DEVELOPER & MONITORING TIPS

INTRODUCTION

Working with designers for many years, there appears to be a recurring pattern of oversights that tend to worsen performance. To try to head-off problems, we have identified some simple practices.

In addition, we have included some scripts to use in monitoring stuck sessions, to try to see why the session is "stuck."

DON'T BEGIN WITH PARALLELISM

This may be the biggest problem we encounter. Many queries submitted for performance analysis have Parallel hints, but the efficiency of the Sql has been neglected. This is understandable, because it's so easy to add a Parallel hint, but much more difficult to make the Sql efficient. Invoking parallelism disguises the inefficiency sometimes, since a ton of parallel processes can often (if only temporarily) cover-up the problem with the Sql.

USE MODERATE PARALLELISM

Oracle can perform a full scan of a typical billion-row table in about 7 minutes using Parallel 6. So, limiting the parallelism to 6 or 8 is a good rule of thumb, except for extreme cases (such as rebuilding a monstrous index when you have the entire system to yourself.) It's true that degrees beyond 6 or 8 provide more improvement, but the benefit begin to fall-off, as you consume more resources. Then, those resources are not available to other jobs. If you do use parallelism, be sure to specify the degree; do not let Oracle decide how many processes to invoke--it could be way more than you expect.

DON'T BEGIN WITH SQL HINTS

It's tempting to start throwing Sql hints into the code, but if you're not sure about the hints, this will confuse the DBA, or future developers. It is very common for the DBA to have to remove Sql hints that don't do anything (or do the wrong thing.)

CONSIDER QUERY SUB-FACTORING

This is the "With" syntax. Using this feature makes your code much easier to follow, and much easier to debug--both functionality-wise and performance-wise. Use of a lot of inline-views can make the code nearly impossible to understand. Here's what it looks like--note that each "view" is enclosed within parentheses:

```
With View1 as (Select * from Table1 Where Col1 = 'ABC'),
     View2 as (Select * from Table2 Where Col1 = 'DEF')
Select Count(*) from View1, View2 Where View1.Col1 = View2.Col1;
```

CONSIDER MAINTAINABILITY

There is not any just one thing to consider when thinking of maintainability. This is a complex subject. Keep in mind, thought that being functionally correct is not good enough. Think of how the next developer can modify your code, if the need arises. If there are lots of inline-views, how will someone else know what to do? If your Sql is hundreds of lines long, how can anyone ever understand it?

DEVELOPER & MONITORING TIPS

USE BINDS

When developers use bind variables, it makes life a lot easier for the DBA. Here's why: When a Sql uses bind variables (and not "literals), then the "Sql_id" is the same. So the DBA can compare performance for months back. If there are literals in the Sql, then it's technically a "new" Sql every time. Comparison just got a lot more difficult. Additionally, when literals are in the Sql, the DBA cannot use a "Stored Outline," which is a very powerful tool for fixing performance.

AVOID LOOPING SQL

It's tempting to divide a complicated problem up into little pieces, but that's oftentimes not the best approach. Sending lots of Sql commands to the database usually slows down a job a LOT, because there is a minimum time (perhaps 50 ms) for return trip. Not a big deal, of course, if you're only looping a few thousand times.

CONSIDER SET FUNCTIONS

It's very common to see the operation, "Not In." If you are just checking a small result set, that is fine; however, for larger result sets, it's far better to use the set operation, "MINUS." For example, to find the rows with names in one table, but not in another:

```
Select Name from Table1
MINUS
Select Name from TAble2;
```

Similarly, one can use INTERSECT to find the rows in common.

ENABLE PARTITIONING

Partitioning can be really tricky, and many times it actually makes performance *worse*. To get a gain from performance, the partition divisions in the table must match how the data is queried. So, if you have a report that extracts (or summarizes) all data in a month, then partitioning a table by month is a good idea. Keep in mind that the "Where Clause" needs to reference the Partition_Key. If it doesn't, Oracle will have no choice but to scan *all* the partitions.

REALIZE ADVANTAGE OF FULL TABLE SCANS

Oracle is capable of scanning hundreds of blocks at one time. This is what happens when the optimizer decides to do a full table (or index) scan. That's why it's always better to shift to full-table scans if the result set is large--say over 100,000 rows. It's very common to have to use a sql hint to change the optimizer plan to a full scan. For example,

```
Select /*+FULL(A) */  * from Tablea A;
```

SEE WHAT'S ACTIVE

Is useful to see all the sessions on the entire RAC cluster, and how long their sql has been running. Here's one way to do that:

```
Select inst_id,sid, username,(sysdate-sql_exec_start)*24*60*60 SECS,
logon_time,sql_id
```

```
from gv$session
where type != 'BACKGROUND'  and event not like 'SQL*Net%' and
      event not like 'Streams AQ: waiting%'
and status = 'ACTIVE'
order by username;
```

```
INST_ID    SID USERNAME            SECS LOGON_TIM SQL_ID
-------- ------ ------------ ---------- --------- -------------
       6   2061 CCBPEMAPP                26-SEP-14
       2   3897 CCBPONLINE              1 26-SEP-14 c7trsz950gsnk
       1   5229 CCBPONLINE              0 26-SEP-14 c7trsz950gsnk
       1   4660 CCBPONLINE              0 26-SEP-14 4pf27xnwgvwnx
       1   3708 CCBPONLINE              0 26-SEP-14 gqrd5zqu5zxke
       1    289 CCBPONLINE              0 26-SEP-14 b1n4vyks7fcsc
       1   5702 CCBPONLINE              0 26-SEP-14 d8st16huy2g7x
       8    154 DA                 242493 23-SEP-14 gwtr7ssqk54ky
       1   4944 SYS                   138 26-SEP-14 38zftsxvxr3d6
       3   2570 SYS                   138 26-SEP-14 38zftsxvxr3d6
       7   2666 SYS                   138 26-SEP-14 38zftsxvxr3d4
       5   2700 SYS                   139 26-SEP-14 38zftsxvxr3d3
```

SEE WHAT'S BEEN RUNNING A LONG TIME

This is similar to the prior script, but in this one, we show the sql for long running sessions. Just change the threshold, as shown below:

```
COL FORMAT A12
SET LINESIZE 200
COL INST FORMAT 99
COL SQL FORMAT A40

Select username, s.inst_id INST,module, sid, s.sql_id, (sysdate-
sql_exec_start)*24*60*60 SECS,
substr(sql_text,1,40) sql
from gv$session s, gv$sqltext t
where s.sql_id = t.sql_id
and s.inst_id = t.inst_id
and piece = 0
and (sysdate-sql_exec_start)*24*60*60 > 99  --LOOK FOR 99 SECONDS
and module not like 'SQL D%'
and module not like 'TOAD%'
and sql_text not like '%collect_session%'
and sql_text not like '%TOTAL_WAITS%';
```

```
USERNAME      INST MODULE          SID SQL_ID              SECS SQL
------------ ---- ------------ ------ ------------- ---------- ----------------
CCBPONLINE       2 CMLPAPHL       4658 207a0fvh77w2w        105 SELECT A.ACCT_ID
CCBPONLINE       2 CMLPAPHL       3897 207a0fvh77w2w        184 SELECT A.ACCT_ID
CCBPONLINE       2 CMLPAPHL       2093 207a0fvh77w2w        166 SELECT A.ACCT_ID
```

DEVELOPER & MONITORING TIPS

SEE IF A SESSION IS "READING UNDO"

Oftentimes, a session is stuck reading undo. That happens when a long-running query is running at the same time that another session is changing the contents of the needed tables. For sessions stuck on undo, the job will usually have to be stopped and restarted.

Once you know the SID, here's a way to see what a session is waiting on:

```
select Sid, event, P1, seconds_in_wait SEC
from V$session_wait
where sid = [SID]
and wait_time = 0;
```

For sessions reading undo, the event will be "sequential file read." When reading from disk, the value in "P1" is the File_id of the Oracle .dbf file. For sessions which are stuck on undo, P1 will identify an undo-tablespace file. The file number might change, but it constantly points to an undo file.

Here's how you get the filename once you have a P1 value:

```
Select File_name from Dba_Data_files where File_id = [P1];
```

SEE HISTORICAL RUNTIME OF A CERTAIN SQL

If you know the exact sql_id of a problem sql, it's easy to see how it's been running over time. Here's an example:

```
Select snap_id, instance_number INST,
Executions_delta EXEC , rows_processed_delta NUMROWS,
Round(elapsed_time_delta/1000/(executions_delta+.01),1) timepermsec
From dba_hist_sqlstat s
where sql_id = 'TBD'
Order by 1;
```

SNAP_ID	INST	EXEC	NUMROWS	TIMEPERMSEC
64606	7	23816	23780	4039.7
64606	5	23378	23327	3985.2
64606	4	22065	22020	4103.6
64606	3	23832	23777	3781.1
64606	6	23011	22957	3968.1

You can also add other fields, such as, *Plan_Hash_Value* or *Disk_Reads* if you want to see if the execution plan has changed over time.