database. May be considered for development databases where recoverability is determined to be a non-issue. This is suitable when you can simply restore a copy of a production database and re-apply any DDL differences to re-create a development environment.
RAID 1	Ideal for online and archived redo logs, Leaves the write-head at the location of the last write. On most systems, you will need 3 volumes for the online redo logs (for 3 groups) and 1 volume for the archived redo logs.
RAID 0+1 or 1+0	Ideally suited for datafiles that require read/write performance especially for On Line Transaction Processing (OLTP) or Hybrid Systems, where read/write performance is important. Pick 1+0 over 0+1 when possible.
RAID 3	Ideal for data mart/data warehouse applications with few users that require mostly range/full scans on its tables and indexes. Everything else remaining constant, RAID 3 provides better data-transfer than RAID 5.
RAID 5	Ideal for data mart/data warehouse applications with many users that require mostly unique scans on its tables and indexes. RAID 5 provides better IOPS than RAID 3.
RAID 7	Ideal for data mart/data warehouse applications with support for more users than RAID 3. The application requires mostly range/full scans on its tables and indexes. If your application requires RAID 3 and better support for IOPS and you can afford it, RAID 7 is your key.

Figure 8

Fundamentals Of Configuring Disk Arrays

So much for theory, now let’s get down to actually figuring out how to configure your RAID disk arrays. Configuring RAID disk arrays must be done in a systematic and methodical fashion. The manner in which the disk arrays are configured is largely dependent on the number of drives, partial data availability requirements, and support for parallelism. The number of drives is primarily dependent on the required data-transfer rates or IOPS (as the case may be) and the amount of data that needs to be stored.

WHAT? Well, obviously you have to have enough space, but the more important point is having enough disks to support the I-O access patterns of your application and your users.

Remember, with RAID we can get additive overall throughput, by grouping disks into volumes. This is a natural outgrowth of the tried and true approach that shows we can get more out of many smaller disks, than just a few big disks. Something to think about!

Here are some of the core issues that require consideration:

1) Determine the suitable level of RAID for the application, based on known/projected access patterns.
2) When available, hardware-level RAID should be used over software-level RAID.
3) Configure the disk controller's cache using a 60:40 ratio for Writes:Reads. For predominantly read-only environments, the write cache can be scaled down.
4) When possible, disable the write cache on volumes that support the online redo logs of the database, unless you have tested the workings of the battery backup for the cache and have verified it to be foolproof. The integrity of your database is at stake if Oracle expects redo entries to be on disk and does not find it at a later time.
5) Procure the smallest drive money can buy, keeping in mind scalability, limits of the host machine, the disk array and growth projections for the database. This is a tough one these days, with 18 GB drives considered as small drives.
6) Bigger and faster drives are not always better than smaller slower drives, as the seek times for larger and faster drives with larger form factors, may be more than their smaller and slower counterparts. This is not that big of an issue, if your drives support a built-in track buffer cache for storing an entire track's worth of data from read request(s).
7) Do not overload the controller(s).
8) Assume that all drives supported by the controller will engage at the same time.
9) Do not "*daisy-chain*" disks or disk arrays.
10) Determine the I-O bandwidth of the controller.
11) Determine the I-O bandwidth of each disk.
12) Spread your volumes across as many controllers on your storage system to get better distribution of I-O request servicing and better availability (A controller should not become a Single Point Of Failure in your system)
13) Ensure that the number of drives per controller is configured per the following formula:

Number of disks per controller = (I-O bandwidth of controller)/(I-O bandwidth of each disk)

Note:

It is useful to note here that in any disk array configuration, the bandwidth of the disk controller and the speed of each individual disk, need to be in focus during its configuration. For example, if the controller supports a bandwidth of 100 MB/sec. and if each disk supports a transfer rate of 10 MB/sec., then no more than 10 drives should be configured on 1 controller. This factor should be considered keeping in mind the read activity on the database, the type and frequency of the reports that are generated, and the number of users requesting such reports.

Fundamentals Of Disk Striping

There is a popular saying that the 3 rules for success in the real estate business are "*Location, Location & Location*". Well, it turns out that the 3 rules for success in optimal RAID implementations are "*Striping, Striping & Striping*". There cannot be enough importance given to disk striping, as it is an integral part of the art of configuring optimal RAID volumes.

Disk Striping employs the simple concept of breaking up data into pieces and disbursing them across multiple drives of a given RAID volume, to create stripes. This concept albeit simple, is very powerful and is one of the most crucial elements in I-O performance scalability. In simple terms, employing many drives to service multiple I-O requests is better than a single drive doing all the work. In addition to providing scalable I-O performance, disk striping also offers better manageability (fewer filesystems to create, mount and maintain).

Given that each drive's I-O bandwidth ceiling is fixed, it is very evident that the degree of striping is what provides scalable I-O performance, when compared to no striping at all. While it may be argued that performing manual Oracle striping of the various tablespaces across multiple drives will achieve similar I-O performance boosts, but this does add a significant amount of administrative overhead in database storage baby sitting. This is something that most sites cannot afford. I sincerely hope you have better things to do with your precious time! It is much more desirable to let the O-S and the hardware do the job of striping than manually doing it.

Also, Oracle DBA 101 (Database Administration Fundamentals) warrants application of common sense in ensuring the independence of the various components of the database, to reduce the amount of interference. Magic will not happen, if your DATA, INDX, RBS, TEMP & SYSTEM tablespaces all are physically located on the same set of disks. It is apparent here that the system should be Optimal Flexible Architecture (OFA) compliant to the extent possible.

Note:

Having made the previous comment, it is important to note here that when ever possible, striping should supercede physical separation of some database components such as SYSTEM, RBS and TEMP. Configuring RBS and TEMP in the same set of physical drives is OK, so long as 99% of the sorts are performed in memory (Need to optimally configure SORT_AREA_SIZE) and it has been verified that there is no evidence of sustained concurrent writes to rollback and temporary segments. For most environments, even sharing the SYSTEM tablespace with RBS and TEMP may be OK.

Tips On Creating Striped Volumes – I

1) Keep the degree of striping as a power of 2 namely (2,4,8,16 and so on). This is to keep in sync with the I-O chunk-size of Oracle (DB_FILE_MULTIBLOCK_READ_COUNT * DB_BLOCK_SIZE). It is important to note here that when using RAID levels 3, 5 and 7, the number of disks required for a volume with a degree of striping of "*n*" is "*n+1*".

2) Determine the degree of striping for the DATA volumes (normally 4 or 8, sometimes 16 or higher). This decides the number of drives that will participate in a given DATA RAID volume.

3) Determine the degree of striping for the INDX volumes (normally 2 or 4, sometimes 8 or higher). This decides the number of drives that will participate in a given INDX RAID volume. INDX requires a lower degree than DATA, as indexes by their inherent nature are smaller than tables and hence require a lower degree of striping

4) The degree of striping in the DATA and INDX volumes should also consider factors such as data/index partitioning, availability requirements and support for parallel operations. Please note that 16 datafiles placed on 16 individual drives can support a 16-way parallel operation (if you have memory and CPU to support it and your controller is not out of capacity because all 16 drives engaged at the same time). The same cannot be concluded with 16 datafiles placed on a 16-way RAID volume with 16 drives. The degree of parallelism that you deploy on a 16-way volume will be significantly less, when compared to the former configuration of 16 individual drives. This factor should be factored, when making "degree of striping" decisions, which in turn controls the number of volumes on your system.

5) Determine and configure the maximum I-O transfer rate of the O-S. On Solaris, the /etc/system kernel parameter that needs to be configured is "*maxphys*". This parameter is set in bytes and defaults to 128K and is appropriate for most systems where the I-O requests are transactional in nature. But for systems that are hybrid in nature and need to support large data-intensive transfers, it is recommended to set this to at least 1 MB (or even 4MB in some cases). For environments that utilize Veritas filesystems, the "*vxio:vol_maxio*" needs to be configured. This parameter is set in 512-byte units and it is recommended to set this at 2048 (which translates to 1 MB) or 8192 (which translates to 4MB).

6) Determine and configure the read-ahead cluster size for filesystems by using a rule of thumb of no more than 200 I-O operations per second on the storage device. Setting this parameter is relevant for volumes that will service large data-intensive transfers. For example, if a storage array can deliver 200 MB/sec., with the 200 I-O operations per threshold, the cluster size for the filesystems on that storage array should be set to 1 MB or 4MB as the case may be. On Solaris, this can be achieved by setting the "*maxcontig*" parameter in the mkfs command. For existing filesystems, the "*fstyp –v* devicename" command can be executed to determine the cluster size. The maxcontig parameter is set in filesystem blocks and the fstyp command will provide you the block-size (bsize) on the device. The read-ahead cluster size is calculated as (maxcontig * bsize). The cluster size for existing filesystems can be modified using the tunefs/vxtunefs commands. This step of determining and configuring the read-ahead cluster size is not relevant if the O-S is configured to support direct I-O, as in that case all "read-ahead" algorithms are disabled.

7) The results from steps 2 -5 will facilitate in calculating the stripe-width that is optimal for the volume. Given the recommendations in the previous steps, it is obvious that stripe-width = cluster size, for volumes that service data-intensive transfers. On volumes that are transactional in nature, the stripe-width can be configured with the value of the default cluster size (128K). However, if the I-O size (maxphys) is modified to 1 MB, then stripe-width should be set at 1 MB, to utilize and benefit from this increased ceiling.

Stripe-width Configuration:

The stripe-width configuration for RAID devices should be done keeping in mind 2 important conditions.

1) Ensure that all drives configured in a given volume are engaged during sequential I/O operations that are bulk in nature (full table scans of large tables). However, note that you should do this keeping in mind that the degree of concurrency of your sequential scans, which will be affected by the number of I/O requests that can be serviced by the volume at any given time. Thought should also be given for concurrency of single-block I/O requests. Single-block

I/O requests should be serviced by a single drive, without all drives engaging in the I/O operation.

2) Ensure that the stripe width of a given RAID volume is a multiple of the maximum I/O size (Step 5 in “Tips on Creating Striped Volumes, Part 1”) of the hardware and/or the I/O drivers. Care needs to be taken with this condition that it does not negate the previous stripe width condition. Remember, the key in reaping the benefits of striping is to ensure a high degree of concurrency for I/O requests on a given volume. This is true in the case of breaking up one very large I/O request into many smaller ones and also for the support of many concurrent small I/O requests

For example, in a given implementation if the maximum I-O size is 1 MB and the number of drives in the RAID array is 8, then the stripe-width should be set at 8 MB and the stripe-unit-size/interlace-size at 1MB. Across many implementations on many different operating systems, it has been observed that a stripe-width of 128K is usually a good starting value. More recent versions of the many O-Ss (such as Solaris 2.6/2.7) support a much higher I-O size (via maxphys) and this should be put to use in an appropriate manner. Stripe-width should always be dependent on the maximum I-O size of the O-S where the RAID volume is implemented.

Caveat:
Keep in mind the dependency between stripe-width and the Oracle initialization parameter – db_file_multiblock_read_count. This Oracle parameter is set based on the value of the stripe-width. A wider stripe-width can support a higher value for this parameter, and the effect of that will be improved performance for full table or index range scans.

Even though configuring volumes with large stripe-widths provide better batch processing performance, care needs to be taken, not to set db_file_multiblock_read_count at the instance level, to a value which will cause the Oracle Optimizer to choose full table scans as the preferred method of executing a query.

From Oracle 7.3 and up this issue can be easily solved, by setting this parameter conservatively at the instance level, and then modifying it to a higher value at the session level (if required).

Tips On Creating Striped Volumes – II

1) Configure the necessary volume groups.

2) Configure the necessary logical volumes.

3) Create and mount filesystems on the logical volumes (1 filesystem per logical volume).

4) Slicing and dicing the logical volumes and creating multiple filesystems under a single RAID volume, does not provide any gain in performance. In fact, it may add some administrative overhead. Creating filesystems in 10s of gigabytes in size is not a problem.

5) Create the filesystems with their “*minfree*” parameters set to 0, as datafiles in Oracle are usually pre-allocated and usually do not benefit from this feature. The benefit of setting the filesystem storage parameter “*minfree*” to 0, is to gain 10% of disk storage which otherwise is potentially wasted.

6) Ensure the following formula is always adhered to:

DB_BLOCK_SIZE = Filesystem Block Size >= O-S Page Size

Most filesystems today default to a block size of 8K. Care needs to be taken so that the Oracle database is not created with a DB_BLOCK_SIZE of less than 8K.

7) Test the I-O data-transfer rate on all volumes and visually examine the storage array to ensure that all drives in the volume are engaged in a sequential read operation. You should see your storage array light up like a Christmas tree. This will provide evidence that the volume is functioning optimally and the stripe-width configuration is working. This test can be done by copying a datafile from a volume to /dev/null. The following is an example of how data-transfer rates can be calculated:

   timex dd if=/u08/oradata/acme/data01.dbf of=/dev/null bs or bsize=stripe-width

   The above command will provide the necessary numbers to calculate the data-transfer rates using the formula:

   (# of records processed * bs)/Time Elapsed)

Operating System Configuration

While most operating systems do not require any special kernel configuration parameters to implement RAID volumes, the following section outlines some of the core issues that require consideration.

Miscellaneous

1) In some O-Ss, the kernel needs to be modified to set the "*write behind*" parameters to support fewer bulk writes, instead of many small writes. On most modern filesystems, it is a parameter that can be tuned, e.g., "*write_pref_io*" on Veritas and this needs to be set to the same value as the read-ahead cluster size.

2) The O-S and Oracle need to support "*large files*" so that files larger than 2048 MB can be configured and utilized for VLDBs. All tests and production implementations referred in this paper are on files with sizes not exceeding 4000 MB.

3) It is also note worthy here that the upper bound for the filesystem buffer cache (which is usually configured by a kernel parameter), should usually be limited to 10-15% of total RAM, as failure to configure this parameter on some flavors of UNIX (e.g., HP-UX), can cause significant degradation in performance. This degradation is noticed in the form of intense paging, process inactivation and swapping. This is because in some cases, default kernel parameters allow up to 50% of total RAM for the filesystem buffer cache.

4) It is also important that while writing to RAID volumes that require parity calculations, the write-penalty is the least. This can be achieved if the size of the writes is equal to the value of the stripe-width on those volumes. This is possible when the stripe-width is set to the same value as the read-ahead cluster size. But if the writing of data begins half-way in a stripe, then partially writing 2 stripes will be much more expensive than writing 1 full-stripe. This is due to the increased overhead in reading the data, re-computing the parity, writing the new parity and then the modified data. For this purpose, it is important to configure "*write alignment*" which is supported on some filesystems. On Veritas filesystems write alignment can be achieved by aligning the clustered writes with the stripe on a preset boundary. This can be achieved by creating the filesystem with the align option and setting this to a value of 512 bytes.

Raw Devices vs. Filesystems

When possible, advanced file systems such as xfs, jfs, or vxfs should be preferred over ufs file systems. This is to utilize the improved journal and performance features (such as elimination of double buffering) that advanced file systems offer over ufs. It must be noted that if you configure your system using ufs file systems the necessary file system level parameters need to be configured and tuned, as the default values will not provide optimal performance.

When using Veritas vxfs file systems, Quick I/O can provide raw device comparable performance without losing the benefits of a file system. This is because the Quick I/O driver intercepts all DBWR writes (when enabled) thus bypassing the file system buffer cache. This provides raw device comparable performance, as the classic problems of double buffering and wasted CPU cycles in managing the file system buffer cache are avoided. When using other types of advanced file systems, Direct I/O can be preferred if supported by the OS, as it too provides raw device comparable performance. The use of raw devices does not add any significant value when compared to advanced file systems with Quick/Direct I/O drivers (wherever applicable).

Raw devices do add a level of operational complexity without justifiable performance benefits, when compared to advanced file systems configured with Veritas Quick I/O or Direct I/O (supported by the OS). In these cases, the configuration and use of raw devices should be reserved for Oracle Parallel Server implementations.

Asynchronous I-O

Oracle configured with asynchronous I/O has been found to work effectively only on raw devices across most flavors of UNIX. Asynchronous I/O on regular file systems is supported on some operating systems such as Solaris. Depending on your I/O system configuration, you may observe that Direct I/O or Quick I/O can offer comparable performance when using specialized file systems such as Veritas (vxfs). You may also have the option of configuring the relevant Oracle instance parameters for multiple database writer support. But normally you either enable asynchronous I/O or multiple database writers, not both.

Database Configuration

Regardless of the storage vendor and their claims, common sense must prevail on issues such as separation of the DATA and the INDX tablespaces. As mentioned in previous sections, for query intensive applications that heavily utilize indexes, the DATA and INDX tablespaces should be separated. There are no two ways about it.

In some applications you may find that a few tables or indexes are "hotter" than most all of the other tables and indexes. Separating these tables or indexes from the others by moving them to new tablespaces placed on new or different RAID volumes remains a "best" practice. Time spent re-balancing I-O by moving datafiles is time well spent. However, the frequency of re-balancing will be far and few between, if you configure your volumes keeping in mind the issues discussed in the aforementioned sections.

A scientific comparison needs to be made to determine whether it is better to create fewer volumes with "*thin-wide stripes*" or more volumes with relatively "*thicker-narrower*" stripes. This is dependent on issues such as data/index partitioning, required support parallelism for core operations and any service-level agreements on high or partial availability. While thin-wide stripes are a very attractive solution, the constraining factors of parallelism, availability and data/index partitioning, make it not that appealing. My goal is to meet somewhere halfway between thin-wide stripes and thick-narrow stripes.

For applications that are write intensive, the DATA, INDX and RBS tablespaces should be separated. Also, the various groups of the online redo logs should be separated on different disks/volumes, when possible. Further, the location where archived redo logs are written by the ARCH process should be different from the location of the online redo logs. Also, it may be justifiable to put the RBS and TEMP tablespaces in the same volumes, if SORT_AREA_SIZE is configured optimally and there is no evidence of sustained simultaneous generation of "*sorts on disk*" and "*undo entries for rollback*".

One too many database administrators find themselves "cheating" with their redo and archive logs. If your database is in NOARCHIVELOG mode, you probably can get away with a single RAID 1 volume for your redo-logs. However, for databases in ARCHIVELOG mode you will need to further separate your redo-log groups from each other and from your archive logs. An effective arrangement uses three RAID 1 volumes. Nothing else should be put on any of these volumes. This is where some administrators start to get uncomfortable. If your system is like most, it is consists of 9 GB or 18 GB drives. Now two of those went into each volume, but because they are mirrored you still only have the capacity of one. Therefore you just used up six 9GB drives to make three 9GB volumes. One of those is for the archived logs so you can now keep quite a few logs on line. But those other two volumes have only a few redo logs on each one. And since your redo logs may be quite small, say 250MB – 1 GB. That leaves a lot of disk space barren. LEAVE IT ALONE. Putting other active components of an Oracle database, on the same volume(s) will only hamper performance. You may however make use of that space for storing export files and/or any other administrative items that will not interfere with the Oracle database.

The SYSTEM tablespace can be physically located on any volume and in most cases can share the same physical volume as the TEMP and RBS tablespaces. This is because contention amongst these components of the database is usually very minimal. The optimal configuration of the SHARED_POOL_SIZE and DB_BLOCK_BUFFERS initialization parameters is essential to reduce I-O on the SYSTEM tablespace.

Instance Configuration

The following are the key Oracle initialization parameters that require configuring, when deploying RAID on Oracle Systems. The Oracle initialization parameter settings below (Figure 9) are for a database with 4 independent logical volumes, a database block-size of 8K and an I-O size of 128K.

Parameter Name	Example Setting	Description
db_block_checkpoint_write_batch	16	Write Chunk-size for DBWR during checkpoint writes, Set it to stripe-width
db_file_multiblock_read_count	16	Read Chunk-size used when performing full-table scans and index fast full scans, Set it to stripe-width
db_writer_processes	8	Number of DBWR processes attached to the instance. Configure this to no more than (2 * # of physically independent volumes)
dbwr_io_slaves	N	Where N the number of I-O slaves required for optimal configuration. This is usually needed only when the number of processes that perform database writes needs to be > 8.

disk_asynch_io	False	Turns off Asynchronous I-O, Does not work as advertised on most flavors of UNIX and on most types of filesystems with the exception of Solaris and AIX.
hash_multiblock_io_count	16	I-O Chunk-size used when performing hash joins
use_direct_io	True	Performs Direct I-O bypassing the filesystem buffer cache. Configure this when available.

Figure 9

RAID & Oracle VLDBs – Core Issues

While most issues related to RAID are common across most applications and databases, there are some that affect VLDBs more than others. The following need consideration:

1) Depending on whether the system is a hybrid system or OLTP, core initialization parameters such as db_block_size (which cannot be modified without database re-creation) need special consideration. When in doubt, go with the bigger block-size. This provides you with better I-O sub-system performance (fewer system calls for servicing your I-O requests) and reduces the number of blocks in your tables and indexes. In the case of indexes, this does have a positive effect on the height of the index.

2) Depending on the nature of the application, i.e. transaction-heavy or reporting-heavy, the level of RAID needs to be carefully considered and configured. Further the nature of the application should also be researched to determine whether the I-O patterns are sequential or random. Re-doing this after-the-fact may be impossible.

3) The ratio of (Used database size vs. Total database size) is very relevant to the selection process of the appropriate level of RAID and all the relevant I-O parameters. Pareto's principle is probably working in your environment. 80% of your I-O is generated from 20% of your data. It is to your benefit to determine which objects comprise that 20%.

4) The ability to segregate "*read only*" tablespaces from *"read write*" tablespaces will further influence the use of different levels of RAID that either read-friendly or write-friendly, as the case may be.

5) The number of users that access the application and hence the number of concurrent sessions supported by the database also determine many configuration issues in the I-O and I-O sub-system.

6) Any service level agreements that relate to "*availability*" need to be factored into the final equation when determining the balance between cost and availability. This issue determines the allowable Mean Time To Recover (MTTR) and thus affects the frequency of backups. This issue also has a direct correlation to the "*degree of striping*" decisions that you need to make.

7) The level and complexity of data replication that is required for the application for protection against failure(s) and disaster recovery will have long-reaching effects on the

RAID configuration decisions that are made with respect to a given system. How much and how frequently does your data change?

8) The options to perform software-based replication versus hardware-based replication need to be determined.

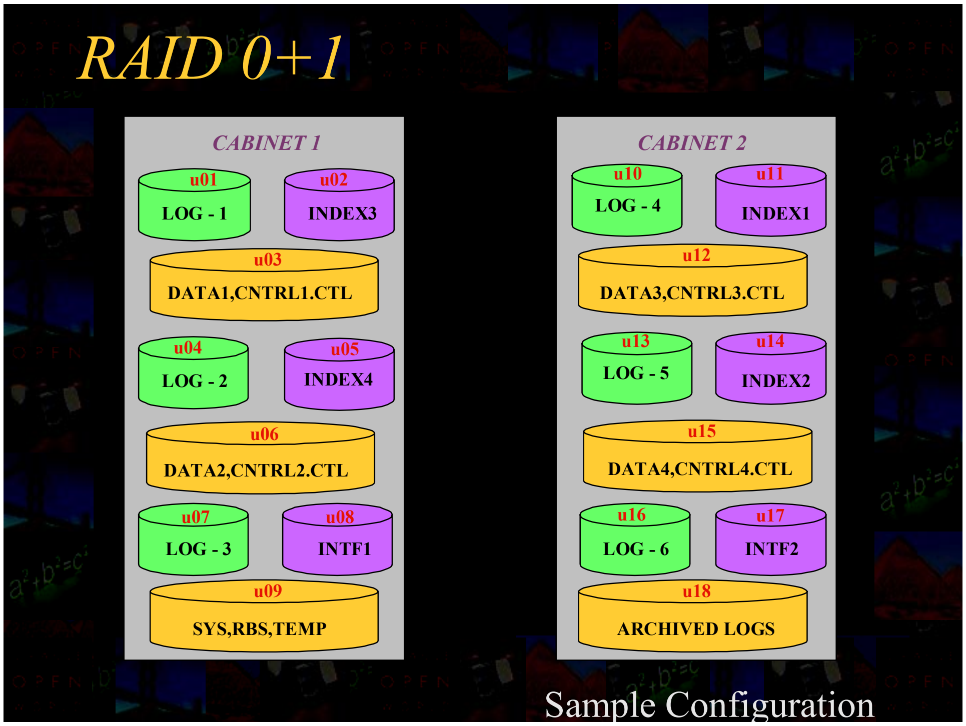
Sample RAID Configurations

The following are sample RAID configurations that are currently supporting live production databases. These configurations have all been designed for optimal I-O performance. While cost is an important factor when designing optimal RAID configurations, the primary focus for all the following configurations were I-O scalability, predictable performance, availability, data & index partitioning and parallelism of operations.

Volume Legend:

Color	*Degree of StripingMirroring*		*Parity*
Green	None	Yes	No
Violet	2	Yes	No
Yellow	4	Yes	No
Orange	4	No	Yes
White	8	No	Yes

The following graphic (Figure 10) is a sample RAID 0+1 configuration consisting of eighty-four 18-GB disks, each capable of transfer rates of 5-7 MB/sec. and 6 controllers (3 per cabinet) each capable of an I-O bandwidth of 100 MB/sec:



RAID 0+1

CABINET 1

u01 LOG - 1

u02 INDEX3

u03 DATA1,CNTRL1.CTL

u04 LOG - 2

u05 INDEX4

u06 DATA2,CNTRL2.CTL

u07 LOG - 3

u08 INTF1

u09 SYS,RBS,TEMP

CABINET 2

u10 LOG - 4

u11 INDEX1

u12 DATA3,CNTRL3.CTL

u13 LOG - 5

u14 INDEX2

u15 DATA4,CNTRL4.CTL

u16 LOG - 6

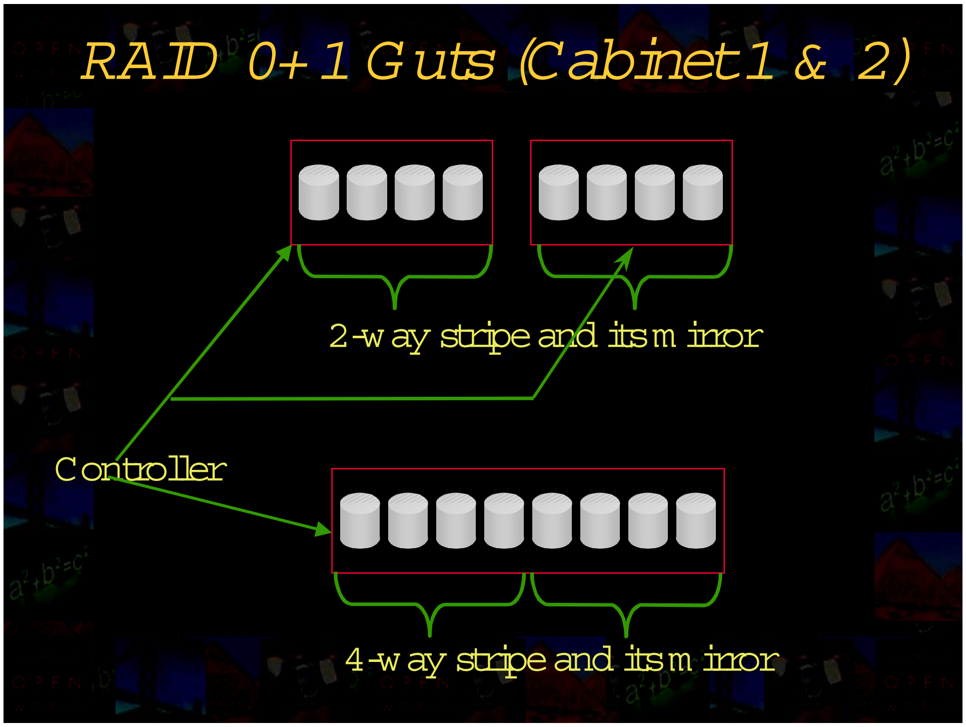
u17 INTF2

u18 ARCHIVED LOGS

Sample Configuration

Figure 10

Figure 11 sheds light on the guts of the RAID 0+1 volume:



RAID 0+1 Guts (Cabinet 1 & 2)

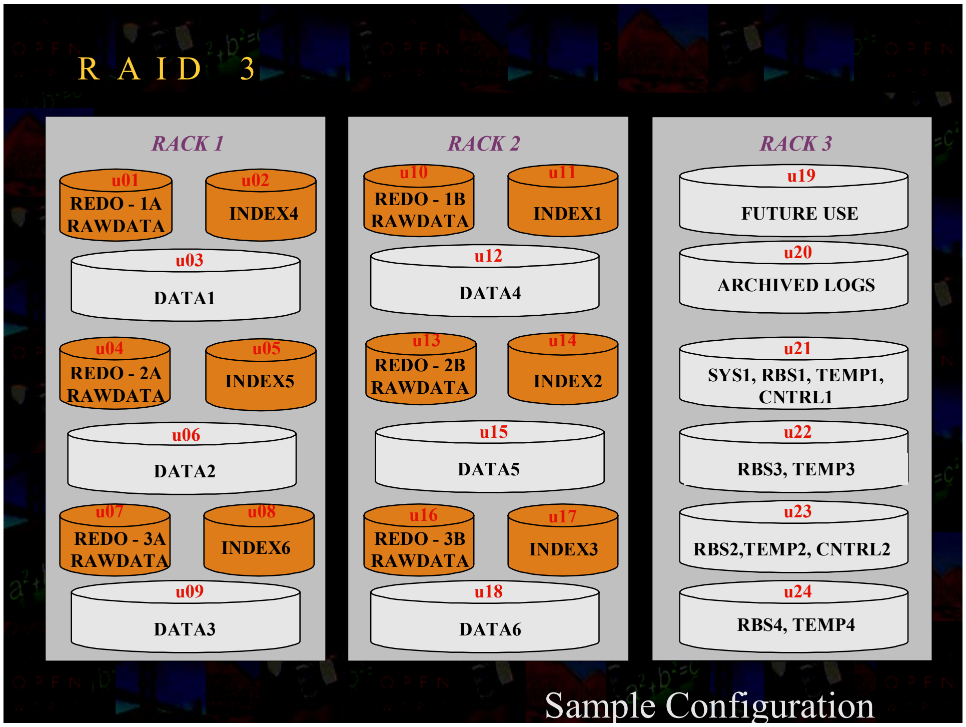
2-way stripe and its mirror

Controller

4-way stripe and its mirror

Figure 11

The following graphic (Figure 12) is a sample RAID 3 configuration consisting of one hundred and eighty 9-GB disks, each capable of transfer rates of 10 MB/sec. and 9 controllers (3 per rack) each capable of an I-O bandwidth of 100 MB/sec:



RAID 3

RACK 1

u01 REDO - 1A RAWDATA

u02 INDEX4

u03 DATA1

u04 REDO - 2A RAWDATA

u05 INDEX5

u06 DATA2

u07 REDO - 3A RAWDATA

u08 INDEX6

u09 DATA3

RACK 2

u10 REDO - 1B RAWDATA

u11 INDEX1

u12 DATA4

u13 REDO - 2B RAWDATA

u14 INDEX2

u15 DATA5

u16 REDO - 3B RAWDATA

u17 INDEX3

u18 DATA6

RACK 3

u19 FUTURE USE

u20 ARCHIVED LOGS

u21 SYS1, RBS1, TEMP1, CNTRL1

u22 RBS3, TEMP3

u23 RBS2,TEMP2, CNTRL2

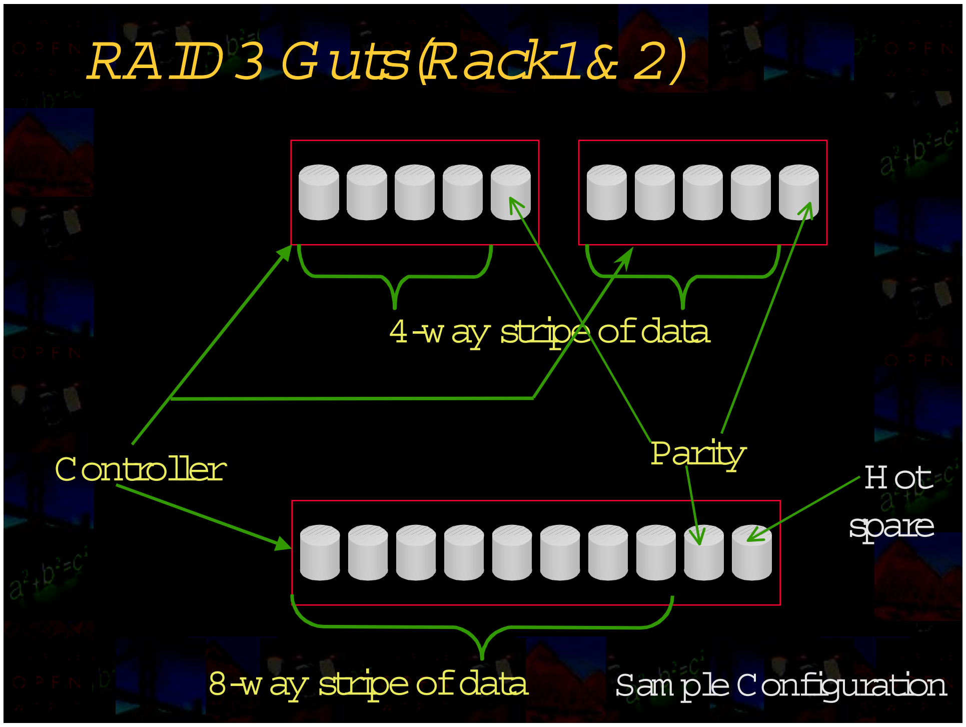
u24 RBS4, TEMP4

Sample Configuration

Figure 12

Figure 13 sheds light on the guts of the RAID 3 volume:

Figure 13



RAID 3 Guts (Rack 1 & 2)

4-way stripe of data

Controller

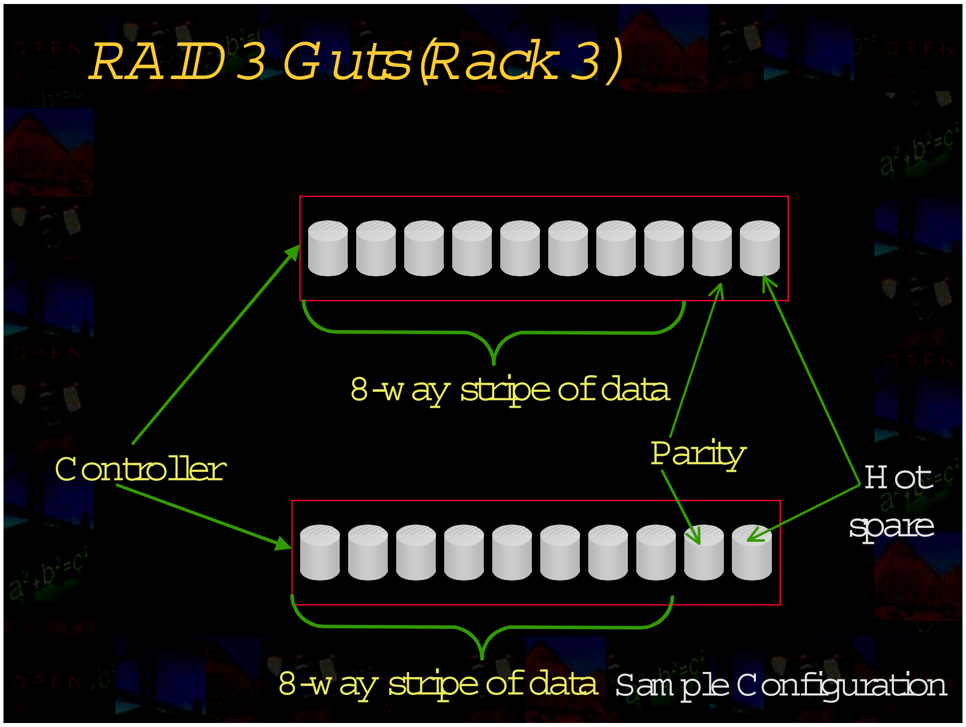
Parity

Hot spare

8-way stripe of data

Sample Configuration

Figure 14 sheds more light on the guts of the RAID 3 volume:



RAID 3 Guts (Rack 3)

8-way stripe of data

Controller

Parity

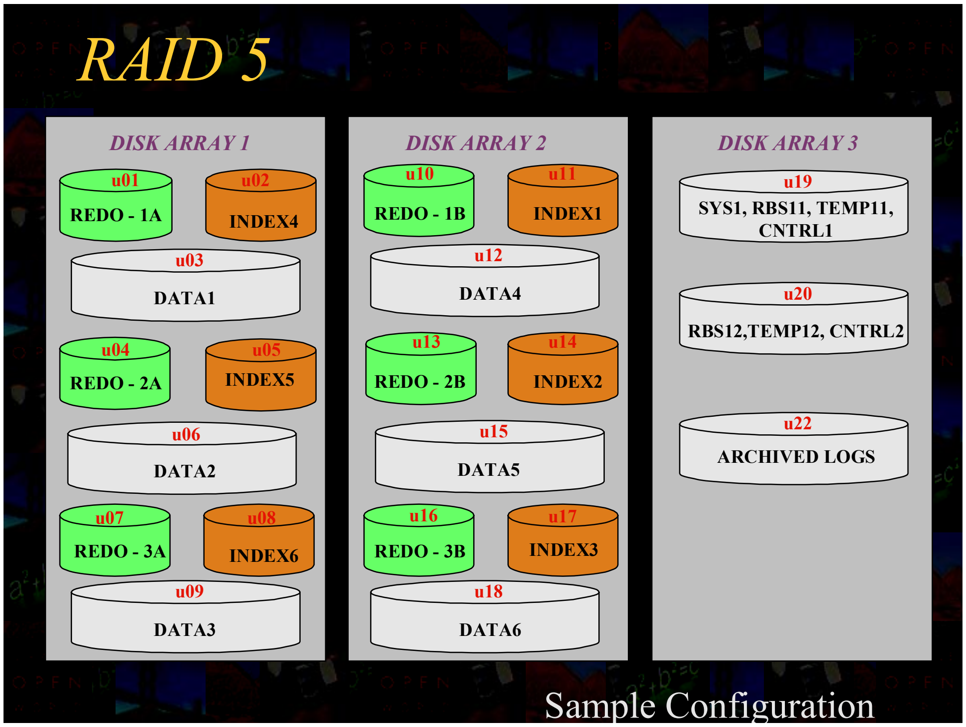
Hot spare

8-way stripe of data

Sample Configuration

Figure 14

The following graphic (Figure 15) is a sample RAID 5 configuration consisting of one hundred and fifty 18-GB disks, each capable of transfer rates of 5-7 MB/sec. and 8 controllers (3 in Disk Array #1 & #2 and 2 in Disk Array #3) each capable of an I-O bandwidth of 100 MB/sec:



RAID 5

DISK ARRAY 1

u01 REDO - 1A

u02 INDEX4

u03 DATA1

u04 REDO - 2A

u05 INDEX5

u06 DATA2

u07 REDO - 3A

u08 INDEX6

u09 DATA3

DISK ARRAY 2

u10 REDO - 1B

u11 INDEX1

u12 DATA4

u13 REDO - 2B

u14 INDEX2

u15 DATA5

u16 REDO - 3B

u17 INDEX3

u18 DATA6

DISK ARRAY 3

u19 SYS1, RBS11, TEMP11, CNTRL1

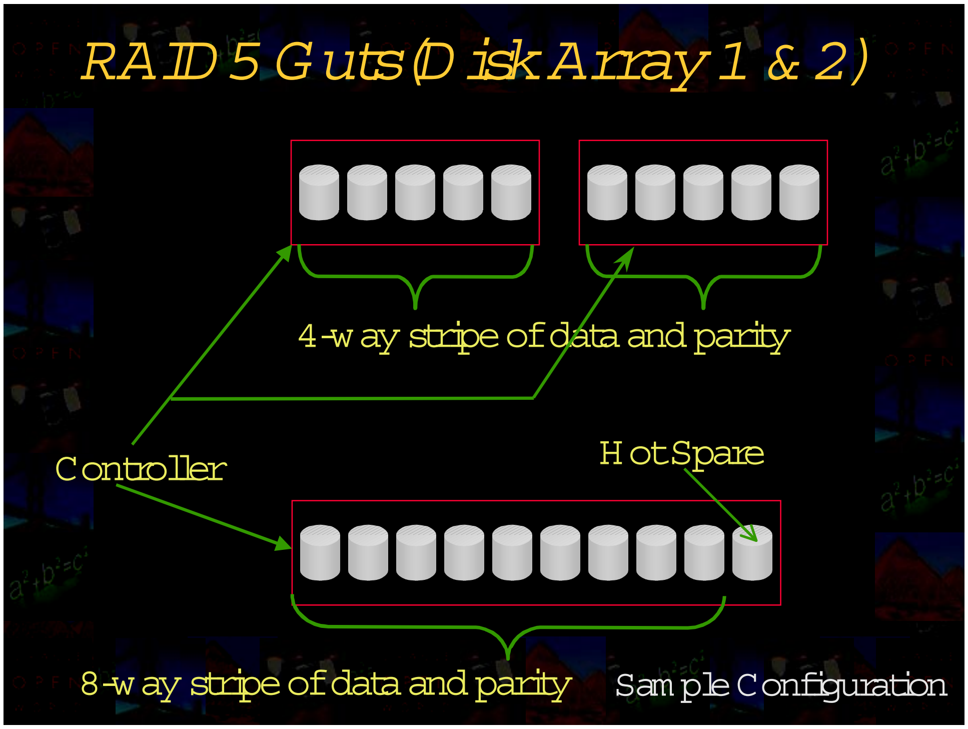
u20 RBS12,TEMP12, CNTRL2

u22 ARCHIVED LOGS

Sample Configuration

Figure 15

Figure 16 sheds light on the guts of the RAID 5 volume:

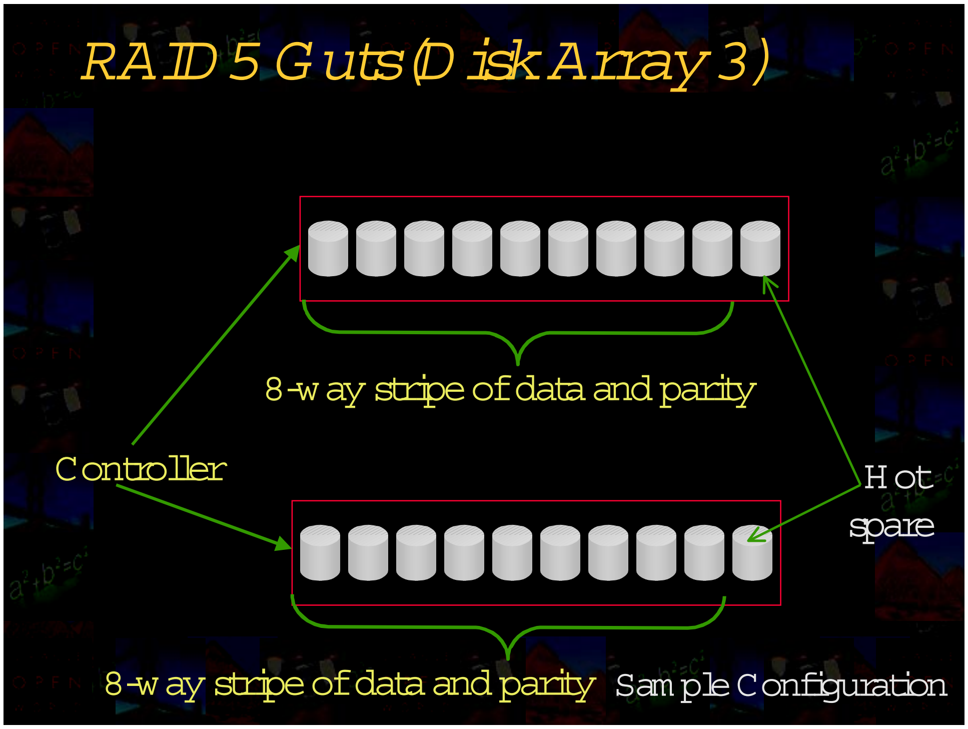


RAID 5 Guts (Disk Array 1 & 2)

4-way stripe of data and parity
Controller
Hot Spare
8-way stripe of data and parity
Sample Configuration

Figure 16

Figure 17 sheds more light on the guts of the RAID 5 volume:



RAID 5 Guts (Disk Array 3)

8-way stripe of data and parity

Controller

Hot spare

8-way stripe of data and parity

Sample Configuration

Figure 17

Conclusion

There is more to RAID than a leading name brand insecticide. RAID provides the technology to scale I-O and system performance of applications and provide high availability. Proactive design, architecture and deployment of optimal RAID configurations are quintessential to gaining scalable I-O performance. It is important to understand the RAID solutions offered by each hardware vendor, before any implementation is done. A solid understanding of the application to be supported plays a large part in RAID selection. Researching the application and the details of your vendor's RAID offerings must be the starting point of any I-O sub-system design.

As always, DBA 101 common sense needs to be applied on a continuous basis, some things never change – DATA tablespaces needs to be separated from INDX tablespaces. The 3 rules for successful optimal RAID configurations are "*Striping, Striping, & Striping*". Oh, one last thing, regardless of what their claims may be, mass storage vendors are definitely not the "*Gods of Disks*".

Credits

Many people deserve credit for the material that went into this paper. First and foremost, to the various Fortune 500 customers, who hired my services to make their Oracle lives better…I sincerely hope you got what you paid for. This paper would be nonexistent without you. Thank you for your trust and faith. Next, to the many System Administrators and Oracle Database Administrators…it was great working with you folks. Lastly, to John Kostelac, who techno-proof-read this paper and contributed his humor and technical insight. Thank you to all of you from the bottom of my heart.

About The Author

Gaja Krishna Vaidyanatha is the Director of Storage Management Products with Quest Software Inc., providing technical and strategic direction for the storage management product line. He has more than 10 years of technical expertise, with 9 years of industry experience working with Oracle systems.

Prior to joining Quest, Gaja worked as a Technical Manager at Andersen Consulting, where he specialized in Oracle Systems Performance Management and lead the Oracle Performance Management SWAT Team within the Technology Product Services Group. In a prior life to that, Gaja was also a Consultant and Instructor at Oracle Corporation specializing in the core technologies of Oracle.

Gaja is the co-author of *Oracle Performance Tuning 101* published by Osborne McGraw-Hill (Oracle Press), which focuses on the rudiments of Oracle Performance Management. His key areas of interests include performance architectures, scalable storage solutions, highly-available systems and system performance management for data warehouses and transactional systems. Gaja holds a Masters Degree in Computer Science from Bowling Green State University, Ohio. He can be reached at gajav@yahoo.com or gaja@quest.com.

References

Massiglia, P. *RAID for Enterprise Computing*. Veritas Software Corporation White Paper, 2000. http://www.veritas.com/

McDougall,R. *Getting to know the Solaris filesystem, Part 2*. Sun Microsystems White Paper, 1999. http://www.sunworld.com/sunworldonline/swol-06-1999/swol-06-filesystem2_p.html

Shah, R. *What is RAID? – Details of this popular storage technology*. Sun Microsystems White Paper, 1999. http://www.sunworld.com/swol-06-1999/f_swol-06-connectivity.html

Vaidyanatha, G. *Implementing RAID on Oracle Systems*. Proceedings of the International Oracle User Group – Americas Live 2000, 2000. http://www.ioug.org

Wong, B. *RAID: What does it mean to me?*. Sun Microsystems White Paper, 1995. http://www.sunworld.com/sunworldonline/swol-09-1995/swol-09-raid5_p.html