Oracle Internals Tips and Technical Trips

Ch. V.N.Sanyasi Rao, Tiruveedula Gopi Krishna

Abstract—The Study or knowledge of internal structures of Oracle is necessary for an advanced performance tuning and trouble-shooting. To understand Oracle Optimizes these internal structures are useful. Oracle uses hidden parameters also. It is advisable to know them but not to play with them. Understanding online backup functionality the knowledge of internals helps a lot. Oracle documentation on tuning the database, a good reading helps to Improve Database Performance. This paper describes oracle internals mainly into the following aspects such as parsing oracle statements, locks contention internals, cursor sharing internals, soring operations Internals and shared pool internals.

Index Terms— oracle internals; parsing; lock contention; cursor sharing; shared pool.

I. INTRODUCTION

Why are people so intensely interested in Oracle internals? Partly because internals information can be useful for tuning and troubleshooting. But also because Oracle Corporation has kept most of the internals hidden secretly, while revealing just enough to tantalize. In fact, Oracle internals information is needed only for advanced performance tuning. It's true that basic application tuning is the kind of tuning that's most often needed, and the kind that has the biggest impact. Nevertheless, there are times when advanced performance tuning is necessary, and that is when you need a deep understanding of how Oracle works. This Paper provides some of the foundations for that understanding. Even though SQL statements and their compiled execution plans are just declarations of what work needs to be done, when Oracle gets to execution of a SQL statement, all processing is done procedurally. Every row-source, join method and access path is internally just a loop, continuing as long as there is data flowing from its child row sources. This session gives you in-depth understanding of how Oracle SQL plan execution physically works, a step beyond just interpreting the logical view of execution plan trees. This should give you good foundation for troubleshooting complex SQL performance issues and understanding things like in which cases one join method is better than another. After covering the internals, we will look into tools and techniques for profiling SQL execution plans, for finding out in which part of the execution plan most of the response time is spent and where optimizer has wrongly estimated the amount of rows to process. It is easy to write SQL that has to run inefficiently, and it is easy to write SQL that results in very unstable performance. In this paper I'll take an example of each type of problem and explain why sometimes you have to do a little extra work and introduce some extra complexity in the front-end code to allow the database to respond efficiently and consistently.

Manuscript Received on March 2015.

Ch.V. N. Sanyasi Rao, Assoc. Project Manager, HCL, Bangalore, India.
Tiruveedula Gopi Krishna, Sirt University Lecturer, Department of Computer Science, Hoon, Libya.

From this discussion you will learn to recognize SQL that is likely to run inefficiently, or vary dramatically in performance from day to day, or even minute to minute, and learn strategies for handling such code with a minimum of effort [1].

II. PARSING IN ORACLE

Whenever a statement is executed, Oracle follows a methodology to evaluate the statement in terms of syntax, validity of objects being referred and of course, privileges to the user. Apart from this, Oracle also checks for identical statements that may have been fired, with the intention of reducing processing overheads [2]. All this takes place in a fraction of a second, even less, without the user knowing what is happening to the statement that was fired. This process is known as Parsing. All statements, DDL or DML, are parsed whenever they are executed. The only key fact is that whether it was a Soft (statement is already parsed and available in memory) or a Hard (all parsing steps to be carried out) parse. Soft parse will considerably improve the system performance whereas frequent Hard parsing will affect the system. Reducing Hard parsing will improve the resource utilization and optimize the SQL code. Types of parsing as follows:

A. Parsing process:

- Syntactical check. The query fired is checked for its syntax.
- Semantic check. Checks on the validity of the objects being referred in the statement and the privileges available to the user firing the statement. This is a data dictionary check.
- Allocation of private SQL area in the memory for the statement.
- Generating a parsed representation of the statement and allocating Shared SQL area. This involves finding an optimal execution path for the statement.

In point four, Oracle first checks if the same statement is already parsed and existing in the memory. If found, the parsed representation will be picked up and the statement executed immediately (Soft parse). If not found, then the parsed representation is generated and stored in a shared SQL area (Part of shared pool memory in SGA), the statement is then executed (Hard parse). This step involves the optimization of the statement, the one that decides the performance [2].

B. Identical statements

Oracle does the following to find identical statements to decide on a soft or a hard parse.

- When a new statement is fired, a hash value is generated for the text string. Oracle checks if this new hash value matches with any existing hash value in the shared pool.

- Next, the text string of the new statement is compared with the hash value matching statements. This includes comparison of case, blanks and comments present in the statements.
- If a match is found, the objects referred in the new statement are compared with the matching statement objects. Tables of the same name belonging to different a schema will not account for a match.
- The bind variable types of the new statement should be of same type as the identified matching statement.
- If all of the above is satisfied, Oracle re-uses the existing parse (soft). If a match is not found, Oracle goes through the process of parsing the statement and putting it in the shared pool (hard).

C. Reduce hard parsing

The shared pool memory can be increased when contention occurs, but more important is that such issues should be addressed at the coding level. Following are some initiatives that can be taken to reduce hard parsing [2].

- Make use of bind variables rather than hard-coding values in your statements.
- Write generic routines that can be called from different places. This will also eliminate code repetition.
- Even with stringent checks, it may so happen that same statements are written in different formats. Search the SQL area periodically to check on similar queries that are being parsed separately. Change these statements to be look-alike or put them in a common routine so that a single parse can take care of all calls to the statement.

D. Identifying unnecessary parse calls at system level

```
select parse_calls, executions,
   substr(sql_text, 1, 300)
from v$sqlarea
where command_type in (2, 3, 6, 7);
```

Check for statements with a lot of executions. It is bad to have the PARSE_CALLS value in the above statement close to the EXECUTIONS value. The above query will fire only for DML statements (to check on other types of statements use the appropriate command type number). Also ignore Recursive calls (dictionary access), as it is internal to Oracle.

E. Identifying unnecessary parse calls at session level

```
select b.sid, a.name, b.value
from v$sesstat b, v$statname a
where a.name in ('parse count (hard)', 'execute count')
and b.statistic# = a.statistic#
Order by sid;
```

F. . Identify the sessions involved with a lot of re-parsing

(VALUE column). Query these sessions from V$SESSION and then locate the program that is being executed, resulting in so much parsing.

```
select a.parse_calls, a.executions, substr(a.sql_text, 1, 300)
from   v$sqlarea a, v$session b
where b.schema# = a.parsing_schema_id and   b.sid = <:sid>
order  by 1 desc;
```

The above query will also show recursive SQL being fired internally by Oracle.

4. Provide enough private SQL area to accommodate all of the SQL statements for a session. Depending on the requirement, the parameter OPEN_CURSORS may need to be reset to a higher value. Set the SESSION_CACHED_CURSORS to a higher value to allow more cursors to be cached at session level and to avoid re-parsing.

G. Identify how many cursors are being opened by sessions

```
select a.username, a.sid, b.value
from   v$session a, v$sesstat b, v$statname c
where  b.sid = a.sid
and    c.statistic# = b.statistic#
and    c.name = 'opened cursors current'
order  by 3 desc;
```

The VALUE column will identify how many cursors are open for a session and how near the count is to the OPEN_CURSORS parameter value. If the margin is very small, consider increasing the OPEN_CURSORS parameter.

H. Evaluate cached cursors for sessions as compared to parsing

```
select a.sid, a.value parse_cnt,
      (select x.value
       from   v$sesstat x, v$statname y
       where  x.sid = a.sid
       and    y.statistic# = x.statistic#
       and    y.name = 'session cursor cache hits') cache_cnt
from   v$sesstat a, v$statname b
where  b.statistic# = a.statistic#
and    b.name = 'parse count (total)'
and    value > 0;
```

The CACHE_CNT ('session cursor cache hits') of a session should be compared to the PARSE_CNT ('parse count (total)'), if the difference is high, consider increasing the SESSION_CACHED_CURSORS parameter.

The following parse related information is available in V$SYSSTAT and V$SESSTAT views, connect with V$STATNAME using STATISTIC# column.

```
SQL> select * from v$statname where name like '%parse%';

STATISTIC# NAME                      CLASS
---------- ------------------------- ----------
       217 parse time cpu             64
       218 parse time elapsed         64
       219 parse count (total)        64
       220 parse count (hard)         64
       221 parse count (failures)     64
```

5. Shared SQL area may be further utilized for not only identical but also for some-what similar queries by setting the initialization parameter CURSOR_SHARING to FORCE. The default value is EXACT. Do not use this parameter in Oracle 8i, as there is a bug involved with it that hangs similar query sessions because of some internal processing.

If you are on 9i, try out this parameter for your application in test mode before making changes in production. [2].

6. Prevent large SQL or PL/SQL areas from ageing out of the shared pool memory. Ageing out takes place based on Least recently used (LRU) mechanism. Set the parameter SHARED_POOL_RESERVED_SIZE to a larger value to prevent large packages from being aged out because of new entries. A large overhead is involved in reloading a large package that was aged out.

7. Pin frequent objects in memory using the DBMS_SHARED_POOL package. This package is created by default. It can also be created explicitly by running DBMSPOOL.SQL script; this internally calls PRVTPOOL.PLB script. Use it to pin most frequently used objects that should be in memory while the instance is up, these would include procedure (p), functions (p), packages (p) and triggers (r). Pin objects when the instance starts to avoid memory fragmentation (Even frequently used data can be pinned but this is a separate topic) [2].

To view a list of frequently used and re-loaded objects.

```
select loads, executions, substr(owner, 1, 15) "Owner",
      substr(namespace, 1, 20) "Type", substr(name, 1, 100)
"Text"
from v$db_object_cache
order by executions desc;
```

To pin a package in memory

```
SQL>exec dbms_shared_pool.keep('standard', 'p');
```

To view a list of pinned objects

```
select substr(owner, 1, 15) "Owner",
      substr(namespace, 1, 20) "Type",
      substr(name, 1, 100) "Text"
from v$db_object_cache
where kept = 'YES';
```

8. Increasing the shared pool size is an immediate solution, but the above steps need to be carried out to optimize the database in the long run. The size of the shared pool can be increased by setting the parameter SHARED_POOL_SIZE in the initialization file.

III. REDUCING LOCK CONTENTION

Lock, or enqueue waits occur when a session waits to obtain a lock. In most cases, this occurs because of a lock on a table or row that the session wants to lock or modify. In some circumstances, the lock involved may be an Oracle internal lock (for instance, the Space Transaction enqueue). If the database is well tuned and the application design sound, enqueue waits should be negligible.

Common causes of excessive enqueue waits are:

- Contention for a specific row in the database. The application design may require that many processes update or lock the same row in the database. One common example of this is when primary keys are generated using a sequence table.
- Table locks caused by foreign keys that have not been indexed. If a non-indexed foreign key is updated, then the parent table is subjected to a table lock until the transaction is complete.
- Old-style temporary tablespaces. If the tablespace named as the temporary tablespace has not been created with the TEMPORARY clause (introduced in Oracle 7.3), sessions may contend for the space transaction lock.
- The space reserved for transactions within a data block is too small. By default, only one transaction slot for tables or two for indexes is allocated when the table or index is created. The number of transaction slots is determined by the INITRANS clause in the CREATE TABLE or CREATE INDEX statement. If additional transaction slots are required, they are created, providing there is free space in the block. However, if all transaction slots are in use (and there is no free space in the block), a session that needs to lock a row in the block encounters an enqueue wait. This occurs even if the row in question is not actually locked by another process. This can occur if both PCTFREE and INITRANS were set too low [2].

IV. INTERNALS OF ORACLE SORT OPERATIONS

Oracle provides several SQL directives that cause sorting of row data.

- **Order by** - An operation that sorts a set of rows for a query with an order by clause
- **Join** - An operation that sorts a set of rows before a merge join
- **Group by** - An operation that sorts a set of rows into groups for a query with the group by clause
- **Aggregate** - A retrieval of a single row that is the result of applying a group function to a group of selected rows
- **Select unique** - An operation that sorts a set of rows to eliminate duplicates
- **Select distinct** - An operation that forces duplicate row to be eliminated from the result set and requires sorting to identify the duplicate rows
- **Create Index** - The creation of an index always invokes a sort of the symbolic keys and ROWID values.

Sorting is a frequently overlooked aspect of Oracle tuning. In general, an Oracle database will automatically perform sorting operations, and the SQL tuning expert must understand all of the SQL directives that invoke Oracle sorting [3].

Sorting involves the below points

1. Oracle Database always performs an automatic sorting operation when a GROUP BY clause is used in a SQL statement, when an index is created, and when an ORDER BY clause is used in a SQL statement.
2. The cheapest method is always used by Oracle Database when ordering a resultset.
3. A hinted execution plan involving a single table, with a /*+ index */ hint, will always retrieve the rows in the sorted order of the index.
4. If a SQL statement requires a single sort operation that completes in memory, t

hat SQL statement will not

use any space in the TEMP tablespace when the rows are retrieved – with the assumption that a hash join did not spill to disk.

5. The CPU_COST parameter causes Oracle Database to favor the pre-sorted ordering of an index over a discrete sorting operation.

6. The value of the SORT_AREA_SIZE parameter or the PGA_AGGREGATE_TARGET parameter if used influences Oracle Database's decision to prefer the pre-sorted ordering of an index over a discrete sorting operation.

7. The clustering factor of an index influences Oracle Database's decision to prefer the pre-sorted ordering of an index over a discrete sorting operation.

8. The default database block size in use by the database influences Oracle Database's decision to prefer the pre-sorted ordering of an index over a discrete sorting operation.

9. A sort operation will only spill to disk when RAM is exhausted.

10. "At the time a session is established with Oracle, a private sort area is allocated in memory for use by the session for sorting, based on the value of the sort_area_size initialization parameter." Supporting evidence:

11. For sort intensive tasks it is not possible to adjust the amount of memory allocated to those tasks by adjusting the SORT_AREA_SIZE parameter at the session level.

12. The entire database can be slowed down due to a disk sort in the TEMP tablespace because sorts to disk are I/O intensive.

13. A good general rule is that the SORT_AREA_SIZE parameter should be adjusted to eliminate sorts to disk caused by GROUP BY operations.

14. Buffer pool blocks are allocated to hold or manage the blocks that are in the TEMP tablespace.

15. An optimal workarea execution, completed entirely in memory, is always preferred over a one-pass or multi-pass workarea execution.

16. Free buffer waits can be caused by excessive sorts to disk, which cause data blocks needed by other sessions to be paged out of the buffer.

17. One percent is an acceptable ratio of disk sorts to the total number of sorts.

18. When the PGA_AGGREGATE_TARGET parameter is specified, the total work area size cannot exceed 200MB.

19. No task may use more than 10MB for sorting.

20. A DBA should modify two hidden (underscore) parameters to permit up to 50MB of memory to be used for an in-memory sort operation for a SQL statement

V. CURSOR_SHARING AND BIND SHARING INTERNALS

The CURSOR_SHARING parameter basically influences the extent to which SQL statements (or cursors) can be shared.

The possible values are EXACT which is the default value and SIMILAR and FORCE.

The official definitions of SIMILAR and FORCE are:

cursor_sharing=similar: "Causes statements that may differ in some literals, but are otherwise identical, to share a cursor, unless the literals affect either the meaning of the statement or the degree to which the plan is optimized." [3].

cursor_sharing=FORCE: "Forces statements that may differ in some literals, but are otherwise identical, to share a cursor, unless the literals affect the meaning of the statement."

So basically it means that the SQL statement will be shared even though literals are different as long as it does not affect the execution plan of the statement. This is the case for SIMILAR setting.

The cursor_sharing=SIMILAR parameter has been deprecated in 11gR2 because it has been found that the use of this parameter could potentially have a lot of performance implications related to the number of child cursors created for the single parent.

In versions prior to 11g R2, there was a limit on the number of child cursors which can be associated with a single parent. It was 1024 and once this number was crossed the parent was marked obsolete and this invalidated it was well as all associated child cursors.

But not having this upper limit was being found to have caused a lot of CPU usage and waits on mutexes and library cache locks in 11gR2 caused by searching the library cache for matching cursors and it was felt that having many child cursors all associated with one parent cursor could perform much worse than having many parent cursors that would be seen with having the default setting of cursor_sharing = EXACT

Using this parameter also was bypassing one of the big improvements made in 11g which was Adaptive Cursor Sharing.

Let us examine the behaviour with CURSOR_SHARING set to SIMILAR as opposed to FORCE and note the differences between the two.

```
SQL> alter system set cursor_sharing='SIMILAR';

System altered.
```

VI. INTERNAL WORKING OF SHARED POOL

As we have seen from above topics, we now know that shared pool is a collection of objects like tables, cursors, views, procedures, functions or packages (Pl/SQL package).

Size of single object will be small but when you have a PL/SQL package, the size would grow in MB as it includes many objects and its attributes. Example: A package body will have many procedures which in turn will occupy more space when we run a simple SQL statement.

Each shared pool object is not a single allocation unit (AU) and it is partitioned into independent memory allocations called "HEAPS". Number of Heaps for the object varies depends upon the type of objects. Eg: for a SQL cursor, there will be 2 heaps a) Smaller Heap for library cache metadata and b) Larger Heap for executable representation of the cursor.

Each HEAP is comprised of 1 or more chunks of memory of Standard Allocation Units (AU) and these reduce the problem of memory fragmentation [3,4].

When memory is allocated for objects, the memory is not allocated in contiguous fashion. i.e., the chunks will not be contiguous as you can see from the figure.

But the memory (free lists – Pointers to memory chunks) must be contiguous.

Eg. Consider, we need 5KB size of Heap memory and each chunk is 4K and we need 1K to complete memory allocation, so the remaining memory is allocated to the heap from another memory chunk and hence the memory chunks are not contiguous.

When the chunks are not contiguous and if they work this way, it will avoid fragmentation.

Generally, the chunk sizes are of 1k and 4K creating more uniform memory allocations. So, when same objects are allocated and aged out, same size of chunks are allocated and aged out. By doing so, we can avoid memory fragmentation and memory usage effectively.

VII. LARGE POOL

RMAN, parallel query and shared servers uses allocation of more than 5KB and they need large allocation of memory. Hence the use other memory pool called LARGE POOL.

VIII. RESERVED POOL

1) Sometimes SQL, PL/SQL packages, if they are over 5KB, they might require larger contiguous chunks of memory. Default settings of reserved pool is 4,400 bytes
2) If there is not enough free space in the shared pool, then oracle must search for free enough memory to satisfy the request. Oracle might even have to age out the old objects to satisfy it. In such cases, oracle will hold the latch resource for a period until it tries to find the memory. Till then, it will cause a minor disruption to the other requests (concurrent request) at memory allocation.
3) So, what oracle does is, it internally configures a small space called RESERVERED POOL so that when we have operations involving PL/SQL and trigger compilations or to load any java objects, this will be used. When the operation is over, the memory freed will be returned back to reserved pool [4].
4) When ORA-4031 error occurs.
5) ORA-4031 occurs in any of the memory pools in the SGA when oracle cannot find a memory chunk large enough to satisfy the internal allocation request on behalf of users operation.
   1. A user executes a query.
   2. An object needs to be allocated in shared pool.
   3. The memory allocated for chunks to be contiguous and this will be allocated to the objects.
   4. If it is small memory size, then the Heap manager scans through freelists and if it is small, it will allocated.
6) If the requested amount of contiguous memory is not available for chunks, then Heap Manager iterates through the shared pool's LRU list and it attempts to create a contiguous chunk of requested size (this is done by aging out the LRU objects).
7) But, ORA-4031 error occurs even if the large/free space created by the heap manager is.

IX. NOT CONTIGUOUS

In case if ASMM/AMM is enabled, an additional granule of memory is requested if sufficient contiguous memory can't be found.

X. CONCLUSION

Understanding of how Oracle works will be deeepened with clear explanations of many of Oracle's international data structures and algorithms. You will be altered to potential performance problems that are not mentioned in the documentation and you will expand your repertoire of tuning solutions and troubleshooting techniques by learning how to use numerous hidden parameters and other undocumented features. Minimizing parsing is critically important both to reduce the CPU overhead of parsing, and to reduce the latch/mutex contention that usually accompanies excessive hard parse rates. Using bind variables in application code is the best way to achieve this, though the CURSOR_SHARING parameter can also be very effective. This discussion has new diagnostics to help you identify and correct bottlenecks caused by excessive parse rates.

ACKNOWLEDGMENT

We would like to thank for our organization to help their technical support and facilitate lab frequently.

REFERENCES

1. http://www.articles.freemegazone.com/oracle-sorting.php
2. oracle-training.cc/oracle_tips_sort_operations.htm
3. http://books.google.com/books?id=gsFC1D1LmvQC&pg=PA306&lpg=PA306#v=onepage&q=&f=false
4. http://www.oracle-training.cc/oracle_tips_sort_operations.htm.
5. tech.e2sn.com/virtual.../systematic-oracle-sql-optimization-in-real-life

AUTHORS PROFILE

CH. V. N. Sanyasirao, Received B.Sc, M.Sc, Mtech, Mphil, from Andhra University,Panjabi University ,Vinayaka mission and in (1997,1999) from India respectively. Currently pursuing Ph.D through Rayalaseema University, India, Registered 2010, and Research interests in Specto Temporal Databases. Currently working as a Associate Project Manager in HCL Technologies 2010.And Worked With huge volumes of databases and Involved into development of many products like Oracle flexcube .Published research paper in www.ijcset.net.

Tiruveedula Gopi Krishna, Received B.Sc, M.Sc, M.E, from Andhra University,Anna University and in (1997,2001,2004,2010) from India respectively. Currently pursuing Ph.D through Rayalaseema University, India, Registered 2010, and Research interests in Data Mining. Currently working as a faculty of computer science for Sirt University, Libya since 2007.And several Research Papers published in various International Reputed high cited impact factor Journals and national journal papers published Computer Science and Engineering. Presented many research papers and participated in international conferences.