Resolving common Oracle Wait Events using the Wait Interface

Wait Event	Possible Causes	Actions	Remarks
db file sequential reads	Use of an unselective index Fragmented Indexes High I/O on a particular disk or mount point Bad application design Index reads performance can be affected by slow I/O subsystem and/or poor database files layout, which result in a higher average wait time	Check indexes on the table to ensure that the right index is being used Check the column order of the index with the WHERE clause of the Top SQL statements Rebuild indexes with a high clustering factor Use partitioning to reduce the amount of blocks being visited Make sure optimizer statistics are up to date Relocate 'hot' datafiles Consider the usage of multiple buffer pools and cache frequently used indexes/tables in the KEEP pool Inspect the execution plans of the SQL statements that access data through indexes Is it appropriate for the SQL statements to access data through index lookups? Is the application an online transaction processing (OLTP) or decision support system (DSS)? Would full table scans be more efficient? Do the statements use the right driving table? The optimization goal is to minimize both the number of logical and physical I/Os.	The Oracle process wants a block that is currently not in the SGA, a for the database block to be read into the SGA from disk. Significant *db file sequential read* wait time is most likely an applicati If the DBA_INDEXES.CLUSTERING_FACTOR of the index approaches th blocks in the table, then most of the rows in the table are ordered. Th However, if the clustering factor approaches the number of rows in t means the rows in the table are randomly ordered and thus it require complete the operation. You can improve the index's clustering facto the table so that rows are ordered according to the index key and re index thereafter. The OPTIMIZER_INDEX_COST_ADJ and OPTIMIZER_INDEX_CA initialization parameters can influence the optimizer to favour the nes operation and choose an index access path over a full table scan. Tuning I/O related waits Note# 223117.1 db file sequential read Reference Note# 34559.1
db file scattered reads	The Oracle session has requested and is waiting for multiple contiguous database blocks (up to DB_FILE_MULTIBLOCK_READ_COUNT) to be read into the SGA from disk. Full Table scans Fast Full Index Scans	Optimize multi-block I/O by setting the parameter DB_FILE_MULTIBLOCK_READ_COUNT Partition pruning to reduce number of blocks visited Consider the usage of multiple buffer pools and cache frequently used indexes/tables in the KEEP pool Optimize the SQL statement that initiated most of the waits. The goal is to minimize the number of physical and logical reads. Should the statement access the data by a full table scan or index FFS? Would an index range or unique scan be more efficient? Does the query use the right driving table? Are the SQL predicates appropriate for hash or merge join? If full scans are appropriate, can parallel query improve the response time? The objective is to reduce the demands for both the logical and physical I/Os, and this is best achieved through SQL and application tuning. Make sure all statistics are representative of the actual data. Check the LAST_ANALYZED date	If an application that has been running fine for a while suddenly clock on the *db file scattered read* event and there hasn't been a code cha want to check to see if one or more indexes has been dropped or be unusable. db file scattered read Reference Note# 34558.1
log file parallel write	LGWR waits while writing contents of the redo log buffer cache to the online log files on disk I/O wait on sub system holding the online redo log files	Reduce the amount of redo being generated Do not leave tablespaces in hot backup mode for longer than necessary Do not use RAID 5 for redo log files Use faster disks for redo log files Ensure that the disks holding the archived redo log files and the online redo log files are separate so as to avoid contention Consider using NOLOGGING or UNRECOVERABLE options in SQL statements	Reference Note# 34583.1
log file sync	Oracle foreground processes are waiting for a COMMIT or ROLLBACK to complete	Tune LGWR to get good throughput to disk eg: Do not put redo logs on RAID5 Reduce overall number of commits by batching transactions so that there are fewer distinct COMMIT operations	Reference Note# 34592.1 High Waits on log file sync Note# 125269.1 Tuning the Redolog Buffer Cache and Resolving Redo Latch Conten Note# 147471.1
buffer busy waits	Buffer busy waits are common in an I/O-bound Oracle system. The two main cases where this can occur are: Another session is reading the block into the buffer Another session holds the buffer in an incompatible mode to our request These waits indicate read/read, read/write, or write/write contention. The Oracle session is waiting to pin a buffer. A buffer must be pinned before it can be read or modified. Only one process can pin a buffer at any one time. This wait can be intensified by a large block size as more rows can be contained within the block This wait happens when a session wants to access a database block in the buffer cache but it cannot as the buffer is "busy It is also often due to several processes repeatedly reading the same blocks (eg: if lots of people scan the same index or data block)	The main way to reduce buffer busy waits is to reduce the total I/O on the system Depending on the block type, the actions will differ **Data Blocks** Eliminate HOT blocks from the application. Check for repeatedly scanned / unselective indexes. Try rebuilding the object with a higher PCTFREE so that you reduce the number of rows per block. Check for 'right- hand-indexes' (indexes that get inserted into at the same point by many processes). Increase INITRANS and MAXTRANS and reduce PCTUSED This will make the table less dense . Reduce the number of rows per block **Segment Header** Increase of number of FREELISTs and FREELIST GROUPs	A process that waits on the *buffer busy waits* event publishes the rea the P3 parameter of the wait event. The Oracle Metalink note # 34405.1 provides a table of reference - c 220 are the most common. Resolving intense and random buffer busy wait performance problen 155971.1

		Undo Header Increase the number of Rollback Segments	
free buffer waits	This means we are waiting for a free buffer but there are none available in the cache because there are too many dirty buffers in the cache Either the buffer cache is too small or the DBWR is slow in writing modified buffers to disk DBWR is unable to keep up to the write requests Checkpoints happening too fast – maybe due to high database activity and under-sized online redo log files Large sorts and full table scans are filling the cache with modified blocks faster than the DBWR is able to write to disk If the number of dirty buffers that need to be written to disk is larger than the number that DBWR can write per batch, then these waits can be observed	Reduce checkpoint frequency - increase the size of the online redo log files Examine the size of the buffer cache – consider increasing the size of the buffer cache in the SGA Set disk_asynch_io = true set If not using asynchronous I/O increase the number of db writer processes or dbwr slaves Ensure hot spots do not exist by spreading datafiles over disks and disk controllers Pre-sorting or reorganizing data can help	Understanding and Tuning Buffer Cache and DBWR Note# 62172.1 How to Identify a Hot Block within the database Buffer Cache. Note# 163424.1
enqueue waits	This wait event indicates a wait for a lock that is held by another session (or sessions) in an incompatible mode to the requested mode. **TX Transaction Lock** Generally due to table or application set up issues This indicates contention for row-level lock. This wait occurs when a transaction tries to update or delete rows that are currently locked by another transaction. This usually is an application issue. **TM DML enqueue lock** Generally due to application issues, particularly if foreign key constraints have not been indexed. **ST lock** Database actions that modify the UET$ (used extent) and FET$ (free extent) tables require the ST lock, which includes actions such as drop, truncate, and coalesce. Contention for the ST lock indicates there are multiple sessions actively performing dynamic disk space allocation or deallocation in dictionary managed tablespaces	Reduce waits and wait times The action to take depends on the lock type which is causing the most problems Whenever you see an *enqueue* wait event for the TX enqueue, the first step is to find out who the blocker is and if there are multiple waiters for the same resource Waits for TM enqueue in Mode 3 are primarily due to unindexed foreign key columns. Create indexes on foreign keys < 10g Following are some of the things you can do to minimize ST lock contention in your database: Use locally managed tablespaces Recreate all temporary tablespaces using the CREATE TEMPORARY TABLESPACE TEMPFILE… command.	Maximum number of enqueue resources that can be concurrently lo controlled by the ENQUEUE_RESOURCES parameter. Reference Note# 34566.1 Tracing sessions waiting on an enqueue Note# 102925.1 Details of V$LOCK view and lock modes Note:29787.1
Cache buffer chain latch	This latch is acquired when searching for data blocks Buffer cache is a chain of blocks and each chain is protected by a child latch when it needs to be scanned Hot blocks are another common cause of *cache buffers chains* latch contention. This happens when multiple sessions repeatedly access one or more blocks that are protected by the same child *cache buffers chains* latch. SQL statements with high BUFFER_GETS (logical reads) per EXECUTIONS are the main culprits Multiple concurrent sessions are executing the same inefficient SQL that is going after the same data set	Reducing contention for the cache buffer chains latch will usually require reducing logical I/O rates by tuning and minimizing the I/O requirements of the SQL involved. High I/O rates could be a sign of a hot block (meaning a block highly accessed). Exporting the table, increasing the PCTFREE significantly, and importing the data. This minimizes the number of rows per block, spreading them over many blocks. Of course, this is at the expense of storage and full table scans operations will be slower Minimizing the number of records per block in the table For indexes, you can rebuild them with higher PCTFREE values, bearing in mind that this may increase the height of the index. Consider reducing the block size Starting in Oracle9*i* Database, Oracle supports multiple block sizes. If the current block size is 16K, you may move the table or recreate the index in a tablespace with an 8K block size. This too will negatively impact full table scans operations. Also, various block sizes increase management complexity.	The default number of hash latches is usually 1024 The number of hash latches can be adjusted by the parameter _DB_BLOCKS_HASH_LATCHES What are latches and what causes latch contention
Cache buffer LRU chain latch	Processes need to get this latch when they need to move buffers based on the LRU block replacement policy in the buffer cache The *cache buffer lru chain* latch is acquired in order to introduce a new block into the buffer cache and when writing a buffer back to disk, specifically when trying to scan the LRU (least recently used) chain containing all the dirty blocks in the buffer cache. Competition for the *cache buffers lru chain* latch is symptomatic of intense buffer cache activity caused by inefficient SQL statements. Statements that repeatedly scan large unselective indexes or perform full table scans are the prime culprits. Heavy contention for this latch is generally due to heavy buffer cache activity which can be caused, for example, by: Repeatedly scanning large unselective indexes	Contention in this latch can be avoided implementing multiple buffer pools or increasing the number of LRU latches with the parameter DB_BLOCK_LRU_LATCHES (The default value is generally sufficient for most systems). Its possible to reduce contention for the *cache buffer lru chain* latch by increasing the size of the buffer cache and thereby reducing the rate at which new blocks are introduced into the buffer cache	

Direct Path Reads	These waits are associated with direct read operations which read data directly into the sessions PGA bypassing the SGA The "direct path read" and "direct path write" wait events are related to operations that are performed in PGA like sorting, group by operation, hash join In DSS type systems, or during heavy batch periods, waits on "direct path read" are quite normal However, for an OLTP system these waits are significant These wait events can occur during sorting operations which is not surprising as direct path reads and writes usually occur in connection with temporary tsegments SQL statements with functions that require sorts, such as ORDER BY, GROUP BY, UNION, DISTINCT, and ROLLUP, write sort runs to the temporary tablespace when the input size is larger than the work area in the PGA	Ensure the OS asynchronous IO is configured correctly. Check for IO heavy sessions / SQL and see if the amount of IO can be reduced. Ensure no disks are IO bound. Set your PGA_AGGREGATE_TARGET to appropriate value (if the parameter WORKAREA_SIZE_POLICY = AUTO) Or set *_area_size manually (like sort_area_size and then you have to set WORKAREA_SIZE_POLICY = MANUAL Whenever possible use UNION ALL instead of UNION, and where applicable use HASH JOIN instead of SORT MERGE and NESTED LOOPS instead of HASH JOIN. Make sure the optimizer selects the right driving table. Check to see if the composite index's columns can be rearranged to match the ORDER BY clause to avoid sort entirely. Also, consider automating the SQL work areas using PGA_AGGREGATE_TARGET in Oracle9*i* Database. Query V$SESSTAT> to identify sessions with high "physical reads direct"	Default size of HASH_AREA_SIZE is twice that of SORT_AREA_SI Larger HASH_AREA_SIZE will influence optimizer to go for hash joi nested loops Hidden parameter DB_FILE_DIRECT_IO_COUNT can impact the di performance.It sets the maximum I/O buffer size of direct read and w operations. Default is 1M in 9i How to identify resource intensive SQL statements?
Direct Path Writes	These are waits that are associated with direct write operations that write data from users' PGAs to data files or temporary tablespaces Direct load operations (eg: Create Table as Select (CTAS) may use this) Parallel DML operations Sort IO (when a sort does not fit in memory	If the file indicates a temporary tablespace check for unexpected disk sort operations. Ensure <Parameter:DISK_ASYNCH_IO> is TRUE . This is unlikely to reduce wait times from the wait event timings but may reduce sessions elapsed times (as synchronous direct IO is not accounted for in wait event timings). Ensure the OS asynchronous IO is configured correctly. Ensure no disks are IO bound	
Latch Free Waits	This wait indicates that the process is waiting for a latch that is currently busy (held by another process). When you see a *latch free* wait event in the V$SESSION_WAIT view, it means the process failed to obtain the latch in the willing-to-wait mode after spinning _SPIN_COUNT times and went to sleep. When processes compete heavily for latches, they will also consume more CPU resources because of spinning. The result is a higher response time	If the TIME spent waiting for latches is significant then it is best to determine which latches are suffering from contention.	A latch is a kind of low level lock. Latches apply only to memory structures in the SGA. They do not apply to database objects. An Oracle SGA has many latches, and they exist to protect various memory structures from potential corruption by concurrent access. The time spent on latch waits is an effect, not a cause; the cause is that you are doing too many block gets, and block gets require cache buffer chain latching What are Latches and what causes Latch contention Database Lock and Latch Information Knowledge Browser Product Page
Library cache latch	The *library cache latches* protect the cached SQL statements and objects definitions held in the library cache within the shared pool. The *library cache latch* must be acquired in order to add a new statement to the library cache Application is making heavy use of literal SQL- use of bind variables will reduce this latch considerably	Latch is to ensure that the application is reusing as much as possible SQL statement representation. Use bind variables whenever possible in the application You can reduce the *library cache* latch hold time by properly setting the SESSION_CACHED_CURSORS parameter Consider increasing shared pool	Larger *shared pools* tend to have long free lists and processes that need to allocate space in them must spend extra time scanning the long free lists while holding the *shared pool* latch if your database is not yet on Oracle9*i* Database, an oversized shared pool can increase the contention for the *shared pool* latch.
Shared pool latch	The *shared pool* latch is used to protect critical operations when allocating and freeing memory in the shared pool Contentions for the *shared pool* and *library cache* latches are mainly due to intense hard parsing. A hard parse applies to new cursors and cursors that are aged out and must be re-executed The cost of parsing a new SQL statement is expensive both in terms of CPU requirements and the number of times the library cache and *shared pool latches* may need to be acquired and released.	Ways to reduce the *shared pool latch* are, avoid hard parses when possible, parse once, execute many. Eliminating literal SQL is also useful to avoid the shared pool latch. The size of the shared_pool and use of MTS (shared server option) also greatly influences the shared pool latch. The workaround is to set the initialization parameter CURSOR_SHARING to FORCE. This allows statements that differ in literal values but are otherwise identical to share a cursor and therefore reduce latch contention, memory usage, and hard parse.	**<Note 62143.1>** explains how to identify and correct problems with the shared pool, and shared pool latch.
Row cache objects latch	This latch comes into play when user processes are attempting to access the cached data dictionary values.	It is not common to have contention in this latch and the only way to reduce contention for this latch is by increasing the size of the shared pool (SHARED_POOL_SIZE). Use Locally Managed tablespaces for your application objects especially indexes Review and amend your database logical design , a good example is to merge or decrease the number of indexes on tables with heavy inserts	Configuring the library cache to an acceptable size usually ensures that the data dictionary cache is also properly sized. So tuning Library Cache will tune Row Cache indirectly