OPTIMAL STORAGE CONFIGURATION MADE EASY

Juan Loaiza, Oracle Corporation

COMPLEXITY

Configuring storage subsystems for the Oracle database is an unnecessarily complex process. In the conventional methodology, a storage configuration is custom designed for each site based on detailed knowledge of the application. This methodology assigns a specific load to various subsets of the database such as log files, rollback segments, index tablespaces, data tablespaces, etc. and assigns disks to these subsets based on their load profile. Many tradeoffs must be made when designing a storage configuration using this methodology. If an IO estimate or assignment is incorrect then performance can suffer. This methodology also requires ongoing load balancing as the database grows and the application is modified.

Storage configuration does not have to be this complicated.

A simple, efficient, and highly available storage configuration is possible. The basic idea of this configuration is to make extensive use of striping across large sets of disks using a methodology that we will describe later. To achieve high availability the disks should be mirrored. We call this methodology **S.A.M.E.** SAME stands for Stripe and Mirror Everything.

The SAME configuration produces close to optimal performance for ALL workloads: OLTP, Warehouse, and Batch. We believe that the SAME configuration will become the predominant configuration for Oracle databases. Oracle is working with leading storage vendors to optimize and automate this methodology.

This SAME methodology is based on careful analysis of current and future storage technology, combined with a thorough understanding of the performance and availability requirements of the Oracle database. This paper describes the thinking and reasoning behind the SAME methodology. An implementation of the ideas described in this paper must take into account the capabilities and limitations of the chosen storage management products. A companion paper discusses work that was jointly performed by Oracle, Veritas, and EMC to test and document a specific implementation of the SAME methodology. That paper serves as a cookbook for implementing the SAME methodology using the Veritas VxVM volume manager and the EMC Symmetrix. That paper can be found at http://www.veritas.com/oracle or http://technet.oracle.com/deploy/availability/listing.htm#Collateral.

DISK BASICS AND TRENDS

The SAME configuration is designed to achieve the best possible utilization of the disk drives in a storage configuration. SAME does not address issues of connectivity between the host and the drives. In practice, disk drives are usually the bottleneck for random operations but connectivity is sometimes the bottleneck for sequential throughput. SAME optimizes for both random access and sequential throughput. Since the SAME methodology does not require making a tradeoff between access rate and throughput, it works well even if connectivity limits the maximum throughput to less than what the disk drives can deliver.

The primary goal of the SAME methodology is to optimize disk utilization. Therefore, it is important to understand the basic characteristics of disks and how disk technology is evolving. This section describes some disk drive basics and trends.

Disk drives have a few basic properties that control and limit their performance. A typical disk on the market today has these properties:

Capacity	35G to 75G, soon 100G to 200G drives will be available.
Rotation Speed	10000 RPM = 3 millisecond average rotation delay
Seek Time	1 millisecond to 11 millisecond depending on distance seeked
Transfer Rate	Varies by location -- 12 MB/sec for the inside tracks, 20 MB/sec for the outside tracks

There are a few clear trends that are driving the disk and storage industries. These are summarized in the following table:

Trend	Implication
Disk Capacity is increasingly rapidly	Bigger databases on fewer disk drives
CPU speed is increasing rapidly	More IOs issued per second per CPU
RAM capacity is increasing slower than disk capacity	Cache size will grow smaller relative to disk storage
Disk performance is increasing slowly	Performance bottlenecks will increase

Fewer drives, more IOs per second, and modest disk performance increases will cause database performance to become increasingly disk limited. It will be commonplace to configure many more disks for a database than are required to hold the database data. The extra disks will be deployed purely for the extra throughput they provide, not the storage capacity. This adds to the expense of deploying a high performance database. Therefore it is imperative that the disk configuration chosen use all available disk bandwidth.

OPTIMIZING DISK BANDWIDTH

To use all available disk bandwidth we need to address two issues. One is how to use all the available disks. The other is how to make most effective use of each disk. We will discuss these issues in turn.

USING ALL AVAILABLE DISKS

To use all available disks the data to be accessed must be spread across as many disks as possible. All disks in the disk farm should have roughly equivalent utilization. Any disk that is used more than the other disks will become a bottleneck to performance.

One way to spread data across many disks is to create roughly one database file per disk and use careful segment allocation and partitioning at the database level to spread the database tables across these files. While this can be done successfully it is a very tedious and customized process.

A more practical method is to make extensive use of volume level striping across disk drives. Modern volume managers can stripe across many tens or hundreds of disks while adding very little overhead to IO operations.

An extra benefit of using a volume manager to spread access across disks is that database capabilities such as partitioning can be focused on increasing database level performance and availability instead of spreading load across disks. For example, partitioning can be used to isolate recent data from historical data.

OPTIMIZING SINGLE DISK SEQUENTIAL PERFORMANCE

Many people think that optimizing sequential throughput requires scanning sequentially through an entire disk or a large portion of a disk. This is not true.

To optimize sequential throughput it is only necessary to sequentially scan enough data that the time it takes to position the disk head to the data is a small percentage of the time to transfer the data. This is a key observation. We only need to make sure that

$$\text{Transfer time} > 5 * \text{positioning time}$$

to achieve high sequential throughput. For example, 100 random reads of 10 megabytes each are almost as efficient as a single scan of 1000 megabytes of data.

The following table shows the relative overhead of positioning for various transfer sizes:

IO Size	Positioning Time	Transfer Time	% of Time Transferring Data
16K	10ms	1ms	10%
64K	10ms	3ms	23%
256K	10ms	12ms	55%
1M	10ms	50ms	83%
2M	10ms	100ms	91%

From this table we can see that a sequential transfer that is 1 megabyte or larger achieves high throughput and efficiency. Larger transfers than this produce modest improvements. Therefore our goal should be to ensure that sequential operations access at least a megabyte of sequential data. Smaller sequential accesses are much less efficient. Larger sequential accesses improve performance by only a small factor.

VARIABLE TRANSFER RATE

The transfer rate for a disk drive is not the same for all portions of a disk The outer sectors of a disk drive move by the disk head faster than the inner sectors leading to a faster transfer rate for the outer sectors. This is simply because of the circular shape of a disk drive.

The outside portions of a disk drive also have more area than the inside portions. This means that the outside portions of a disk drive hold more data than the inside portions. Typically more than 60% of the capacity of the disk is on the outside half of the disk.

The following table summarizes the transfer rate at various regions of a typical disk drive based on the percentage of *capacity* at that region. Thus it combines these two factors to show the achieved transfer rates for user data stored on the drive.

Position on Disk (measured in terms of capacity)	Transfer Rate at this Position (MB/sec)
Outer Edge	22
Outer Quarter	21
Mid Point	19
Inner Quarter	16
Inner Edge	11

What we can conclude from this table is that the transfer rate of a disk drives varies by a factor of two from the inner edge of the drive to the outer edge. Therefore it is beneficial to place frequently accessed data toward the outer edge of a disk drive. Note that most of the performance drop occurs in the inner most tracks of the disk. Therefore, a fine level of positioning is not needed. If frequently used data is placed randomly on the outer half of a disk drive, then sequential access to that data achieves over 90% of the best possible throughput. Thus, placing most frequently accessed data towards the outside half of a disk, and less frequently accessed data towards the inside half suffices to achieve close to optimal sequential access.

Optimizing Random Access

To optimize random access on a disk drive it helps to limit the length that the disk head moves (seeks) between data accesses.

A typical disk drive today has a minimum seek time of approximately 1ms for seeking to the next track, and a maximum seek time of approximately 11ms for seeking the entire width of the disk. The seek time varies roughly linearly between these values based on the length of the seek. If seeks are confined to a region of a disk, then to seek at random within that region takes on average one third of the time to seek from one end of the region to the other. So to seek at random within the entire drive takes roughly 4ms.

To read a block of data, the disk head must seek to the correct track and then must wait for the data to rotate to below the disk head. For a 10000 RPM drive the average rotation latency is 3ms. Therefore, for small seek distances the time to position the disk head is dominated by the rotation delay (1ms to seek, 3ms to rotate). This means that positioning data at a very fine level to reduce seek time does not help much. It is enough to position data that is accessed frequently roughly in the same half or quarter of a disk drive.

Also recall that disk drives hold more data towards the outer edge of the drive. This implies that to seek the same distance measured in megabytes towards the outer edge of a disk is faster than towards the inner edge of a disk. Thus we should prefer to place more frequently accessed data towards the outer portions of disk drives to reduce seek distance between frequently accessed blocks.

The following table summarizes the effect of confining seek distance to the outer portions of a disk drive. This table displays the time it takes to seek within a region of the disk The idea of the table is to calculate the average read time and space used if we limited ourselves to using *only* the outside part of the disk. This will give us a calculation of the maximum benefit we can get by placing data carefully on the disk. For example, we can see in the table that if only the outer 60% of the disk drive is used, the average read time is 5.7ms.

Percent of Disk Used (in distance from outer edge)	Average Seek Distance with Region	Percent of Disk Space Used by Region	Average Read Time within Region (seek plus rotation)
25%	8%	33%	4.8ms
50%	17%	60%	5.7ms
75%	25%	85%	6.5ms
100%	33%	100%	7.3ms

We can conclude from this table that placing frequently accessed data towards the outside half of a disk achieves most of the performance benefit that can be achieved by positioning data to reduce seek time. Because the rotation latency dominates the total access time when the seek distance is very small, placing data at a very fine level within a disk achieves little extra performance gain. The read time for a *minimum* length seek access is 4ms. The average read time for a random read within the outer half (by capacity) of a disk drive is approximately 5.3 ms.

Summary of Optimizing Disk Bandwidth

In summary, to optimize disk bandwidth we should:

- Stripe data to equalize the workload across disks and eliminate hotspots
- Stripe data to enable many disks to serve requests for any subset of data
- Ensure that sequential access occurs in at least 1 megabyte units to achieve high sequential bandwidth
- Place frequently used data on the outer half of disks to provide the fastest transfer rate
- Place frequently used data on the outer half of disks to minimize seek overhead

These recommendations will change slowly as disk technology evolves since they do not depend on factors that are increasing rapidly. The primary trend to be aware of is that since transfer speed is increasing faster than seek rate, the minimum unit of sequential access should increase with time.

THE ORACLE IO WORKLOAD

The Oracle IO workload is very complex.

Oracle has many file types (data, log, temp, archive, undo, system, control, backup, etc.) and many operation types (scan, lookup, load, insert, create index, join, LOB, sort, hash, backup, recovery, batch write, etc.). Understanding the IO workload to each file type for each operation and its relative importance is extremely difficult.

The Oracle IO workload is also application dependent. Like an Operating System, the Oracle Database acts on behalf of an application. The workload for an OLTP application is different from a warehousing application. The workload for a batch job is different from an online user. The hit rate of the buffer cache varies by application. The tables, indexes, and queries in a database are totally application dependent.

Because of this complexity, designing an efficient workload based storage configuration is extremely difficult. The SAME configuration completely avoids getting into this level of detail. It is not necessary since the underlying disk technology is independent of this detail. SAME does not optimize for a specific workload. Instead, it spreads the IO load across the disks and makes sure each of the disks is utilized efficiently. There are just a few key attributes of the Oracle workload that must be understood to see how SAME accomplishes this feat. This section describes these attributes.

SEQUENTIAL IO IS TREATED SPECIALLY BY ORACLE

Oracle treats sequential IO specially. Sequential IO occurs for table scans, direct loads, logging, backups, sorts, etc. Oracle recognizes when one of these operations is occurring and issues large IO operations that span block boundaries. The size of the IO operation issued is controlled by parameters such as db_file_multiblock_read_count.

Recall that, to achieve high sequential disk bandwidth, it is necessary to ensure that sequential access to the disk drive occurs in at least one megabyte units. Many IO subsystems can detect a pattern of sequential access and will perform read-ahead using large IO operations. However, detection of sequential access takes time. The best way to ensure that large IO operations occur at the disk drive is to issue large IOs at the Oracle level and ensure that these IOs are not broken up between Oracle and the disk. To achieve this, parameters such as db_file_multiblock_read_count should be set to one megabyte, stripe widths should be set to one megabyte, and OS IO size limits should be set to at least one megabyte.

DIRECT READ AND WRITE OPERATIONS PERFORM READ AHEAD

Oracle can perform large scan or load operations directly to disk, bypassing the Oracle buffer cache. These are performed for parallel table scans, index creation, direct loads, sorts, etc. This type of operation is termed *direct IOs*. For direct IOs, Oracle will issue multiple asynchronous IO operations. This is sometimes called asynchronous read ahead or double buffering. The goal of these operations is to achieve maximum disk throughput so that the operation becomes CPU bound instead of IO bound.

To achieve maximum throughput, it is best if each of these IO operations accesses a different disk. Accessing separate disks allows the scan or load operation to use the bandwidth of multiple disks. This implies that the stripe size chosen should be less than or equal to the IO size.

PARALLEL EXECUTION

Oracle can automatically parallelise many operations. The operations that can be parallelised include scan, sort, join, hash, load, create index, etc. A parallel operation can use the full execution power of all the CPUs on the host. On a large multiprocessor or cluster, the IO rate that is generated by a parallel scan can reach many gigabytes per second.

The most important point to understand about parallel execution is that it can focus intense IO activity on any subset of the database. The chosen table, index, or partition receives the aggregate CPU power of the host system. If very

high IO bandwidth is not available for the subset, then parallel execution will not scale. This implies that any subset of the database must be spread across many disks since we cannot predetermine which subset of the database will have a parallel operation invoked on it.

Many databases are configured to have subsets of data that are spread across just one, or a small number of disks. This storage configuration eliminates the possibility of executing a fast parallel operation on the subset of data. Often, administrators are not aware that parallelism is being limited when the storage configuration is designed. This is a terrible waste of one of the most powerful features of the Oracle database.

The most extreme case of IO parallelism occurs in the parallel nested loops join operation. In this operation each CPU can issue tens to hundreds of random IO operations in parallel. For a large multiprocessor, this can result in hundreds of random IOs issued in parallel. If the data subset being joined is not spread across very many disks, the storage subsystem will bottleneck the performance of the join operation.

To ensure that parallelism is not constrained by the IO configuration, all subsets of database data should be spread across as many disks as possible. For best performance, all data subsets should be spread across *every* available disk. This ensures that performance is limited by the total disk farm, not by a sub-optimal choice that was made when the storage subsystem was configured.

Log Files

It is generally more efficient and flexible to parallelise IO operations using parallel execution at the Oracle level than using small stripe widths at the storage level. However, online log file writes cannot be parallelised at the Oracle level. They must be parallelised at the storage system level. If an online log file is located on a single disk, then operations that make changes very rapidly such as parallel updates, parallel index creations, parallel loads, etc. may become bottlenecked on the log disk. Therefore the online log file should be spread across multiple disks using striping.

In general it is easiest and most efficient to stripe the logs across all the disks just like the data files. Sometimes people worry that placing the log files on the same disks as the data files will cause *interference* between data accesses and log writes. This is because the disk head may have to move to a new position when the log is written. As we discussed in the previous section, for relatively small seeks the rotational latency of the IO will dominate the seek time. So, if the log file is placed along with the other frequently accessed data on the outside half of the disk, interference will not be a significant problem. Striping across too few disks is a bigger problem in practice.

In a later section we will discuss the availability issues around striping the log files along with the data files.

The choice of stripe width for the log files is somewhat more tricky. Ideally we would like to stripe the log files using the same one megabyte stripe width as the rest of the files. However, the log files are written sequentially, and many storage systems limit the maximum size of a single write operation to one megabyte (or even less). If the maximum write size is limited, then using a one megabyte stripe width for the log files may not work well. In this case, a smaller stripe width such as 64K may work better.

Caching RAID controllers are an exception to this. If the storage subsystem can cache write operations in non-volatile RAM, then a one megabyte stripe width will work well for the log files. In this case, the write operation will be buffered in cache and the next log writes can be issued before the previous write is destaged to disk.

Archive log files have somewhat similar issues. However, unlike online logs it is possible to parallelise archive log writes at the Oracle level. This can be done in Oracle release 8.1 or later by configuring multiple archiver processes. In previous releases, archiving can be parallelised by issuing the 'archive log current' command. Alternately, the archive log files can also be striped using a 64K stripe width.

Summary of the Oracle IO Workload

The Oracle IO workload is complex, application dependent, operation dependent, and data dependent. Therefore, designing a load dependent storage configuration is very difficult. Fortunately most of this complexity can be ignored.

The main attributes of the Oracle IO workload that must be considered are the following:

- Sequential operations issue large IOs, random operation issue small IOs. Therefore, if the storage subsystem does not break up the IO operations, sequential operations will execute very efficiently.
- Direct read and write operations perform asynchronous read ahead. Therefore, for the fastest possible execution, ensure that the read ahead operations run on separate disks.
- Parallel execution generates very high IO rates. Therefore, data must be striped across many disks so that the storage configuration does not become a bottleneck to performance.
- Log writes are not parallelised at the Oracle level. Therefore the online log files should be striped. The width of the stripe depends on whether a caching RAID controller is being used.

THE SAME CONFIGURATION

With the background given in the previous sections, we are now ready to discuss the SAME configuration in detail. The goal of the SAME configuration is to be as simple as possible. There are only four basic rules in SAME. They are:

1. Stripe all files across all disks using a one megabyte stripe width
2. Mirror data for high availability
3. Place frequently accessed data on the outside half of the disk drives
4. Subset data by partition, not disk

STRIPE ALL FILES ACROSS ALL DISKS USING A ONE MEGABYTE STRIPE WIDTH

Striping all files across all disks ensures that the full bandwidth of all the disk drives is available for any operation. Therefore, parallel execution and other IO intensive operations will not be unnecessarily bottlenecked because of the disk configuration. You cannot reconfigure your storage without a great deal of effort, so striping across all disks is the safest and most future-proof choice you can make.

Some people base their storage configuration on average or expected IO activity. This is a very dangerous thing to do because it limits maximum throughput. Maximum throughput effects many things including the speed of batch jobs and recovery operations. This translates into delays completing critical business processing and more down time. Therefore, basing a configuration on expected IO activity costs the business money and reduces service levels.

By striping all files across all disks we equalize the load across disk drives and eliminate hot-spots. This improves response time by shortening disk queues. In theory, this extreme striping can cause "interference" between different jobs accessing the same disk, thus increasing response time. However, in practice this does not occur. By using large IOs for sequential access and concentrating frequently accessed data on the outside of disks we greatly reduce this effect. In practice the additional bandwidth and reduced queuing provided by extreme striping more than compensate for interference effects.

One of the most important benefits of striping across all disks is that it reduces administrative burden. With full striping, it is no longer necessary to constantly move files around in order to compensate for long disk queues caused by over utilized disks. Some sites spend significant resources "managing" this problem. Full striping eliminates this issue altogether.

Striping across all disks using a one megabyte stripe width is fast for sequential access. The large stripe width ensures that the disks spend most of their time transferring data, not positioning the disk head. The use of striping also allows Oracle level read-ahead to make use of multiple disks to speed the transfer rate.

Striping across all disks is great for random IO operations. A single disk can only perform about one hundred random IOs per second. Striping across all disks spreads this load across the entire disk farm. This ensures that there are not a few disk arms that bottleneck the performance of the database, while others sit idle.

Striping across all disks is the ideal, but for very large databases it may not be possible to stripe all data across all disks. This is because of limitations on the number of volumes and disks supported by current volume managers. The exact limits depend on the software and hardware stack in use. Often, high tens of disks works well, but limits are reached at several hundred disks.

Why does the SAME configuration recommend a one megabyte stripe width? Let's examine the reasoning behind this choice.

Why not use a stripe depth smaller than one megabyte? Smaller stripe depths can improve disk throughput for a single process by spreading a single IO across multiple disks. However IOs that are much smaller than a megabyte can cause seek time to becomes a large fraction of the total IO time. Therefore, the overall efficiency of the storage system is reduced. In some cases it may be worth trading off some efficiency for the increased throughput that smaller stripe depths provide. In general it is not necessary to do this though. Parallel execution at database level achieves high disk throughput while keeping efficiency high. Also, remember that the degree of parallelism can be dynamically tuned, whereas the stripe depth is very costly to change.

Why not use a stripe depth bigger than one megabyte? One megabyte is large enough that a sequential scan will spend most of its time transferring data instead of positioning the disk head. A bigger stripe depth will improve scan efficiency but only modestly. One megabyte is small enough that a large IO operation will not "hog" a single disk for very long before moving to the next one. Further, one megabyte is small enough that Oracle's asynchronous read-ahead operations access multiple disks. One megabyte is also small enough that a single stripe unit will not become a hot-spot. Any access hot-spot that is smaller than a megabyte should fit comfortably in the database buffer cache. Therefore it will not create a hot-spot on disk.

As discussed above, there are several competing goals that must be balanced when choosing a stripe depth. A one megabyte stripe depth satisfies all these goals without short changing any of them. This is not to say that there is something "magical" about the value of one megabyte. A value that is somewhat smaller or larger will also work well. However, values that are very different will leave some goal under optimized.

As disk technology advances, the stripe depth will need to be gradually increased to a value larger than one megabyte. This is because the transfer rate for disks is increasing faster than the positioning time is decreasing. This is a gradual change but it should be considered when deploying newer generation disk technology.

Mirror Data for High Availability

The simplest way to ensure that data is not lost is to implement mirroring at the storage subsystem level.

It is generally easiest to implement mirroring at the disk or partition level and then implement striping on top of the mirrored disks

The only way to lose data that is mirrored is to have multiple disk failures. Current disk drives are highly reliable so the probability of multiple failures is minute. Also, many systems allow a spare drive to be configured so that repair occurs rapidly after a mirror failure.

Some people worry that extensive use of striping will increase the probability of data loss. This is not true. Striping increases the damage that occurs when a double disk failure occurs but it does not make it more probable. If all files, are striped across all disks then any double disk failure that strikes both disks in a mirror will cause the entire database to need recovery instead of just the files that were on the failed disks. This lengthens the time to perform recovery. However, restore from tape and recovery is always a slow process, even for a single disk.

There are many failure scenarios that are much more likely than a double disk failure. These include operator or application error, data corruption, etc. The best way to protect against these more likely failures is to implement a standby database. A standby database will also protect against a double disk failure. Thus, achieving extreme high availability requires a standby database, and this also solves the problem of double disk failure. Hence, striping should not decrease the availability of a well managed database.

Striping the online log files across many disks does increase the probability that a double disk failure will destroy an online log. Loss of an online log will cause unrecoverable loss of the most recent updates. Again, the probability of this is very low, and a standby or remote mirror of the online log is the best way to protect against it. If a standby is too complex or expensive to implement and you want to reduce the very low probability of data loss, then you should implement additional mirroring of the online logs, control file, and archive logs. We call these three sets of files the recovery set. These are the files that are needed to perform a full recovery from a backup.

Ideally you should create an additional mirror of all files in the recovery set at the Oracle level. Mirroring by Oracle using file multiplexing is somewhat more resilient to corruptions than mirroring at the storage subsystem level[1]. The additional copy of the recovery set should be separated from the main copies in every way possible in order to reduce the incidence of correlated failures. For example, the additional mirror should be on a separate files system, which resides on a separate volume, which is on different disks, accessed by different controllers, and ideally on a separate RAID device.

Raid5 and its equivalents can also be used to prevent data loss. For databases that have large amounts of highly read intensive data, Raid5 provides a more cost effective solution than mirroring. However, as we move into a world of fewer very high capacity disks, the storage benefits of Raid5 will become less significant, and its performance costs will become more significant. Also, mirror splits provide a very nice way to perform a quick on-disk copy or backup of a database or tablespace. Mirror splits cannot be done in a Raid5 configuration. In addition, Raid5 is a more complicated technology that has more failure modes than mirroring. Therefore, we believe that Raid5 technology will become less important in the future. To simplify the storage configuration we recommend simple mirroring for most databases.

PLACE FREQUENTLY ACCESSED DATA ON THE OUTSIDE HALF OF THE DISK DRIVES

As was discussed before, by limiting most accesses to the outside half of disk drives, the random read time is reduced to close to the minimum, and the disk transfer rate is increased to close to the maximum.

One way to accomplish this is to measure or predict the IO rates of the various tablespaces in the database, and position tablespaces with higher than median IO rates on the outside half of the disk drives. In this solution, the online logs and archive logs should be placed on the outside half of the disk drive since they can receive a lot of IO activity during updates.

Another way to accomplish this is to just leave the inside half of the disk drives empty. Since disk drive capacity is increasing much faster than disk performance, it will become increasingly common to purchase many more disk drives than are necessary to just hold the data. In this solution, it makes sense to use the remaining space to store on-disk backups of the database. If disk trends continue for many more years, disks might start looking like geologic strata. The outer layers of the disk will hold newer data while the inner layers contain increasingly older data.

Some storage systems allow a primary mirror copy to be designated. The primary mirror copy is preferentially used for read operations while write operations happen to both disk drives. On these storage systems it may be beneficial to place the primary copy of the database on the outside half of the disk drives, and the non-primary copy on the inside half. Since most databases are quite read intensive, this may provide a high degree of locality to the outer half of the disk drive. In this solution, the logs files and their mirrors should be placed on the outside half of the disk drives since those are known to be write intensive.

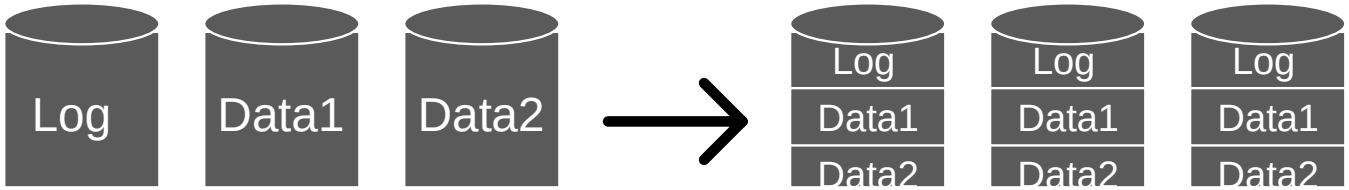
SUBSET DATA BY PARTITION NOT DISK

In some cases it is necessary to separate some database files from others. For example, some sites create a remote mirror of the online logs to prevent data loss in a disaster. Other sites want to separate read only data from writeable data. Others want to implement extra mirroring for a subset of the database. Often these data files are placed in a separate volume that is treated differently from the rest of the volumes on the system.

When creating these separate volumes it is common to place them on a separate set of disks from the rest of the database. Unfortunately, this isolates the files onto a small set of disks that can then become a performance bottleneck for some operations. Therefore, it is better to create a partition across all the existing disks, and concatenate and stripe the partition to form the separate volumes. This separates the data files logically while maintaining physical access to all the disks.

---

[1] Because Oracle multiplexing issues two entirely independent IOs. A failure in the IO stack at any level is less likely to effect both IOs. Also, Oracle performs logical validation of the blocks on read, and can correct corruptions by reading the other plex.



The subset by partition rule is not really a separate rule. It is a variant of the first rule. It really is just a way to preserve striping across all disks while creating subsets of data.

ADDING DISKS

The primary issue with the SAME methodology is that adding (or subtracting) disk space is more difficult to do. If the database is striped across all of the existing disks then adding a new disk means restriping the database. For a large database this can take a long time. There are a number of ways to handle this issue.

The best way to handle the issue of adding disks is to not do it. Ideally, careful planning is done ahead of time to ensure that sufficient space is allocated for the expansion needs of the database. With disk capacity increasing rapidly over time, it will become commonplace to over allocate storage space for databases to ensure there are enough disks to satisfy performance goals. Even if it is not possible to allocate all the needed space up front, it should be possible to add space in big increments so that space addition is a rare event.

Another option is to deploy storage technology that allows online reorganization of data. For example, recent releases of the Veritas volume manager allow online restriping of volumes. The restriping activity occurs in background and has negligible performance impact on online operations. Background restriping can take a long time to complete, but adding disks should occur very infrequently, so it should not matter how long it takes. This is also a very good solution because it preserves the benefits of the SAME methodology with little or no compromises.

Often, the need is for more capacity, not more performance. In this case it is often possible to replace the existing disks with a new generation of larger disks. Here the data on the existing disks is copied to the new disks and then new partitions are created with the remaining space on the new disks. These new partitions can then be striped to form a new volume(s) to use for additional database data.

Some times when more space is added, a new storage device (RAID box) is deployed to replace the old one. In this case, the storage on the new device can be configured and then a backup of the database can be restored onto the new device. Media recovery can then be applied to the database until it is almost caught up with the production database. At this point, the production database can be shut down and the remaining logs can be applied to the new storage device. Once this is done it is ready to replace the old storage device. Essentially a standby database is created from the production database and graceful switch over is performed to the standby database.

Another option is to create groups of disk and stripe only within the group. This is sometimes called a striping group. For example, you can separate the disk pool into groups of 8 disks. Striping is done within the eight disk striping group. Additional disks must be added in groups of eight. This is a compromise solution between complete striping and no striping. It simplifies the problem of adding disks, but brings back the issues of IO imbalances among the groups, and of parallel operations being limited by the throughput of one group. If you choose to go this route then make the number of disks per group as large as possible. The more CPUs you have the larger the group should be. Try to have at least four disks per CPU and preferably eight or more.

One of the problems with restriping when a new disk is added is that it requires reorganizing the entire database. If you have eighty disks and want to add ten more, it is expensive to reshuffle the data on the existing eighty disks. A way to avoid this is to create multiple partitions per disk. When more disks are added, some of the partitions on the existing disks can be moved to the new disks, and then the freed space can be used for creating fresh partitions. This will not produce files that are ninety way striped, but it will preserve the eighty way striping. We call this approach "sliding the stripe". Contrast this with striping groups in which to add ten disks requires restricting the striping to span no more than ten disks.

A detailed example will help in understanding this last option. Suppose you have six disks each of which is broken up into three partitions as shown in figure 1 below. The partitions labeled with the same number are then striped to form a volume. So, for example, all the partitions with the number '1' are striped to form a volume. These volumes are then added to a single file system on which the database is created.

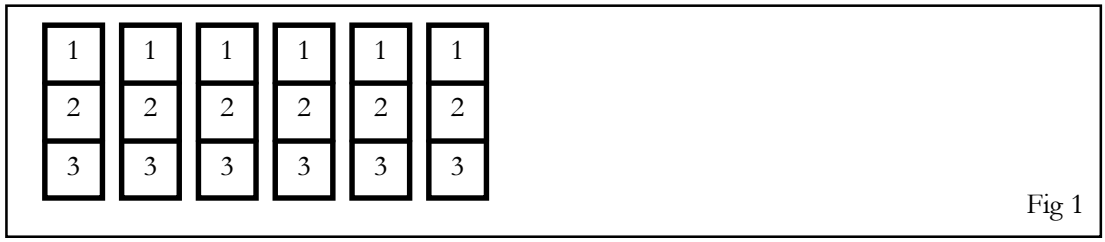


Fig 1

The number of partitions per disk determines the percentage we must increment the disks by in each step. In this example we have three partitions per disk so we would add 33% more disks. Since there are six disk in this example, we would add disks in sets of two. After we add the two disks we can move six partitions over, and create six new ones as shown in figure 2. If we had used six partitions per disk, we could add one disk at a time.

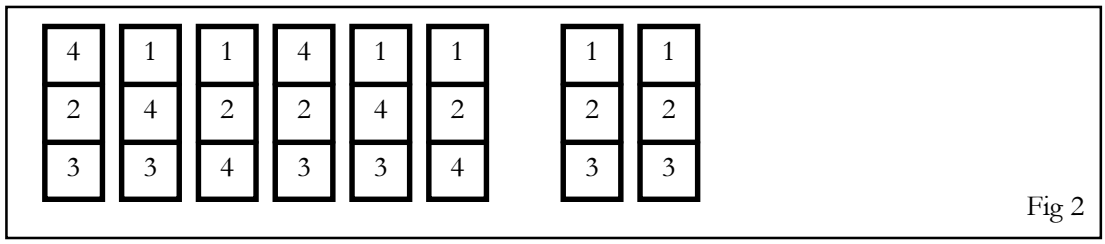


Fig 2

Note that we have created a new volume labeled number '4' that is striped across all the original disks. The old volumes or now striped across the new disks and some of the original disks.

If we add two more disks the configuration would look like figure 3:

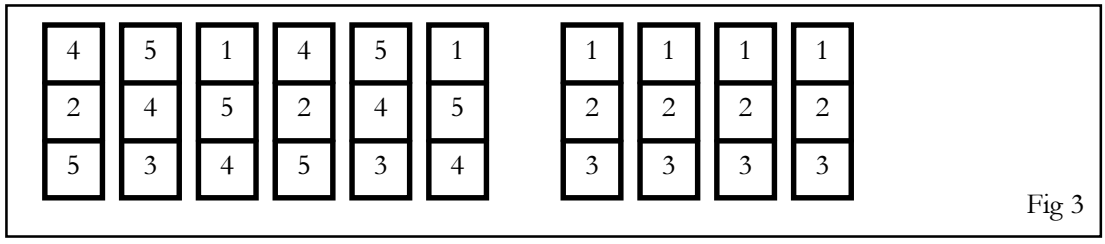


Fig 3

If two more disks are added the final result is shown in figure 4.

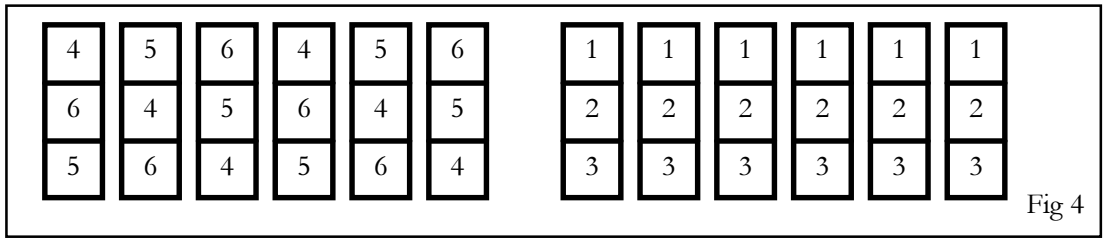


Fig 4

Note that the contents of the original six disks are now on the six new disks. If we want to keep adding disks we can apply the same process to these disks. This scheme is more complicated than simply adding striping groups but it does a better job of preserving the benefits of extensive striping. The degree of striping that existed on the original disk configuration is preserved as new disks are added. In the example, we always preserved six way striping as new disks were added. If you use a volume manager that allows partitions to be moved online, then this entire process can be performed with no downtime.

With today's technology there is sometimes a tradeoff between the effort required to modify a configuration and daily performance and monitoring. Remember that modifying a storage configuration should be a rare operation so it often makes sense to optimize for the normal case at the expense of the reorganization case.

DISCUSSION OF SAME

The best thing about the SAME configuration is that it is very simple and it works for all database workloads. You don't have to get into the details of the application. SAME works well for OLTP, warehouse, and batch workloads. SAME works well for any application, operation or data subset. SAME produces maximum IO bandwidth to allow parallel execution to scale.

To set up a storage configuration using SAME you only need to know the total disk space needed by the application and the approximate total IO throughput needed. These determine the number of disks. Once the number of disks are known you can simply mirror the disks, stripe the resulting volumes and then build a file system on top of this.

The SAME configuration is viable for a number of reasons:

- Current volume managers allow efficient striping across many disks
- Mirrored disks with hot replacement provide highly reliable storage. This balances the risk of using extensive striping.
- Sequential access from Oracle issues large IOs. This minimizes seek overhead.
- Parallel execution produces very high IO parallelism rates. This removes the need for fine grain striping.
- RAID caches absorb high write rates. This eliminates the need for fine grain striping of log files.

KEEP IT SIMPLE

It is the goal of the SAME configuration to be as *simple* as possible. We believe there is great power in simplicity. Many people try to micro-optimize a storage configuration. In our experience this usually creates complexity and extra administrative burden without providing any significant gains. Therefore, if you believe in the ideas presented in this paper, try to adopt the configuration without adding any embellishments unless you can demonstrate that the basic configuration will not work well.

A few of the recommendations in this paper are contrary to current practice. In particular, people often question the wisdom of striping across *all* disks, and of including the log files in the same stripe set as the data files. We have performed a number of experiments in which the logs were separated into distinct disks, and data files were grouped into distinct sets of disks. In all the experiments, the simple "stripe all files over every disk" approach performed as well or better than the best subset that experienced experts were able to come up with. Fundamentally, striping across all disks allows *any* operation to use *all* the available disk bandwidth. Separating data into islands of disks causes disk bottlenecks to occur for some operations.

It is also important to *not* organize the storage configuration around optimizing rare events such as double disk failure and disk space addition. It is better to optimize around daily performance and scalability, even if it means more complexity for some rare event.

BACKUP AND RECOVERY

The SAME configuration can improve the performance of backup and recovery. Backup and recovery are very IO intensive operations. The intensive striping that is used in the SAME configuration allows these operations to run at full speed with minimal disk bottlenecks.

CONCLUSION

Storage configuration and maintenance does not have to be complex and labor intensive. By following some simple guidelines you can create a storage configuration that is simple, has excellent performance, and good availability. The SAME configuration has only four rules.

1. Stripe all files across all disks using a one megabyte stripe width
2. Mirror data for high availability
3. Place frequently accessed data on the outside half of the disk drives
4. Subset data by partition, not disk

The SAME methodology is based on careful analysis of current and future disk technology, combined with a thorough understanding of the performance and availability requirements of the Oracle database. SAME is an abstract methodology that is not tied to any particular storage management product in the market today. The basic ideas of SAME can be implemented with technology that exists today. Some of the details described in this paper are not available today, but they may become available over the next few years. Oracle is working with leading storage vendors to optimize and automate this methodology.

ACKNOWLEDGMENTS

Several of the ideas presented in this paper were invented by Bill Bridge. Other ideas resulted from discussions with Garry Lemasa, Carol Colrain, Jegraj Djegaradjane, and Vikram Joshi.