Six Easy Pieces: Essentials of Database Tuning for Packaged Applications

Mike Swing

UTOUG 2012

mswing@trutek.com

Six Easy Pieces

1. Check Initialization Parameters
2. Optimize Hardware
3. Partition Large Tables > 10 GB
4. Use Parallel Execution
5. Create Materialized Views
6. Perform Periodic Maintenance

EBS and Demantra

- The E-Business Suite is considered an OLTP (on line transaction processing) system while Demantra is a planning system, aka DSS (decision support system) and performs more like a data warehouse, but still has some characteristics of an OLTP system.
- For example, with many data warehouses, backups are not performed regularly, because data can be re-loaded from the original data load. This is not the case with Demantra, in addition to the daily and weekly data load, promotions (and other data) are created manually, requiring a valid backup method.

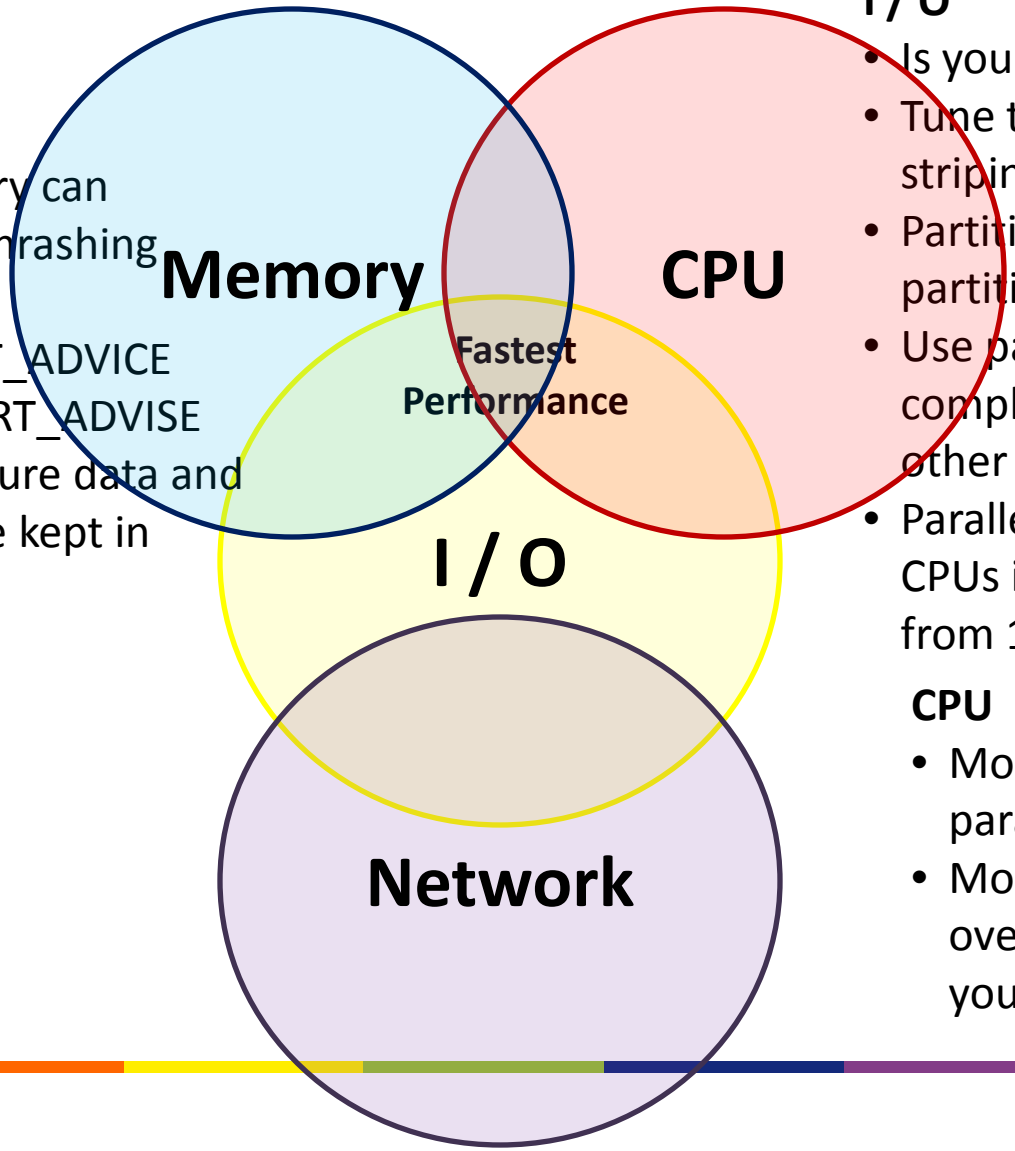
EBS and Demantra

- The workload for the EBS is forms, reports, interfaces (dataloads) and some planning activity, like MRP. The biggest workloads are the planning processes. In Demantra, worksheets, simulations and dataloads characterize the workload.
- Demantra is a planning application. Worksheets read millions of rows of data and the simulations iterate until the best planning solution is found. Worksheets are very IO intensive, while the simulations demand more memory and CPUs.
- Both EBS and Demantra benefit from more memory for the SGA and PGA, because most of the data being read from disk is recent data being used by more than one process or user. Without enough memory, IO thrashing occurs.

Hardware Tuning

Memory

- More memory can reduce I/O Thrashing
- Use the SGA_TARGET_ADVICE and PGA_TART_ADVISE tables to ensure data and programs are kept in memory.



I / O

- Is your system IO bound?
- Tune the hardware, block size, striping, number of disks
- Partitioning can reduce IO with partitioning pruning
- Use parallel execution to complete full table scans and other operations in parallel.
- Parallel Execution with more CPUs is useless if all reads come from 1 disk.

CPU

- More CPUs allows for more parallel operations
- More CPUs won't help overall performance if you're I/O bound.

Some Goals from this Presentation

Learn how to use the following to improve the performance of packaged applications:

Solid State Disk – to improve hardware performance
Increase Memory – to keep SGA and PGA in memory
Partition – use partitioning pruning to improve SQL
Parallel Query / Parallel Execution – assuming you have the I/O and CPU capacity
Maintenance - ReOrg your data by primary access path
Correctly set the initialization parameters

Optimizing Hardware Resources and Configuration

More Memory

Memory utilization- more data in memory reduces the strain on the IO subsystem. More data in memory provides better CPU utilization.

More IOPS

Verify Optimal IO configuration – check block sizes, stripe size and width, number of disks

Reduce Demand for IO

- Keep more Data in Memory
- Reduce Row Chaining, Row Migration, and Improve the Clustering Factor

Use Solid State Disk for files with a high volume of transactions

- All dbf's, control files, log files, /tmp. Separate the log files on another SSD if possible.

More CPUs

More CPUs can actually make the problem worse if the IOPS or Memory are insufficient. However, in order to fully utilize the power of parallel execution, more CPUs may be necessary.

Partitioning and Parallel Execution

Partitioning

- Partition Large Tables for Better Selectivity with Partition Pruning
- Enable Parallel DML for Load Programs
- The database licensing option costs money, but can be used to help fix bad SQL.

Parallel Query – Parallel Execution

- More CPUs – Normally each worksheet uses only one CPU. With Parallel Query, more CPUs can be used for each worksheet, thereby decreasing the worksheet execution times.

Materialized Views and Maintenance

Materialized Views

- Replace complex views or subqueries with a materialized view
- Determine refresh rate

Maintenance

- Gather Statistics
- Remove Row Chaining using “Alter Tablespace Move Table”
- ReOrg Tables to improve the clustering factor
- Rebuild Tables and move null columns to the end of the table
- Rebuild Indexes
- Periodically Rebuild Partitions

Initialization Parameters

Optimizer_index_caching = 0
Optimizer_index_cost_adj = 100
Optimizer_mode = ALL_ROWS
Processes =1200, sessions=1000
_btree_bitmap_plans = FALSE
FILESYTEMIO_OPTIONS
Use the DEFAULT BUFFER_POOL
Use a larger database block size for Demantra, 32K block size is great in our tests
PARALLEL_MAX_SERVERS
PARALLEL_MIN_SERVERS
PARALLEL_MIN_PERCENT
PARALLEL_ADAPTIVE_MULTI_USER

Overview of the Tuning Process

1. Characterize the workload, identify tests that adequately simulate the workload. Run these tests after each change to measure the effect.
2. Identify differences between systems - "Why is the DEV server faster than PROD?“
   - Database configuration difference
   - Initialization parameters, for example:

Compare Initialization Parameters Across Instances

log_archive_trace	0	0	0	0	Establish archivelog operation tracing level
log_buffer	268423168	14489600	888608	14465024	redo circular buffer size
log_checkpoint_interval	0	0	0	0	# redo blocks checkpoint threshold
log_checkpoint_timeout	0	0	0	0	Maximum time interval between checkpoints in seconds
log_checkpoints_to_alert	FALSE	FALSE	FALSE	FALSE	log checkpoint begin/end to alert file
log_file_name_convert					logfile name convert patterns and strings for standby/clone db
logmnr_max_persistent_sessions	1	1	1	1	maximum number of threads to mine
max_commit_propagation_delay	0	0	0	0	Max age of new snapshot in .01 seconds
max_dispatchers					max number of dispatchers

Find the Differences - Continued

Memory Utilization
Swap space
Memory - paging?
Tablespace configuration
Hardware differences
Number of CPUs, Disk configuration, number of disks, speed of disks, number of controllers stripe size & stripe width

Overview of the Tuning Process

3. **Resolve the differences**

Make changes, one at a time, and measure the effect of the change, by timing known worksheets.

If the platforms all perform consistently the same, because the configuration is relatively the same (we could have less CPUs and less memory, but we need a similar disk configuration), then we can make changes in the DEV system and be confident the changes will positively affect the PROD system.

Overview of the Tuning Process

4. Identify the SYSTEM WAIT events.

Determine any hardware bottlenecks.
Check for SQL using unselective index scans
Check for Foreign Keys with missing indexes
A larger buffer cache can help – really, test large sizes of SGA_TARGET and PGA_AGGREGATE_TARGET

Overview of the Tuning Process

5. Modify procedures to improve performance. Parallelize the load programs. Implement parallel query / parallel execution /
6. Implement partitioning for very large tables or to improve parallel loads.
7. If software changes don't improve performance enough, then make hardware changes.
8. First, add more memory from the current setting of 32 GB to 128 GB, then, if performance is still waiting on the data to be initially read from disk, add solid state drives for the database files: 2 - 250GB SSD mirrored. Add more CPUs and use PX /PQ.
9. Configure the database for new hardware.
10. Test

Implement and Measure Change

Often at the end of a tuning exercise, it isn't possible to identify the exact changes that fixed the problem.

To identify the changes that provide the most benefit, it is recommended to implement only one change at a time.

Most companies with a performance crisis implement a number of changes at once, and can't identify the changes that provided the benefit.

Performance tuning is an iterative process, because solving one bottleneck often uncovers another (sometimes worse) problem.

Implement and Measure Change

Wait Events Statistics

To get an indication of where the database time is spent, follow these steps:

- Examine the data collection for V$SYSTEM_EVENT. The events of interest should be ranked by wait time.
- The V$SESSION, V$SESSION_WAIT, V$SESSION_EVENT, and V$SYSTEM_EVENT views provide information on what resources were waited for, and, since the configuration parameter TIMED_STATISTICS is set to true, how long each resource or wait event was waited for.

Wait Events Statistics

The highest level view is the V$SYSTEM_EVENT view and the most detailed is the V$SESSION view.

- **V$SYSTEM_EVENT** lists the events and times waited for by the whole instance (that is, all session wait events data rolled up) since instance startup.
- **V$SESSION_EVENT** lists the cumulative history of events waited for on each session. After a session exits, the wait event statistics for that session are removed from this view.
- **V$SESSION_WAIT** is a current state view. It lists either the event currently being waited for or the event last waited for on each session
- **V$SESSION** lists session information for each current session. It lists either the event currently being waited for or the event last waited for on each session. This view also contains information on blocking sessions.

Wait Events Statistics V$SYSTEM_EVENT

EVENT	TOTAL WAITS	TOTAL_TIMEOUTS	TIME WAITED	AVERAGE TIME	WAIT_CLASS
rdbms ipc message	2858798	2456614	784906366	274.56	Idle
SQL*Net message from client	942932	0	558364537	592.16	Idle
pmon timer	195359	195351	57195149	292.77	Idle
Streams AQ: waiting for messages in the queue	116749	116710	57186568	489.82	Idle
Streams AQ: qmn slave idle wait	20869	0	57052208	2733.83	Idle
Streams AQ: qmn coordinator idle wait	41446	20971	57050456	1376.5	Idle
virtual circuit status	19535	19534	57033084	2919.53	Idle
dispatcher timer	9767	9767	56837766	5819.37	Idle
smon timer	6349	1693	55332756	8715.19	Idle
Streams AQ: waiting for time management or cleanup tasks	442	320	32930851	74504.19	Idle
Backup: sbtwrite2	5835727	0	21135800	3.62	Administrative
db file sequential read	29602868	0	21073645	0.71	User I/O
read by other session	47138782	0	19276829	0.41	User I/O
Backup: sbtbackup	107	0	3219974	30093.22	Administrative
jobq slave wait	7131	7120	2094307	293.69	Idle
RMAN backup & recovery I/O	3607441	0	1791398	0.5	System I/O
PX Idle Wait	8403	8477	1636636	194.77	Idle
PL/SQL lock timer	48	48	630438	13134.13	Idle
PX Deq: Execution Msg	4410	2693	539149	122.26	Idle
control file parallel write	207980	0	528055	2.54	System I/O

Wait Events Statistics

The db file scattered read wait event is the number of full table scans and the db file sequential read is the number of range scans. The ratio of db file sequential read / db file scattered read is 55.0.

The other disturbing wait event is “read by other session”; this wait event indicates the number of times another session has waited for a block being used by a current session.

The “**read by other session**” wait event indicates several processes repeatedly reading the same blocks, e.g. many sessions scanning the same index or performing full table scans on the same table. Tuning this issue is a matter of finding and eliminating this contention.

Index Range Scans

An index range scan is a common operation for accessing selective data. It can be bounded (bounded on both sides) or unbounded (on one or both sides).

Data is returned in the ascending order of index columns. Multiple rows with identical values are sorted in ascending order by rowid.

If data must be sorted by order, then use the ORDER BY clause, and do not rely on an index. If an index can be used to satisfy an ORDER BY clause, then the optimizer uses this option and avoids a sort.

V$SESSION_WAIT

Because V$SESSION_WAIT is a **current state view**, it also contains a **finer-granularity** of information than V$SESSION_EVENT or V$SYSTEM_EVENT.

It includes additional identifying data for the current event in three parameter columns: P1, P2, and P3.

For example, V$SESSION_EVENT can show that a session had many waits on the db file sequential read, but it does not show the file and block number.

V$SESSION_WAIT

The view V$SESSION_WAIT shows the file number in P1, the block number read in P2, and the number of blocks read in P3.

This is an example of querying based on a specific wait event; in this case the ‘read by other session’ wait event indicates multiple sessions are trying to read the same block:

```
SELECT p1 "file#", p2 "block#", p3 "class#"
FROM v$session_wait
WHERE event = 'read by other session';
```

This is an indication of “hot blocks”. This can usually be resolved by adding more memory to the SGA_TARGET initialization parameter and PGA_AGGREGATE_TARGET initialization parameter.

V$SESSION_WAIT

The "**db file sequential read**" - a sequential read reads data into contiguous memory (a scattered read reads multiple blocks and scatters them into different buffers in the SGA).
A sequential read is a single-block read or more typically an index range scan. In our case, this means that indexes are being used, just used very often.

The following query will give the name and type of the object:

```
SELECT relative_fno, owner, segment_name, segment_type
FROM dba_extents
WHERE file_id = &file
AND &block BETWEEN block_id AND block_id + blocks - 1
```

V$SESSION_WAIT

The view V$SESSION_WAIT shows the file number in P1, the block number read in P2, and the number of blocks read in P3.
This is an example of querying based on a specific wait event; in this case the ‘read by other session’ wait event indicates multiple sessions are trying to read the same block:

```
SELECT p1 "file#", p2 "block#", p3 "class#"
FROM v$session_wait
WHERE event = 'read by other session';
```

This is an indication of “hot blocks”. This can usually be resolved by adding more memory to the SGA_TARGET initialization parameter and PGA_AGGREGATE_TARGET initialization parameter.

V$SESSION_WAIT

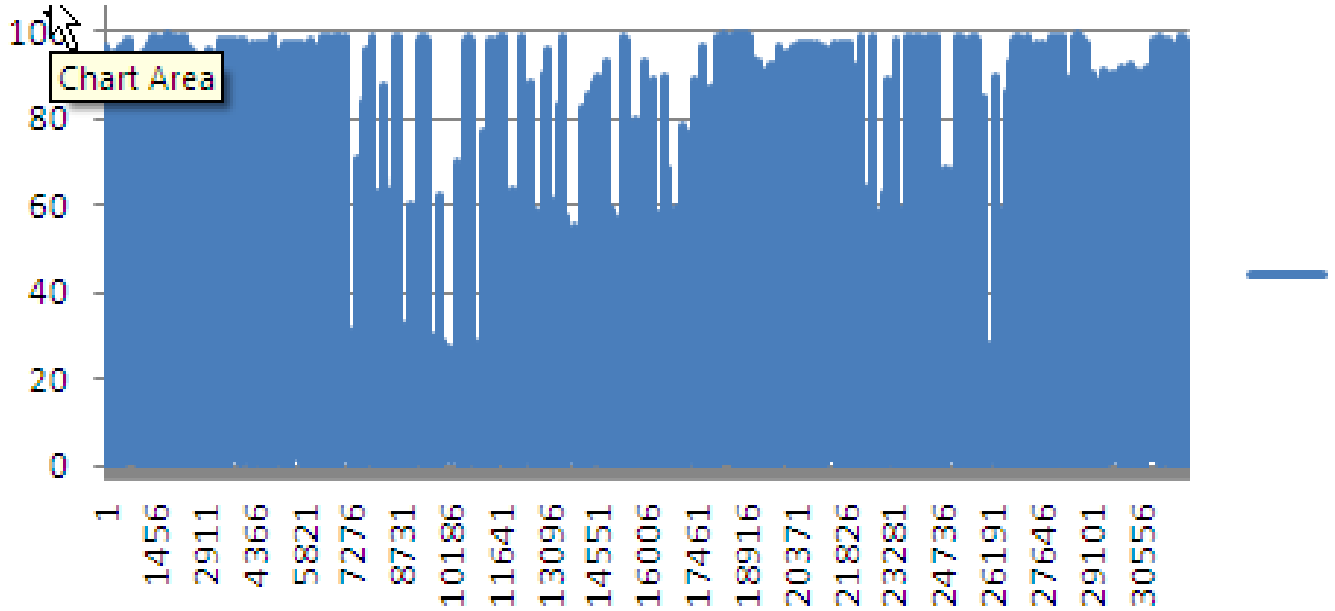
SELECT relative_fno, owner, segment_name, segment_type
FROM dba_extents
WHERE file_id = 21
AND 271986 BETWEEN block_id AND block_id + blocks – 1

RELATIVE_FNO	OWNER	SEGMENT_NAME	SEGMENT_TYPE
21	TPMPROD	SALES_DATA	TABLE

We absolutely have an IO bottleneck. We have two solutions: faster IO or more memory so less IO is required. From above we can see that we are reading the same blocks over and over.

IO Waits

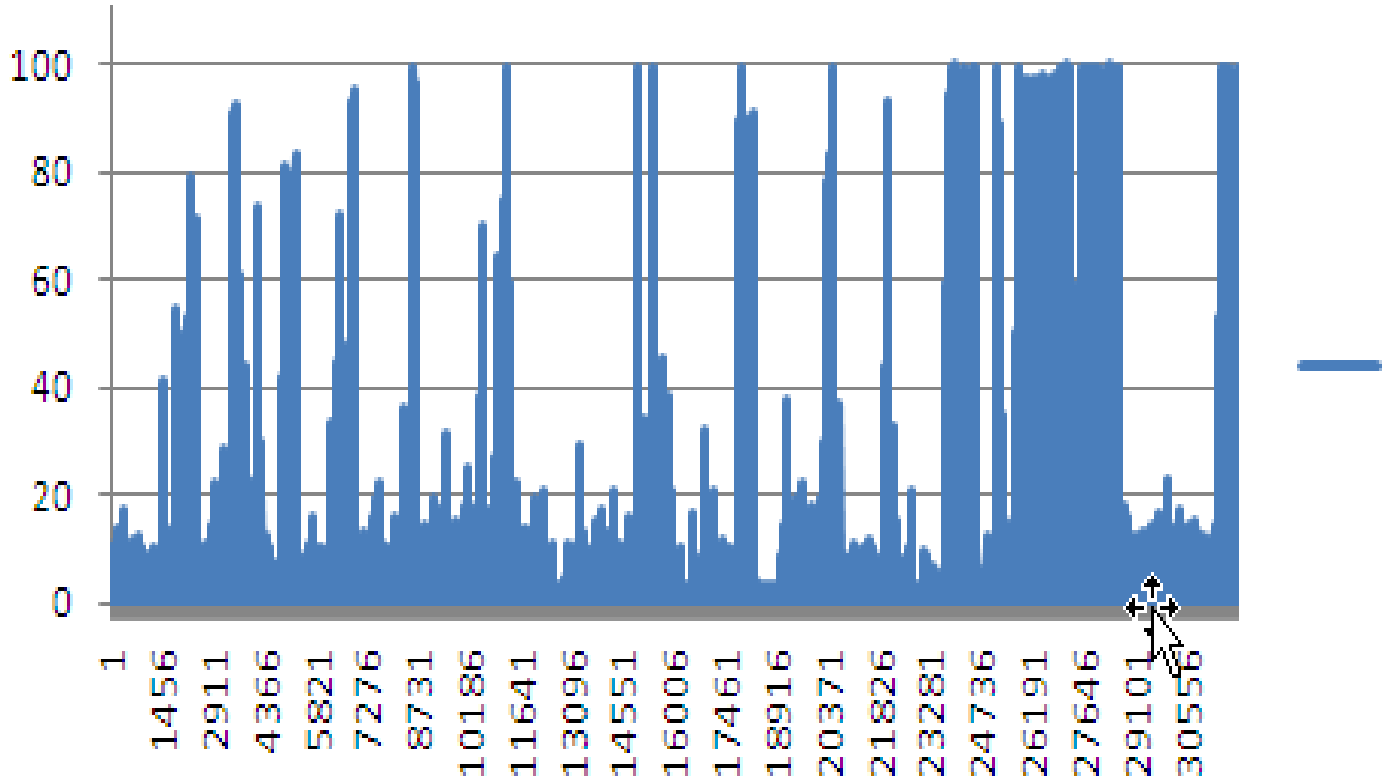
The following diagram shows the percentage of IO Waits over 32000 minutes on PROD. This shows IO Wait percentage consistently over 90%:



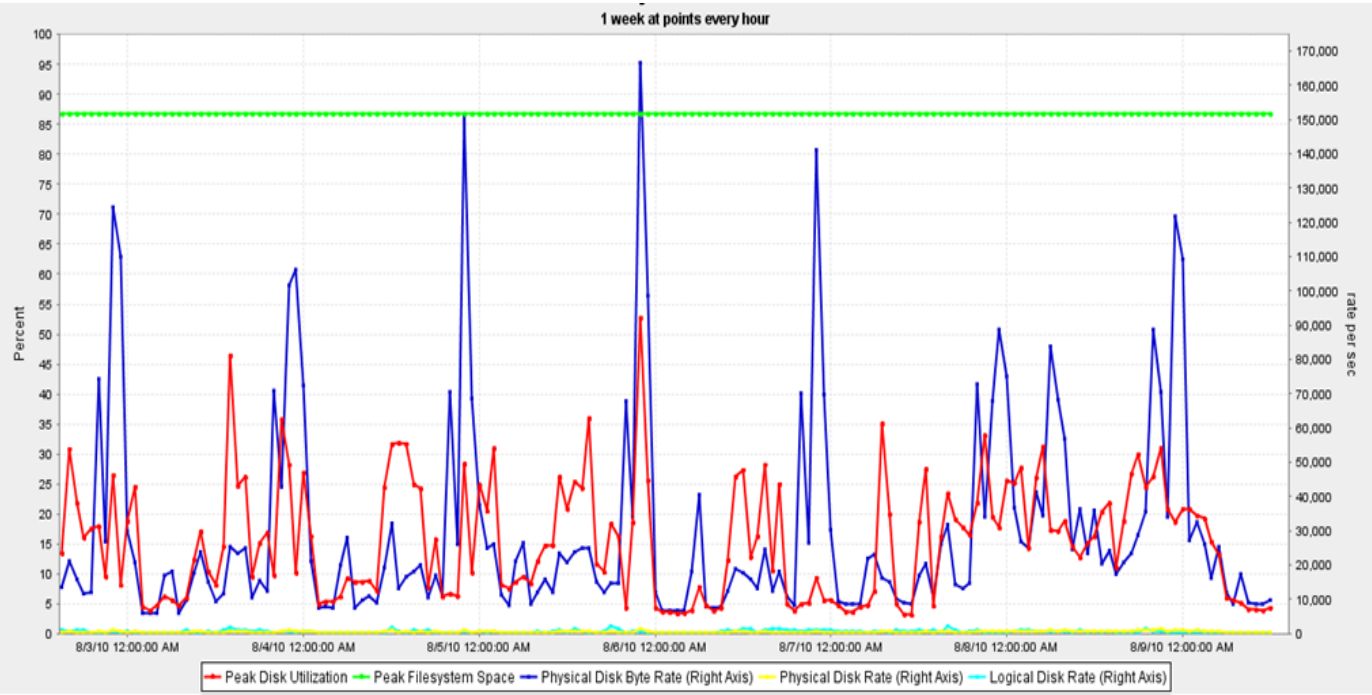
The corresponding CPU usage for the same period of time shows the CPUs relatively idle:

CPU Waits

The corresponding CPU usage for the same period of time shows the CPUs relatively idle:



I/O Management



This summary for a Demantra planning system shows the physical disk byte rate per second exceeds 150K bytes per second. This is typical of a system that doesn't have enough memory for the database. Proper sizing of the SGA and the PGA should reduce the physical disk byte rate per second to below 20K.

Clustered Data

A SQL statement can be executed in many different ways, such as full table scans, index scans, nested loops, and hash joins.

The query optimizer determines the most efficient way to execute a SQL statement after considering many factors related to the objects referenced and the conditions specified in the query.

This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

Clustered Data

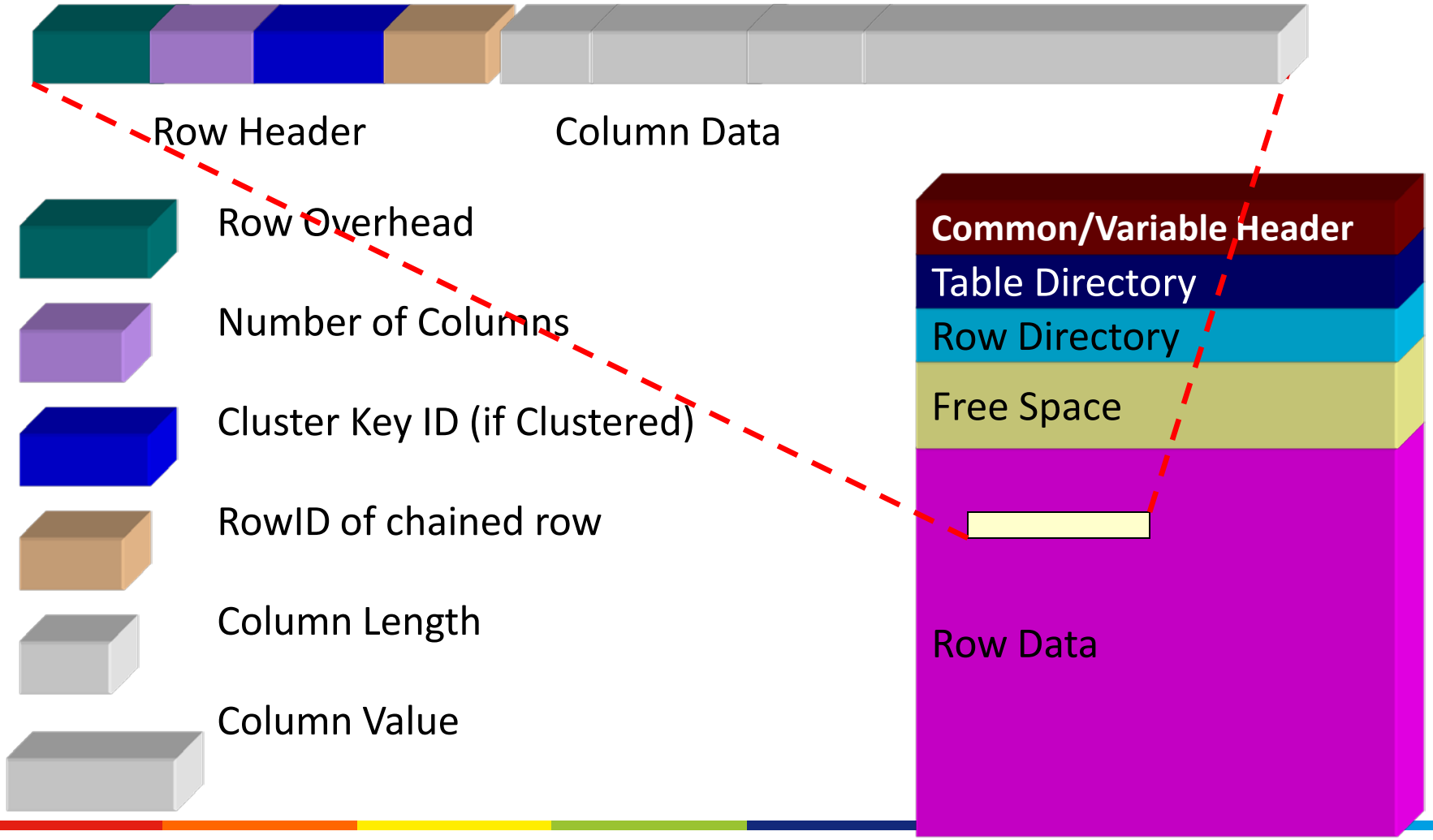
- A less obvious issue which can affect the IO rates is **how well data is clustered physically**. For example, assume that you frequently fetch rows from a table where a column is between two values via an index scan. If there are 100 rows in each index block then the two extremes are:
  - Each of the table rows is in a different physical block (100 blocks need to be read for each index block)
  - The table rows are all located in the few adjacent blocks (a handful of blocks need to be read for each index block)
- Pre-sorting or re-organizing data can help performance.
- It can help to place files with **frequent index scans on disks that are buffered by an O/S file system cache**. Often this will allow some of Oracle's read requests to be satisfied from the OS cache rather than from a real disk IO.
- If the filesystem supports DIRECTIO, use DIRECTIO

I/O by Blocks, not Rows

Oracle does I/O by blocks. The optimizer's decision to use full table scans is influenced by the percentage of blocks accessed, not rows. This is called the index clustering factor. If blocks contain single rows, then rows accessed and blocks accessed are the same.

However, most tables have multiple rows in each block. Consequently, the desired number of rows could be clustered together in a few blocks, or they could be spread out over a larger number of blocks.

How Data Blocks Store Rows



I/O by Blocks, not Rows

Although the clustering factor is a property of the index, the clustering factor actually relates to the spread of similar indexed column values within data blocks in the table.

A lower clustering factor indicates that the individual rows are concentrated within fewer blocks in the table.

Conversely, a high clustering factor indicates that the individual rows are scattered more randomly across blocks in the table.

Therefore, a high clustering factor means that it costs more to use a range scan to fetch rows by rowid, because more blocks in the table need to be visited to return the data.

Oracle Does I/O by Blocks

Therefore, the optimizer's decision to use full table scans is influenced by the percentage of blocks accessed, not rows.

This is called the index clustering factor. If blocks contain single rows, then rows accessed and blocks accessed are the same.

Clustering factor is available in dba_indexes and dba_ind_partitions in the clustering_factor column.

Measure Rows Out of Order

```
SELECT (ROUND(((SELECT COUNT(*) AS CNT
FROM (SELECT SALES_DATE, ITEM_ID, LOCATION_ID
    ,RELATIVE_FNO     ,BLOCK_NUMBER     ,ROW_NUMBER     ,DATA_ROW     ,(LAG(DATA_ROW)
OVER
(PARTITION BY RELATIVE_FNO, BLOCK_NUMBER
 ORDER BY ROW_NUMBER)) AS PREV_DATA_ROW
    FROM (SELECT SALES_DATE, ITEM_ID, LOCATION_ID
        ,RELATIVE_FNO         ,BLOCK_NUMBER         ,ROW_NUMBER         ,(DENSE_RANK() OVER
(PARTITION BY RELATIVE_FNO, BLOCK_NUMBER
 ORDER BY SALES_DATE, ITEM_ID, LOCATION_ID)) AS DATA_ROW
        FROM (SELECT SALES_DATE, ITEM_ID, LOCATION_ID
                ,SUBSTR(ROWID,11,6)||SUBSTR(ROWID,1,3) RELATIVE_FNO
                ,SUBSTR(ROWID,4,6) AS BLOCK_NUMBER
                ,SUBSTR(ROWID,17,3) AS ROW_NUMBER
            FROM SALES_DATA
            ) C
        ) B
    ) A
WHERE DATA_ROW != PREV_DATA_ROW AND DATA_ROW != PREV_DATA_ROW + 1)/
(SELECT COUNT(*) FROM SALES_DATA)),3)*100) AS "Out Of Order Ratio %“ FROM DUAL;
```

Database Block Size = File System Buffer Size

"Although there is every reason ***to use DirectIO,*** if you can't use DirectIO it is important to ensure that the database block size matches the file system buffer size exactly.

If the database block size is smaller than the file system buffer size, then DBWn has to perform partial block writes.

Partial block writes make DBWn work less efficiently and foreground processes will begin to wait for its services at lower workloads.

Foreground processes are seldom directly affected by partial block writes because their writes, normally to temporary segments, are typically large relative to the database block size.

Database Block Size > File System Buffer Size

If the database block size is larger than the file system buffer size, then all single block reads and writes are split into a series of distinct physical I/O operations - one for each file system buffer addressed - and these I/O operations are performed one at a time.

For writes, a full rotational latency is sustained between each pair of file system buffer writes.

For reads, all except the initial file system buffer read can normally be satisfied cheaply from the disk track buffers.

However, the distinct contiguous file system buffer reads trigger the file system read ahead mechanism, even for random reads, causing inappropriate read ahead.

Database Block Size = File System Buffer Size

There are two reasons that the file-system block size and the database block size should match...

if DB_BLOCK_SIZE is larger than file-system block size, then each database I/O will cause multiple file-system blocks to be accessed

- this might be bad if it triggers any "read-ahead" logic within the file-system, causing the ultimate I/O operation to read more data unnecessarily

if DB_BLOCK_SIZE is smaller than file-system block size, then each database I/O will result in multiple database blocks worth of data being accessed all the time, even when you only need one DB block.

If DB_BLOCK_SIZE is not the same size or larger multiple of the file-system block size, then direct I/O (i.e. unbuffered file-system I/O) cannot be used on most UNIX variants

Database Block Size

Reads

- Regardless of the size of the data, the goal is to minimize the number of reads required to retrieve the desired data.
- If the rows are small and access is predominantly random, then choose a smaller block size.
- If the rows are small and access is predominantly sequential, then choose a larger block size.
- If the rows are small and access is both random and sequential, then it might be effective to choose a larger block size.
- If the rows are large, such as rows containing large object (LOB) data, then choose a larger block size.

Database Block Size

Writes

For high-concurrency OLTP systems, consider appropriate values for INITRANS, MAXTRANS, and FREELISTS when using a larger block size. These parameters affect the degree of update concurrency allowed within a block. However, you do not need to specify the value for FREELISTS when using automatic segment-space management.

If you are uncertain about which block size to choose, then try a database block size of 8 KB for most systems that process a large number of transactions. This represents a good compromise and is usually effective. Typically, systems processing LOB data need more than 8 KB.

Stripe Size

“It is the number of spindles available that should matter, not the sizes of stripes, in my opinion.

If you can stripe a volume across 19 physical disk drives or spindles, then it almost certain to be faster than a volume striped across 4-6 spindles. “

Jonathan Lewis

Stripe Size

"A smaller stripe size will guarantee that even low volumes of I/O requests utilize all those spindles, but that is a double-edged sword.

If a single I/O request can make all the spindles busy, then what will happen when many concurrent I/O requests occur, if all spindles are responding to the first I/O request.

That is where larger stripe sizes may help - each I/O request may only activate one or two spindles, but the striping pattern should cause all of spindles to serve all the requests.

If the striping size becomes too large, then hot spots may occur. That is my understanding of why vendors like EMC have settled on stripe sizes of about 1Mb for workloads like Oracle."

ASM

“For Oracle database files (i.e. datafiles, tempfiles, controlfiles, and online redo logfiles), it is recommended to consider discontinuing the use of journaled file-systems, in favor of implementing Oracle’s Automatic Storage Management (ASM) product. “

ASM is a replacement for volume managers and file-systems, and is a stable product even in its introductory releases of Oracle10gR2.

ASM

- The main advantage of ASM is that it was created by Oracle to optimize Oracle database I/O, whereas file-systems were created for more general-purpose non-database usages.
- Two of the most important I/O features for Oracle databases, namely *asynchronous I/O* and *direct I/O*, come native with ASM, while with file-systems these features are either difficult to configure or are not available at all.
- Using asynchronous I/O, ASM and the Oracle database instance permit kernalized asynchronous I/O worker threads to increase the bandwidth of many types of sequential I/O patterns.

ASM

Also, UNIX file-systems utilize the *UNIX buffer cache* (UBC), which is generally consumes all but about 1%-5% of available free physical memory.

Using *direct I/O*, ASM lets the Oracle database instance do all of the memory caching, thus eliminating the double-caching of the UBC with the attendant concurrency bottlenecks, and reducing the usage of the UBC tremendously.

Using Operating System Caches

Operating systems and device controllers provide data caches, and by default all database I/O goes through the file system cache. On some UNIX systems, direct I/O is available to the filestore.

DirectIO allows the database files to be accessed within the UNIX file system, bypassing the file system cache. It saves CPU resources and allows the file system cache to be dedicated to non-database activity, such as program texts and spool files.

Although the operating system cache is often redundant because the Oracle buffer cache buffers blocks, there are a number of cases where Oracle does not use the Oracle buffer cache. **Direct I/O bypasses the operating system cache and does not use the operating system cache**. This may yield worse performance than using operating system buffering.

Avoid Buffered I/O

Buffered datafile I/O wastes memory because Oracle data blocks are cached both in the SGA and in the file system buffer cache with substantial overlap between the two sets of blocks that are cached.

The file system cache should be used to buffer non-Oracle I/O only.

Using it to attempt to enhance the caching of Oracle data just wastes memory, and lots of it. Oracle can cache its own data much more effectively than the operating system can, most of the time.

Avoid Buffered I/O

Oracle has a sophisticated touch count based cache replacement algorithm that is sensitive to both the frequency and of data usage.

Oracle also avoids caching data that is unlikely to be, or will never be reused, and DBAs can fine tune Oracle's caching decisions by allocating segments to the KEEP and RECYCLE buffer pools.

There is also about a 50% savings in CPU time and elapsed time when getting data from the Oracle buffer cache rather than the filesystem buffer cache.

filesystemio_options

The hidden *_filesystemio_options* parameter was introduced in Oracle version 8.1.7 to control the use of program selectable direct I/O and asynchronous I/O against file system based database files. It is no longer a hidden parameter from release 9.2 onwards. The following table shows the values that can be used when setting this parameter.

	Buffered I/O	**Direct I/O**
Synchronous I/O	none	directIO
Asynchronous I/O	asynch	setall

filesystemio_options

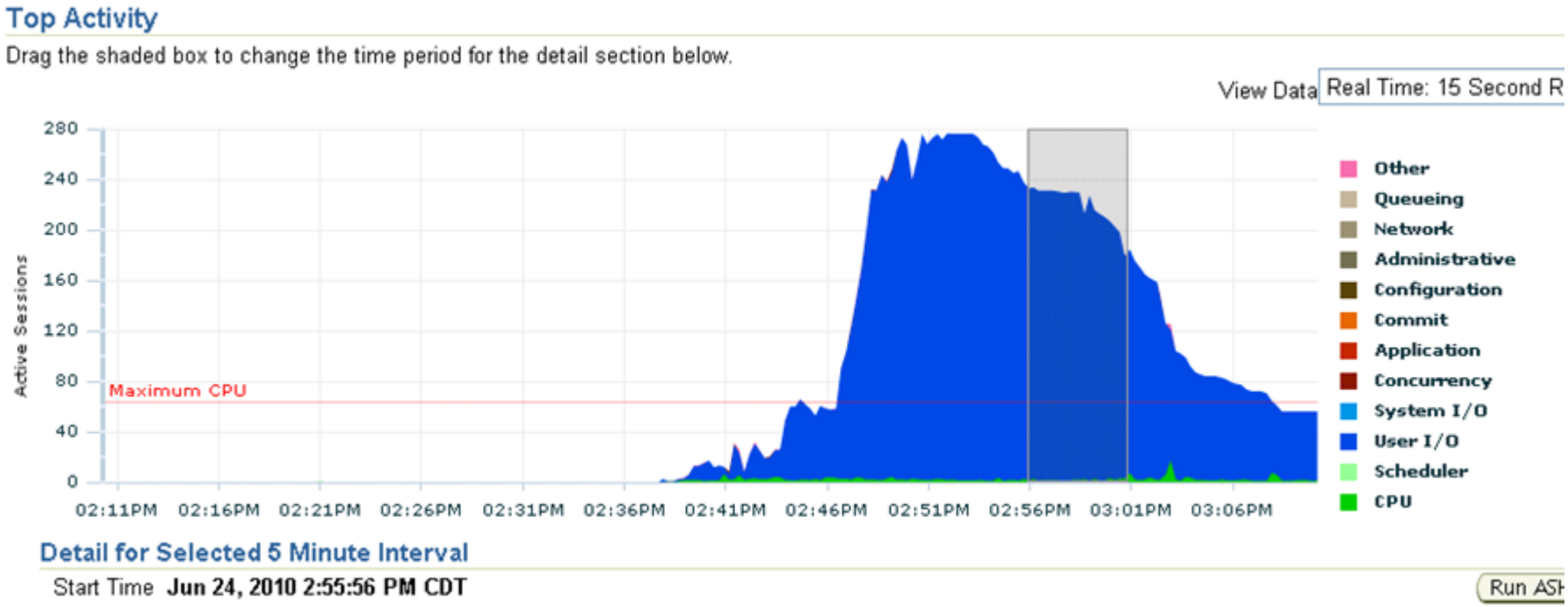
Databases that use a combination of raw log files and raw temp files with filesystem based datafiles can use asynchronous I/O against the raw files,

> but to avoid inefficient threaded asynchronous I/O against the datafiles try setting *disk_asynch_io* to the default of TRUE and setting *filesystemio_options* to either **none** or **directIO**.

The parameter can to be set to either **directIO** or **setall** to avoid buffered I/O against file systems that support direct I/O but do not provide a direct I/O mount option.

The default value for this parameter is **asynch**.

User I/O Dwarfs All Other Activity



With SSD There are Fewer Waits on User I/O

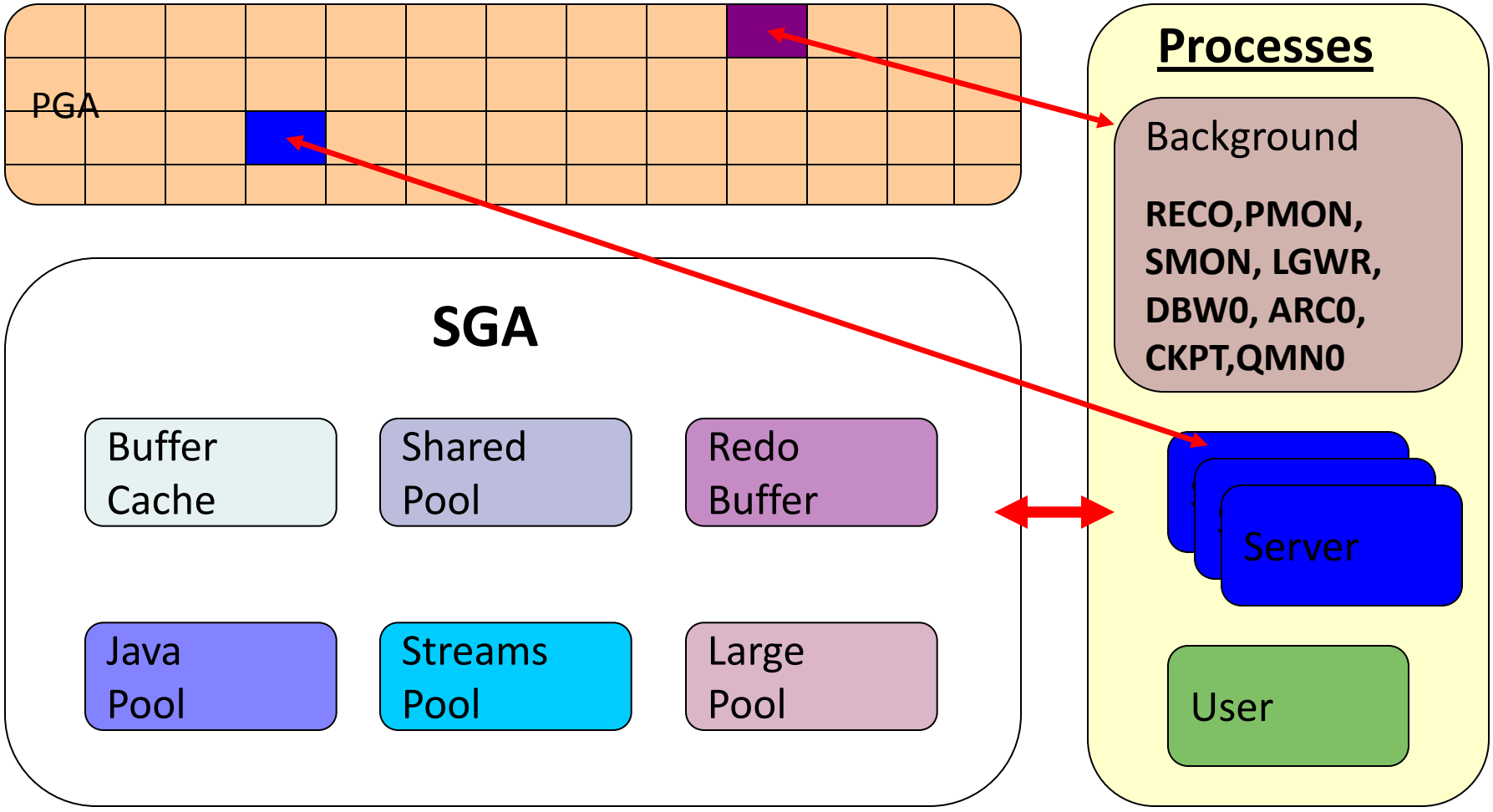
2.5 minutes with 8 CPUs vs 18 min for 1 CPU

How to Use Solid State Disk

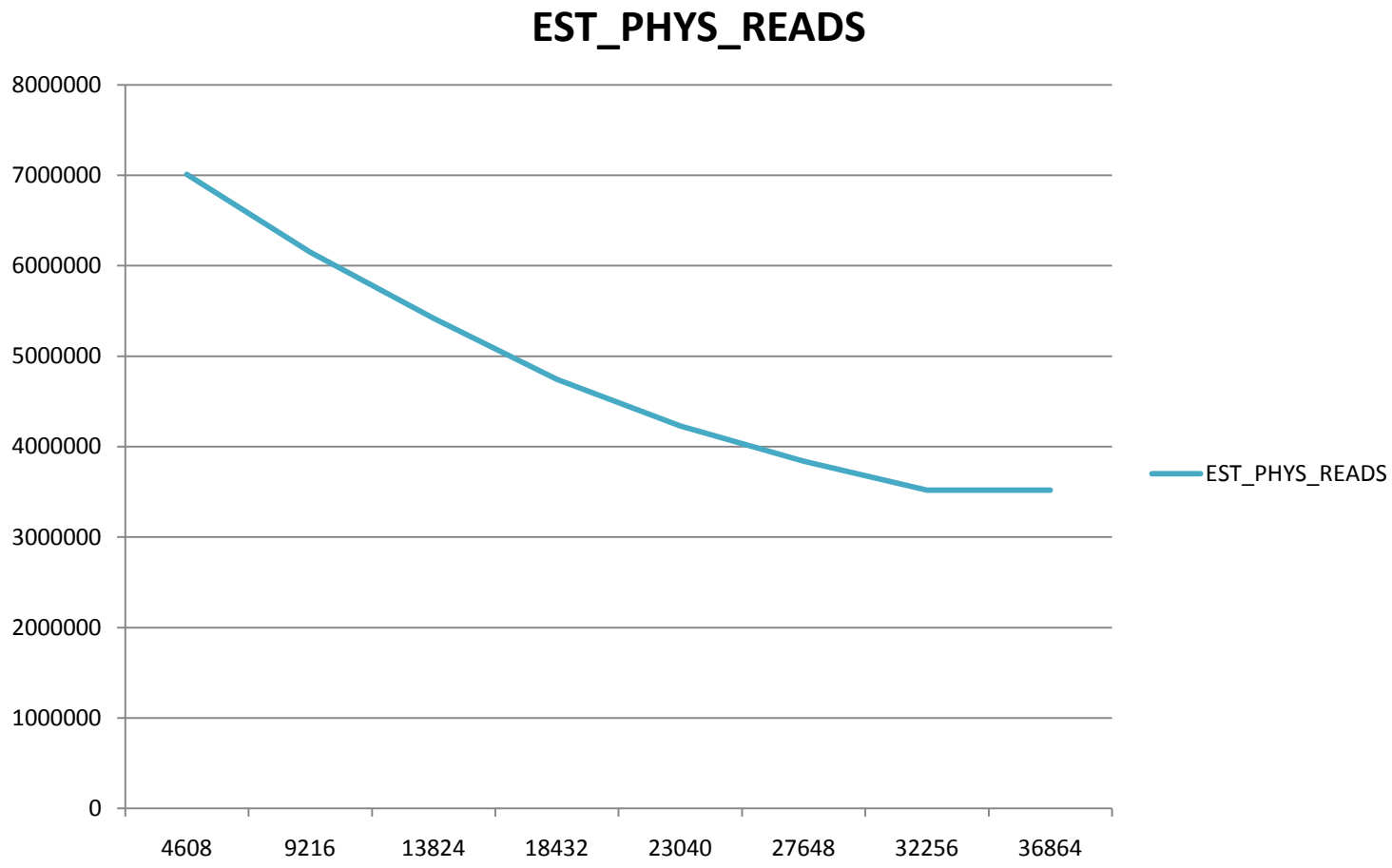
- SSDs are very fast to read from but can be slow for writes. Lots of writes can “wear out” blocks.
- Minimize writes to the SSD.
- Even though your SAN may be very expensive to add SSD drives, local SATA drive on Linux systems are easy to install.
- Put all your log files on small separate disks
- Put all your .dbf’s on Solid State Disks
- Put all your ORACLE_HOMEs on SSD (optional)
- Put your archive logs on normal disks, but make sure they don’t fall behind.

Oracle Memory Structure

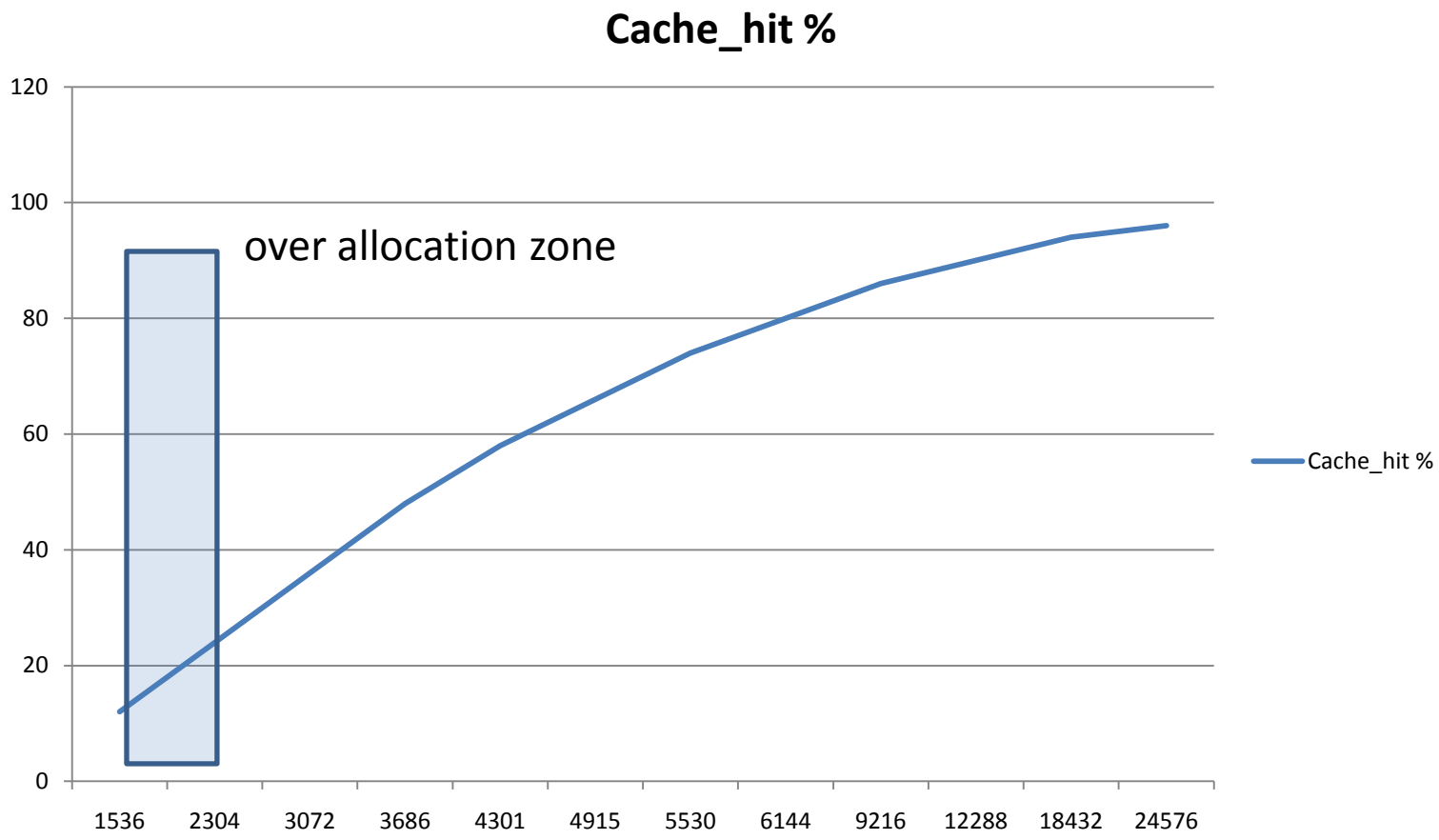
Private to each server and background process; there is one PGA for each process



Querying V$SGA_TARGET_ADVICE



V$PGA_TARGET_ADVICE



Partitioning

- Partitioning can help you tune SQL statements to avoid unnecessary index and table scans, using partition pruning.
- Improve the performance of massive join operations when large amounts of data are joined together by using partition-wise joins.
- Partitioning data greatly improves manageability of very large databases and dramatically reduces the time required for administrative tasks such as backup and restore.

Partitioning

Oracle offers four partitioning methods:

- Range Partitioning
- Hash Partitioning
- List Partitioning
- Composite Partitioning

Each partitioning method has different advantages and design considerations. Thus, each method is more appropriate for a particular situation.

Range Partitioning

Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition.

It is the most common type of partitioning and is often used with dates.

For example, you might want to partition sales data into monthly partitions or weekly partitions.

Range Partitioning

The following statement creates a table that is range partitioned on the sales_date field:

```
 CREATE TABLE DEV.SALES_DATA_NEW(
          SALES_DATE              DATE              NOT NULL,
          ITEM_ID                 NUMBER(10)        NOT NULL,
          LOCATION_ID             NUMBER(10)        NOT NULL)
TABLESPACE TS_SALES_DATA
PARTITION BY RANGE (SALES_DATE)
(PARTITION  p1 values less than ('01-Jan-1995'),
PARTITION  p2 values less than ('08-Jan-1995'),
...
PARTITION  p1879 values less than ('29-Dec-2030'),
PARTITION  pMAX values less than (MAXVALUE))
```

When to Use Range Partitioning

- Range partitioning is a convenient method for partitioning historical data.
- Range partitioning organizes data by time intervals on a column of type DATE. Thus, most SQL statements accessing range partitions focus on timeframes.
- An example of this is a SQL statement similar to "select data from a particular period in time." In such a scenario, if each partition represents data for one month, the query "find data of month 98-DEC" needs to access only the December partition of year 98.
- This reduces the amount of data scanned to a fraction of the total data available, an optimization method called partition pruning.

Avoiding I/O Bottlenecks

To avoid I/O bottlenecks, when Oracle is not scanning all partitions because some have been eliminated by pruning, **spread each partition over several devices**.

If you're planning to use parallel execution, spread each partition over *n* devices, where *n* is the degree of parallelism.

MAXVALUE

You can specify the keyword MAXVALUE for any value in the partition bound *value_list*. This keyword represents a virtual infinite value that sorts higher than any other value for the data type, including the NULL value.

For example, you might partition the OFFICE table on STATE (a CHAR(10) column) into three partitions with the following partition bounds:

- VALUES LESS THAN ('I'): States whose names start with A through H
- VALUES LESS THAN ('S'): States whose names start with I through R
- VALUES LESS THAN (MAXVALUE): States whose names start with S through Z, plus special codes for non-U.S. regions

Guidelines for Partitioning Indexes

When deciding how to partition indexes on a table, consider the mix of applications that need to access the table. There is a trade-off between performance on the one hand and availability and manageability on the other. Here are some of the guidelines you should consider:

For OLTP applications:

- **Global indexes and local prefixed indexes provide better performance** than local nonprefixed indexes because they minimize the number of index partition probes.
- **Local indexes support more availability when there are partition or subpartition maintenance operations** on the table. Local nonprefixed indexes are very useful for historical databases

Guidelines for Partitioning Indexes

- **For DSS applications, local non-prefixed indexes can improve performance** because many index partitions can be scanned in parallel by range queries on the index key.
- For historical tables, indexes should be local if possible. This limits the impact of regularly scheduled drop partition operations.
- **Unique indexes on columns other than the partitioning columns must be global** because unique local nonprefixed indexes whose key does not contain the partitioning key are not supported.

Why a Full Table Scan Can Be Faster for Accessing Large Amounts of Data

Full table scans are cheaper than index range scans when accessing a large fraction of the blocks in a table.

This is because full table scans can use larger I/O calls, and making fewer large I/O calls is cheaper than making many smaller calls.

When the Optimizer Uses Full Table Scans

Lack of Index

- If the query is unable to use any existing indexes, then it uses a full table scan. For example, if there is a function used on the indexed column in the query, the optimizer is unable to use the index and instead uses a full table scan.
- If you need to use the index for case-independent searches, then either do not permit mixed-case data in the search columns or create a function-based index, such as UPPER(last_name), on the search column.

Large Amount of Data

If the optimizer thinks that the query will access most of the blocks in the table, then it uses a full table scan, even though indexes might be available.

When the Optimizer Uses Full Table Scans

Small Table

If a table contains less than DB_FILE_MULTIBLOCK_READ_COUNT blocks under the high water mark, which can be read in a single I/O call, then a full table scan might be cheaper than an index range scan, regardless of the fraction of tables being accessed or indexes present.

High Degree of Parallelism

A high degree of parallelism for a table skews the optimizer toward full table scans over range scans. Examine the DEGREE column in DBA_TABLES for the table to determine the degree of parallelism.

Parallel Query / Parallel Execution

Full Table Scan Hints

Use the hint FULL(*table alias*) if you want to force the use of a full table scan.

Parallel Query Execution

When a full table scan is required, response time can be improved by using multiple parallel execution servers for scanning the table.

When Oracle runs SQL statements in parallel, multiple processes work together simultaneously to run a single SQL statement.

By dividing the work necessary to run a statement among multiple processes, Oracle can run the statement more quickly than if only a single process ran it. This is called **parallel execution**.

Parallel Query / Parallel Execution

Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time. An example of this is when 12 processes handle 12 different months in a year instead of one process handling all 12 months by itself.

Parallel execution helps systems scale in performance by making optimal use of hardware resources. If your system's CPUs and disk controllers are already heavily loaded, you need to alleviate the system's load or increase these hardware resources before using parallel execution to improve performance.

Measuring the Performance Impact of Increasing the Degree of Parallelism

Running the worksheets with multiple CPUs in parallel improved performance by 8 times. We decided to evaluate the effect of varying degrees of parallelism. In our example, we have 64 processor threads available.

We found that the degree of parallelism for tables could be high, but that indexes generally should have a degree of 1, to avoid issues with the CBO. However, in specific cases, by setting the appropriate SALES_DATA indexes to a higher degree we were able to increase the CPU utilization of the worksheet. For example, if the SALES_DATA table is set to a parallel degree of 8, 8 different CPUs will execute the parallel query.

Measuring the Performance Impact of Increasing the Degree of Parallelism

The following diagram shows the same Capacity Planning worksheet run twice: once for 2:00 minutes with 8 CPUs and the second run for 15:45 minutes with 1 CPU. The green color indicates the process was mostly consuming CPU. There is a small component of User I.O shown in dark blue, in the upper left corner of the process running at 9:35 with 8 CPUs.

Measuring the Performance Impact of Increasing the Degree of Parallelism

The CAPACITY_PLANNING worksheet uses the parallel index SALES_DATA_144_IDX. The following example shows how increasing the parallelism of the worksheet decreases the runtime of the worksheet, almost linearly. Since, the data access path includes this index, the index and the table must be set to the same DOP to use 8 CPUs for this query. If the Index DOP is set to less than the table DOP, then the least common value will be used for the DOP. In this way, we can decrease the effective DOP just by altering certain indexes.

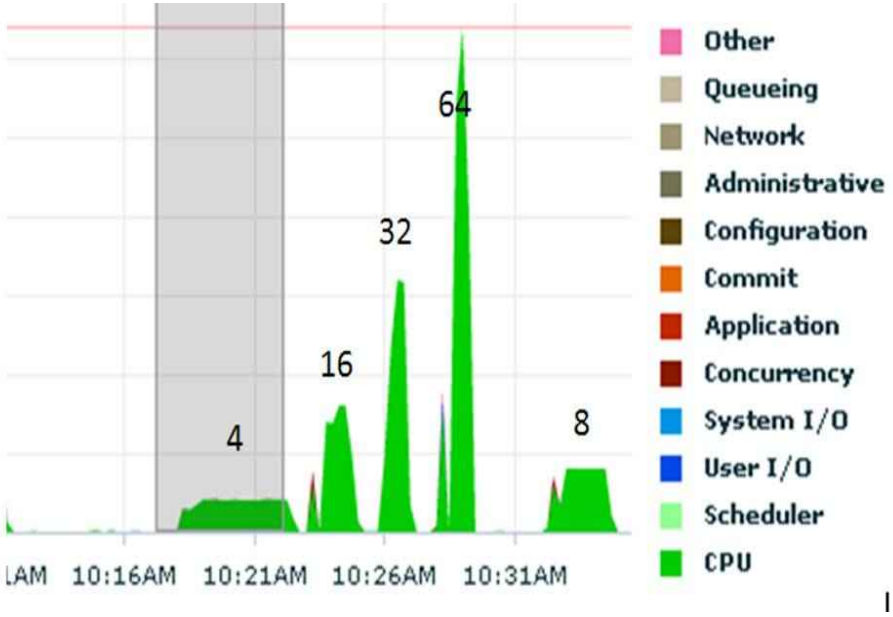
SALES_DATA_1IDX		N	NO	1	SYNCRO_SIG	Asc	1
SALES_DATA_1IDX		N	NO	1	ITEM_ID	Asc	2
SALES_DATA_1IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_144_IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_144_IDX		N	NO	1	LOCATION_ID	Asc	2
SALES_DATA_144_IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_144_IDX		N	NO	1	IS_PROMOTION	Asc	4
SALES_DATA_1436_IDX		N	YES	1	ITEM_ID	Asc	1
SALES_DATA_1436_IDX		N	YES	1	LOCATION_ID	Asc	2
SALES_DATA_1436_IDX		N	YES	1	SALES_DATE	Asc	3
SALES_DATA_1436_IDX		N	YES	1	IS_SL_MARKETING	Asc	4
IS_T_EP_CTO_1001_IDX		N	NO	1	IS_T_EP_CTO	Asc	1
IS_SUPPLY_PLAN_632_IDX		N	NO	1	IS_SUPPLY_PLAN	Asc	1
IS_SL_MARKETING_1412_IDX		N	YES	1	IS_SL_MARKETING	Asc	1
IS_SCENARIO_RESOURCE_663_IDX		N	NO	1	IS_SCENARIO_RESOURCE	Asc	1

Measuring the Performance Impact of Increasing the Degree of Parallelism

ALTER INDEX SALES_DATA_144_IDX (parallel 8)

ALTER INDEX SALES_DATA_144_IDX (parallel 16)

ALTER INDEX SALES_DATA_144_IDX (parallel 64)



Measuring the Performance Impact of Increasing the Degree of Parallelism

The associated worksheet runtimes to this graph are as follows:

		Efficiency	If 100% efficient
4:	00:03:56		3 :56
8:	00:01:55	103%	1:58
16:	00:01:20	61%	:57.5 sec
32:	00:00:53	67%	:40 sec
64:	00:00:44	33%	:26.5 sec

Note that as the number of processors doubles, the time should be reduced by half. For example, with a parallel degree of 8, 8 CPUs are used to process the worksheet and the runtime is 1:55. By doubling the CPUs running a worksheet, we might expect the runtime to be about 1:00 minute. However, the runtime with 16 processors is 1:20, meaning this is a non-linear process with some loss due to 'friction' and approximately 63% efficient.

Measuring the Performance Impact of Increasing the Degree of Parallelism

You can also change the DOP when you rebuild the index:

```
ALTER INDEX index_name REBUILD PARALLEL 6
```

The other issue is related to PQ / PE and independent data access. If PQ / PE is turned on, but the query is reading all the data from one physical disk, then it takes longer to complete the query.

Oracle provides several initialization parameters to control how parallel execution works. The most basic of these parameters are PARALLEL_MAX_SERVERS, PARALLEL_MIN_SERVERS, and PARALLEL_MIN_PERCENT. These parameters control the number of parallel-execution server processes in the pool and the number of such processes acquired by an operation when sufficient processes are not available in the pool.

The most interesting of all the parallel execution-related initialization parameters is **PARALLEL_ADAPTIVE_MULTI_USER**, which determines the behavior of parallel execution when parallel operations are performed in a multiuser scenario.

Enabling Parallel DML

A DML statement can be parallelized only if you have explicitly enabled parallel DML in the session with the ENABLE PARALLEL DML clause of the ALTER SESSION statement. This mode is required because parallel DML and serial DML have different locking, transaction, and disk space requirements.

The default mode of a session is DISABLE PARALLEL DML. When parallel DML is disabled, no DML will be executed in parallel even if the PARALLEL hint is used.

Enabling Parallel DML

When parallel DML is enabled in a session, all DML statements in this session will be considered for parallel execution.

However, even if parallel DML is enabled, the DML operation may still execute serially if there are no parallel hints or no tables with a parallel attribute or if restrictions on parallel operations are violated.

The session's PARALLEL DML mode does not influence the parallelism of SELECT statements, DDL statements, and the query portions of DML statements.

Thus, if this mode is not set, the DML operation is not parallelized, but scans or join operations within the DML statement may still be parallelized.

ALTER SESSION ENABLE PARALLEL DML 4

In systems with enough I/O bandwidth and CPUs to use parallel execution a simple trigger can enable parallel dml for each connection and insert a tag into the CLIENT_INFO column in V$SESSION.

```
CREATE OR REPLACE TRIGGER DEMANTRA.trace_trig after logon on demantra.schema
DECLARE sqlstr VARCHAR2(200) := 'ALTER SESSION ENABLE PARALLEL DML 4';
BEGIN
  IF (USER = 'DEMANTRA') THEN execute immediate sqlstr;
    dbms_application_info.set_client_info('parallel dml 4 enabled');
  END IF;
END trace_trig;
```

There CLIENT_INFO column on V$SESSION is VARCHAR2(64), so you can specify any string but only the first 64 chars will populate in V$SESSION.

After this trigger fires, you can query that column for the value "parallel dml 4 enabled".

Materialized Views

The concept is simple. Anywhere there exists a complex, long running in-line view, it can be potentially replaced with a materialized view, resulting in much faster performance.

Tom Kyte writes on his blog:
“So, if you:
select * from (some-really-hard-to-execute-query-that-takes-5-minutes)
vs
create materialized view mv as
(some-really-hard-to-execute-query-that-takes-5-minutes)
and then
select * from (select * from mv)

it is going to be night and day different of course. One answered the query at runtime, the other answered the query last night and we just browse the answer.”

Materialized Views

A materialized view is like a query with a result that is materialized and stored in a table. When a user query is found compatible with the query associated with a materialized view, the user query can be rewritten in terms of the materialized view. This technique improves the execution of the user query, because most of the query result has been pre-computed. The use of materialized views to rewrite a query is cost-based. That is, the query is not rewritten if the plan generated without the materialized views has a lower cost than the plan generated with the materialized views.

Define materialized views on OLTP systems, when you can REFRESH ON DEMAND or periodically when the load is light.

"Materialized views with “Refresh on Commit” have a significant overhead on commit."

Materialized Views

“The Demantra worksheet queries are enormous in sheer text size, and any materialized view to support such massive queries would have required a "defining query" exceeding almost 100 Kb in length. Unfortunately, Oracle enforces a maximum query text size of 64 Kb for materialized views. I was thus unable to create a large enough materialized view to feasibly support the Demantra "worksheet" queries simply due to this limitation. This is a real shame, because the scenario of these "worksheet" queries is a perfect use-case for single-table aggregated materialized view.

Maybe, create a view to represent all of the columns in the SALES_DATA table more concisely, using shorter columns names create the materialized view using that view”

Maintenance

Gather Statistics
Fix Row Migration and Row Chaining
ALTER TABLESPACE MOVE TABLE – to repair chained rows
Rebuild tables with more than 255 columns and place null columns at the end of table
Clustering Factor – Rows out of Order
Reorg of Tables based on Primary Key
Find Foreign Keys with Missing Indexes

Gather Statistics

From the Demantra seeded package REBUILD_SCHEMA:

-- *Same as ANALYZE TABLE T1 COMPUTE STATISTICS;*

```
        IF get_is_table_exists(rec1.table_name) = 1 THEN

dbms_stats.GATHER_TABLE_STATS (OWNNAME => NULL,
TABNAME => rec1.table_name, CASCADE => TRUE,
method_opt => 'FOR ALL COLUMNS SIZE 1');

        END IF;
```

Row Migration

Row migration occurs **when an update to that row would cause it to not fit on the block anymore** (with all of the other data that exists there currently).

A row migration means that the entire row will move and we just leave behind the "forwarding address".

The original block just has the rowid of the new block and the entire row is moved.

Row Migration

Full Table Scans are not affected by migrated rows

The forwarding addresses are ignored. We know that as we continue the full scan, we'll eventually get to that row so we can ignore the forwarding address and just process the row when we get there. Hence, in a full scan migrated rows don't cause us to really do any extra work -- they are meaningless.

Index Read will cause additional IO's on migrated rows

When we Index Read into a table, then a migrated row will cause additional IO's. That is because the index will tell us “goto file X, block Y, slot Z to find this row”. But when we get there we find a message that says “really go to file A, block B, slot C to find this row”. We have to do another IO (logical or physical) to find the row.

Row Chaining

A row is too large to fit into a single database block.

For example, if you use a 4KB blocksize for your database, and you need to insert a row of 8KB into it, Oracle will use 3 blocks and store the row in pieces. Some conditions that will cause row chaining are:

- Tables whose rowsize **exceeds** the blocksize.
- Tables with LONG and LONG RAW columns are prone to having chained rows.
- **Tables with more then 255 columns** will have chained rows as Oracle break wide tables up into pieces. So, instead of just having a forwarding address on one block and the data on another we have data on two or more blocks.

Row Chaining

Chained rows affect us differently. Here, it depends on the data we need. If we had a row with two columns that's spread over two blocks, the query:

```
SELECT c1 FROM t1
```

where c1 is in Block 1, would not cause any "table fetch continued row". It would not actually have to get c256, it would not follow the chained row all of the way out.

On the other hand, if we ask for:

```
SELECT c2 FROM t1
```

and c256 is in Block 2 due to row chaining, then you would in fact see a "table fetch continued row"

Fix Row Chaining in Tables

The dynamic Demantra procedure REBUILD_SCHEMA contains the following PLSQL:

```
IF get_is_table_exists(rec2.table_name) = 1
      THEN
                  dynamic_ddl('ALTER TABLE '||rec2.table_name||' MOVE
                  TABLESPACE '||rec2.tablespace_name||' NOLOGGING ');

  dynamic_ddl('ALTER TABLE '||rec2.table_name||' LOGGING ');
  rebuild_indexes(rec2.table_name);
  -- Refresh the stats
  -- Same as ANALYZE TABLE T1 COMPUTE STATISTICS;

dbms_stats.GATHER_TABLE_STATS (OWNNAME => NULL, TABNAME =>
rec2.table_name, CASCADE => TRUE, method_opt => 'FOR ALL COLUMNS SIZE 1');

END IF;
```

ALTER INDEX … REBUILD with COMPUTE STATISTICS

"One optimization for the rebuild of indexes following the reload of data into the SALES_DATA and MDP_MATRIX is to use the COMPUTE STATISTICS clause of the CREATE INDEX or ALTER INDEX … REBUILD commands.

During the creation of an index it is possible to obtain a "free compute" of statistics for the cost-based optimizer, since all of the data in the index is viewed during a build.

Using COMPUTE STATISTICS removes the need to issue subsequent commands like DBMS_STATS.GATHER_INDEX_STATS, and also allows the reduction of the DBMS_STATS.GATHER_TABLE_STATS command to specify the CASCADE=>FALSE command, so ensure that statistics are not gathered recursively on indexes associated with the table."

Re-organization of Tables Based on Primary Key

A less obvious issue which can affect the IO rates is how well data is clustered physically. For example, assume that you frequently fetch rows from a table where a column is between two values using an index scan.

If there are 100 rows in each index block then the two extremes are:

- Each of the table rows is in a different physical block (100 blocks need to be read for each index block)
- The table rows are all located in the few adjacent blocks (a handful of blocks need to be read for each index block)

Re-organization of Tables Based on Primary Key

After making a copy of the table to be rebuilt using "create table as select" (CTAS), run a script that uses a parallel insert to reload the data so the primary key has the highest clustering factor.

In this case, the SALES_DATA_PK is ordered on item_id, location_id and sales_date.

Re-organization of Tables Based on Primary Key

```
--  Copy the data from the renamed table
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL ( INS_TBL 8) APPEND */
INTO DEV.SALES_DATA INS_TBL
(ITEM_ID              ,
LOCATION_ID       ,
SALES_DATE
.
.
SELECT  /*+  PARALLEL ( SEL_TBL 8) INDEX (SEL_TBL SALES_DATA_PK) */
ITEM_ID ,
 LOCATION_ID       ,
 SALES_DATE
FROM SALES_DATA
ORDER BY ITEM_ID, LOCATION_ID, SALES_DATE
```

Rebuild Indexes

You might want to re-create an index to compact it and minimize fragmented space, or to change the index's storage characteristics. When creating a new index that is a subset of an existing index or when rebuilding an existing index with new storage characteristics, Oracle might use the existing index instead of the base table to improve the performance of the index build.

Use the ALTER INDEX ... REBUILD statement to reorganize or compact an existing index or to change its storage characteristics. The REBUILD statement uses the existing index as the basis for the new one.

Avoid calling DBMS_STATS after the index creation or rebuild, by including the COMPUTE STATISTICS statement on the CREATE or REBUILD.

Oracle Foreign Keys Without Indexes

Identify missing indexes. The script output indicates we have 297 foreign keys without indexes. This not only slows down the data access, but can be crippling on updates, because the table can't perform a row level lock and must perform a table level lock.

Here is a partial list of the report generated by TOAD:

FK's w/o Matching Indexes (or an unusable one)

Foreign Key: DEV.APPLICATIONS_MODULE_ID_FK
On Table: APPLICATIONS
Columns: MODULE_ID
Foreign Key: DEV.ATTRIBUTES_LEVEL_ID_FK
On Table: NOTE_POPULATION
Columns: LEVEL_ID
Foreign Key: DEV.AUDIT_DATA_AUDIT_ID_FK
On Table: AUDIT_DATA
Columns: AUDIT_ID

Find Missing Indexes in EBS

```
select    owner, table_name, column_name
from    (select c.owner, c.table_name, cc.column_name, cc.position
    from dba_constraints c, dba_cons_columns cc
    where c.owner = 'APPS‘      and cc.owner = c.owner
    and cc.table_name = c.table_name
    and cc.constraint_name = c.constraint_name
    and c.constraint_type = 'R‘      and c.table_name not in
       (SELECT table_name FROM   user_tables WHERE
       (substr(table_name,1,3)  = 'TMP‘ OR  substr(table_name,1,5)  = 'TEMP_'))
    minus
    select table_owner owner, table_name, column_name, column_position
position
    from dba_ind_columns      where table_owner = 'APPS'
    where table_owner = ( select application_short_name from
applsys.fnd_application)
    order by 1, 2, 4)
```

Demantra Tables with FKs That are Missing Indexes

TABLE_NAME	COLUMN_NAME
COMPUTED_FIELDS_BASE	LOOKUP_SECURITY_TYPE_ID
CUSTOMER_TYPE_RELATION	CUSTOMER_RHS
DM_WIZ_IMPORT_FILE_DEF	SRC_TABLE_CODE
MARKETING_EVENT_DATA	ITEM_ID
MARKETING_EVENT_MATRIX	ITEM_ID
PROMOTION_MATRIX	PROMOTION_TRANSFER_STAT_ID
SCENARIO_RESOURCE_DATA	ITEM_ID
SUPPLY_PLAN_DATA	ITEM_ID
TREE_VIEW_EDIT_PROPAGATION	TARGET_SERIES_ID
T_EP_CTO_DATA	ITEM_ID

Summary

I/O is the Bottleneck

Measure changes and only change one variable at a time

Improve Hardware

- Add Memory
- Use SSD
- Add more CPUs

Understand the Workload and the number of users.

Parallel Execution can decrease query execution time.

Make sure there are enough IOPS and CPUs for PX/PQ

Use Partitioning for large tables

Maintain the Application

- Fix Row Chaining and Row Migration
- Re-Org tables
- Find Missing Indexes

Use tested Initialization Parameters

Questions

www.trutek.com

mswing@trutek.com

blog.trutek.com

Meet me at the 2e2 booth for Free books, or outside the meeting hall after my presentation.

Use “dd” to Test I/O

Create a file-system with a larger block size, copy a large file from the existing file-system into the new (larger block-size) file-system, then perform some read and write tests using the UNIX "dd" utility...

read test: dd if=<*file-name*> of=/dev/null bs=32768
write test: dd if=/dev/zero of=<*file-name*> bs=32768

```
read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 32768)          = 32768
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 32768)         = 32768
read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 32768)          = 32768
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 32768)         = 32768
read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 32768)          = 32768
read(3( Detaching from process 7918 ("dd if=/dev/zero of=/test1/test.file bs=32768
count=100000") )
```

The Pool of Parallel Slave Processes

Oracle maintains a pool of parallel slave processes for each instance.

The parallel coordinator for a SQL statement assigns parallel tasks to slave processes from this pool.

These parallel slave processes remain assigned to a task until its execution is complete.

After that, these processes return to the pool and can be assigned tasks from some other parallel operation.

A parallel slave process serves only one SQL statement at a time.

PARALLEL_MIN_SERVERS

The following parameters control the number of parallel slave processes in the pool:

- **PARALLEL_MIN_SERVERS**

Specifies the minimum number of parallel slave processes for an instance.

When an instance starts up, it creates the specified number of parallel slave processes.

The default value for this parameter is 0, meaning that no slave processes would be created at startup.

PARALLEL_MAX_SERVERS

- Specifies the maximum number of parallel slave processes that an instance is allowed to have at one time. The default value for PARALLEL_MAX_SERVERS is platform-specific.
- It takes time and resources to create parallel slave processes.
- Since parallel slave processes can serve only one statement at a time, you should set PARALLEL_MIN_SERVERS to a relatively high value if you need to run lots of parallel statements concurrently.
- That way, performance won't suffer from the need to constantly create slave processes.

PARALLEL_MAX_SERVERS

- You also need to consider how to set PARALLEL_MAX_SERVERS. Each parallel slave process consumes memory.
- Setting PARALLEL_MAX_SERVERS too high may lead to memory shortages during peak usage times.
- On the other hand, if PARALLEL_MAX_SERVERS is set too low, some operations may not get a sufficient number of parallel slave processes.

Degree of Parallelism

The parallel execution coordinator may enlist two or more of the instance's parallel execution servers to process a SQL statement.

The number of parallel execution servers associated with a single operation is known as the **degree of parallelism**.

ALTER TABLE SALES_DATA PARALLEL (degree 8);

Parallel Execution Views

- V_$PQ_SESSTAT - lists system statistics for parallel queries. After you have run a query or DML operation, you can use the information derived from V$PQ_SYSSTAT to view the number of slave processes used, and other information for the system.
- V_$PQ_SLAVE - lists statistics for each of the active parallel execution servers on an instance

Parallel Execution Views

V_$PQ_SYSSTAT - lists system statistics for parallel queries. After you have run a query or DML operation, you can use the information derived from V$PQ_SYSSTAT to view the number of slave processes used, and other information for the system.

V_$PQ_TQSTAT - contains statistics on parallel execution operations. The statistics are compiled after the query completes and only remain for the duration of the session. It displays the number of rows processed through each parallel execution server at each stage of the execution tree. This view can help determine **skew** problems in a query's execution. (Note that for PDML, information from V$PQ_TQSTAT is available only after a commit or rollback operation.)

```
CREATE TABLE DEMANTRA.TRIG_FIRE
(  TIMESTAMP  DATE )
TABLESPACE TS_DP
PCTUSED    0
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL        80K
            MINEXTENTS     1
            MAXEXTENTS     UNLIMITED
            PCTINCREASE    0
            BUFFER_POOL    DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;
```

Coarse Grain Striping

In a system with a high degree of concurrent small I/O requests, such as in a traditional OLTP environment, it is beneficial to keep the stripe depth large.

Using stripe depths larger than the I/O size is called coarse grain striping.

In high-concurrency systems, the stripe depth can be *n* * DB_BLOCK_SIZE where *n* is greater than 1.

Coarse Grain Striping

Coarse grain striping allows a disk in the array to service several I/O requests. In this way, a large number of concurrent I/O requests can be serviced by a set of striped disks with minimal I/O setup costs.

Coarse grain striping strives to maximize overall I/O throughput. Multiblock reads, as in full table scans, **will benefit when stripe depths are large and can be serviced from one drive.**

Parallel query in a DSS environment is also a candidate for coarse grain striping. This is because there are many individual processes, each issuing separate I/Os.

Fine Grained Striping

- In a system with a few large I/O requests, such as in a traditional DSS environment or a low-concurrency OLTP system, **then it is beneficial to keep the stripe depth small**.
- This is called fine grain striping. In such systems, the stripe depth is *n* * DB_BLOCK_SIZE, or smaller than, 32 * 32K = 1M where *n* is smaller than the multiblock read parameters, DB_FILE_MULTIBLOCK_READ_COUNT.
- **Fine grain striping** allows a single I/O request to be serviced by multiple disks. Fine grain striping strives to maximize performance for individual I/O requests or response time.

Stripe Everything Across Every Disk

The simplest approach to I/O configuration is to build one giant volume, striped across all available disks. To account for recoverability, the volume is mirrored (RAID 1).

The striping unit for each disk should be larger than the maximum I/O size for the frequent I/O operations. This provides adequate performance for most cases.

Fine Grained Striping

- In a system with a few large I/O requests, such as in a traditional DSS environment or a low-concurrency OLTP system, **then it is beneficial to keep the stripe depth small**.
- This is called fine grain striping. In such systems, the stripe depth is *n* * DB_BLOCK_SIZE, or smaller than, 32 * 32K = 1M where *n* is smaller than the multiblock read parameters, DB_FILE_MULTIBLOCK_READ_COUNT.
- **Fine grain striping** allows a single I/O request to be serviced by multiple disks. Fine grain striping strives to maximize performance for individual I/O requests or response time.

Optimizer_mode = ALL_ROWS

The optimizer uses a cost-based approach for all SQL statements in the session regardless of the presence of statistics and optimizes with a goal of best throughput (minimum resource use to complete the entire statement). This is the default value.

Stripe Size

"I generally find finer striping (i.e. 32K) acceptable for very low concurrency operations, such as LGWR, but really bad for high-concurrency operations (i.e. datafile I/O) as the number of user sessions ramps up. For high-concurrency data structures, I've found larger stripes of 1Mb or so to be optimal.

Since DB_BLOCK_SIZE is 32K and one of the possible stripe sizes is 32K, what happens during multi-block reads during full table scans or fast full index scans is that a single I/O request from a single session lights up all the drives at once. Two sessions doing that concurrently will likely cause all or part of one session to queue on the other. As the concurrency increases, it stands to reason that all of the drives will busy all the time. So, smaller stripes will probably benchmark well with low volumes of concurrent I/O requests, but possibly start to choke as volume increases."

Gather Statistics

One major weakness of using BLOCK_SAMPLE=>TRUE, namely that **tables that experience mass deletions of data could have sparsely-populated blocks with few or no rows**.

Thus, BLOCK_SAMPLE=>TRUE may encounter those sparsely-populated blocks and thus result in inaccurate or inappropriate statistics.

So, **be sure to use BLOCK_SAMPLE=>TRUE only on tables that do not experience large-scale or mass deletion operations."**

Using Operating System Caches

Operating systems and device controllers provide data caches, and by default all database I/O goes through the file system cache. On some UNIX systems, direct I/O is available to the filestore.

DirectIO allows the database files to be accessed within the UNIX file system, bypassing the file system cache. It saves CPU resources and allows the file system cache to be dedicated to non-database activity, such as program texts and spool files.

Although the operating system cache is often redundant because the Oracle buffer cache buffers blocks, there are a number of cases where Oracle does not use the Oracle buffer cache. **Direct I/O bypasses the operating system cache and does not use the operating system cache**. This may yield worse performance than using operating system buffering.

Asynchronous I/O

With synchronous I/O, when an I/O request is submitted to the operating system, the writing process blocks until the write is confirmed as complete. It can then continue processing. With asynchronous I/O, processing continues while the I/O request is submitted and processed. Use asynchronous I/O when possible to avoid bottlenecks.

Some platforms support asynchronous I/O by default, others need special configuration, and some only support asynchronous I/O for certain underlying file system types.

FILESYSTEMIO_OPTIONS Init Parameter

You can use the FILESYSTEMIO_OPTIONS initialization parameter to enable or disable asynchronous I/O or direct I/O on file system files. This parameter is platform-specific and has a default value that is best for a particular platform. It can be dynamically changed to update the default setting.

FILESYTEMIO_OPTIONS can be set to one of the following values:
ASYNCH: enable asynchronous I/O on file system files, which has no timing requirement for transmission

DIRECTIO: enable direct I/O on file system files, which bypasses the buffer cache
SETALL: enable both asynchronous and direct I/O on file system files
NONE: disable both asynchronous and direct I/O on file system files

Avoid Buffered I/O

Buffered datafile I/O wastes memory because Oracle data blocks are cached both in the SGA and in the file system buffer cache with substantial overlap between the two sets of blocks that are cached.

The file system cache should be used to buffer non-Oracle I/O only.

Using it to attempt to enhance the caching of Oracle data just wastes memory, and lots of it. Oracle can cache its own data much more effectively than the operating system can, most of the time.

Avoid Buffered I/O

Oracle has a sophisticated touch count based cache replacement algorithm that is sensitive to both the frequency and of data usage.

Oracle also avoids caching data that is unlikely to be, or will never be reused, and DBAs can fine tune Oracle's caching decisions by allocating segments to the KEEP and RECYCLE buffer pools.

There is also about a 50% savings in CPU time and elapsed time when getting data from the Oracle buffer cache rather than the filesystem buffer cache.

filesystemio_options

The hidden *_filesystemio_options* parameter was introduced in Oracle version 8.1.7 to control the use of program selectable direct I/O and asynchronous I/O against file system based database files. It is no longer a hidden parameter from release 9.2 onwards. The following table shows the values that can be used when setting this parameter.

	Buffered I/O	**Direct I/O**
Synchronous I/O	none	directIO
Asynchronous I/O	asynch	setall

filesystemio_options

Databases that use a combination of raw log files and raw temp files with filesystem based datafiles can use asynchronous I/O against the raw files, but to avoid inefficient threaded asynchronous I/O against the datafiles try setting *disk_asynch_io* to the default of TRUE and setting *filesystemio_options* to either **none** or **directIO**.

The parameter can to be set to either **directIO** or **setall** to avoid buffered I/O against file systems that support direct I/O but do not provide a direct I/O mount option. The default value for this parameter is **asynch**.

filesystemio_options

Databases that use a combination of raw log files and raw temp files with filesystem based datafiles can use asynchronous I/O against the raw files,

> but to avoid inefficient threaded asynchronous I/O against the datafiles try setting *disk_asynch_io* to the default of TRUE and setting *filesystemio_options* to either **none** or **directIO**.

The parameter can to be set to either **directIO** or **setall** to avoid buffered I/O against file systems that support direct I/O but do not provide a direct I/O mount option.

The default value for this parameter is **asynch**.

Move Archive Logs to Different Disks

If archive logs are striped on the same set of disks as other files, then any I/O requests on those disks could suffer when redo logs are being archived. Moving archive logs to separate disks provides the following benefits:

- The archive can be performed at very high rate (using sequential I/O).
- Nothing else is affected by the degraded response time on the archive destination disks.
- The number of disks for archive logs is determined by the rate of archive log generation and the amount of archive storage required.

Move Redo Logs to Separate Disks

- In high-update OLTP systems, the redo logs are write-intensive. Moving the redo log files to disks that are separate from other disks and from archived redo log files, is able to write sequentially, until it is interrupted. Minimize the interruptions.
- Increasing the number of disks for redo logs does not improve performance, because of the sequential nature of the writes.
- Using RAID 5 for redo and archive logs can create bottlenecks in write speeds.

filesystemio_options

Databases that use a combination of raw log files and raw temp files with filesystem based datafiles can use asynchronous I/O against the raw files, but to avoid inefficient threaded asynchronous I/O against the datafiles try setting *disk_asynch_io* to the default of TRUE and setting *filesystemio_options* to either **none** or **directIO**.

The parameter can to be set to either **directIO** or **setall** to avoid buffered I/O against file systems that support direct I/O but do not provide a direct I/O mount option. The default value for this parameter is **asynch**.

Querying V$SGA_TARGET_ADVICE

If I wanted to spend about 66% of my time waiting on IO, I could reduce my SGA
size to 3.5 GB from the current value of 14 GB.

If we reduce the SGA size to about 2 GB, we can spend 100% of our time waiting
for IO.

CACHE SIZE	Size Factor	Buffers for Est	Read Factor	Estd Phys Reads	Read Time	% of db time for reads
1616	0.1	51106	2.0723	131470019	1481786	171.1
3232	0.2	102212	1.806	114576040	1284858	148.4
4848	0.3	153318	1.6294	103371559	1154251	133.3
6464	0.4	204424	1.5063	95560959	1063205	122.8
8080	0.5	255530	1.4069	89254720	989695	114.3
9696	0.6	306636	1.3139	83355318	920927	106.3
11312	0.7	357742	1.2421	78800900	867838	100.2
12928	0.8	408848	1.1511	73026620	800529	92.4
14544	0.9	459954	1.077	68323860	745710	86.1
16160	1	511060	1	63441765	688801	79.5
17776	1.1	562166	0.9438	59878699	647267	74.7
19392	1.2	613272	0.8764	55598427	597373	69
21008	1.3	664378	0.8038	50996766	543733	62.8
22624	1.4	715484	0.7429	47127927	498635	57.6
24240	1.5	766590	0.6919	43897015	460973	53.2
25856	1.6	817696	0.6496	41213165	429689	49.6
27472	1.7	868802	0.6104	38723687	400669	46.3
29088	1.8	919908	0.5756	36516608	374942	43.3
30704	1.9	971014	0.5503	34914017	356261	41.1
32320	2	1022120	0.5122	32493037	328041	37.9

ASM

ASM is new and can therefore be daunting to configure and maintain. It also blurs the once well-drawn divisions of responsibility between UNIX systems administrators and Oracle database administrators.

Previously, UNIX systems administrators would set up file-systems on logical volumes for use by Oracle databases. With ASM, UNIX system administrators now make storage LUNs available from which database administrators build disk groups, into which database files are created.

It seems different, but ASM uses many of the same concepts as file-systems and volume-managers, so it is not too difficult to adapt."

Automatic Storage Management – ASM

- Because Oracle-managed files require the use of a file system, DBAs give up control over how the data is laid out. Therefore, it is important to correctly configure the file system.
- The Oracle-managed file system should be built on top of an LVM that supports striping. For load balancing and improved throughput, the disks in the Oracle-managed file system should be striped.
- Oracle-managed files work best if used on an LVM that supports dynamically extensible logical volumes. Otherwise, the logical volumes should be configured as large as possible.
- Oracle-managed files work best if the file system provides large extensible files.

Range Partitioning

- Range partitioning maps rows to partitions based on ranges of column values.
- Range partitioning is defined by the partitioning specification for a table or index in PARTITION BY RANGE(column_list) and by the partitioning specifications for each individual partition in VALUES LESS THAN(value_list), where column_list is an ordered list of columns that determines the partition to which a row or an index entry belongs.
- These columns are called the partitioning columns.
- The values in the partitioning columns of a particular row constitute that row's partitioning key.

SALES_DATA Indexes

Index Name		Unique	Logging	Degree	Column Name	Order	Position
SD_LOAD_SIG_IDX		N	NO	1	LOAD_SIG	Asc	1
SALES_DATA_PK		Y	NO	1	ITEM_ID	Asc	1
SALES_DATA_PK		Y	NO	1	LOCATION_ID	Asc	2
SALES_DATA_PK		Y	NO	1	SALES_DATE	Asc	3
SALES_DATA_6IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_6IDX		N	NO	1	SALES_DATE	Asc	2
SALES_DATA_6IDX		N	NO	1	BRANCH_ID	Asc	3
SALES_DATA_687_IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_687_IDX		N	NO	1	LOCATION_ID	Asc	2
SALES_DATA_687_IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_687_IDX		N	NO	1	IS_SCENARIO_RESOURCE	Asc	4
SALES_DATA_656_IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_656_IDX		N	NO	1	LOCATION_ID	Asc	2
SALES_DATA_656_IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_656_IDX		N	NO	1	IS_SUPPLY_PLAN	Asc	4
SALES_DATA_4IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_4IDX		N	NO	1	SALES_DATE	Asc	2
SALES_DATA_3IDX		N	NO	1	SALES_DATE	Asc	1
SALES_DATA_2IDX		N	NO	1	LOCATION_ID	Asc	1
SALES_DATA_2IDX		N	NO	1	SALES_DATE	Asc	2

Database Block Size = File System Buffer Size

TEST1

/test1 on /dev/vg02/lvtest1
ioerror=mwdisable,largefiles,mincache=direct,delaylog,nodatainlog,convosync
=direct,dev=40020002 on Tue Nov 16 09:33:20 2010

Stripe Size (Kbytes) 32
Fs block size 1024

root@DEV333:/test1> time dd if=/dev/zero of=/test1/test.file bs=32768
count=1000>
100000+0 records in
100000+0 records out
real 1:32.0
user 0.0
sys 2.5

Database Block Size = File System Buffer Size

TEST 2

/test2 on /dev/vg02/lvtest2
ioerror=mwdisable,largefiles,delaylog,nodatainlog,convosync=direct,dev=400200
03 on Tue Nov 16 09:33:22 2010

Stripe Size (Kbytes) 32
 Fs block size 1024

root@DEV333:/test2> time dd if=/dev/zero of=/test2/test.file bs=32768 coun>
100000+0 records in
100000+0 records out
real 28.8
user 0.0
sys 7.9

Reorg Tables for Better Clustering Factor

Index Name		Unique	Logging	Degree	Column Name	Order	Position
SD_LOAD_SIG_IDX		N	NO	1	LOAD_SIG	Asc	1
SALES_DATA_PK		Y	NO	1	ITEM_ID	Asc	1
SALES_DATA_PK		Y	NO	1	LOCATION_ID	Asc	2
SALES_DATA_PK		Y	NO	1	SALES_DATE	Asc	3
SALES_DATA_6IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_6IDX		N	NO	1	SALES_DATE	Asc	2
SALES_DATA_6IDX		N	NO	1	BRANCH_ID	Asc	3
SALES_DATA_687_IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_687_IDX		N	NO	1	LOCATION_ID	Asc	2
SALES_DATA_687_IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_687_IDX		N	NO	1	IS_SCENARIO_RESOURCE	Asc	4
SALES_DATA_656_IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_656_IDX		N	NO	1	LOCATION_ID	Asc	2
SALES_DATA_656_IDX		N	NO	1	SALES_DATE	Asc	3
SALES_DATA_656_IDX		N	NO	1	IS_SUPPLY_PLAN	Asc	4
SALES_DATA_4IDX		N	NO	1	ITEM_ID	Asc	1
SALES_DATA_4IDX		N	NO	1	SALES_DATE	Asc	2
SALES_DATA_3IDX		N	NO	1	SALES_DATE	Asc	1
SALES_DATA_2IDX		N	NO	1	LOCATION_ID	Asc	1
SALES_DATA_2IDX		N	NO	1	SALES_DATE	Asc	2

Rebuild Tables and Move Null Columns to the End of the Table

Some tables can have more than 255 columns. For example, in Demantra the number of columns in the SALES_DATA table can vary, but my current customer has 588 columns in their SALES_DATA table. Typically, Oracle breaks up rows at the 255 column mark; this causes massive chaining, and forces db file sequential reads instead of full table scans.

However, not all the columns are used and most of them are null. By moving the non-null columns into the first 255 columns, Oracle will skip the extra IO caused by chaining.

By changing the column order in the sales data table, we can reduce chaining, reduce the table size by about 50%, and reduce logical and physical IO.

Rebuild Tables and Move Null Columns to the End of the Table

Tim Gorman says:
"Trailing NULL columns are not stored, but columns with NULL values which are followed by non-NULL columns will be held by a place-holder of one byte.
So, if you have a table with the format...

```
COL_A    number,
COL_B    number,
COL_C    number,
COL_D    number,
COL_E    number
```

And you have a row where all columns except COL_C have non-NULL data values, then all columns will be populated, though of course COL_C will have a 1-byte placeholder for it's NULL value. However, if all columns except COL_E have non-NULL data values, then that row will consume storage just for the four leading non-NULL columns, and no storage will be used for the trailing NULL column COL_E."

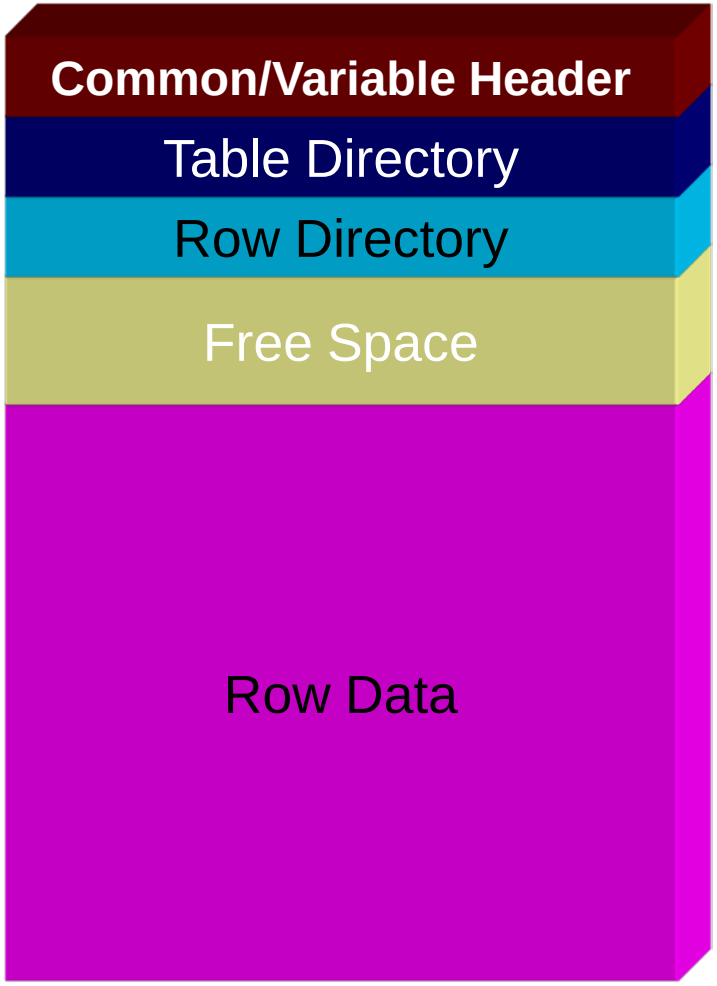
Rebuild Tables and Move Null Columns to the End of the Table

```
select count(column_name), table_name from dba_tab_columns
where owner = 'DEV'
and TABLE_NAME not like 'SIMU%'
and TABLE_NAME not like 'COEFF%'
and TABLE_NAME not like '%TEST%'
having count(column_name) > 255 group by table_name
```

Column Count	Table Name
593	SALES_DATA
530	ADS_CTO_SALES_DATA
350	SIM_MATRIX_V
305	ADS_MDP_MATRIX
461	BRANCH_DATA_ITEMS
509	PROMOTION_DATA
351	MDP_MATRIX

Data Blocks

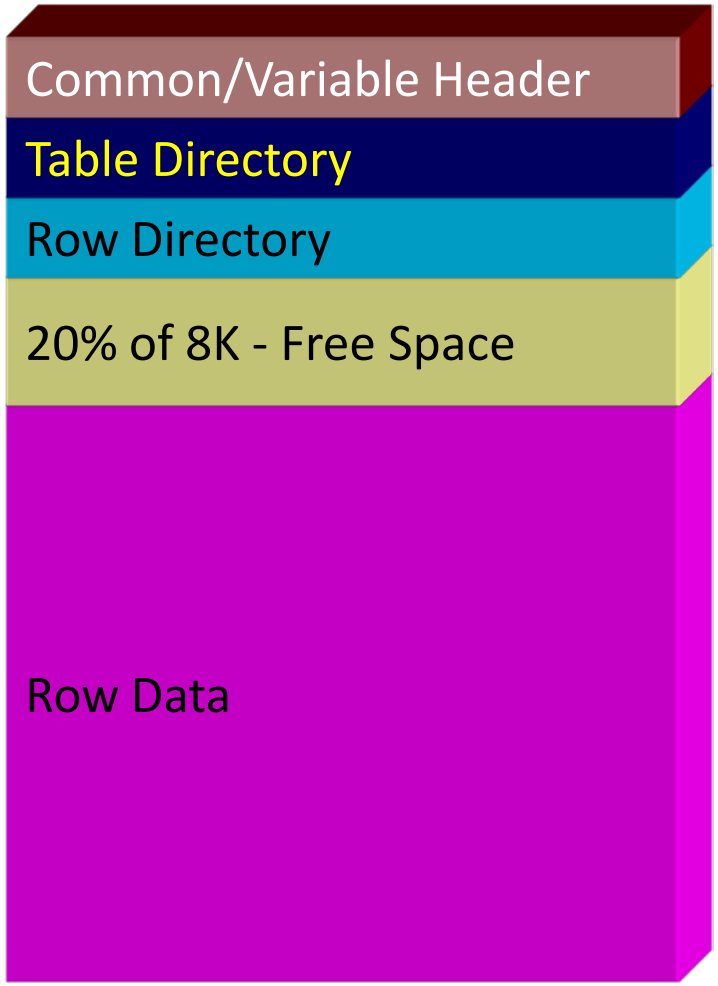
- Oracle manages the storage space in the datafiles of a database in units called **data blocks**. A data block is the smallest unit of data used by a database. In contrast, at the physical, operating system level, all data is stored in bytes. Each operating system has a **block size**. Oracle requests data in multiples of Oracle data blocks, not operating system blocks



Percent Free

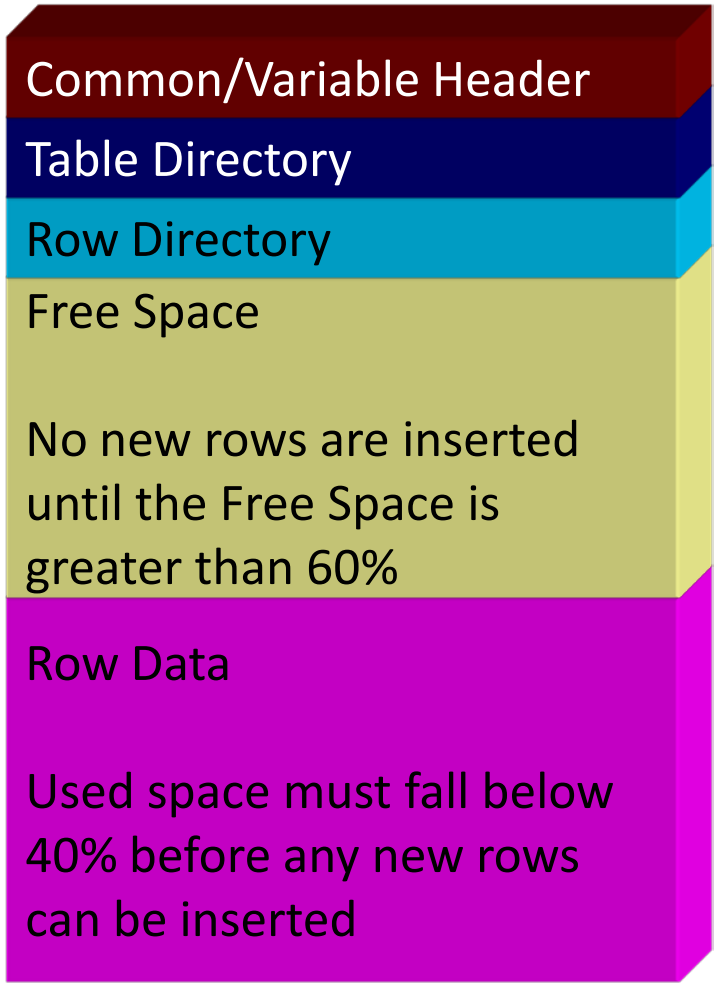
CREATE TABLE statement:

PCTFREE 20 This states that 20% of each data block in this table's data segment be kept free and available for possible updates to the existing rows already within each block. New rows can be added to the row data area, and corresponding information can be added to the variable portions of the overhead area, until the row data and overhead total 80% of the total block size.



Percent Used

PCTUSED 40: In this case, a data block used for this table's data segment is considered unavailable for the insertion of any new rows until the amount of used space in the block falls to less than 40%, (assuming that the block's used space has previously reached PCTFREE).



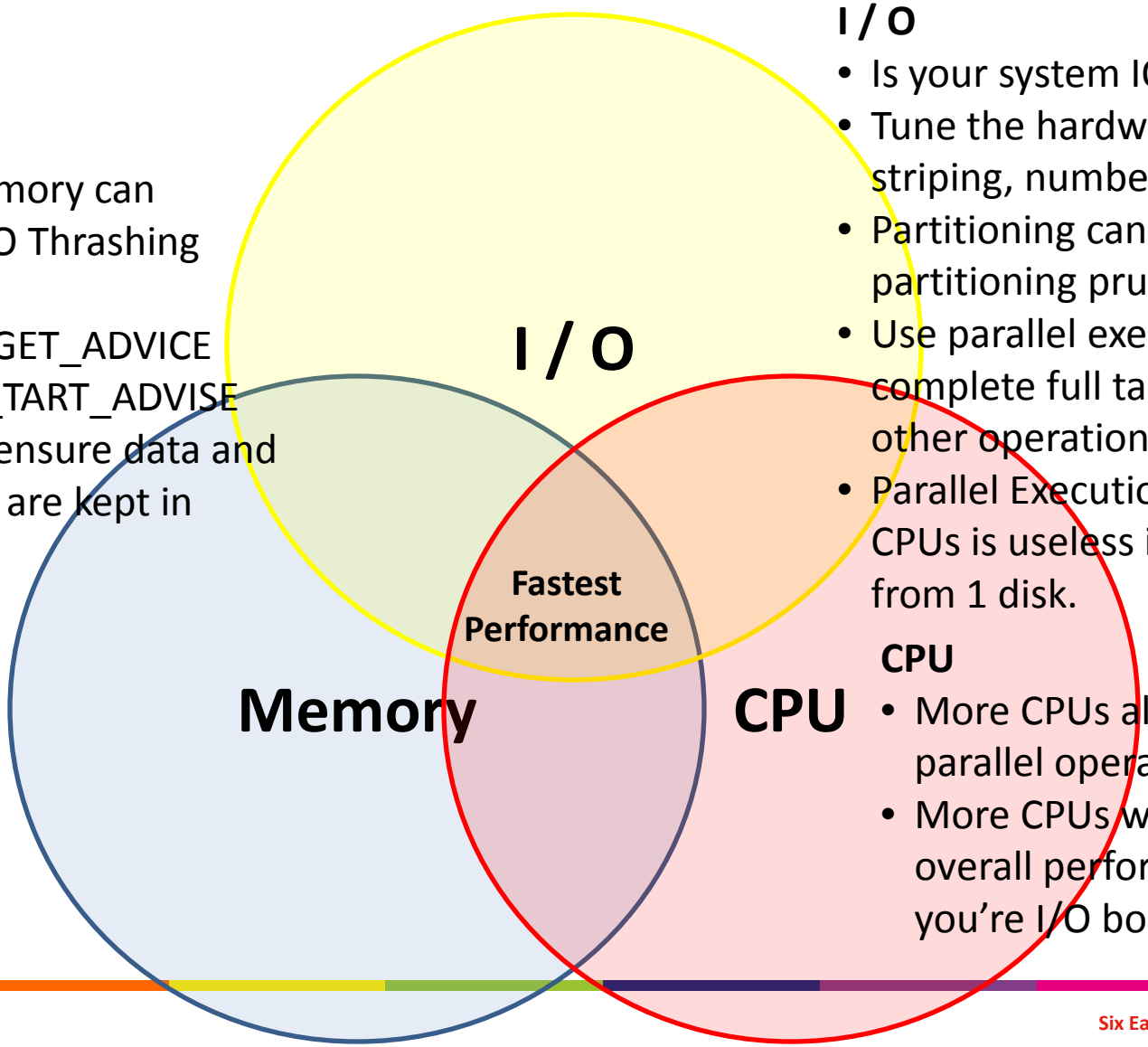
Degree of Parallelism

No more than two sets of parallel execution servers can run simultaneously. Each set of parallel execution servers may process multiple operations. Only two sets of parallel execution servers need to be active to guarantee optimal inter-operation parallelism.

Parallel execution is designed to effectively use multiple CPUs and disks to answer queries quickly. When multiple users use parallel execution at the same time, it is easy to quickly exhaust available CPU, memory, and disk resources.

Hardware Tuning

Memory
- More memory can reduce I/O Thrashing
- Use the SGA_TARGET_ADVICE and PGA_TART_ADVISE tables to ensure data and programs are kept in memory.



I / O
- Is your system IO bound?
- Tune the hardware, block size, striping, number of disks
- Partitioning can reduce IO with partitioning pruning
- Use parallel execution to complete full table scans and other operations in parallel.
- Parallel Execution with more CPUs is useless if all reads come from 1 disk.

CPU
- More CPUs allows for more parallel operations
- More CPUs won't help overall performance if you're I/O bound.

Controlling the Behavior of the Query Optimizer

CURSOR_SHARING

This parameter converts literal values in SQL statements to bind variables. Converting the values improves cursor sharing and can affect the execution plans of SQL statements. The optimizer generates the execution plan based on the presence of the bind variables and not the actual literal values.

DB_FILE_MULTIBLOCK_READ_COUNT

This parameter specifies the number of blocks that are read in a single I/O during a full table scan or index fast full scan. The optimizer uses the value of DB_FILE_MULTIBLOCK_READ_COUNT to cost full table scans and index fast full scans. Larger values result in a cheaper cost for full table scans and can result in the optimizer choosing a full table scan over an index scan.

Controlling the Behavior of the Query Optimizer

OPTIMIZER_INDEX_CACHING

This parameter controls the costing of an index probe in conjunction with a nested loop. The range of values 0 to 100 for OPTIMIZER_INDEX_CACHING indicates percentage of index blocks in the buffer cache, which modifies the optimizer's assumptions about index caching for nested loops and IN-list iterators. A value of 100 infers that 100% of the index blocks are likely to be found in the buffer cache and the optimizer adjusts the cost of an index probe or nested loop accordingly. Use caution when using this parameter because execution plans can change in favor of index caching.

OPTIMIZER_INDEX_COST_ADJ

This parameter can be used to adjust the cost of index probes. The range of values is 1 to 10000. The default value is 100, which means that indexes are evaluated as an access path based on the normal costing model. A value of 10 means that the cost of an index access path is one-tenth the normal cost of an index access path.

Controlling the Behavior of the Query Optimizer

OPTIMIZER_MODE
This initialization parameter sets the mode of the optimizer at instance startup. The possible values are RULE, CHOOSE, ALL_ROWS, FIRST_ROWS_*n*, and FIRST_ROWS.

PGA_AGGREGATE_TARGET
This parameter automatically controls the amount of memory allocated for sorts and hash joins. Larger amounts of memory allocated for sorts or hash joins reduce the optimizer cost of these operations.

STAR_TRANSFORMATION_ENABLED
This parameter, if set to true, enables the query optimizer to cost a star transformation for star queries. The star transformation combines the bitmap indexes on the various fact table columns

Parallel Operations - DML

The following operations can be parallelized in Oracle:

Parallel Query (SELECT statements)

Parallel DML

INSERT, UPDATE, DELETE, and MERGE operations

- Parallel DML operations are mainly used to speed up large DML operations against large database objects.
- Parallel DML is useful in a DSS environment where the performance and scalability of accessing large objects are important.
- Parallel DML complements parallel query in providing you with both querying and updating capabilities for your DSS databases.

Parallel Operations - DDL

The overhead of setting up parallelism makes parallel DML operations infeasible for short OLTP transactions. However, parallel DML operations can speed up batch jobs running in an OLTP database.

Parallel DDL

CREATE TABLE AS SELECT

The CREATE TABLE statement for an index-organized table can be parallelized either with or without an AS SELECT clause

CREATE INDEX,

ALTER INDEX REBUILD

If the table is partitioned,

ALTER TABLE MOVE or [SPLIT or COALESCE]

If the index is partitioned,

ALTER INDEX REBUILD [or SPLIT]

Parallel Operations - SQLLoad

All of these DDL operations can be performed in NOLOGGING mode for either parallel or serial execution.

Different parallelism is used for different operations. **Parallel create (partitioned) table as select and parallel create (partitioned) index run with a degree of parallelism equal to the number of partitions.**

Parallel operations require accurate statistics to perform optimally.

- Gathering table or index statistics
- Refreshing materialized views
- SQL*Loader
  - SQLLOAD scott/tiger CONTROL=ctl1.ctl **DIRECT=TRUE PARALLEL=TRUE**
  - SQLLOAD scott/tiger CONTROL=ctl2.ctl **DIRECT=TRUE PARALLEL=TRUE**