Diagram

Description automatically generated

Diagram

Description automatically generatedGraphical user interface, diagram

Description automatically generated

Graphical user interface, application

Description automatically generated Diagram

Description automatically generated with medium confidence

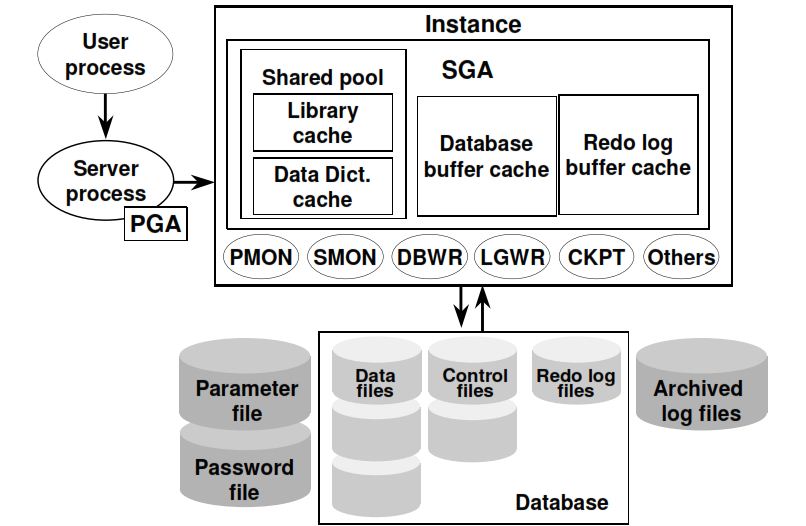
Diagram

Description automatically generated

Diagram

Description automatically generated

Oracle Instance architecture

Oracle

Performance Tuning Tools

Performance tools can examine execution strategies and can be used for [SQL tuning](http://www.datadisk.co.uk/html_docs/oracle/sql_optimization.htm), the tools can give you a good estimate of resource usage used by the queries.

The common tuning tools are

* *explain plan* - lets you see the execution plan used by the query
* *autotrace* - automatically produces an execution plan when you execute any SQL.
* *SQL trace* - traces the execution of SQL statements
* *TKPROF* - formats the trace files into a readable form
* *dbms\_monitor* - setup up end-to-end tracing
* Statspack - performance diagnosis tool
* AWR - automatic load repository

**EXPLAIN Plan**

The *explain plan* lets you see the execution plan used by the optimizer when it executed your query, it is idea when you are using hints as you can see if the hint is being used or not. The output of the explain plan goes into a table called *plan\_table*, the explain plan out will tell you

* The tables used in the query and the order in which they are accessed
* The operations performed on the output of each step of the plan.
* The specific access and join methods used for each table mentioned
* The cost of each operation.

To create your own explain plan you must execute a oracle script which will create the plan table where the output of the explain plan is stored.

|  |  |
| --- | --- |
| Setting up explain plan | @$ORACLE\_HOME/rdbms/admin/utlxplan.sql |
| Creating the EXPLAIN plan | explain plan for   select \* from employees where lname = 'valle'; |
| Display explain plan | select \* from plan\_table (dbms\_xplan.display);  Note: i have used the dbms\_xplan package in the above statement |

When reading the plan there are some principle to consider

* Each step in the plan returns output in the form of a set of rows to the parent step
* Read the plan outward starting from the line that is indented the most
* If two operations are at the same level in terms of indentation, read the top one first
* The numbering of the steps in the plan is misleading, start reading the plan output from inside out, read the most indented operations first

|  |  |
| --- | --- |
| example | select statement   hash join     nested loops       table access full department       index unique scan employee\_pk     index fast full scan dept\_id\_pk   1. Oracle does a full table scan of the department table 2. Oracle performs an index unique scan of the employees table using its primary key index 3. Oracle performs a nested loop operation to join the rows from step 1 and 2 4. Oracle performs a fast full index scan of the department\_id using its primary key 5. The final step oracle performs a hash join of the set from step3 and the rows resulting from step 4 |

**Autotrace**

The autotrace facility enables you to produce *explain plans* automatically when you execute a SQL statement. make sure that the plan table has been created (see above regarding running script utlxplan.sql).

|  |  |
| --- | --- |
| Set privilege | grant plustrace to vallep;  Note: you can also grant to public |
| Turn off | set autotrace off |
| Turn on | set autotrace on  Note: this turns on explain and statistics |
| Turn on explain | set autotrace {on|off|trace[only]} explain |
| Turn on statistics | set autotrace {on|off|trace[only]} statistics |
| Traceonly | set autotrace traceonly |

**SQL Trace and TKPROF**

SQL trace helps you trace the execution of SQL statements, TKPROF formats the output from the trace into a readable format. SQL trace enables you to track the following variables in a SQL statement

* CPU and elapsed times
* Parsed and execution counts for each SQL statement
* Number of physical and logical reads
* Execution plan for all the SQL statements
* Library cache hit ratios

The explain plan gives you important information regarding the access path that the optimizer used, the SQL trace gives you the breakdown of the resources used CPU and I/O.

Collecting trace statistics imposes a performance penalty, you can control the collection of statistics by two parameters

|  |  |
| --- | --- |
| Turn on statistics collection | alter system set statistics\_level = typical; alter system set statistics\_level = all; |
| Turn off statistics collection | alter system set statistics\_level = basic; |
| Turn on timed statistics | alter system set time\_statistics = true;  Note: default is set to false, even if the statistics\_level is set to basic (off) time statistics will be collected |

You can turn on tracing for both the session or instance level, remember that turning on for the instance will use a lot of disk space and system resources.

|  |  |
| --- | --- |
| Instance | alter system set sql\_trace = true; alter system set sql\_trace = false; |
| Session | alter session set sql\_trace = true; alter session set sql\_trace = false;  dbms\_system.set\_sql\_trace\_in\_session(<sid><serial#>,true); dbms\_system.set\_sql\_trace\_in\_session(<sid><serial#>,false); |

The trace will create a file located in the *user\_dump\_dest* and will format the file as *db\_name\_ora\_nnnnn.trc*, the file size will generally be much larger in size that other files in this area.

TKPROF uses the trace file along with the following parameters

|  |  |
| --- | --- |
| **FILENAME** | input trace file |
| **EXPLAIN** | the *explain plan* for the SQL statement |
| **RECORD** | creates a SQL script with all the nonrecursive SQL statements |
| **WAITS** | Records a summary of wait events |
| **SORT** | Presents sort data based on one or more items |
| **TABLE** | The name of the table TKPROF temporarily puts the executions plans |
| **SYS** | Enables and disables listing SQL statements issued by SYS |
| **PRINT** | Lists only a specified number of SQL statements instead of all statements |
| **INSERT** | Creates scripts that stores the trace information in the database. |

|  |  |
| --- | --- |
| TKPROF examples | tkprof finance\_ora\_16340.trc test.txt sys=no explain=y  Note: the output will be dumped into the test.txt file |

**End-to-End Tracing**

By using a new attribute *client\_identifier* you can trace a users session through multiple database sessions. you use the package *dbms\_monitor* or OEM to setup the tracing. You require three attributes to trace the session

* Client Identifier
* Service Name
* Combination of service name, module name and action name.

Below is an example on how to use the end-to-end tracing

|  |  |
| --- | --- |
| setup the service name, module name, action name and the client id | dbms\_monitor.serv\_mod\_act\_trace\_enable (   service\_name => 'myservice',   module\_name => 'batch\_job',   action\_name => 'batch\_insert' );  dbms\_monitor.client\_id\_trace\_enable (   client\_id => 'vallep' ); |
| set the UID using a trigger | create or replace trigger logon\_trigger   after logon on database declare   user\_id varchar(64); begin   select ora\_login\_user || ':' || sys\_context('userenv','os\_user') into user\_id from dual;   dbms\_session.set\_identifier(uid); end; |
| Obtain the sid and serial# | dbms\_monitor.session\_trace\_enable (   session\_id => 111,   serial\_num => 23,   waits => true,   binds => false ); |
| Combine the multiple trace files in one file | c:\> trcsess output="vallep.trc" service="myservice" module="batch\_job" action="batch\_insert" |
| run TKPROF against the consolidated file | c:\> tkprof vallep.trc output=vallep.rpt sort=(EXELA, PRSELA, FCHELA)  Note: there are many options to the sort parameter please see the Oracle documentation for more information.  EXELA - elapsed time executing PRSELA - elapsed time parsing FCHELA - elapsed time fetching |

**Inefficient SQL**

You can use the view *v$sql* to find inefficient SQL code, the view gathers important information regarding the disk reads and memory reads for a SQL statement. This view holds information on statements since startup, it also ages out older statements. The view will give you information on the following

* *rows\_processed* - total number of rows processed by the statement
* *sql\_text* - the SQL text of the statement (first 1,000 characters)
* *buffer\_gets* - total number of logical reads (high CPU use)
* *disk\_reads* - total number of disk reads (high I/O use)
* *sorts* - number of sort for the statement (high sort ratios)
* *cpu\_time* - total parse and execution time
* *elapsed\_time* - elapsed parse and execution time
* *parse\_calls* - combined soft and hard parse calls
* *executions* - number of times the statement was executed
* *loads* - number of times the statement was flushed out of the shared pool then reloaded
* *sharable\_memory* - total memory used by the shared cursor
* *persistant\_memory* - total persistent memory used by the cursor
* *runtime\_memory* - total runtime memory used by the cursor

|  |  |
| --- | --- |
| High disk reads | select sql\_text, executions, buffer\_gets, disk\_reads from v$sql    where buffer\_gets > 100000 or disk\_reads > 100000 order by buffer\_gets + 100 \* disk\_reads desc; |
| High disk reads and parsed calls and row processed | select sql\_text, rows\_processed, buffer\_gets, disk\_reads, parsed\_calls from v$sql    where buffer\_gets > 100000 or disk\_reads > 100000 order by buffer\_gets + 100 \* disk\_reads desc; |
| TOP 5 CU time and elapsed time | select sql\_text, executions,   round(elapsed\_time/1000000, 2) elapsed\_seconds,   round(cpu\_time/1000000, 2) cpu\_secs from (select \* from v£sql order by elapsed\_time desc) where rownum < 6; |

**SQL Tuning Advisor**

When you have identified bad SQL, you can use the [SQL tuning advisor](http://www.datadisk.co.uk/html_docs/oracle/advisors.htm) to perform an in depth analysis to come up with a better execution plan.

See the [Advisors](http://www.datadisk.co.uk/html_docs/oracle/advisors.htm) for more detailed information.

**Statspack**

Statspack is a diagnostic tool that captures and stores the V$ table information and allows you generate reports at a later date, although it has been replaced with [AWR](http://www.datadisk.co.uk/html_docs/oracle/awr.htm) many dba's still use this tool, I will only give a brief overview of this tool as you will more than likely start using the [AWR](http://www.datadisk.co.uk/html_docs/oracle/awr.htm) tool as its now Oracle preferred method of collecting statistics.

To install statspack you simply run the following as sys with sysdba privilege "$oracle\_home\rdbms\admin\spcreate.sql", you can use "spdrop.sql" to remove it. The install script will ask you three pieces of information

* The PERFSTAT user password
* The default tablespace that you will use for the PERFSTAT schema
* The default temporary tablespace that you will use for the PERFSTAT schema

Once the installation has finished check the "spcpkg.lis" file for any errors, the below commands can be run to obtain snapshots and generate reports.

|  |  |
| --- | --- |
| Create snapshot | exec statspack.snap |
| Run report | @$oracle\_home\rdbms\admin\spreport.sql  Note: when you run the report it will ask for two snapshot points to compare. |

**AWR**

* Check Dispatcher Setting
* Enable Connection Pooling

RAC Performance

I have already discussed basic [Oracle tuning](http://www.datadisk.co.uk/html_docs/oracle/tuning_tools.htm), in this section I will mainly discuss Oracle RAC tuning. First lets review the best practices of a Oracle design regarding the application and database

* Optimize connection management, ensure that the middle tier and programs that connect to the database are efficient in connection management and do not log on or off repeatedly
* Tune the SQL using the available tools such as [ADDM](http://www.datadisk.co.uk/html_docs/oracle/addm.htm) and SQL Tuning Advisor
* Ensure that applications use **bind variables**, *cursor\_sharing* was introduced to solve this problem
* Use [packages and procedures](http://www.datadisk.co.uk/html_docs/oracle/pl_sql_packages.htm) (because they are compiled) in place of anonymous PL/SQL blocks and big SQL statements
* Use [locally managed tablespaces](http://www.datadisk.co.uk/html_docs/oracle/tablespaces.htm) and [automatic segment space management](http://www.datadisk.co.uk/html_docs/oracle/assm.htm) to help performance and simplify database administration
* Use automatic [undo management](http://www.datadisk.co.uk/html_docs/oracle/undo.htm) and [temporary tablespace](http://www.datadisk.co.uk/html_docs/oracle/tablespaces.htm) to simplify administration and increase performance
* Ensure you use large caching when using [sequences](http://www.datadisk.co.uk/html_docs/oracle/synon_seq.htm), unless you cannot afford to lose sequence during a crash
* Avoid using DDL in production, it increases invalidations of the already parsed SQL statements and they need to be recompiled
* Partition tables and indexes to reduce index leaf contention (buffer busy global cr problems)
* Optimize contention on data blocks (hot spots) by avoiding small tables with too many rows in a block

Now we can review RAC specific best practices

* Consider using application partitioning (see below)
* Consider restricting DML-intensive users to using one instance, thus reducing cache contention
* Keep read-only tablespaces away from DML-intensive tablespaces, they only require minimum resources thus optimizing Cache Fusion performance
* Avoid auditing in RAC, this causes more shared library cache locks
* Use full tables scans sparingly, it causes the [GCS](http://www.datadisk.co.uk/html_docs/rac/architecture.htm#kernel) to service lots of block requests, see table *v$sysstat*column "table scans (long tables)"
* if the application uses lots of logins, increase the value of *sys.audsess$* sequence

**Partitioning Workload**

Workload partitioning is a certain type of workload that is executed on an instance, that is partitioning allows users who access the same set of data to log on to the same instance. This limits the amount of data that is shared between instances thus saving resources used for messaging and [Cache Fusion](http://www.datadisk.co.uk/html_docs/rac/cache_fusion.htm) data block transfer.

You should consider the following when deciding to implement partitioning

* If the CPU and private interconnects are of high performance then there is no need to partition
* Partitioning does add complexity, thus if you can increase CPU and the interconnect performance the better
* Only partition if performance is betting impacted
* Test both partitioning and non-partitioning to what difference it makes, then decide if partitioning is worth it

**RAC Wait Events**

An *event*is an operation or particular function that the Oracle kernel performs on behalf of a user or a Oracle background process, events have specific names like *database event*. Whenever a session must wait for something, the wait time is tracked and charged to the event that was associated with that wait. Events that are associated with all such waits are known as *wait events*. They are a few wait classes

* Commit
* Scheduler
* Application
* Configuration
* User I/O
* System I/O
* Concurrency
* Network
* Administrative
* Cluster
* Idle
* Other

There are over 800 different events spread across the above list, however you probably will only deal with about 50 or so that can improve performance.

When a session requests access to a data block it sends a request to the lock master for proper authorization, the request does not know if it will receive the block via Cache Fusion or a permission to read from the disk. Two placeholder events

* *global cache cr request (consistent read - cr)*
* *global cache curr request (current - curr)*

keep track of the time a session spends in this state. There are number of types of wait events regarding access to a data block

|  |  |  |
| --- | --- | --- |
| **Wait Event** | **Contention type** | **Description** |
| **gc current block 2-way** | write/write | an instance requests authorization for a block to be accessed in current mode to modify a block, the instance mastering the resource receives the request. The master has the current version of the block and sends the current copy of the block to the requestor via Cache Fusion and keeps a Past Image (.PI)  If you get this then do the following   * Analyze the contention, segments in the "current blocks received" section of [AWR](http://www.datadisk.co.uk/html_docs/oracle/awr.htm) * Use application partitioning scheme * Make sure the system has enough CPU power * Make sure the interconnect is as fast as possible * Ensure that socket send and receive buffers are configured correctly |
| **gc current block 3-way** | write/write | an instance requests authorization for a block to be accessed in current mode to modify a block, the instance mastering the resource receives the request and forwards it to the current holder of the block, asking it to relinquish ownership. The holding instance sends a copy of the current version of the block to the requestor via Cache Fusion and transfers the exclusive lock to the requesting instance. It also keeps a past Image (PI).  Use the above actions to increase the performance |
| **gc current block 2-way** | write/read | The difference with the one above is that this sends a copy of the block thus keeping the current copy. |
| **gc current block 3-way** | write/read | The difference with the one above is that this sends a copy of the block thus keeping the current copy. |
| **gc current block busy** | write/write | The requestor will eventually get the block via cache fusion but it is delayed due to one of the following   * The block was being used by another session on another session * was delayed as the holding instance could not write the corresponding redo record immediately   If you get this then do the following   * Ensure the [log writer](http://www.datadisk.co.uk/html_docs/oracle/redo.htm) is tuned |
| **gc current buffer busy** | local | This is the same as above (gc current block busy), the difference is that another session on the same instance also has requested the block (hence local contention) |
| **gc current block congested** | none | This is caused if heavy congestion on the GCS, thus CPU resources are stretched |

**Enqueue Tuning**

Oracle RAC uses a queuing mechanism to ensure proper use of shared resources, it is called Global Enqueue Services (GES). Enqueue wait is the time spent by a session waiting for a shared resource, here are some examples of enqueues:

* updating the control file (CF enqueue)
* updating an individual row (TX enqueue)
* exclusive lock on a table (TM enqueue)

Enqueues can be managed by the instance itself others are used globally, GES is responsible for coordinating the global resources. The formula used to calculate the number of enqueue resources is as below

                     GES Resources = DB\_FILES + DML\_LOCKS + ENQUEUE\_RESOURCES + PROCESS + TRANSACTION x (1 + (N - 1)/N)  
  
                      N = number of RAC instances

|  |  |
| --- | --- |
| displaying enqueues stats | SQL> column current\_utilization heading current SQL> column max\_utilization heading max\_usage SQL> column initial\_allocation heading initial SQL> column resource\_limit format a23;  SQL> select \* from v$resource\_limit; |

**AWR and RAC**

I have already discussed [AWR](http://www.datadisk.co.uk/html_docs/oracle/awr.htm) in a single instance environment, so for a quick refresh take a look and come back here to see how you can use it in a RAC environment.

From a RAC point of view there are a number of RAC-specific sections that you need to look at in the AWR, in the report section is a AWR of my home RAC environment, you can view the whole report [here](http://www.datadisk.co.uk/html_docs/rac/awrrpt.txt).

|  |  |  |
| --- | --- | --- |
| **RAC AWR Section** | **Report** | **Description** |
| **Number of Instances** | [instances](http://www.datadisk.co.uk/html_docs/rac/rac_instances.txt) | lists the number of instances from the beginning and end of the AWR report |
| **Instance global cache load profile** | [global cache](http://www.datadisk.co.uk/html_docs/rac/global_cache_profile.txt) | information about the interinstance cache fusion data block and messaging traffic, because my [AWR report](http://www.datadisk.co.uk/html_docs/rac/awrrpt.txt) is lightweight here is a more heavy used RAC example  Global Cache Load Profile ~~~~~~~~~~~~~~~~~~~~~~~~~            Per Second        Per Transaction                                                  ---------------          --------------- Global Cache blocks received:       315.37                   12.82 Global Cache blocks served:          240.30                   9.67 GCS/GES messages received:          525.16                   20.81 GCS/GES messages sent:                765.32                   30.91  The first two statistics indicate the number of blocks transferred to or from this instance, thus if you are using a 8K block size          Sent:        240 x 8,192 = 1966080 bytes/sec = 2.0 MB/sec         Received:  315 x 8,192 = 2580480 bytes/sec = 2.6 MB/sec  to determine the amount of network traffic generated due to messaging you first need to find the average message size (this was 193 on my system)      select sum(kjxmsize \* (kjxmrcv + kjxmsnt + kjxmqsnt)) / sum((kjxmrcv + kjxmsnt + kjxmqsnt)) "avg Message size" from x$kjxm         where kjxmrcv > 0 or kjxmsnt > 0 or kjxmqsnt > 0;  then calculate the amount of messaging traffic on this network      193 (765 + 525) = 387000 = 0.4 MB  to calculate the total network traffic generated by cache fusion       = 2.0 + 2.6 + 0.4 = 5 MBytes/sec      = 5 x 8 = 40 Mbits/sec  The DBWR Fusion writes statistic indicates the number of times the local DBWR was forced to write a block to disk due to remote instances, this number should be **low.** |
| **Glocal cache efficiency percentage** | [global cache efficiency](http://www.datadisk.co.uk/html_docs/rac/global_cache_efficiency.txt) | this section shows how the instance is getting all the data blocks it needs. The best order is the following   * Local cache * Remote cache * Disk   The first two give the *cache hit ratio* for the instance, you are looking for a value less than 10%, if you are getting higher values then you may consider [application partitioning](http://www.datadisk.co.uk/html_docs/rac/performance.htm#partitioning). |
| **GCS and GES - workload characteristics** | [GCS and GES workload](http://www.datadisk.co.uk/html_docs/rac/gcs_ges_workload.txt) | this section contains timing statistics for global enqueue and global cache. As a general rule you are looking for   * All timings related to CR (Consistent Read) processing block should be less than 10 msec * All timings related to CURRENT block processing should be less than 20 msec |
| **Messaging statistics** | [messaging](http://www.datadisk.co.uk/html_docs/rac/messaging.txt) | The first section relates to sending a message and should be less than 1 second.  The second section details the breakup of direct and indirect messages, direct messages are sent by a instance foreground or the user processes to remote instances, indirect are messages that are not urgent and are pooled and sent. |
| **Service statistics** | [Service stats](http://www.datadisk.co.uk/html_docs/rac/service_stats.txt) | shows the resources used by all the service instance supports |
| **Service wait class statistics** | [Service wait class](http://www.datadisk.co.uk/html_docs/rac/service_wait_class.txt) | summarizes waits in different categories for each service |
| **Top 5 CR and current block segements** | [Top 5 CR and current blocks](http://www.datadisk.co.uk/html_docs/rac/top_5_cr_current_blocks.txt) | conatns the names of the top 5 contentious segments (table or index). If a table or index has a very high percentage of CR and Current block transfers you need to investigate. This is pretty much like a normal single instance. |

**Cluster Interconnect**

As I stated above the interconnect it a critical part of the RAC, you must make sure that this is on the best hardware you can buy. You can confirm that the interconnect is being used in Oracle 9i and 10g by using the command *oradebug* to dump information out to a trace file, in Oracle 10g R2 the cluster interconnect is also contained in the *alert.log* file, you can view my information from [here](http://www.datadisk.co.uk/html_docs/rac/interconnect.txt).

|  |  |
| --- | --- |
| interconnect | SQL> oradebug setmypid SQL> oradebug ipc  Note: look in the user\_dump\_dest directory, the trace will be there |

## **Identifying Contention for Shared Servers**

* 1. Steadily increasing wait times in the requests queue indicate contention for shared servers.
  2. If you detect resource contention with shared servers, then first ensure that this is not a memory contention issue by examining the shared pool and the large pool.
  3. If performance remains poor, then you might want to create more resources to reduce shared server process contention

## **Measuring Database Performance**

* Time model statistics use time to identify quantitative effects about specific actions performed on the database, such as logon operations and parsing.
* The most important time model statistic is database time, or DB time.
* This statistic represents the total time spent in database calls and is an indicator of the total instance workload.
* DB time is measured cumulatively from the time of instance startup and is calculated by aggregating the CPU and wait times of all sessions not waiting on idle wait events (non-idle user sessions).
* **Ultimately, the objective in tuning an Oracle database is to reduce the time that users spend in performing an action on the database, or to simply reduce DB time**
* **Active Session History Statistics**
  + Active Session History (ASH) enables you to examine and perform detailed analysis on both current data in the V$ACTIVE\_SESSION\_HISTORY view and historical data in the DBA\_HIST\_ACTIVE\_SESS\_HISTORY view, often avoiding the need to replay the workload to trace additional performance information. ASH also contains execution plan information for each captured SQL statement. You can use this information to identify which part of SQL execution contributed most to the SQL elapsed time.
* **Wait Events Statistics**
  + Wait events are statistics that are incremented by a server process or thread to indicate that it had to wait for an event to complete before processing could continue. Wait event data reveals various symptoms of problems that might be impacting performance, such as latch contention, buffer contention, and I/O contention.
  + To enable easier high-level analysis of wait events, Oracle Database groups events into the following classes:
    - Administrative
    - Application
    - Cluster
    - Commit
    - Concurrency
    - Configuration
    - Idle
    - Network
    - Other
    - Scheduler
    - System I/O
  + User I/OThe wait classes are based on a common solution that usually applies to fixing a problem with the particular wait event. For example, exclusive TX locks are generally an application-level issue and HW locks are generally a configuration issue. The following list includes common examples of wait events in some of the wait classes:
    - Application: locks waits caused by row level locking or explicit lock commands
    - Commit: waits for redo log write confirmation after a commit
    - Idle: wait events that signify the session is inactive, such as SQL\*Net message from client
    - Network: waits for data to be sent over the network
    - User I/O: wait for blocks to be read off a disk

## **Interpreting Database Statistics**

* Using Hit Ratios
  + When tuning, it is common to compute a ratio that helps determine to identify whether a performance bottleneck exists, examine other related performance data.
  + Such ratios may include the
    - buffer cache hit ratio
    - the soft-parse ratio
    - the latch hit ratio
* Using Wait Events with Timed Statistics
  + This data is useful for comparing the total wait time for an event to the total elapsed time between the data collections.
  + For example, if the wait event accounts for only 30 seconds out of a 2-hour period, then very little performance improvement can be gained by investigating this event
* Comparing Database Statistics with Other Factors
  + it is important to consider other factors that may influence whether the statistic is of value. Such factors may include the user load and hardware capability.

## **About Gathering Database Statistics**

**AWR**

AWR collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This gathered data is stored both in memory and in the database and is displayed in both reports and views.The statistics collected and processed by AWR include:

* Object statistics that determine both access and usage statistics of database segments
* Time model statistics based on time usage for activities, displayed in the V$SYS\_TIME\_MODEL and V$SESS\_TIME\_MODEL views
* Some of the system and session statistics collected in the V$SYSSTAT and V$SESSTAT views
* SQL statements that are producing the highest load on the system, based on criteria such as elapsed time and CPU time
* Active Session History (ASH) statistics, representing the history of recent sessions activity

**Snapshots**

* Snapshots are sets of historical data for specific time periods that are used for performance comparisons by Automatic Database Diagnostic Monitor (ADDM). By default, Oracle Database automatically generates snapshots of the performance data once every hour and retains the statistics in AWR for 8 days. You can also manually create snapshots or change the snapshot retention period, but it is usually not necessary.

## **Instance Tuning Steps**

#### **CPU Usage**

If there is high CPU usage, then determine whether the CPU is being used effectively.

Is most of the CPU usage attributable to a small number of high-CPU using programs, or is the CPU consumed by an evenly distributed workload?

If a small number of high-usage programs use the CPU, then look at the programs to determine the cause. Check whether some processes alone consume the full power of one CPU. Depending on the process, this could indicate a CPU or process-bound workload that can be tackled by dividing or parallelizing process activity.

If a small number of Oracle processes consumes most of the CPU resources, then use SQL\_TRACE and TKPROF to identify the SQL or PL/SQL statements to see if a particular query or PL/SQL program unit can be tuned.

#### **Identifying I/O Problems**

An overly active I/O system can be evidenced by disk queue lengths greater than two, or disk service times that are over 20-30ms.

Check the Oracle wait event data in V$SYSTEM\_EVENT to see whether the top wait events are I/O related. I/O related events include

* db file sequential read
* db file scattered read
* db file single write
* db file parallel write
* log file parallel write

These are all events corresponding to I/Os performed against data files and log files. If any of these wait events correspond to high average time, then investigate the I/O contention.

If the I/O system is overly active, then check for potential hot spots that could benefit from distributing the I/O across more disks.

Also identify whether the load can be reduced by lowering the resource requirements of the programs using those resources.

If the I/O problems are caused by Oracle Database, then I/O tuning can begin.

If Oracle Database is not consuming the available I/O resources, then identify the process that is using up the I/O.

Determine why the process is using up the I/O, and then tune this process.

#### **Identifying Network Issues**

Using operating system utilities, look at the network round-trip ping time and the number of collisions. If the network is causing large delays in response time, then investigate possible causes.

### Examine the Oracle Database Statistics

#### **Wait Events**

Wait events are grouped into classes. The wait event classes include:

Administrative,

Application,

Cluster,

Commit,

Concurrency,

Configuration, Idle, Network, Other, Scheduler, System I/O, and User I/O.

### Table of Wait Events and Potential Causes

### Graphical user interface, text, application, email Description automatically generated