Oracle Performance Diagnostic Guide
Slow Database

Version 3.1.0 January 13, 2009

Welcome to the Oracle Performance Diagnostic Guide This guide is intended to help you resolve query tuning, hang/locking, and slow database issues. The guide is not an automated tool but rather seeks to show methodologies, techniques, common causes, and solutions to performance problems.

Most of the guide is finished but portions of the content under the Hang/Locking tab is still under development.
Your feedback is very valuable to us - please email your comments to: Vickie.Carbonneau@oracle.com

Contents

Slow Database > Identify the Issue > Overview
- Recognize a Slow Database Issue
- Clarify the Issue
- Verify the Issue
- Next Step - Data Collection

Slow Database > Identify the Issue > Data Collection
- Gather Database Performance Data
- Gather Operating System (OS) Performance Data
- Gather RDA Report
- Gather Application Logs (Optional)
- Next Step - Analyze

Slow Database > Identify the Issue > Analysis
- Verify Oracle OS Resource Usage
- Verify The Database is Slow
- Next Step - Determine a Cause
- Would You Like to Stop and Log a Service Request?

Slow Database > Determine a Cause >Overview

Slow Database > Determine a Cause >Data Collection
- Gather an Extended SQL Trace
- Next Step - Analyze

Slow Database > Determine a Cause >Analysis
- Determine the Type of Performance Problem
- Choose a Tuning Strategy
- Open a Service Request with Oracle Support Services
- Give Us Your Feedback

Slow Database > Reference

Causes and Solutions
Tools

Feedback

We look forward to your feedback. Please email any comments, suggestion to help improve this guide, or any issues that you have encountered with the tool usage to Vickie.Carbonneau@oracle.com, Technical Advisor, Center of Expertise (CoE).

Slow Database > Identify the Issue > Overview

To properly identify the issue we want to resolve, we must do three things:

- **Recognize** the issue
- **Clarify** the details surrounding the issue
- **Verify** that the issue is indeed the problem
  - This will be done in the *Data Collection* and *Analysis* steps that follow

Recognize a Slow Database Issue

What is a slow database issue?

A slow database issue can manifest itself as:

- A large number of sessions that run slower than usual
- The database permits logons and seems to be working (not hung) but takes much longer than usual to show results
- Many different types of activity all slow down at around the same time

You might have identified this behavior from:

- benchmarking/testing
- user complaints
- statspack or AWR reports showing less throughput (e.g., transactions/sec)
- statspack, AWR, or ASH reports showing much higher wait and/or CPU times than normal
- OS data that shows more CPU consumption or I/O by Oracle processes than is normal

These problems might appear after:

Documentation
- Performance Tuning Overview

Notes
- Database Performance FAQ

White Papers
- Yet Another Performance Profiling Method
- The COE Performance Method

Case Studies

- schema changes
- changes in statistics gathering
- changes in data volumes
- changes in application
- database upgrades

- Resolving High CPU usage in Oracle Servers
- The Mysterious Performance Drop
- Intense and Random Buffer Busy Wait Performance
- Diagnosing Another Buffer Busy Waits Issue
- Using Real-Time Diagnostic Tools to Diagnose Intermittent Database Hangs

Clarify the Issue

A clear problem statement is critical. You need to be clear on exactly what the problem is. It may be that in subsequent phases of working through the issue, the real problem becomes clearer and you have to revisit and re-clarify the issue.

Notes

- ODM Reference: Identify the Issue

To clarify the issue, you must know as much as possible of the following:

- When the system was slow and when it was OK.
- Any related changes that coincide with the bad performance
- The sequence of events leading up to the problem
- Where/how was it noticed
- The significance of the problem
- What IS working
- What is the expected or acceptable result?
- What have you done to try to resolve the problem

As an example:

- A system performs poorly every morning between 10am and 12pm; it is OK at all other times.
- The problem occurred after the latest application version was installed
- It was noticed by end users.
- It is making the application run slowly and preventing our system from taking orders.
- System performs well except between 10am and 12pm
- Orders are typically processed by the database in 200 mSec; during problem, they take 10 seconds
- We tried re-gathering stats, but it did not make any difference.

Why is this step mandatory?
Skipping this step will be risky because you might attack the wrong problem and waste significant time and effort.
A clear problem statement is critical to begin finding the cause and solution to the problem.

Verify the Issue

Our objective in this step of the diagnostic process is to ensure the database shows symptoms of a performance problem. At this point, you need to collect data that verifies the existence of a problem.

Notes

- ODM Reference: Identify the Issue

To verify the existence of the issue you must collect :

- systemwide evidence using statspack, AWR, and/or ASH report when performance was *good* and bad
- specific evidence of the poor performance for a session or several queries
- extended SQL trace of one or more sessions during periods of good and bad performance

The data above should also be collected during good performance to demonstrate the impact of the problem

Further examples and advice on what diagnostic information will be needed to resolve the problem will be discussed in the DATA COLLECTION section.

Once the data is collected, you will review it to either verify there is a slow database issue, or decide it is a different issue.

Why is this step mandatory?
If you skip this step, you might assume the database is the problem, but the problem may actually reside in the client or network. The effort involved in tuning the database would be wasted in this case.

Next Step - Data Collection

When you have done the above, click "NEXT" to get some guidance on collecting data that will help to validate that you are looking at the right problem and that will help in diagnosing the cause of the problem.

Slow Database > Identify the Issue > Data Collection

In this step, we will collect data to verify the performance problem is due to the database and not external to it.
Note: Collect data when performance is good as well as when it was bad.

Gather Database Performance Data

Always collect instance-wide performance data, and in addition, collect extended SQL trace data if only certain sessions are slow but many others don't have a performance problem.

Be sure to gather data when performance was good AND bad so the traces can be compared.

Is the Entire Database Slow?

If the entire database seems slow, then instance-wide performance data is very useful for getting an overall context of performance in the database. It can give a distorted picture if the database workload is very mixed (e.g., lots of OLTP and decision support SQL at the same time), but in general it is helpful to have this data.

Depending on your database version, the data may be collected as follows:

- 10.2.x or higher, all of the following are preferred:
  - Active Session History (ASH) reports
  - Automatic Diagnostic Database Monitor (ADDM) reports
  - Statspack reports
- 10.1.x or higher, all of the following are preferred:
  - ADDM reports
  - Statspack reports
- 8.1.6 - 9.2.x: Statspack reports

For accuracy, the time interval should be as short as possible to capture the problem (around 10 - 15 minutes is ideal).

In 10g or higher versions of the database, the AWR and statspack reports (statspack is preferred) include OS data. This means you can avoid more in-depth OS data collection **if you collect the AWR or statspack snapshots during the performance problem**.

See the document references on this page for details on obtaining ASH / AWR / statspack reports.

Are Just Some Sessions Slow?

Documentation

- Understanding SQL Trace and TKProf
- Automatic Workload Repository
- Automatic Workload Repository Reports
- Active Session History Reports

Reference

- Statspack Complete Reference
- Oracle Diagnostic Pack Licensing Information

How-To

If only certain sessions are slow, we'll need to focus on those sessions using an **extended SQL trace (Event 10046, level 12)**. SQL trace data is extremely useful for diagnosing performance problems because we'll know exactly which SQL statements are most impacted and how they're impacted (CPU, wait, or idle time). The key is getting the trace from the most important and most impacted sessions as completely as possible.

The following process will help you collect SQL trace data properly:

1. ***Choose a session to trace***

Target the most important / impacted sessions

- ***Users that are experiencing the problem most severely; e.g., normally the transaction is complete in 1 sec, but now it takes 30 sec.***
- ***Users that are aggressively accumulating time in the database***

The following queries will allow you to find the sessions currently logged into the database that have accumulated the most time on CPU or for certain wait events. Use them to identify potential sessions to trace using 10046.

These queries are filtering the sessions based on logon times less than 4 hours and the last call occurring within 30 minutes. This is to find more currently relevant sessions instead of long running ones that accumulate a lot of time but aren't having a performance problem. You may need to adjust these values to suit your environment.

Find Sessions with the Highest CPU Consumption

```
-- sessions with highest CPU consumption
SELECT s.sid, s.serial#, p.spid as "OS PID",s.username, s.module, st.value/100 as "CPU sec"
FROM v$sesstat st, v$statname sn, v$session s, v$process p
WHERE sn.name = 'CPU used by this session' -- CPU
AND st.statistic# = sn.statistic#
AND st.sid = s.sid
AND s.paddr = p.addr
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
ORDER BY st.value;

       SID    SERIAL# OS PID       USERNAME
MODULE                                           CPU sec
---------- ---------- ------------ --------------------
---------------------------------------- ----------
       141       1125 15315        SYS                 sqlplus@coehq2 (TNS V1-
V3)                     8.25
       147        575 10577        SCOTT
SQL*Plus                                          258.08
       131        696 10578        SCOTT
SQL*Plus                                          263.17
       139        218 10576        SCOTT
SQL*Plus                                          264.08
```

- How To Collect 10046 Trace Data
- Recommended Method for Obtaining 10046 trace for Tuning
- How To Generate TKProf reports
- OTN: Oracle By Example: Monitoring and Tuning the Database

Scripts and Tools

- Collect 10046 Traces Automatically with LTOM

```
       133        354 10583        SCOTT
SQL*Plus                                    265.79
       135        277 10586        SCOTT
SQL*Plus                                    268.02
```

Find Sessions with Highest Waits of a Certain Type

```
-- sessions with the highest time for a certain wait
SELECT s.sid, s.serial#, p.spid as "OS PID", s.username, s.module, se.time_waited
FROM v$session_event se, v$session s, v$process p
WHERE se.event = '&event_name'
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
AND se.sid = s.sid
AND s.paddr = p.addr
ORDER BY se.time_waited;

SQL> /
Enter value for event_name: db file sequential read

       SID    SERIAL# OS PID       USERNAME
MODULE                                    TIME_WAITED
---------- ---------- ------------ --------------------
----------------------------------------- -----------
       141       1125 15315        SYS                  sqlplus@coehq2 (TNS V1-
V3)                          4
       147        575 10577        SCOTT
SQL*Plus                                       45215
       131        696 10578        SCOTT
SQL*Plus                                       45529
       135        277 10586        SCOTT
SQL*Plus                                       50288
       139        218 10576        SCOTT
SQL*Plus                                       51331
       133        354 10583        SCOTT
SQL*Plus                                       51428
```

10g or higher: Find Sessions with the Highest DB Time

```
```

```
-- sessions with highest DB Time usage
SELECT s.sid, s.serial#, p.spid as "OS PID", s.username, s.module, st.value/100 as "DB Time
(sec)"
, stcpu.value/100 as "CPU Time (sec)", round(stcpu.value / st.value * 100,2) as "% CPU"
FROM v$sesstat st, v$statname sn, v$session s, v$sesstat stcpu, v$statname sncpu, v$process p
WHERE sn.name = 'DB time' -- CPU
AND st.statistic# = sn.statistic#
AND st.sid = s.sid
AND  sncpu.name = 'CPU used by this session' -- CPU
AND stcpu.statistic# = sncpu.statistic#
AND stcpu.sid = st.sid
AND s.paddr = p.addr
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
AND st.value > 0;

       SID    SERIAL# OS PID       USERNAME MODULE                                   DB Time
(sec) CPU Time (sec)      % CPU
---------- ---------- ------------ -------- ----------------------------------------
------------- -------------- ----------
       141       1125 15315        SYS      sqlplus@coehq2 (TNS V1-V3)
12.92           9.34      72.29


Note: sometimes DB Time can be lower than CPU Time when a session issues long-running
recursive calls. The DB Time statistic doesn't update until the top-level call is finished
(versus the CPU statistic that updates as each call completes).
```

Obtain a complete trace

- ***Ideally, start the trace as soon as the user logs on and begins the operation or transaction. Continue tracing until the operation is finished.***
- ***Try to avoid starting or ending the trace in the middle of a call unless you know the call is not important to the solution***

2. ***Collect the trace and generate a TKProf report***

See the document references on this page for details on obtaining extended SQL trace data. Read Recommended Method for Obtaining 10046 trace for Tuning first

- ***Trace a Connected Session***
  This is the most common way to get a trace file.
  - ***Start tracing on a connected session***
  - ***Coordinate with the user to start the operation***
  - ***Let the trace collect while the operation is in progress***
  - ***Stop tracing when the operation is done***
  - ***Gather the trace file from the "user_dump_dest" location (you can usually identify the file just by looking at the timestamp).***

- ***Alternative: Trace Using a Test Script***
  Sometimes you may be able to script a reproducible test case.
  - ***Put ALTER SESSION commands to start / stop the tracing in the test script***
  - ***Run the test script and collect the trace file from the "user_dump_dest" location (you can usually identify the file just by looking at the timestamp).***

- ***Other Considerations***
  - ***Shared Servers: Tracing shared servers could cause many separate trace files to be produced as the session moves to different Oracle processes during execution. Use the 10g utility, "trcsess" to combine these separate files into one.***

- ***Generate a TKProf report and sort the SQL statements in order of most elapsed time using the following command:***

  tkprof <trace file name> <output file name> sort=fchela,exeela,prsela

3. ***Make sure trace file contains only data from the recent test***

If this session has been traced recently, there may be other traces mixed in the file with the recent trace collected

- ***We should extract only the trace data that is part of the recent tests. See the place in the sample trace below where it says "Cut away lines above this point".***

Trace file from a long running process that has been traced intermittently over several days

```
. . .
*** 2006-07-24 13:35:05.642 <== Timestamp from a previous tracing
WAIT #8: nam='SQL*Net message from client' ela= 20479935 p1=1650815232 p2=1 p3=0
=====================
PARSING IN CURSOR #9 len=43 dep=0 uid=57 oct=3 lid=57 tim=1007742062095 hv=4018512766 ad='97039a58'
select e.empno, d.deptno<== Previous cursor that was traced
from emp e, dept d
END OF STMT
PARSE #9:c=630000,e=864645,p=10,cr=174,cu=0,mis=1,r=0,dep=0,og=4,tim=1007742062058
BINDS #9:
EXEC #9:c=0,e=329,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1007742062997
WAIT #9: nam='SQL*Net message to client' ela= 18 p1=1650815232 p2=1 p3=0
. . .
FETCH #9:c=10000,e=513,p=0,cr=1,cu=0,mis=0,r=15,dep=0,og=4,tim=1007742148898
WAIT #9: nam='SQL*Net message from client' ela= 2450 p1=1650815232 p2=1 p3=0
WAIT #9: nam='SQL*Net message to client' ela= 7 p1=1650815232 p2=1 p3=0
FETCH #9:c=0,e=233,p=0,cr=0,cu=0,mis=0,r=10,dep=0,og=4,tim=1007742152065
. . .

====> CUT AWAY LINES ABOVE THIS POINT - THEY AREN'T PART OF THIS TEST <====
*** 2006-07-24 18:35:48.850
<== Timestamp for the tracing we want (notice its about 5 hours later)


=====================
PARSING IN CURSOR #10 len=69 dep=0 uid=57 oct=42 lid=57 tim=1007783391548 hv=3164292706 ad='9915de10'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
. . .
=====================
PARSING IN CURSOR #3 len=68 dep=0 uid=57 oct=3 lid=57 tim=1007831212596 hv=1036028368 ad='9306bee0'
select e.empno, d.dname<== Cursor that was traced
from emp e, dept d
where e.deptno = d.deptno
END OF STMT
PARSE #3:c=20000,e=17200,p=0,cr=6,cu=0,mis=1,r=0,dep=0,og=4,tim=1007831212566
BINDS #3:
EXEC #3:c=0,e=321,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1007831213512
WAIT #3: nam='SQL*Net message to client' ela= 15 p1=1650815232 p2=1 p3=0
WAIT #3: nam='db file sequential read' ela= 7126 p1=4 p2=11 p3=1
. . .
FETCH #3:c=10000,e=39451,p=12,cr=14,cu=0,mis=0,r=1,dep=0,og=4,tim=1007831253359
WAIT #3: nam='SQL*Net message from client' ela= 2009 p1=1650815232 p2=1 p3=0
WAIT #3: nam='SQL*Net message to client' ela= 10 p1=1650815232 p2=1 p3=0
FETCH #3:c=0,e=654,p=0,cr=1,cu=0,mis=0,r=13,dep=0,og=4,tim=1007831256674
WAIT #3: nam='SQL*Net message from client' ela= 13030644 p1=1650815232 p2=1 p3=0
STAT #3 id=1 cnt=14 pid=0 pos=1 obj=0 op='HASH JOIN (cr=15 pr=12 pw=0 time=39402 us)'
=====================
PARSING IN CURSOR #7 len=55 dep=0 uid=57 oct=42 lid=57 tim=1007844294588 hv=2217940283 ad='95037918'
alter session set events '10046 trace name context off' <== tracing turned off
END OF STMT
```

If you are unsure about how to edit the trace file, it is best to capture the trace again using a session that does not have a trace file already. To confirm, check the OS PID of the session you intend to trace and look for a file with that PID in the user_dump_dest.

4. *Make sure the trace is complete*

If the trace started or ended during a call, its best to rethink how the trace is started to ensure this doesn't happen.

You can get an idea for the the amount of time attributed to the call that was in progress at the beginning or end of the trace by looking at the timestamps to find the total time spent prior to the first call and comparing it to the call's elapsed time (although if there were other fetch calls before the first one in the trace, you'll miss those) . The following trace file excerpt was taken by turning on the trace after the query had been executing for a few minutes.

```
*** 2006-07-24 15:00:45.538 <== Time when the trace was started
WAIT #3: nam='db file scattered read' ela= 18598 p1=4 p2=69417 p3=8 <== Wait
*** 2006-07-24 15:01:16.849 <== 10g will print timestamps if trace hasn't been written to in a while
WAIT #3: nam='db file scattered read' ela= 20793 p1=4 p2=126722 p3=7
. . .
*** 2006-07-24 15:27:46.076
WAIT #3: nam='db file sequential read' ela= 226 p1=4 p2=127625 p3=1 <== Yet more waits
WAIT #3: nam='db file sequential read' ela= 102 p1=4 p2=45346 p3=1
WAIT #3: nam='db file sequential read' ela= 127 p1=4 p2=127626 p3=1
WAIT #3: nam='db file scattered read' ela= 2084 p1=4 p2=127627 p3=16
. . .
*** 2006-07-24 15:30:28.536 <== Final timestamp before end of FETCH call
WAIT #3: nam='db file scattered read' ela= 5218 p1=4 p2=127705 p3=16 <== Final wait
WAIT #3: nam='SQL*Net message from client' ela= 1100 p1=1650815232 p2=1 p3=0
=====================
PARSING IN CURSOR #3 len=39 dep=0 uid=57 oct=0 lid=57 tim=1014506207489 hv=1173176699 ad='931230c8'
select count(*) from big_tab1, big_tab2 <== This is not a real parse call, just printed for convenience
END OF STMT

FETCH #3:c=0,e=11,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1014506207466 <== Completion of FETCH call
Notice the FETCH reports 11 microSec elapsed. This is wrong as you can see from timestamps -
It should be around 30 minutes. Maybe this is a feature?
```

As you can see at the top of the file, the trace was started in the middle of a call that was reading from a file and causing waits. When the call completed, the amount of time for the fetch was incorrectly reported.

5. *Repeat to Capture Good Performance Trace*

Repeat the above steps during a period of good performance if possible. This will give you a reference trace to compare against.

Gather Operating System (OS) Performance Data

OS data is needed to see the overall performance of the machine(s) where Oracle is running.

Automatically Using Tools

This section will discuss how to gather data using scripts or tools. Describe when/why you'd use the particular tool, but use the sidebar refs to point to a doc that gives the details on how to setup and use.

10g or higher: If you have obtained a 10gR2 or higher statspack report, you do not need to collect detailed OS data as described below to verify CPU or memory saturation. However, the data captured using the following tools are thorough and may improve the quality of the diagnosis in some cases.

1. ***OS Watcher (OSW) (Preferred Method)***

OS Watcher (OSW) is a collection of UNIX shell scripts intended to collect and archive operating system and network metrics to aid support in diagnosing performance issues. OSW operates as a set of background processes on the server and gathers OS data on a regular basis, invoking such Unix utilities as vmstat, netstat and iostat.

OSW is the preferred way of gathering data on Unix-based systems because it is very simple to install and will collect files that can be analyzed later by Oracle engineers.

Please read the OS Watcher User's Guide for more information on setting up OSW and collecting data. Use OSWg to graph the data for quick analysis.

2. ***LTOM***

The Lite Onboard Monitor (LTOM) is a java program designed as a real-time diagnostic platform for deployment to a customer site. LTOM differs from other support tools, as it is proactive rather than reactive. LTOM provides real-time automatic problem detection and data collection. LTOM is very well suited to transient and unpredictable performance issues..

Please read the LTOM - The On-Board Monitor User's Guide for more information.

3. ***Enterprise Manager***

Enterprise Manager's performance management pages in Grid Control or DB Control include charts that show CPU and memory performance data. To see these charts, click on the host target name (in the "General" section) and then click on the "Performance" page link. From here you will see charts for CPU, memory, and I/O utilization as well as detailed process information.

Enterprise Manager is very good for real-time analysis; however, screen captures will be required for later analysis by Oracle Support.

Manually

OS Watcher is preferred over manual methods. But if you are unable to use OS Watcher and wish to manually collect OS data, please read How to use OS commands to diagnose Database Performance issues for more information.

How-To

- How to use OS commands to diagnose Database Performance issues

Scripts and Tools

- OS Watcher User's Guide
- LTOM - The On-Board Monitor User's Guide

Note:
Data should be gathered at the same time as the database performance data is gathered

Gather RDA Report

The RDA will collect many different bits of data about your system that will be used at various points of this effort. Please read the Remote Diagnostic Agent (RDA) 4 - Main Man Page for more information.

Reference
• Remote Diagnostic Agent (RDA) 4 - Main Man Page

Gather Application Logs (Optional)

Applications will often record when a call was made to the database and how long it took. This can be valuable for determining if the database call is taking too long. If the database calls are quick, then its possible that the problem is due to the client or network. If database calls appear slow, then the problem could well be in the database.

Caution: Application logs are not always instrumented properly and they might incorrectly imply a slow database. This can happen when:

- The "call" to the database includes many other actions not directly involved with the call to Oracle
- The client is on a slow machine where all calls are bogged down, including those to the database

Nevertheless, it is a good idea to obtain these logs to help diagnose the problem. Contact your application developer or administrator for information on how to enable and collect these logs.

Next Step - Analyze

When you have collected the data, click "NEXT" to receive guidance on analyzing the data to verify if the performance problem is with the database or external to it.

Slow Database > Identify the Issue > Analysis

This step will analyze the data collected in the previous step to verify if the database is slow or the problem lies outside of the database.

Verify Oracle OS Resource Usage

This step will verify that:

- There are enough CPU and memory resources for Oracle processes, or if not, then at least Oracle is using those resources and more detailed analysis of the database is required
- OR, non-Oracle processes are using most of the CPU or memory; this is not an Oracle tuning issue

Scripts and Tools

- Oracle® Enterprise Manager Concepts, Host Performance Page
- OSWg User's Guide
- OS Watcher User's Guide
- LTOM - The On-Board Monitor User's Guide

Check CPU Consumption

A system needs sufficient CPU for good, solid performance. Analyze CPU utilization by answering the following questions:

1. ***Are CPU resources scarce?***

We will check CPU usage by looking for:

- ***Total CPU utilization (USER + SYS) should be less than 90%***
  - ***Batch or reporting applications can use higher CPU utilization to maximize throughput, but generally OLTP must have some headroom to permit a stable response time.***
- ***Run queue size per CPU less than 4***
  - ***The run queue size is a very good indicator of CPU utilization problems. In general, when the run queue per CPU is 4 or higher, the system may experience performance problems (of course, the higher the run queue size the worse the problem). This is an indicator of how many processes must wait for a CPU on average and can be used as a gauge on the scarcity of CPU resources.***

Check CPU usage depending on how you collected OS data:

10gR2 Statspack Reports
See the following section:

```
Host CPU  (CPUs: 1)
~~~~~~~~              Load Average
                      Begin     End      User  System    Idle     WIO     WCPU
                    ------- -------   ------- ------- ------- ------- --------
                       2.83   10.16     83.54   16.09    0.37    0.24 #######
```

The "Load Average" begin/end values will tell you the approximate run queue size per CPU. In this case, it was around 10.16 (saturated).

The "User" + "System" values will tell you the total CPU utilization. In this case it was 99.63% (83.54 + 16.09) (saturated).

10gR1: AWR or Statspack Reports

1. ***Go to the OS Statistics or Operating System Statistics section of the report, E.g.:***

```
OS Statistics  DB/Inst: DB10GR2/DB10gR2  Snaps: 20-21
-> ordered by statistic type (CPU use, Virtual Memory, Hardware Config), Name

Statistic                                     Total
------------------------- ----------------------
BUSY_TIME                                    14,257
IDLE_TIME                                        53
IOWAIT_TIME                                      34
SYS_TIME                                      2,302
USER_TIME                                    11,955
LOAD                                             10
OS_CPU_WAIT_TIME                            156,500
VM_IN_BYTES                                       0
PHYSICAL_MEMORY_BYTES                 2,081,890,304
NUM_CPUS                                          1
```

2. ***Calculate CPU Utilization = 100% * BUSY_TIME / (BUSY_TIME + IDLE_TIME)***
3. ***Approximate the run queue size per CPU as:***
   RunQ/CPU = OS_CPU_WAIT_TIME / (NUM_CPUS * BUSY_TIME)

   In this case, it was:

```
        CPU Utilization = 100* 14257 / (14257 + 53) = 99.6%
        RunQ / CPU = 156500 / (1 * 14257) = 10.98
```

Note: in 10gR1, *_TIME was *_TICKS

Enterprise Manager

1. ***In the General section of the database tab, click on the name of your host***
2. ***On the host page, examine the chart for CPU utilization. Select the "CPU Details" view from the pull-down menu. This will show you charts of CPU utilization and CPU load.***

OS Watcher

- ***Graph the data using OSWg and inspect the CPU utilization and run queue charts.***

LTOM

- ***Use the LTOM profiler to generate the CPU charts***

If CPU is not saturated, then proceed to check memory utilization below; otherwise check if Oracle processes are using most of the CPU in the next step.

2. ***What processes are using most of the CPU?***

Examine the data you collected to find out what kind of process is using most of the CPU. The analysis method depends on which type of data you collected - see below:

10gR2 Statspack

Look for this section:

```
Instance CPU
~~~~~~~~~~~~
             % of total CPU for Instance:   74.27
             % of busy  CPU for Instance:   74.55
  %DB time waiting for CPU - Resource Mgr:
```

The "% of busy CPU for Instance" will tell you approximately how much of the host CPU is used by this instance. In this case, it was 74.55%; this instance is the main cause for the host's CPU saturation.

10gR1 Statspack

1. ***Obtain the total CPU utilized for the host (in cs): Find the OS Statistics section, BUSY_TICKS***
2. ***Obtain the total CPU utilized by the instance's foreground sessions (in cs): Find the Instance Activity Stats section, read the total value of the CPU used by this session statistic***
3. ***Calculate % of busy CPU for instance = 100% * "CPU used by this session" / BUSY_TICKS***

For example:

```
        ...
OS Statistics  DB/Inst: DB10GR2/DB10gR2  Snaps: 20-21
```

```
-> ordered by statistic type (CPU use, Virtual Memory, Hardware Config), Name

Statistic                                     Total
------------------------- ----------------------
BUSY_TIME                                    14,257
        ...

Instance Activity Stats  DB/Inst: DB10GR2/DB10gR2  Snaps: 20-21

Statistic                                      Total     per Second    per Trans
-------------------------------- ------------------ -------------- ------------
CPU used by this session                      10,724           75.5        446.8
...
```

In the example above,
% CPU utilized by this instance = 100 * 10724 / 14257 = 75.2%

If most of the CPU is used by this instance, then you have verified that Oracle is responsible for the CPU consumption.

OSWatcher

1. ***Navigate to the OSW archive directory and then to the directory with "top" reports, oswtop***
2. ***Examine the files there which correspond to the time of the performance problem; look at the top processes using CPU***

   For example:

```
        zzz ***Thu Feb 8 15:03:14 PST 2007
load averages:  3.57,  4.23,  3.32    15:03:15
105 processes: 98 sleeping, 6 running, 1 on cpu

Memory: 2048M real, 29M free, 4316M swap in use, 1847M swap free


   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
 19003 oracle     1  32    0    0K    0K run      6:37 25.13% oracle
  6446 oracle     1   0    0    0K    0K run      0:05 21.31% oracle
  1980 oracle    30  29   10  200M   53M sleep    1:23  4.83% java
  6408 oracle     1  59    0   26M   12M sleep    0:01  2.68% perl
 26471 oracle     1  59    0    0K    0K sleep    0:01  1.48% oracle
  6424 oracle     1  59    0    0K    0K sleep    0:00  0.81% oracle
   697 oracle    14  59   -5    0K    0K sleep  340:59  0.67% ocssd.bin
  6455 oracle     1  59    0    0K    0K sleep    0:00  0.56% oracle
```

```
 28744 oracle     1  59    0    0K    0K sleep    3:50  0.25% oracle
```

The top two Oracle processes use around 46% of the CPU. However, because topdoesn't give complete information on the process, you'll still need to determine which instance the oracle processes belong to (if more than one on the machine).

3. ***If most of them are Oracle processes, then check which instance they belong to (if you have more than one instance on the machine)***
4. ***Determine if they are all part of the same instance or not; you may have more than once instance that is affecting performance***

If most of the CPU is used by this instance, then you have verified that Oracle is responsible for the CPU consumption.

LTOM Profiler

Under construction

Enterprise Manager

1. ***In the General section of the database tab, click on the name of your host***
2. ***On the host page, click on the Performance tab***
3. ***Review the list of processes in the Process section at the bottom of the page***
4. ***If most of them are Oracle processes, then check which instance they belong to (if you have more than one instance on the machine)***
5. ***Determine if they are all part of the same instance or not; you may have more than once instance that is affecting performance***

If most of the CPU is used by this instance, then you have verified that Oracle is responsible for the CPU consumption.

Are Oracle processes using most of the CPU?

- ***If Oracle processes use most of the CPU, continue to the next item below to check memory consumption.***
- ***If non-Oracle processes are using most of the CPU, the problem appears to be outside of Oracle; this tuning effort should be aborted and the non-Oracle processes should be investigated or more CPUs should be added to the system.***

Check Memory Consumption

Database performance can be very slow and unpredictable when the system is short on memory. Answer the two questions below to determine if there is a memory problem on your system.

1. ***Is there a memory shortage?***

Regardless of the tool used to collect the data, you will need to consider the following metrics when looking for a memory shortage:

- ***Memory Utilization (% or free KB): measures how much physical memory has been allocated to processes. When this is around 100% the system will utilize more and more swap; the severity of the shortage is evident by the following two metrics.***
- ***Memory Page Scan Rate (pages/s): a measure of how hard the page scanner is working to reclaim memory. When this is in the hundreds/sec, its likely that there is a memory shortage.***
- ***Swap Utilization (% or free KB): how much of the swap device is being used. As physical memory becomes scarce this percentage goes up. Compare this value to a baseline to see if swap usage increased beyond a normal amount for the system. As swap approaches 100% utilization, the memory crunch gets worse and the system becomes unstable (and could crash).***

You are seeing a memory shortage If you see memory utilization close to 100%, the scan rate in the hundreds / sec, and a large percentage of the swap device utilized.

These metrics can be seen using the following tools or data collection:

Enterprise Manager

1. ***In the General section of the database tab, click on the name of your host***
2. ***On the host page, examine the chart for memory utilization. Select the "Memory Details" view from the pull-down menu. This will show you charts of memory utilization, swap utilization, and page scanner activity.***

OS Watcher

1. ***Use OSWg to create graphs of the OS data, specifically the memory graphs***
2. ***Examine the graphs for "Memory: Available Swap" and "Memory: Scan Rate"***

LTOM Profiler

1. ***Create an LTOM Profiler output directory***
2. ***Click on the link for Operating System Memory***
3. ***Examine the graphs for "Memory: Available Swap" and "Memory: Scan Rate"***

If there is NOT a memory shortage, then proceed to verify if the database is slow. Otherwise, find the processes using memory with the following step.

2. ***What is using most of the memory?***

Determine which processes are using most of the memory (Oracle or non-Oracle) using the following methods/data collection.

10gR2 Statspack

1. ***Look for this section:***

```
Memory Statistics                     Begin          End
~~~~~~~~~~~~~~~~~                ------------ ------------
                  Host Mem (MB):     1,985.4      1,985.4
                   SGA use (MB):       228.0        228.0
                   PGA use (MB):        55.4         54.5
    % Host Mem used for SGA+PGA:        14.3         14.2
```

2. ***If the value of "% Host Mem used for SGA+PGA" is high, then you have verified that this instance is responsible for the memory consumption***

Enterprise Manager

1. ***In the General section of the database tab, click on the name of your host***
2. ***On the host page, click on the Performance tab***
3. ***Sort the list of processes in the Process section at the bottom of the page by "Memory Utilization %" (selectable in the "View by" dropdown list)***
4. ***If most of them are Oracle processes, then check which instance they belong to (if you have more than one instance on the machine)***
5. ***Determine if they are all part of the same instance or not; you may have more than once instance that is affecting performance***

OSWatcher

1. ***Go to the OSWatcher "archive/oswps" directory; this directory has output of the "ps" command taken at every sample interval***
2. ***Choose a ps output file during the performance problem***
3. ***Look at one of the ps outputs that was obtained during the problem***

```
       zzz ***Thu Feb 8 15:00:08 PST 2007
 F S      UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN    STIME TTY      TIME CMD
 8 R   oracle 21150     1  0  79 20        ?    241          14:43:19 pts/5    0:00 /usr/bin/
ksh ./OSWatcher.sh
 8 S   oracle 14258     1  0  40 20        ?  44958        ?   Jan 25 ?       0:01
ora_q001_DB10gR2
```

```
 8 O   oracle  4057  4055  0  69 20        ?    151          15:00:08 pts/5    0:00 ps -elf
 8 S   oracle 28748     1  0  40 20        ?  34961        ? 16:19:40 ?        0:00
ora_d000_DB9iR2
 8 R   oracle 19003 19002 27  77 20        ? 439930          14:41:33 ?        5:49
oracleDB10gR2 (DESCRIPTION=(LOCAL=Y
 8 S   oracle  8842     1  0  40 20        ?   4017        ?   Dec 26 ?        0:41 /u01/app/
oracle/product/DB10gR2/bin
 8 S   oracle  4048  4045  0  59 20        ?    173        ? 15:00:08 pts/5    0:00 iostat -xn
1 3
```

4. ***Sort the processes by the "SZ" column to see which ones are using the most memory (this includes mapped shared memory so care must be taken to subtract the size of the SGA from each value)***
5. ***Determine if they are all part of the same instance or not; you may have more than once instance that is affecting performance***

Are Oracle processes using most of the memory?

- ***If Oracle processes use most of the memory, you have verified an Oracle performance problem. Click NEXT to continue to the next phase, "Determine a Cause" and proceed to: Analysis > Choose a Tuning Strategy > Oracle Memory Consumption.***
- ***If non-Oracle processes are using most of the memory, the problem appears to be outside of Oracle; this tuning effort should be aborted and the non-Oracle processes should be investigated or more memory should be added to the system.***

Verify The Database is Slow

A performance problem may appear to be due to the database but may actually be caused by a slow client (usually middle-tier) or network. This step will help you verify if the database is really the cause for the slow performance, or if you should look elsewhere.

The main idea in these comparison is to compare the total "DB Time" between the "good" and "bad" reports. DB Time is the total time spent in the database either working (using CPU) or waiting for a non-idle event. When there is a performance problem, DB Time increases (usually because there are many more sessions waiting for non-idle events).

Analyze the following reports depending on what you've collected in the previous step:

Analyze the ASH, AWR, or Statspack Reports

Analyze the ASH , AWR, or statspack reports you collected to verify if the database is causing the performance problem.

Choose **one** of the following analysis methods, depending on what you've collected:

1. ***10.2.x: Compare ASH Reports Between Good and Bad Performance Periods***

Compare the average active sessions in an ASH report during good performance and bad performance. The average active sessions will show you how many sessions were either on CPU or waiting for a non-idle event. When performance is bad due to the database, the number of active sessions will be higher than when performance was good. This is because when there is some resource bottleneck, more sessions will need to actively wait for that resource and this increases the number active sessions.

For example, here is an ASH report summary for a database during a busy period when performance was good:

```
         Analysis Begin Time:   13-Feb-07 10:28:03
           Analysis End Time:   13-Feb-07 10:43:03
                Elapsed Time:        15.0 (mins)
                Sample Count:       2,387
     Average Active Sessions:        2.65
  Avg. Active Session per CPU:        2.65
                Report Target:   None specified

Top User Events              DB/Inst: DB10GR2/DB10gR2  (Feb 13 10:28 to 10:43)

                                                          Avg Active
Event                             Event Class     % Activity   Sessions
---------------------------------- --------------- ---------- ----------
db file sequential read            User I/O             40.13       1.06
db file parallel read              User I/O             29.91       0.79
CPU + Wait for CPU                 CPU                   8.97       0.24
          -------------------------------------------------------------
```

Here is the same server and application during a period of bad performance:

```
         Analysis Begin Time:   12-Feb-07 15:12:49
           Analysis End Time:   12-Feb-07 15:27:55
                Elapsed Time:        15.1 (mins)
                Sample Count:       9,161
     Average Active Sessions:       10.11
  Avg. Active Session per CPU:       10.11
                Report Target:   None specified

Top User Events              DB/Inst: DB10GR2/DB10gR2  (Feb 12 15:12 to 15:27)
```

```
                                                          Avg Active
Event                               Event Class     % Activity   Sessions
----------------------------------- --------------- ---------- ----------
read by other session               User I/O             53.40       5.40
CPU + Wait for CPU                  CPU                  37.24       3.77
db file scattered read              User I/O              4.90       0.50
          -------------------------------------------------------------
```

Notice the increase in the Average Active Sessions when performance was bad - it is about a 5X increase (from 2.65 to 10.11). Notice also the change in the Top User Events; the main waits went from db file sequential read to read by other session and CPU + Wait for CPU along with db file scattered read. This was due to an index being dropped on an important table. A query that normally used the index was performing full table scans during the period of bad performance.

2. ***10.2.x: Use AWR Diff Report to Compare Good and Bad Performance Periods***

1. ***Use the AWR difference script (using $OH/rdbms/admin/awrddrpt.sql) to generate a report***
2. ***Check if DBTime increased during bad period (shown in the top of the report or in the "Time Model" section)***

For example:

```
Snapshot Set  Begin Snap Id ...      Elapsed Time (min)       DB Time (min)    Avg Active Users
------------ -------------- ... -------------------------- ---------------- -------------------
1st                    2519 ...                      60.27             1.62                0.03
2nd                    2518 ...                      59.54            17.48                0.29
```

Note: You must compare snapshot periods of the same length!

3. ***10.1.x or higher: Compare AWR Reports Between Good and Bad Performance Periods***

1. ***Generate an AWR report (using $OH/rdbms/admin/awrrpt.sql) for the good and bad periods***
2. ***In the Time Model Statistics section, look for "DB TIME" in each report***

For example:

```
Time Model Statistics              DB/Inst: DB10GR2/DB10gR2  Snaps: 2468-2469
-> Total time in database user-calls (DB Time): 14.2s
-> Statistics including the word "background" measure background process
   time, and so do not contribute to the DB time statistic
-> Ordered by % or DB time desc, Statistic name

Statistic Name                                       Time (s) % of DB Time
------------------------------------------ ------------------ ------------
```

```
sql execute elapsed time                                14.1         99.2
DB CPU                                                  10.0         70.6
parse time elapsed                                       7.9         56.0
hard parse elapsed time                                  7.4         52.0
hard parse (sharing criteria) elapsed time               1.3          9.5
PL/SQL execution elapsed time                            1.1          8.0
sequence load elapsed time                               0.0           .2
repeated bind elapsed time                               0.0           .1
hard parse (bind mismatch) elapsed time                  0.0           .0
DB time                                                 14.2          N/A
          -------------------------------------------------------------
```

3. ***Check if DBTime increased during bad period***

4. ***Compare Statspack Reports Between Good and Bad Performance Periods***

Manually compare the files to see if the total DB Time increased during the bad period. Do this for both good and bad periods:

1. ***Generate a statspack report (using $OH/rdbms/admin/spreport)***
2. ***Determine the total DB Time:***

<u>10g or higher statspack:</u>

Use the Time Model statistics:

```
Time Model System Stats  DB/Inst: DB10GR2/DB10gR2  Snaps: 31-32
-> Ordered by % of DB time desc, Statistic name

Statistic                                       Time (s) % of DB time
---------------------------------- -------------------- ------------
sql execute elapsed time                           11.3         99.8
DB CPU                                              6.6         58.0
parse time elapsed                                  5.2         45.7
hard parse elapsed time                             5.1         45.1
PL/SQL execution elapsed time                       0.0           .2
repeated bind elapsed time                          0.0           .1
DB time                                            11.3
          ---------------------------------------------------------------
```

<u>Any version of statspack:</u>

Calculate "Total DB Time" by adding the top 5 non-idle, foregroundtimed events

```
Top 5 Timed Events                                                    Avg %Total
~~~~~~~~~~~~~~~~~~                                                   wait   Call
Event                                        Waits    Time (s)   (ms)   Time
----------------------------------------- ----------- ---------- ------ ------
CPU time                                                      7          84.1
db file sequential read                          325          1      2    9.1
log file parallel write                           12          0     17    2.6
control file parallel write                        7          0     22    1.9
SQL*Net break/reset to client                      6          0     19    1.4
          -------------------------------------------------------------
```

Foreground (FG)event time can be estimated by subtracting the Background (BG)wait time for each event you see in the top 5 timed events. Some events will not have a BG component; in general you can ignore BG events for a quick estimate.

For example, for the Top 5 Timed Events above, here were the foreground and background waits:

```
Wait Events  DB/Inst: DB10GR2/DB10gR2  Snaps: 31-32

                                                        Avg
                                        %Time Total Wait   wait    Waits
Event                            Waits  -outs   Time (s)   (ms)     /txn
-------------------------------- ------------ ------ ---------- ------ --------
db file sequential read               325      0          1      2     29.5
log file parallel write                12      0          0     17      1.1 <== has BG
event component
control file parallel write             7      0          0     22      0.6 <== has BG
event component
SQL*Net break/reset to client           6      0          0     19      0.5
log file sync                           8      0          0      4      0.7
. . .

Background Wait Events  DB/Inst: DB10GR2/DB10gR2  Snaps: 31-32
-> %Timeouts:  value of 0 indicates value was < .5%.  Value of null is truly 0
-> Only events with Total Wait Time (s) >= .001 are shown
-> ordered by Total Wait Time desc, Waits desc (idle events last)

                                                        Avg
                                        %Time Total Wait   wait    Waits
Event                            Waits  -outs   Time (s)   (ms)     /txn
-------------------------------- ------------ ------ ---------- ------ --------
log file parallel write                13      0          0     17      1.2
control file parallel write             7      0          0     22      0.6
. . .
```

The log file parallel writeand control file parallel writeevents are 100% background events. That means the total DB Time can be approximated as:
Est. Total DB Time = CPU time (7 sec) + db file sequential read (1 sec)= 8 seconds

3. ***Check if DBTime increased during bad period***

5. ***Analyze a Single Statspack / AWR Report***

Using a single statspack is not reliable for issue verification; if you decide to use this, please be advised that there is a chance the problem is outside of the database and your tuning efforts may be wasted.

With a single statspack, we can look at the type of timed events that are in the "Top 5" and see if they're "unhealthy", e.g., concurrency or RAC waits:

Some unhealthy events are:

- ***Enqueues***
- ***Latches***
- ***Buffer busy waits***
- ***Row cache lock waits***
- ***Free buffer waits***
- ***RAC waits (having to do with GC)***
- ***Library cache lock or pin waits***
- ***Shared cursor S to X waits***

Unhealthy events need more diagnostic effort to determine what is causing them. E.g., a library cache latch problem will not be solved by tuning a query to perform fewer buffer gets. Solutions to these type of problems are sometimes difficult to implement; e.g., rewrite parts of the application to avoid locking conflicts.

Some "healthy" events are:

- ***CPU***
- ***db file sequential read***
- ***db file scattered read***
- ***direct path read***
- ***direct path write***

The presence of healthy events mean that you will need to either add capacity like CPUs or I/O, or maybe tune the SQL. Either way, the solution is achievable by some common solutions.

How to Proceed

- If DBTime is greater during bad period, it is likely the database is causing the problem; you have verified the problem is in the database.
  - Note where the database is spending its time: CPU or top few wait events
  - click the NEXT button to proceed to the next step
- Otherwise, check the clients / mid-tiers. Ideally, the client or mid-tiers have a diagnostic log with timing data for calls made to the database. Read this log to see the database performance from the client's point of view.
- If no bottlenecks are found in the clients / mid-tiers (no CPU or memory bottlenecks), you may continue with this process and assume the database has the performance problem. An extended SQL trace (see the next phase of this process) will be very important for finding the cause.
  - Note where the database is spending its time: CPU or top few wait events
  - click the NEXT button to proceed to the next step

Analyze the Extended SQL Trace

Analyze the SQL trace data you collected to verify if the database is causing the performance problem.

1. ***Preferred: Compare Two TKProfs (Good and Bad Performance)***

We can verify if the database is the cause of the performance problem by comparing the same operation's trace files when performance was good and when it was bad.

Follow this process to verify the database has a performance problem :

a. ***Start with the "good-performing" trace file***

b. ***Go to the end of the TKProf and note the values for the following (for both non-recursive and recursive):***
   - ***total elapsed time***
   - ***total calls***
   - ***idle time, i.e., SQL*Net Message from client wait, total waited***
   - ***total number of rows returned***

c. ***Derive the following metrics:***
   - ***elapsed time / call = (total elapsed recursive + total elapsed non-rec) / total calls***
   - ***rows / call = total rows / total calls***
   - ***total idle time / call =[idle time (non-recursive) + idle time (recursive)] / total calls***

d. ***Repeat steps (b) and (c) for the "bad-performing" trace file***

e. ***Compare each of the derived metrics between the "good" trace and "bad" trace and see which are higher, similar, or lower***

f. ***Look in the table for the combination of symptoms that matches what you see in your comparison:***

Elapsed Time / Call	Rows / Call	Total Idle Time	Verification Result
Similar	Similar	Similar	No Problem
Higher	Similar	Similar	Database Problem
Similar	Similar	Higher	Client Problem (General)
Similar or Higher	Lower	Higher	Client Problem (Arraysize)

Summary of Verification Results

No Problem: The trace files were similar. This indicates neither the database nor the database client appeared to be slow. The problem may be in front of the database client / mid-tier; i.e., a user's browser or network might be slow. Stop the database tuning effort and instead focus on the clients.

Database Problem: The trace file comparison shows higher elapsed time per call which means the database is taking longer to do the same work.
The database has been verified as the performance problem. Click NEXT to determine a cause for this problem.

Client Problem (General): The trace file comparison shows the database as being more idle and not taking more elapsed time per call. This usually means either the database client or network is taking longer to send requests to the database, and hence appears slower from the end user's point of view.
Stop the database tuning effort and instead investigate the database clients to see why they are slower.

Client Problem (Arraysize): The client is processing fewer rows per call with the database. This is ineffecient behavior because it introduces delays while more calls are needed to move rows in or out of the database from the client and causes the database to work harder (more logical reads, block pins, context switches, etc). To fix this problem, the client will need to request more rows per call.
Stop the database tuning effort and instead focus on the clients.

Analysis Example

The following TKProf was collected during a period of good performance:

```
. . .

********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      0      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        0      0.00       0.00          0          0          0           0
                           ^
                      total elapsed time

Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  PL/SQL lock timer                              198        1.13        193.94


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute   1964      1.23       5.88        184         38       3205         784
Fetch     1180      4.97      33.31       8185      51952          0       15886
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     3144      6.20      39.19       8369      51990       3205       16670
           ^                    ^                                           ^
      total calls       total elapsed time                          total rows


Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  db file sequential read                       1736        1.41        16.15
  db file parallel read                          821        0.96        13.36
  read by other session                            5        0.03         0.05

*** no idle events for this session's trace ***

    2  user  SQL statements in session.
    0  internal SQL statements in session.
    2  SQL statements in session.
********************************************************************************
```

The derived metrics for the data collected in the previous example are:

```
elapsed time / call = 39.19 / 3144 = 0.0125 sec/call
rows / call = 16670 / 3144 = 5.3 rows/call
total idle time / call = 0 / 3144 = 0 secs / call
```

The derived metrics for the data collected during the bad period (not shown) are:

```
elapsed time / call = 284.16 / 90 = 3.16 sec/call
rows / call = 440 / 90 = 4.89 rows/call
total idle time / call = 0 / 90 = 0 secs
```

Note: both TKProfs were based on trace files for the same amount of time (5 minutes) of a typical session.

When we compare the good and bad TKProfs we get:

```
elapsed time / call was HIGHER (0.0125 vs. 3.16)
rows / call was SIMILAR (5.3 vs. 4.89)
total idle time / call was SIMILAR (0 vs. 0)
```

We conclude this was a DATABASE PROBLEM (higher, similar, similar) after looking up the combination of symptoms in the table above.

2. ***Alternative: Analyze One TKProf (Bad Performance)***

This method is not as accurate as comparing two trace files, but we can usually get a good idea if the database is causing most of the problem or not. For this analysis we will review just the trace of the bad performing case and look for how much time was spent in the database versus waiting for a client.

Follow this process to verify the database has a performance problem :

a. ***Go to the end of the TKProf and note the values for the following (for both non-recursive and recursive):***
   - ***total elapsed time***
   - ***idle time, i.e., SQL*Net Message from client wait, total waited***

b. ***Derive the following metrics:***
   - ***total DB elapsed time = total elapsed time (non-recursive) + total elapsed time (recursive)***
   - ***total idle time = idle time (non-recursive) + idle time (recursive)***

Verification Result

If the total DB elapsed time is greater than the total idle time, then the database is responsible for most of the time

the session spent during the trace.
The database has been verified as the performance problem. Click NEXT to determine a cause for this problem.

If the total elapsed time is less than the total idle time, then the client is responsible for most the time the session spent during the trace.
Stop the database tuning effort and instead investigate the database clients to see why they are slower.

Check the average number of rows per call (divide total rows by total calls) and see if it is less than 10. Generally, a low number of rows / call indicates room for improvement with regard to array processing; larger array sizes should reduce the client and DB time (although if you make the arraysize too large, response time may suffer for OLTP applications). Also, keep in mind that to really determine if the client time is significant, one must know how the application is being used, if the user's "think time" is expected and natural, and what is the expected database call time.

Some typical causes and solutions for client waits can be found in this guide under:
Slow Database > Determine a Cause >Analysis > Reduce Client Bottlenecks

Next Step - Determine a Cause

If the analysis above has verified the problem is with the database, click "NEXT" to move to the next phase of this process where you will receive guidance to determine a cause for the slow database performance.

Would You Like to Stop and Log a Service Request?

We would encourage you to continue until at least the "Determine a Cause", "Data Collection" step, but If you would like to stop at this point and receive assistance from Oracle Support Services, please do the following:

- Please copy and paste the following into the SR:

  Last Diagnostic Step = Performance_Diagnostic_Guide.Slow_Database.
  Issue_Identification.Data_Collection

- Enter the problem statement and how the issue has been verified (if performed)
- Gather the OS and database performance data and prepare to upload it to the SR
- Optionally, gather an RDA
- Gather other relevant information you may have such as timing data for typical queries

The more data you collect ahead of time and upload to Oracle, the fewer round trips will be required for this data and the quicker the problem will be resolved.

Click here to log your service request

Slow Database > Determine a Cause >Overview

At this point we have verified the slow database performance; now, we seek to determine the cause for this. The overall process for this is:

Data Collection

1. You will need the extended SQL trace you collected in the Issue Identification > Data Collection step, OR gather one now by:
    - Identifying the top few sessions that are affected
    - Obtain an extended SQL trace of the top sessions
2. Review the trace to confirm the main DB time components (CPU and/or waits) and how they compare to the overall components from the database-wide data gathered in the prior Issue Identification > Data Collection steps

Analysis

1. Look for common causes for the main DB time components
2. Review possible solutions for the likely cause
3. Implement the best solution
4. Verify that the solution solved the problem or more work is needed

Its very important to remember that every cause that is identified should be justified by the facts we have collected. If a cause cannot be justified, it should not be identified as a cause (i.e., we are not trying to guess at a solution).

Slow Database > Determine a Cause >Data Collection

This phase is very critical to resolving the query performance problem because accurate data about the query's execution plan and underlying objects are essential for us to determine a cause for the slow performance.

Gather an Extended SQL Trace

The extended SQL trace (10046 trace at level 12) will capture execution statistics of all SQL statements issued by a session during the trace. It will show us how much time is being spent per statement, how much of the time was due to CPU or wait events, and what the bind values were. We will be able to see specifically which statements are running slower and which wait events may be applicable.

For detailed information on how to use the 10046 trace event, see the "How To" article on the side called, Recommended Method for Obtaining 10046 trace for Tuning

You may have already gathered the SQL trace in the prior data collection step if only a few sessions were affected; otherwise you will need to gather the data now.

A summary of the steps needed to obtain the 10046 and TKProf are listed below:

Documentation
- Understanding SQL Trace and TKProf

How-To
- How To Collect 10046 Trace Data
- Recommended Method for Obtaining 10046 trace for Tuning

SCRIPTS/TOOLS
- How To Generate TKProf reports
- Collect 10046 Traces Automatically with LTOM

Collecting the Trace

The following process will help you collect SQL trace data properly:

1. ***Choose a session to trace***

Target the most important / impacted sessions

- ***Users that are experiencing the problem most severely; e.g., normally the transaction is complete in 1 sec, but now it takes 30 sec.***
- ***Users that are aggressively accumulating time in the database***

The following queries will allow you to find the sessions currently logged into the database that have accumulated the most time on CPU or for certain wait events. Use them to identify potential sessions to trace using 10046.

These queries are filtering the sessions based on logon times less than 4 hours and the last call occurring within 30 minutes. This is to find more currently relevant sessions instead of long running ones that accumulate a lot of time but aren't having a performance problem. You may need to adjust these values to suit your environment.

Find Sessions with the Highest CPU Consumption

```
-- sessions with highest CPU consumption
SELECT s.sid, s.serial#, p.spid as "OS PID",s.username, s.module, st.value/100 as "CPU sec"
```

```
FROM v$sesstat st, v$statname sn, v$session s, v$process p
WHERE sn.name = 'CPU used by this session' -- CPU
AND st.statistic# = sn.statistic#
AND st.sid = s.sid
AND s.paddr = p.addr
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
ORDER BY st.value;

       SID    SERIAL# OS PID       USERNAME
MODULE                                         CPU sec
---------- ---------- ------------ --------------------
---------------------------------------- ----------
       141       1125 15315        SYS                  sqlplus@coehq2 (TNS V1-
V3)                     8.25
       147        575 10577        SCOTT
SQL*Plus                                      258.08
       131        696 10578        SCOTT
SQL*Plus                                      263.17
       139        218 10576        SCOTT
SQL*Plus                                      264.08
       133        354 10583        SCOTT
SQL*Plus                                      265.79
       135        277 10586        SCOTT
SQL*Plus                                      268.02
```

Find Sessions with Highest Waits of a Certain Type

```
-- sessions with the highest time for a certain wait
SELECT s.sid, s.serial#, p.spid as "OS PID", s.username, s.module, se.time_waited
FROM v$session_event se, v$session s, v$process p
WHERE se.event = '&event_name'
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
AND se.sid = s.sid
AND s.paddr = p.addr
ORDER BY se.time_waited;

SQL> /
Enter value for event_name: db file sequential read

       SID    SERIAL# OS PID       USERNAME
MODULE                                    TIME_WAITED
```

```
---------- ---------- ------------ --------------------
---------------------------------------- -----------
       141       1125 15315        SYS                  sqlplus@coehq2 (TNS V1-
V3)                          4
       147        575 10577        SCOTT
SQL*Plus                                        45215
       131        696 10578        SCOTT
SQL*Plus                                        45529
       135        277 10586        SCOTT
SQL*Plus                                        50288
       139        218 10576        SCOTT
SQL*Plus                                        51331
       133        354 10583        SCOTT
SQL*Plus                                        51428
```

10g or higher: Find Sessions with the Highest DB Time

```
-- sessions with highest DB Time usage
SELECT s.sid, s.serial#, p.spid as "OS PID", s.username, s.module, st.value/100 as "DB Time
(sec)"
, stcpu.value/100 as "CPU Time (sec)", round(stcpu.value / st.value * 100,2) as "% CPU"
FROM v$sesstat st, v$statname sn, v$session s, v$sesstat stcpu, v$statname sncpu, v$process p
WHERE sn.name = 'DB time' -- CPU
AND st.statistic# = sn.statistic#
AND st.sid = s.sid
AND  sncpu.name = 'CPU used by this session' -- CPU
AND stcpu.statistic# = sncpu.statistic#
AND stcpu.sid = st.sid
AND s.paddr = p.addr
AND s.last_call_et < 1800 -- active within last 1/2 hour
AND s.logon_time > (SYSDATE - 240/1440) -- sessions logged on within 4 hours
AND st.value > 0;

       SID    SERIAL# OS PID       USERNAME MODULE                                   DB Time
(sec) CPU Time (sec)      % CPU
---------- ---------- ------------ -------- ----------------------------------------
------------ -------------- ----------
       141       1125 15315        SYS      sqlplus@coehq2 (TNS V1-V3)
12.92           9.34      72.29


Note: sometimes DB Time can be lower than CPU Time when a session issues long-running
recursive calls. The DB Time statistic doesn't update until the top-level call is finished
```

```
(versus the CPU statistic that updates as each call completes).
```

Obtain a complete trace

- ***Ideally, start the trace as soon as the user logs on and begins the operation or transaction. Continue tracing until the operation is finished.***
- ***Try to avoid starting or ending the trace in the middle of a call unless you know the call is not important to the solution***

2. ***Collect the trace and generate a TKProf report***

See the document references on this page for details on obtaining extended SQL trace data. Read Recommended Method for Obtaining 10046 trace for Tuning first

- ***Trace a Connected Session***
  This is the most common way to get a trace file.
  - ***Start tracing on a connected session***
  - ***Coordinate with the user to start the operation***
  - ***Let the trace collect while the operation is in progress***
  - ***Stop tracing when the operation is done***
  - ***Gather the trace file from the "user_dump_dest" location (you can usually identify the file just by looking at the timestamp).***
- ***Alternative: Trace Using a Test Script***
  Sometimes you may be able to script a reproducible test case.
  - ***Put ALTER SESSION commands to start / stop the tracing in the test script***
  - ***Run the test script and collect the trace file from the "user_dump_dest" location (you can usually identify the file just by looking at the timestamp).***
- ***Other Considerations***
  - ***Shared Servers: Tracing shared servers could cause many separate trace files to be produced as the session moves around to various Oracle processes on each call . Use the 10g utility, "trcsess" to combine these separate files into one.***
- ***Generate a TKProf report and sort the SQL statements in order of most elapsed time using the following command:***

  tkprof <trace file name> <output file name> sort=fchela,exeela,prsela

3. *Make sure trace file contains only data from the recent test*

If this session has been traced recently, there may be other traces mixed in the file with the recent trace collected

- *We should extract only the trace data that is part of the recent tests. See the place in the sample trace below where it says "Cut away lines above this point".*

Trace file from a long running process that has been traced intermittently over several days

```
. . .
*** 2006-07-24 13:35:05.642 <== Timestamp from a previous tracing
WAIT #8: nam='SQL*Net message from client' ela= 20479935 p1=1650815232 p2=1 p3=0
=====================
PARSING IN CURSOR #9 len=43 dep=0 uid=57 oct=3 lid=57 tim=1007742062095 hv=4018512766 ad='97039a58'
select e.empno, d.deptno<== Previous cursor that was traced
from emp e, dept d
END OF STMT
PARSE #9:c=630000,e=864645,p=10,cr=174,cu=0,mis=1,r=0,dep=0,og=4,tim=1007742062058
BINDS #9:
EXEC #9:c=0,e=329,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1007742062997
WAIT #9: nam='SQL*Net message to client' ela= 18 p1=1650815232 p2=1 p3=0
. . .
FETCH #9:c=10000,e=513,p=0,cr=1,cu=0,mis=0,r=15,dep=0,og=4,tim=1007742148898
WAIT #9: nam='SQL*Net message from client' ela= 2450 p1=1650815232 p2=1 p3=0
WAIT #9: nam='SQL*Net message to client' ela= 7 p1=1650815232 p2=1 p3=0
FETCH #9:c=0,e=233,p=0,cr=0,cu=0,mis=0,r=10,dep=0,og=4,tim=1007742152065
. . .

====> CUT AWAY LINES ABOVE THIS POINT - THEY AREN'T PART OF THIS TEST <====
*** 2006-07-24 18:35:48.850
<== Timestamp for the tracing we want (notice its about 5 hours later)


=====================
PARSING IN CURSOR #10 len=69 dep=0 uid=57 oct=42 lid=57 tim=1007783391548 hv=3164292706 ad='9915de10'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
. . .
=====================
PARSING IN CURSOR #3 len=68 dep=0 uid=57 oct=3 lid=57 tim=1007831212596 hv=1036028368 ad='9306bee0'
select e.empno, d.dname<== Cursor that was traced
from emp e, dept d
where e.deptno = d.deptno
END OF STMT
PARSE #3:c=20000,e=17200,p=0,cr=6,cu=0,mis=1,r=0,dep=0,og=4,tim=1007831212566
BINDS #3:
EXEC #3:c=0,e=321,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1007831213512
WAIT #3: nam='SQL*Net message to client' ela= 15 p1=1650815232 p2=1 p3=0
WAIT #3: nam='db file sequential read' ela= 7126 p1=4 p2=11 p3=1
. . .
FETCH #3:c=10000,e=39451,p=12,cr=14,cu=0,mis=0,r=1,dep=0,og=4,tim=1007831253359
WAIT #3: nam='SQL*Net message from client' ela= 2009 p1=1650815232 p2=1 p3=0
WAIT #3: nam='SQL*Net message to client' ela= 10 p1=1650815232 p2=1 p3=0
FETCH #3:c=0,e=654,p=0,cr=1,cu=0,mis=0,r=13,dep=0,og=4,tim=1007831256674
WAIT #3: nam='SQL*Net message from client' ela= 13030644 p1=1650815232 p2=1 p3=0
STAT #3 id=1 cnt=14 pid=0 pos=1 obj=0 op='HASH JOIN (cr=15 pr=12 pw=0 time=39402 us)'
=====================
```

```
PARSING IN CURSOR #7 len=55 dep=0 uid=57 oct=42 lid=57 tim=1007844294588 hv=2217940283 ad='95037918'
alter session set events '10046 trace name context off' <== tracing turned off
END OF STMT
```

If you are unsure about how to edit the trace file, it is best to capture the trace again using a session that does not have a trace file already. To confirm, check the OS PID of the session you intend to trace and look for a file with that PID in the user_dump_dest.

4. ***Make sure the trace is complete***

If the trace started or ended during a call, its best to rethink how the trace is started to ensure this doesn't happen.

You can get an idea for the the amount of time attributed to the call that was in progress at the beginning or end of the trace by looking at the timestamps to find the total time spent prior to the first call and comparing it to the call's elapsed time (although if there were other fetch calls before the first one in the trace, you'll miss those) . The following trace file excerpt was taken by turning on the trace after the query had been executing for a few minutes.

```
*** 2006-07-24 15:00:45.538 <== Time when the trace was started
WAIT #3: nam='db file scattered read' ela= 18598 p1=4 p2=69417 p3=8 <== Wait
*** 2006-07-24 15:01:16.849 <== 10g will print timestamps if trace hasn't been written to in a while
WAIT #3: nam='db file scattered read' ela= 20793 p1=4 p2=126722 p3=7
. . .
*** 2006-07-24 15:27:46.076
WAIT #3: nam='db file sequential read' ela= 226 p1=4 p2=127625 p3=1 <== Yet more waits
WAIT #3: nam='db file sequential read' ela= 102 p1=4 p2=45346 p3=1
WAIT #3: nam='db file sequential read' ela= 127 p1=4 p2=127626 p3=1
WAIT #3: nam='db file scattered read' ela= 2084 p1=4 p2=127627 p3=16
. . .
*** 2006-07-24 15:30:28.536 <== Final timestamp before end of FETCH call
WAIT #3: nam='db file scattered read' ela= 5218 p1=4 p2=127705 p3=16 <== Final wait
WAIT #3: nam='SQL*Net message from client' ela= 1100 p1=1650815232 p2=1 p3=0
=====================
PARSING IN CURSOR #3 len=39 dep=0 uid=57 oct=0 lid=57 tim=1014506207489 hv=1173176699 ad='931230c8'
select count(*) from big_tab1, big_tab2 <== This is not a real parse call, just printed for convenience
END OF STMT

FETCH #3:c=0,e=11,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1014506207466 <== Completion of FETCH call
Notice the FETCH reports 11 microSec elapsed. This is wrong as you can see from timestamps -
It should be around 30 minutes. Maybe this is a feature?
```

As you can see at the top of the file, the trace was started in the middle of a call that was reading from a file and causing waits. When the call completed, the amount of time for the fetch was incorrectly reported.

If you have verified the TKProf has been properly collected, then proceed to the next section if you also have an ASH, AWR, or statspack report; otherwise click NEXT to analyze the TKProf in detail.

Comparing the TKProf to ASH / AWR / Statspack

If you have an ASH, AWR, or statspack report (because the whole DB seemed slow), its a good idea to compare the TKProf with the instance-wide report. The point is to confirm that the bottlenecks seen at the whole instance level are consistent with what you see in a session via TKProf.

If the two don't match, then maybe you identified or traced the wrong session. If only a few sessions are slow, then its quite possible that their CPU and wait profiles differ from the overall database. In that case, you can focus on what you've found in the TKProf without needing to correlate it to the overall instance (but you should make sure you traced the correct session).

Compare the files as follows:

1. ***Review the TKProf***

Identify the timed events of the TKProf.
Go to the "OVERALL TOTALS" section and note:

- ***Total CPU; add total CPU for recursive and non-recursive statements***
- ***Total DB Time; Add the total elapsed time for recursive and non-recursive statements***
- ***Find the percentage of the total DB time the CPU and top waits represent***

For example:

```
********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      0      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        0      0.00       0.00          0          0          0           0

Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  read by other session                          14        0.20         1.54
  db file scattered read                         30        0.00         0.01
  db file sequential read                         7        0.00         0.00
  PL/SQL lock timer                               6        1.02         6.00


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute     57      0.06       0.02          0          5         28          24
Fetch       33     24.52     284.13      17025     312555          0         416
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       90     24.58     284.16      17025     312560         28         440

Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  read by other session                         1374        0.96        158.10
  db file scattered read                        2622        0.57          7.11
  db file sequential read                        721        0.23          0.81
  latch: cache buffers chains                      5        0.25          0.51

    2  user  SQL statements in session.
    0  internal SQL statements in session.
    2  SQL statements in session.
********************************************************************************
```

The amount of CPU and DB time is:

Total CPU time = 24.58 sec
Total DB Time = 284.16 sec

As a percentage of DB Time:

Wait: "read by other session" = 158.10 / 284.16 = 55.6%
CPU = 24.58 / 284.16 * 100% = 8.7%
Wait: "db file scattered read" = 7.11 / 284.16 = 2.5%

2. ***Review the ASH, AWR, or Statspack Report***

ASH Report:

- ***Identify the main bottlenecks in the ASH report by viewing the Top User Events (order by "% Activity")***

  For example:

```
Top User Events              DB/Inst: DB10GR2/DB10gR2  (Feb 12 15:12 to 15:27)

                                                          Avg Active
Event                                Event Class     % Activity   Sessions
---------------------------------- --------------- ---------- ----------
read by other session                User I/O             53.40       5.40
CPU + Wait for CPU                   CPU                  37.24       3.77
db file scattered read               User I/O              4.90       0.50
```

AWR / Statspack Reports:

- ***Identify the main bottlenecks of the AWR and statspack reports by viewing the Top 5 Timed Events, % Total Call Time***

  For example:

```
Top 5 Timed Events                                                    Avg %Total
~~~~~~~~~~~~~~~~~~                                                   wait   Call
Event                                            Waits    Time (s)   (ms)   Time
---------------------------------------- ------------ ----------- ------ ------
read by other session                         15,436       1,752    113   79.6
CPU time                                                      268          12.2
db file scattered read                        40,981          98      2    4.5
PL/SQL lock timer                                 60          60   1001    2.7
db file sequential read                       12,504          11      1     .5
```

3. ***Compare the Timed Events***

If the percentage of the top timed events of the TKProf match the percentages in the instance-wide reports, then the trace has captured at a session level what was seen at an instance-wide level.

In the example above, the event read by other session is the most significant wait in the AWR report and the TKProf reports. The 10046 / TKProf are a good representation at the session level for what is happening instance-wide.

Proceed to the analysis section to determine a cause for the timed events seen in the TKProf report if you have confirmed the TKProf / trace is valid.

Next Step - Analyze

In the following step, you will receive guidance on interpreting the data you collected to determine the cause for the performance problem; click "NEXT" to continue.

Slow Database > Determine a Cause >Analysis

The data collected in the previous step will be analyzed in this step to determine a cause.
Database tuning is often an iterative process where bottlenecks are identified and removed with each iteration. If performance doesn't reach your goals after implementing a solution, be sure to re-identify the issue and repeat this process.

Determine the Type of Performance Problem

Using previously collected data, you need to determine the most significant bottleneck in the database. If you have already determined the type of problem, proceed to the *Tuning Strategy* section below.

CPU Consumption

An Oracle CPU performance problem can be identified by symptoms seen at the OS level and at the database level. For details on what to look for, see the section *Slow Database > Identify the Issue > Analysis > Verify Oracle OS Resource Usage*.

In summary:

1. ***OS Symptoms***

OS performance data shows:

- ***OS CPU utilization greater than 90%***
- ***OS run queue size per CPU is greater than 4***
- ***Most of the CPU is used by Oracle processes***

2. ***Database Symptoms***

Database performance data shows:

- ***TKProf shows most of the session's time is spent for CPU rather than waits (or if database waits account for a small amount of the total database elapsed time)***
- ***AWR or statspack shows that CPU is the top component of DB time***
- ***Oracle 10g+: Time model data shows CPU usage is the major component of DB Time***

If your data shows symptoms of a CPU consumption problem, proceed to the section below called, "Choose a Tuning Strategy" and select the "Reduce CPU Consumption" strategy.

Wait Bottleneck

This section will show you how to confirm a wait bottleneck and which waits are slowing down the database.

If your data shows symptoms of a wait bottleneck, proceed to the section below called, "Choose a Tuning Strategy" and select the "Reduce Wait Bottlenecks" strategy.

1. ***TKProf***

Compare overall CPU time to overall elapsed time. If non-idle waits account for most of the elapsed time, then you have a wait bottleneck problem otherwise its a CPU consumption problem.

If you have a wait bottleneck:

- ***Examine the waits to find the largest ones***
- ***Go to the Choose a Tuning Strategy Section, Reduce Wait Bottlenecks to determine a cause for the waits***

For example:

In this case, we see a large bottleneck for the read by other session waits. There are also waits for the CPU (i. e., significant run queue size due to CPU saturation) which accounts for the difference between the total elapsed time and the sum of the CPU time and wait times.

```
********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      0      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        0      0.00       0.00          0          0          0           0

Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  read by other session                          14        0.20         1.54
  db file scattered read                         30        0.00         0.01
  db file sequential read                         7        0.00         0.00
  PL/SQL lock timer                               6        1.02         6.00


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute     57      0.06       0.02          0          5         28          24
Fetch       33     24.52     284.13      17025     312555          0         416
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       90     24.58     284.16      17025     312560         28         440

Misses in library cache during parse: 0

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  read by other session                         1374        0.96        158.10
  db file scattered read                        2622        0.57          7.11
  db file sequential read                        721        0.23          0.81
  latch: cache buffers chains                      5        0.25          0.51

    2  user  SQL statements in session.
    0  internal SQL statements in session.
    2  SQL statements in session.
********************************************************************************
```

2. AWR / Statspack Reports

Review the section, Top 5 Timed Events. If CPU accounts for most of the call time, then you have a CPU consumption problem; otherwise a wait bottleneck.

If you have a wait bottleneck:

- ***Examine the waits to find the largest ones***
- ***Go to the Choose a Tuning Strategy Section, Reduce Wait Bottlenecks to determine a cause for the waits***

For example:

In this case, we see a large bottleneck for the read by other session waits (79.6% of total call time):

```
Top 5 Timed Events                                                    Avg %Total
~~~~~~~~~~~~~~~~~~                                                   wait   Call
Event                                            Waits    Time (s)   (ms)   Time
----------------------------------------- ------------ ----------- ------ ------
read by other session                         15,436       1,752    113   79.6
```

```
CPU time                                          268          12.2
db file scattered read                     40,981          98     2    4.5
PL/SQL lock timer                              60          60  1001    2.7
db file sequential read                    12,504          11     1     .5
```

3. ***ASH Reports***

Review the section, Top User Events, % Activity. If CPU accounts for most of the call time, then you have a CPU consumption problem; otherwise a wait bottleneck.

If you have a wait bottleneck:

- ***Examine the waits to find the largest ones***
- ***Go to the Choose a Tuning Strategy Section, Reduce Wait Bottlenecks to determine a cause for the waits***

For example:

In this case, we see a large bottleneck for the read by other session waits (53.4% of the session activity):

```
Top User Events              DB/Inst: DB10GR2/DB10gR2  (Feb 12 15:12 to 15:27)

                                                                 Avg Active
Event                                 Event Class     % Activity   Sessions
----------------------------------- --------------- ---------- ----------
read by other session                 User I/O             53.40       5.40
CPU + Wait for CPU                    CPU                  37.24       3.77
db file scattered read                User I/O              4.90       0.50
```

Client Bottleneck

Client bottlenecks are detected by observing that sessions spend most of their time waiting for an event outside of the database (usually *SQL*Net message from client*). These can be:

- In between any calls to the database
- In between a set of FETCH calls for one or more of the same cursors

Consider the steps to diagnosing client waits:

1. ***Confirm that client bottlenecks are occurring***

Client waits can be seen in aggregate in a TKProf file by looking at the bottom where the OVERALL TOTALS section shows the total database call elapsed time and the total waits for the SQL*Net message from client events.

For example:

```
********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        6      0.04       0.04          0          0          0           0
Execute      7      0.00       0.00          0          0          0           0
Fetch      100      1.92       2.08       6655       7744          0        1404
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      113      1.96       2.12       6655       7744          0        1404

Misses in library cache during parse: 6
Misses in library cache during execute: 1

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                      109        0.00          0.00
  SQL*Net message from client                    109       39.96        187.32
  db file sequential read                        393        0.00          0.04
  db file scattered read                         490        0.07          0.50
  SQL*Net break/reset to client                    2        0.00          0.00


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      0      0.00       0.00          0          0          0           0
```

```
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        0      0.00       0.00          0          0          0           0

Misses in library cache during parse: 0

    7  user  SQL statements in session.
    0  internal SQL statements in session.
    7  SQL statements in session.
********************************************************************************
```

In this example we see the total database time is only a small percentage of the total time:

Note: Add NON-RECURSIVE and RECURSIVE times

- ***Database elapsed time = total elapsed = 2.12 + 0 = 2.12 sec***
- ***Client Idle time = SQL*Net message from client = 187.32 + 0 = 187.32 sec***
- ***Total Time = Database elapsed time + Client Idle time = 2.12 + 187.32 = 189.44 sec***
- ***% Client Idle Time = Client Idle Time / Total Time = 187.32 / 189.44 * 100% = 98.9%***

Since almost 99% of the time is time waiting for the client, this warrants further investigation to see if the waits can be addressed by reducing the number of fetch calls or speeding up the clients.

2. ***Check if most of the elapsed time is spent waiting between ANY TYPE of call***

The best way to see this is to examine the raw 10046 trace file and observe the waits for SQL*Net message from client events shown on lines beginning with "WAIT #0". These waits aren't part of any open cursor but represent waits for a new call from the client.

For example:

```
FETCH #2:c=0,e=2829,p=5,cr=11,cu=0,mis=0,r=9,dep=0,og=1,tim=4144651383971
WAIT #2: nam='SQL*Net message from client' ela= 646165 driver id=1650815232 #bytes=1 p3=0
obj#=51516 tim=4144652030609
STAT #2 id=1 cnt=100 pid=0 pos=1 obj=51516 op='TABLE ACCESS BY INDEX ROWID BIG1 (cr=167 pr=83
pw=0 time=1057 us)'
STAT #2 id=2 cnt=100 pid=1 pos=1 obj=59541 op='INDEX RANGE SCAN I_BIG1 (cr=10 pr=0 pw=0
time=161 us)'
WAIT #0: nam='SQL*Net message to client' ela= 10 driver id=1650815232 #bytes=1 p3=0 obj#=51516
tim=4144652031862
*** 2007-02-13 19:55:33.322
WAIT #0: nam='SQL*Net message from client' ela= 30933361 driver id=1650815232 #bytes=1 p3=0
obj#=51516 tim=4144682965398
=====================
PARSING IN CURSOR #1 len=74 dep=0 uid=54 oct=3 lid=54 tim=4144682972867 hv=1218416003
```

```
ad='88be1a58'
select object_id,object_name from big1 where object_id between 151 and 200
END OF STMT
PARSE #1:c=10000,e=6895,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=4144682972835
EXEC #1:c=0,e=327,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=4144682973675
WAIT #1: nam='SQL*Net message to client' ela= 13 driver id=1650815232 #bytes=1 p3=0 obj#=51516
tim=4144682973960
WAIT #1: nam='db file sequential read' ela= 148 file#=4 block#=29933 blocks=1 obj#=51516
tim=4144682974666
FETCH #1:c=0,e=768,p=1,cr=4,cu=0,mis=0,r=1,dep=0,og=1,tim=4144682974941
```

In the above trace, we see two kinds of WAITs: WAIT #0 and WAIT #1. The # is associated with a cursor when its greater than 0. In the example, the "WAIT #0, SQL*Net message from client" line is a wait after the last fetch was completed and the database was waiting for a new call. The "WAIT #1, SQL*Net message from client" line was for the client but for an open cursor that was being fetched. If fewer fetches were done, the waits on lines like "WAIT #1" could've been reduced. The "WAIT #0" could only be reduced by a faster client.

3. Check if most of the elapsed time is spent waiting between FETCH calls

Evidence of waits between calls can be spotted by looking at the following:

1) In the TKProf, you will notice the total time spent in the database is small compared to the time waited by the client. You will also see the bulk of the time in "SQL*Net message from client" in the waits section, as shown below:

TKProf of a session where the client used an arraysize of 2 and caused many fetch calls

```
select empno, ename from  emp

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        8      0.00       0.00          0         14          0          14
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       10      0.00       0.00          0         14          0          14


Rows     Row Source Operation
-------  ---------------------------------------------------
     14  TABLE ACCESS FULL EMP (cr=14 pr=0 pw=0 time=377 us)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                        8        0.00          0.00
  SQL*Net message from client                      8       29.36         78.39
```

Notice above: 8 fetch calls to return 14 rows. 78.39 seconds waiting for "SQL*Net message from client" for 8 waits. Each wait corresponds to each fetch call. The total database time was 377 microSeconds, but the total

elapsed time to fetch all 14 rows was 78.39 seconds due to client waits. If you reduce the number of fetches, you will reduce the overall elapsed time. In any case, the database is fine, the problem is really external to the database.

2) To confirm whether the waits are due to a slow client, examine the 10046 trace for the SQL statement and look for WAITs in between FETCH calls, as follows:

```
PARSING IN CURSOR #2 len=29 dep=0 uid=57 oct=3 lid=57 tim=1016349402066 hv=3058029015 ad='94239ec0'
select empno, ename from emp
END OF STMT
PARSE #2:c=0,e=5797,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1016349402036
EXEC #2:c=0,e=213,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1016349402675
WAIT #2: nam='SQL*Net message to client' ela= 12 p1=1650815232 p2=1 p3=0
FETCH #2:c=0,e=423,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=4,tim=1016349403494 <== Call Finished
WAIT #2: nam='SQL*Net message from client' ela= 1103179 p1=1650815232 p2=1 p3=0 <== Wait for client
WAIT #2: nam='SQL*Net message to client' ela= 10 p1=1650815232 p2=1 p3=0
FETCH #2:c=0,e=330,p=0,cr=1,cu=0,mis=0,r=2,dep=0,og=4,tim=1016350507608 <== Call Finished (2 rows)
WAIT #2: nam='SQL*Net message from client' ela= 29367263 p1=1650815232 p2=1 p3=0 <== Wait for client
WAIT #2: nam='SQL*Net message to client' ela= 9 p1=1650815232 p2=1 p3=0
FETCH #2:c=0,e=321,p=0,cr=1,cu=0,mis=0,r=2,dep=0,og=4,tim=1016379876558 <== Call Finished (2 rows)
WAIT #2: nam='SQL*Net message from client' ela= 11256970 p1=1650815232 p2=1 p3=0 <== Wait for client
WAIT #2: nam='SQL*Net message to client' ela= 10 p1=1650815232 p2=1 p3=0
. . .
FETCH #2:c=0,e=486,p=0,cr=1,cu=0,mis=0,r=1,dep=0,og=4,tim=1016409054527
WAIT #2: nam='SQL*Net message from client' ela= 18747616 p1=1650815232 p2=1 p3=0
STAT #2 id=1 cnt=14 pid=0 pos=1 obj=49049 op='TABLE ACCESS FULL EMP (cr=14 pr=0 pw=0 time=377 us)'
```

Notice: Between each FETCH call, there is a wait for the client. The client is slow and responds every 1 - 2 seconds.

Client or network bottlenecks are often associated with these wait events:

- ***SQL*Net message from client; waiting for the client***
- ***SQL*Net more data from client; waiting for additional data from the client***
- ***SQL*net more data to client; lots of data being sent back to the client (maybe not strictly a client wait, but usually best to solve from a client's point of view)***

If the TKProf / trace files show symptoms of a client bottleneck, go to the Tuning Strategy section below under Client Bottlenecks to identify potential causes and solutions.

If your data shows symptoms of a client bottleneck, proceed to the section below called, "Choose a Tuning Strategy" and select the "Reduce Client Bottlenecks" strategy.

Oracle Memory Consumption

Memory consumption problems are described in detail in the section, *Slow Database > Identify the Issue > Analysis > Verify Oracle OS Resource Usage > Check Memory Consumption*. Please review the analysis techniques there.

If your data shows symptoms of a memory consumption problem, proceed to the section below called, "Choose a Tuning Strategy" and select the "Reduce Oracle Memory Consumption" strategy.

Proceed to the next section to analyze the cause for the bottleneck.

Choose a Tuning Strategy

Choose one of the tuning strategies below depending on the kind of performance problem that was verified earlier.

Oracle 10g+ ==> Use ADDM to Tune the Database

Oracle 10g is able to perform automated tuning analysis using ADDM. This is the preferred way to begin a tuning effort if you are using Oracle 10g. You can always tune the database manually as a last resort.

Note: You must be licensed for the "Tuning Pack" to use ADDM.

Reduce CPU Consumption

There are two major types of CPU usage in the database:

- **Parse CPU** : CPU used whenever Oracle parses (and optimizes) a statement.
- **Non-Parse CPU**: CPU usage by Oracle NOT involving parsing. This can be for things like reading blocks in the buffer cache, performing a sort or a join, reading a file, etc. This is also called "Other" CPU.

CPU is also tracked as "Recursive CPU". This means CPU was used by a statement running "underneath" another statement. Typically this is due to SQL issued by PL/SQL or by internal SQL statements that Oracle has to run to process the top level query or operation. In general, one can just focus on parse CPU and non-parse CPU because recursive CPU contains both and is not as useful for tuning.

Determine CPU Usage Type from TKProf

- **Parse CPU**: In the Overall Totals section, add the Parse CPU for both recusive and non-recursive statements
- **Total CPU**: In the Overall Totals section, add the Total CPU for both recusive and non-recursive statements
- **Non-parse CPU**: total CPU - parse CPU
- Determine which is larger: parse CPU or non-parse CPU

Consider the following causes for high parse CPU or other CPU:

1. ***Parse CPU Usage***

Facts Required for Analysis:

- ***Is most of the parse time due to one (or a few queries) OR due to all queries?***
  Check this by generating a new TKProf report sorted by parse CPU time as follows:

  ```
  tkpof trace_file_name output_file sort=prscpu
  ```

  Review the parse CPU values of the queries at the top of the file and work your way down to see if parse CPU usage was widespread.

- ***Are the queries with high parse time being HARD parsed?***
  Look at the queries in the TKProf with high parse CPU and see if "Misses in library cache during parse" is close to the total number of parses for that statement.

Review the following common observations to see if any match your data:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

One or a few queries with High CPU usage during HARD parse

High CPU usage during hard parses are often seen with large statements involving many objects or partitioned objects.

What to look for

1. ***Check if the statement was hard parsed***
2. ***Compare parse cpu time to parse elapsed time to see if parse cpu time is more than 50%***

Cause Identified: Dynamic sampling is being used for the query and impacting the parse time

Dynamic sampling is performed by the CBO (naturally at parse time) when it is either requested via hint or parameter, or by default because statistics are missing. Depending on the level of the dynamic sampling, it may take some time to complete - this time is reflected in the parse time for the statement.

Cause Justification

- ***The parse time is responsible for most of the query's overall elapsed time***
- ***The execution plan output of SQLTXPLAIN, the UTLXPLS script, or a 10053 trace will show if dynamic sampling was used while optimizing the query.***

Solution Identified: Alternatives to Dynamic Sampling

If the parse time is high due to dynamic sampling, alternatives may be needed to obtain the desired plan without using dynamic sampling.

M ***Effort Details***

Medium effort; some alternatives are easy to implement (add a hint), whereas others are more difficult (determine the hint required by comparing plans)

L ***Risk Details***

Low risk; in general, the solution will affect only the query.

Solution Implementation

Some alternatives to dynamic sampling are:

1. ***In 10g or higher, use the SQL Tuning Advisor (STA) to generate a profile for the query (in fact, its unlikely you'll even set dynamic sampling on a query that has been tuned by the STA)***
2. ***Find the hints needed to implement the plan normally generated with dynamic sampling and modify the query with the hints***
3. ***Use a stored outline to capture the plan generated with dynamic sampling***

For very volatile data (in which dynamic sampling was helping obtain a good plan), an approach can be used where an application will choose one of several hinted queries depending on the state of the data (i.e., if data recently deleted use query #1, else query #2).

Documents for hints:

Using Optimizer Hints

Forcing a Known Plan Using Hints

How to Specify an Index Hint

QREF: SQL Statement HINTS

Documents for stored outlines / plan stability:

Using Plan Stability

Stored Outline Quick Reference

How to Tune a Query that Cannot be Modified

How to Move Stored Outlines for One Application from One Database to Another

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Query has many IN LIST parameters / OR statements

The CBO may take a long time to cost a statement with dozens of IN LIST / OR clauses.

Cause Justification

- *The parse time is responsible for most of the query's overall elapsed time*
- *The query has a large set of IN LIST values or OR clauses.*

Solution Identified: Implement the NO_EXPAND hint to avoid transforming the query block

In versions 8.x and higher, this will avoid the transformation to separate query blocks with UNION ALL (and save parse time) while still allowing indexes to be used with the IN-LIST ITERATOR operation. By avoiding a large number of query blocks, the CBO will save time (and hence the parse time will be shorter) since it doesn't have to optimize each block.

L *Effort Details*

Low effort; hint applied to a query.

L *Risk Details*

Low risk; hint applied only to the query and will not affect other queries.

Solution Implementation

See the reference documents.

Optimization of large inlists/multiple OR`s

NO_EXPAND Hint

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Partitioned table with many partitions

The use of partitioned tables with many partitions (more than 1,000) may cause high parse CPU times while the CBO determines an execution plan.

Cause Justification

1. ***The parse time is responsible for most of the query's overall elapsed time***
2. ***Determine total number of partitions for all tables used in the query.***
3. ***If the number is over 1,000, this cause is likely***

Solution Identified: 9.2.0.x, 10.0.0: Bug 2785102 - Query involving many partitions (>1000) has high CPU/ memory use

A query involving a table with a large number of partitions takes a long time to parse, causes rowcache contention, and high CPU consumption. The case of this bug involved a table with greater than 10000 partitions and global statistics ere not gathered.

M ***Effort Details***

Medium effort; application of a patchset.

L ***Risk Details***

Low risk; patchsets generally are low risk because they have been regression tested.

Solution Implementation

Apply patchset 9.2.0.4

Workaround:
Set "_improved_row_length_enabled"=false

Additional bug information:
Bug 2785102

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Many queries being HARD parsed

Hard parsing is costly for the database since it has to create various memory structures in the library cache and also optimize the SQL statement. If many queries are being hard parsed, parse CPU will be high.

What to look for

1. *Check if many statements were hard parsed*

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the presence of literal values.*

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M *Effort Details*

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

*Pro*C/C++ Precompiler Programmer's Guide*

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- ***EXACT: Leave the statement as it was written with literals (default value)***
- ***FORCE: Substitute all literals with binds (as much as possible)***
- ***SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)***

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L ***Effort Details***

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M ***Risk Details***

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; increasing the size of the shared pool is not risky unless:

- *There are many unshared statements due to literals*
  For more details, see: Understanding and Tuning the Shared Pool
- *The machine doesn't have enough physical memory and starts swapping*
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M *Effort Details*

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M *Risk Details*

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*

- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

2. ***Non-Parse CPU Usage***

Non-Parse CPU usage is usually due to poor performing SQL statements or PL/SQL procedures. In this case, CPU is used for these two types of calls:

- ***EXECUTE: tracks CPU usage by PL/SQL, DML, and DDL statements. This also includes CPU used to retrieve rows affected by the DML statement.***
- ***FETCH: tracks CPU usage by SELECT statements when rows are being accessed and being prepared to return to the client. This includes the effort to traverse indexes, read blocks, perform join operations, and basically follow a query's execution plan to obtain rows.***

Facts Required for Analysis:

- ***Is most of the non-parse CPU time due to one (or a few queries) OR due to all queries? Check this by generating a new TKProf report sorted by fetch and execute CPU time as follows:***

  ```
  tkpof trace_file_name output_file sort=fchcpu,execpu
  ```

  Review the fetch and execute CPU values of the queries at the top of the file and work your way down to see if this type of CPU usage was widespread.

- ***Is most of the CPU time spent for fetching or executing?***
  If most time is spent executing, focus attention on DML and PL/SQL, otherwise on SELECT statements

Review the following common observations to see if any match your data:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

One or a few queries use most non-parse CPU

One or a few queries stand out as the heaviest users of non-parse CPU time. This signifies that those particular queries need to be tuned.

What to look for

- *TKProf: Only a few statements consume most of the total CPU usage (top statements when TKProf is sorted by fetch and execute CPU time)*
- *AWR or statspack: Only a few SQL statements are reported to have the highest CPU usage, and these statements' CPU usage is responsible for most of the database's CPU time (as reported in the Top 5 Timed Events section)*

Cause Identified: SQL tuning required

If one or a few statements use most of the fetch or execute time, then these statements need to be tuned.

Cause Justification
Most of the CPU time either in the entire instance (shown in AWR or statspack) or within a session (shown in TKProf) is consumed by one or a few statements.

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To
How to use the Sql Tuning Advisor

Documentation
Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference
SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Reduce Wait Bottlenecks

Review the wait categories below to find the main waits that are affecting your database. The categories contain wait events that are most commonly found in performance issues - the list is NOT exhaustive but does cover events found causing problems in over 90% of wait-related performance problems with the database.

1. ***Wait Events Summary***

Once you have determined the wait bottlenecks, you can examine possible causes and solutions in the sections below. The various wait events are categorized according to the following table:

Event Name	*Category*
buffer busy waits	*Concurrency - Buffer Busy Waits*
db file scattered read	*Reads / Writes*
db file sequential read	*Reads / Writes*
direct path read	*Reads / Writes*
direct path write	*Reads / Writes*
enq: TM - contention	*Concurrency - Enqueues / Locks / Pins*
enq: TX - contention	*Concurrency - Enqueues / Locks / Pins*
enq: TX - row lock contention	*Concurrency - Enqueues / Locks / Pins*
free buffer waits	*Reads / Writes*
global cache cr request	*Cluster*
latch: cache buffers chains	*Concurrency - Latches and Mutexes*
latch: library cache	*Concurrency - Latches and Mutexes*
latch: shared pool	*Concurrency - Latches and Mutexes*
library cache lock	*Concurrency - Enqueues / Locks / Pins*
library cache pin	*Concurrency - Enqueues / Locks / Pins*
log buffer space	*Configuration*
log file sync	*Commit*
read by other session	*Concurrency - Buffer Busy Waits*

See the 10gR2 documentation for Wait Events Statistics for helpful information on specific wait events.

2. ***Cluster***

Waits related to Real Application Cluster resources (for example, global cache resources such as 'gc buffer busy'. Typical events:

Oracle 10g:

- ***gc buffer busy***
- ***gc cr request***
- ***gc cr block 2-way***
- ***gc cr block 3-way***
- ***gc current block busy***
- ***gc current block 2-way***
- ***gc current block 3-way***
- ***gc current block busy***

Oracle 9.2.x:

- ***TBD***

Oracle 9.0.1.x:

- ***TBD***

Facts Required for Analysis:

- ***TKProf, elapsed times for events (Overall Totals, recursive and non-recursive):***
  - ***Total wait time for the event***
  - ***Average wait for the event = total wait time / total waits***

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: global cache CR request

The event is waited for when a session is looking for a consistent read version of a block but cannot find it in its local cache. It also implies that the current block is not cached locally. The wait ends when either a block or a grant arrives. Depending on whether the remote instance has the block cached or not, the requesting instance receives

- *A CR block, resulting in the statistic global cache cr block received to be incremented*
- *A grant, resulting in the statistic global cache gets to be incremented*
- *(9i RAC Only) A current block, resulting in the statistic global cache current blocks received to be incremented.*

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for global cache CR request waits.*
- *AWR or statspack:*
  - *Significant waits for global cache CR request*

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L *Effort Details*

Low effort

L *Risk Details*

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To

How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference

Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Inefficient SQL causing too many block reads across nodes

A poorly performing SQL statement will require an excessive amount of reads. In a RAC database those reads may require bringing blocks from other nodes and waiting for those blocks to arrive.

Cause Justification
TKProf:

- *Significant waits on global cache CR request*
- *SQL statements perform 100 or more logical reads (query + current) per row per execution*
- *Full table scans (in a RAC database) may be seen in the execution plan for a statement that is waiting on this event*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- *Significant waits on waits , and/or for RAC, global cache CR request*
- *SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution*
- *Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event*
- *The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).*

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*

- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: More lock manager processes are needed

The database may require more lock manager processes to meet the demands of the database. When the lock managers are too busy, block transfers will take longer and cause waits for these blocks.

Cause Justification
TKProf:

- *Significant waits on global cache CR request*
- *SQL statements perform 100 or fewer logical reads (query + current) per row per execution, meaning that the statement is reasonably tuned*
- *Full table scans (in a RAC database) are not seen in the execution plan for a statement that is waiting on this event*

OS data:

- *The LMD process is very busy for the instance, possibly using as much as one CPU on a consistent basis*

Solution Identified: Increase the number of lock manager processes

Increase the number of Lock Manager processes for the instance by altering the value of the init.ora parameter _LM_DLMD_PROCS

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk.

Solution Implementation

See the documents below.

TBD

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

3. *Commit*

This wait class only comprises one wait event - wait for redo log write confirmation after a commit (that is, 'log file sync').

- ***log file sync***

Facts Required for Analysis:

- ***TKProf, elapsed times for events (Overall Totals, recursive and non-recursive):***
  - ***Total wait time for the event***
  - ***Average wait for the event = total wait time / total waits***
- ***AWR or statspack report***
  - ***Waits, average wait time for log file sync***
  - ***Instance Statistics, user commits***
  - ***Instance Statistics, user calls***
  - ***Calculate, user calls / commit***

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: log file sync

When a user session commits (or rolls back), the session's redo information must be flushed to the redo logfile by the LGWR background process. This event shows the time that it takes for the LGWR to complete the write and then post the requester. The server process performing the COMMIT or ROLLBACK waits under this event for the write to the redo log to complete.

Wait class: Commit, typically foreground

What to look for

- ***TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for log file sync waits.***
- ***AWR or statspack: log file sync waits is among the top timed events***

Cause Identified: Frequent commits by the application

The application is committing frequently (and possibly unnecessarily

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***In the AWR or Statspack report, the average wait time for log file sync is much higher than the average wait time for log file parallel write - meaning that most of the wait for log writer is NOT due to waiting for the redo to be written***
- ***In the AWR or Statspack report, the average user commits / user call is less than 30 - meaning that commits are happening frequently***

Solution Identified: Reduce the rate of commits or rollbacks

Look into the application and determine if more rows can be processed per commit. Sometimes a developer will allow the underlying language to "auto-commit" by default; this is suboptimal and should be controlled by the developer.

If the ratio of rollbacks to commits is more than 10 percent, investigate if this is unexpected or can be avoided. Rollback operations will cause the logwriter to flush redo and induce waits on log file sync waits just as commits would.

M ***Effort Details***

Medium effort; this will require some work and coordination with developers to examine their code.

L ***Risk Details***

Low risk; however, the business needs must be well understood to commit at the right times.

Solution Implementation

See the documents below.

Reference

WAITEVENT: "log file sync" Reference Note

WAITEVENT: "log file parallel write" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Redolog file write performance problems

Logwriter is not able to write to the redo log files efficiently; writes are taking too long.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *In the AWR or Statspack report, the average wait time for log file sync is very similar to the average wait time for log file parallel write - meaning that most of the wait for log writer is due to waiting for the redo to be written*
- *The average time for the log file parallel write event is more than 20msec*
- *In the AWR or Statspack report, the average user commits / user call is more than 30 - meaning that commits are NOT happening frequently*

Solution Identified: Investigate redolog file write performance

Work with the system administrator to examine the filesystems where the redologs are located. Look for other processes that may be writing to that same location or a capacity problem.

M *Effort Details*

Medium effort; this will require some work and coordination with system administrators to examine the filesystems. The redolog files may need to be moved.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

See the documents below.

Reference

WAITEVENT: "log file sync" Reference Note

WAITEVENT: "log file parallel write" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L *Effort Details*

Low effort

L *Risk Details*

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To
How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference
Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

4. Concurrency - Buffer Busy Waits

Waits for miscellaneous internal database resources used to coordinate operations. Typical events:

- ***buffer busy waits***
- ***read by other session***

Facts Required for Analysis:

The key is to determine which segments and statements are causing the performance problems. Please read the note, How to Identify The Segment Associated with Buffer Busy Waits for more details.

After you determine which segment is associated with the buffery busy waits, examine the table below for common causes.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: Buffer busy waits

Buffer busy waits indicate that there are some buffers in the buffer cache that multiple processes are attempting to either access concurrently while its being read from disk or waiting for another session's block change to complete. In this case (buffer busy wait > data block), the contention is on the actual block where the data is stored, and can be either a table or an index.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for buffer busy waits.***
- ***AWR or statspack:***
  - ***Oracle 9.2 or higher: buffer busy waits is among the top timed events***

Cause Identified: Heavy insert activity with poor freelist configuration

Concurrent INSERTs with a suboptimal freelist configuration can lead to buffer busy wait contention as multiple sessions attempt to insert data into the same block (because it appears on the freelist to them).

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are INSERT statements.***

AWR or statspack reports:

- ***buffer busy wait event is among the top ones***
- ***SQL with highest wait time (derive as elapsed time - cpu time) are INSERT statements***

Solution Identified: Use ASSM or add additional freelists and/or freelist groups

Heavy INSERT activity by concurrent sessions can cause multiple sessions to attempt their insert into the same blocks because automatic segment space management (ASSM) is NOT used AND there is only a single freelist, too few process freelists, and/or no freelist groups.

The best solution is to use ASSM since it is sometimes tricky to arrive at a correct freelist or freelist group setting.

Adding process freelists will help remove contention as each process will map to separate blocks. Freelists can be added at any time without rebuilding the table.

Adding freelist groups will also remove contention by mapping processes to other freelists. This is of greatest benefit in RAC environments where the freelist group block itself will be associated with an instance, but will still help in single instance environments as well. The table must be rebuilt to change the freelist group setting.

L ***Effort Details***

Medium effort; may require rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Freelists

Performance Tuning: Buffer Busy Waits, Segment Header Contention

Admin Guide: Specifying Segment Space Management in Locally Managed Tablespaces

Reference

WAITEVENT: "buffer busy waits" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Heavy insert activity affecting an index segment

Concurrent INSERTs or updates may see contention when a related index has a key that is constantly increasing (e.g., a key based on a sequence number)

Cause Justification

TKProf:
- *Use the TKProf reports sorted by elapsed execute time*
- *Look at the top statements and determine if they are seeing buffer busy waits and are DML statements.*
- *The raw trace shows the buffer busy wait's file (P1) and block (P2) values resolve to an index segment*

AWR or statspack reports:
- *buffer busy wait event is among the top ones*
- *SQL with highest wait time (derive as elapsed time - cpu time) are INSERT statements*
- *9.2+ : the segments with the most buffer busy waits are indexes (as shown on the Top Buffer Busy Waits per Segment section*

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L *Effort Details*

Low effort; will require rebuilding an index.

M *Risk Details*

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M *Effort Details*

Medium effort; requires rebuilding the table.

L *Risk Details*

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Many concurrent SQL statements performing physical reads

Many concurrent physical reads against the same blocks will result in buffer busy waits as one session gets to do the actual physical read and the others will be blocked by the buffer busy wait event until the read completes.

This is usually an indication that the SQL statement must be tuned.

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are SELECT statements or DDL with a SELECT subquery.***
- ***The SQL statement performs many physical reads (i.e., disk on the TKProf); you see events like db file scattered reads or db file sequential reads taking significant amounts of time***

AWR or statspack reports:

- ***buffer busy wait event is among the top ones***
- ***SQL with highest wait time (derive as elapsed time - cpu time) are SELECT statements***
- ***Events like db file scattered reads or db file sequential reads are prominent in the top events lists***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Many concurrent SQL statements performing physical reads and I/O performance is poor

Many concurrent physical reads against the same blocks will result in buffer busy waits as one session gets to do the actual physical read and the others will be blocked by the buffer busy wait event until the read completes.

This is usually an indication that the SQL statement must be tuned. The waits can be amplified greatly when physical reads are slow due to poor I/O subsystem performance.

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are SELECT statements or DDL with a SELECT subquery.***
- ***The SQL statement performs many physical reads (i.e., disk on the TKProf); you see events like db file scattered read or db file sequential read taking significant amounts of time***
- ***The average time for db file scattered read or db file sequential read is around 20 mSec or higher***

(derive from: total wait time / waits).

AWR or statspack reports:

- *buffer busy wait event is among the top ones*
- *SQL with highest wait time (derive as elapsed time - cpu time) are SELECT statements*
- *Events like db file scattered reads or db file sequential reads are prominent in the top events lists*
- *The average time for db file scattered read or db file sequential read is around 20 mSec or higher.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- *Significant waits for the read by other session event*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

5. ***Concurrency - Enqueues / Locks / Pins***

Waits related to enqueues or locks; these usually result from user application code (for example, lock waits caused by row level locking or explicit lock commands). Typical events:

- ***enqueue***
- ***enq: HW - contention***
- ***enq: ST - contention***
- ***enq: TM - contention***
- ***enq: TX - row lock contention***
- ***enq: TX - index contention***
- ***enq: TX - allocate ITL entry***
- ***library cache load lock***
- ***library cache lock***
- ***library cache pin***
- ***PL/SQL lock timer***
- ***row cache lock***

Facts Required for Analysis:

For enqueue waits, the key is to determine which enqueue type (and mode held) is causing the performance problem. Please read the note, [How to Determine The Lock Type and Mode from an Enqueue Wait] for more details.

For all other waits, it will be helpful to at least identify the top SQL statements associated with the waits using this technique:

a. ***In the "Overall Totals" (recursive and non-recursive) section, look for wait events with high elapsed times for the lock or pin waits***
b. ***In the "Overall Totals" section, determine which call type is associated with the highest elapsed time: parse, execute, or fetch***
c. ***Generate a new TKProf report sorted by the call type found for the highest elapsed times in step b. For example:***

   For execute calls:

   ```
   tkpof trace_file_name output_file sort=exeela
   ```

   For fetch calls:

   ```
   tkpof trace_file_name output_file sort=fchela
   ```

d. ***Note the top statements in this new TKProf report - these are the main statements that are waiting.***

Examine the table below for common causes of the wait events you found.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

9.2 or prior: wait: enqueue, type TM
10g+: wait: enq: TM - contention

This could be for various reasons and is identified in pre-10g versions as waits for enqueue for the TM enqueue.
In 10g, the wait is enq: TM - contention
Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enqueue waits for TM enqueue or enq:TM - contention waits.***
  - ***In the raw 10046 trace file, most wait's P1 field decodes to TM***
- ***AWR or statspack:***
  - ***pre-10g: enqueue waits; Enqueue Activities show most waits for TM enqueue***
  - ***10g: enq:TM - contention is among the top timed events***

Cause Identified: Foreign key columns missing an index

Foreign key columns should be indexed to avoid locking issues with the parent or child tables. The exact behavior varies by version but in all versions, Oracle will use indexes to avoid locks or use a more permissive lock mode.

Cause Justification
TKProf:
- ***Lock wait is for TM enqueue, generally in mode 3 or 4***
- ***Statement involves an update to a parent or child table that has an FK constraint on a column being changed***

Solution Identified: Create indexes on the child table's foreign key columns

An index on a foreign key column will permit Oracle to either avoid or minimize lock waits when rows in the parent or child table are changed.

L ***Effort Details***

Low effort; requires creation of an index.

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

Documentation

Concepts (10gR2): No Index on the Foreign Key

Concepts (9iR2): No Index on the Foreign Key

Notes

Referential Integrity and Locking

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: ANALYZE INDEX is blocking DML statements

An index is being analyzed using the ANALYZE INDEX VALIDATE STRUCTURE command while a DML operation on the underlying table is being attempted (requiring a TM lock to be placed).

Cause Justification
The ANALYZE INDEX command acquires a TM enqueue in share mode on the underlying table; this will block other sessions when they attempt to place a TM lock that is incompatible with a share-mode lock.

The following query shows the command type for the session currently blocking another session with the TM enqueue:

```
select s.command
from v$lock l, v$session s
where l.sid = s.sid
and l.block = 1
and l.type='TM'
```

If the command type is 63 (versions 9.2 - 11.x), then an analyze index command is responsible for the blocking.

Solution Identified: Run the ANALYZE INDEX command during a maintenance window or quiet time

There is no workaround such as an "ONLINE" option for the ANALYZE INDEX VALIDATE STRUCTURE command. You will simply need to avoid the contention by scheduling the command when there is no contention likely.

L ***Effort Details***

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

L ***Risk Details***

Low risk; the analyze index command may be interrupted if necessary. The index statistics that it populates do not directly affect execution plans.

Solution Implementation

Not Applicable - solution is trivial.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Parallel DML Being Used While Other DML Performed on Same Objects

Parallel DML will acquire TM enqueues on the partitions involved (share mode) as well as the entire table (row exclusive). No other DML against affected partitions will be allowed until the PDML transaction completes.

Cause Justification
This cause is likely if there are:

- ***waits on the TM enqueue***
- ***sessions waiting are either attempting to perform PDML or are waiting for another session performing PDML***

Solution Identified: Schedule the PDML to occur during a quiet time

Schedule the PDML activity when the system is quiet to avoid impacting users.

L ***Effort Details***

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M ***Risk Details***

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use a custom parallel DML script

Sometimes its possible to avoid contention by controlling which partitions are going to concurrently receive DML through individual sessions rather than a single PDML command. This involves splitting the workload in some way and performing the DML across several sessions.

H ***Effort Details***

It could take some time to split the workload properly and script the job to run across sessions.

L ***Risk Details***

Contention can be stopped by stopping the jobs.

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

9.2 or prior: Wait: "enqueue"; TX Contention, Mode 4

TX enqueue contention in mode 4 (share mode) may be due to a variety of causes. They are not due to specific row locks but for operations related to transaction management like:

- *Lack of ITLs in a block*
- *Foreign key constraints without an index on a child table's key column*
- *Index block splits*

Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for the enqueue event*
  - *In the raw 10046 trace file, most wait's P1 field decodes to type TX, mode 4*
- *AWR or statspack:*
  - *Oracle 9.2 or prior: enqueue is among the top timed events*
  - *Enqueue Activities section shows that TX enqueues account for a significant amount of the enqueue times*
  - *Its not possible to know the modes requested without looking at the raw 10046 trace file or by looking at V$SESSION_WAIT, V$LOCK, or similar during the wait.*

Cause Identified: Insufficient ITLs in a block

Waits for the TX enqueue in mode 4 can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.

Cause Justification
Prior to 9.2.x:
In versions prior to 9.2.x, it is difficult to pinpoint an ITL wait exactly. Consider this justified if you have examined other causes for TX, mode 4 waits and none are justifiable.

9.2.x:
Using statspack snapshots taken at level 7, look in the segment statistics section to see which segments have the highest ITL waits (e.g., Segments by ITL Waits). If these are a significant portion of the TX enqueue waits (see the Enqueue Activities section), then this cause is justified.

In 10g, the wait event itself tells you this is an ITL wait, so it is justified from the wait event.

Solution Identified: Increase the table's INITRANS setting

Increase the table's INITRANS setting to account for the number of concurrent sessions changing an individual block.

L ***Effort Details***

Medium effort; may require rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects except if INITRANS is set too large and the block size is small (this will waste a lot of block space).

Solution Implementation

See the documents below.

Documentation
SQL Ref: INITRANS

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Index contention due to block splits

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).

Cause Justification
To identify which segment is involved:

- ***Look in the TKProf of one or more sessions that experience the most of this kind of wait.***
- ***Find the statement that waited the longest amount of time on the event with long TX, mode 4 waits. This is generally an insert statement.***
- ***Examine the statement to find the indexes involved.***

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L ***Effort Details***

Low effort; will require rebuilding an index.

M ***Risk Details***

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M ***Effort Details***

Medium effort; requires rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

9.2 or prior: Wait: "enqueue"; TX Contention, Mode 6

TX enqueue contention in mode 6 (exclusive mode) usually occurs when one session is updating or deleting a row, while another session wishes to update or delete the same row.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enqueue waits.***
  - ***In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 6***
- ***AWR or statspack:***
  - ***Oracle 9.2 or prior: enqueue is among the top timed events***
  - ***Enqueue Activities section shows that TX enqueues account for a significant amount of the enqueue times***
  - ***Its not possible to know the modes requested***

Cause Identified: Waiting for a row level lock due to faulty application design

Flaws in application design are often the reason for locks being held for a long time. A couple of scenarios to illustrate this are:

1) A user navigates to a certain row on a page and makes a change without committing it. The user then leaves the page for a time while the row is locked. If another user wants to update the same row, he or she will have to wait. This type of situation can be detected by identifying the blocking session (either through V$LOCK or V$SESSION.BLOCKING_SESSION in 10g) and finding out how long it has been idle using the column V$SESSION.LAST_CALL_ET.

2) The application starts a transaction and locks or updates rows then executes one or more long running queries before it commits the changes. This has the effect of holding the row locks a long time; the solution is to tune the SQL in between the row lock and the final commit.

Cause Justification

- ***Use the utllockt.sql script to identify locking problems. Focus on locks where the lock type is TX and LMode is 6 (Exclusive) and check if locks are being held for a long time.***
- ***Trace the lock HOLDER shown in the output of the utllockt.sql script using the 10046 event to see what its doing***
- ***Look for long running queries that cause row locks to be held a long time or other problems in the application***

Solution Identified: SELECT FOR UPDATE locks too many rows

Sometimes a "pessimistic" locking strategy is implemented with SELECT FOR UPDATE statements that are missing predicates and are too "greedy" with their locking. Examine these statements to see if they are locking more rows than they actually need to lock.

M ***Effort Details***

Medium effort; requires access and examination of application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. ***Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).***
2. ***Look for recent cursors that the session has executed and are still open by querying V$OPEN_CURSOR for the session identified in step 1. For example:***

```
SELECT sql_text FROM v$open_cursor WHERE sid = 1234
```

3. ***See if any cursors involve FOR UPDATE, UPDATE, or DELETE***
4. ***Examine these cursors to see if they are selecting too many rows***
5. ***Change the application to lock fewer rows at time. Sometimes this may require splitting up the work into a SELECT statement that finds candidate rows to lock and then a SELECT FOR UPDATE to lock an individual row. There are many ways to implement this kind of change - it all depends on the application.***

Sometimes, the cursor that locked the rows is no longer open and other cursors have executed since then. In these cases it is difficult to find the exact cause of the blocking without looking at the application in depth. One clue that may help is knowing the SQL for the waiting session and then examining the application code for other places and situations where the tables in the SQL statement may be locked. You can find the SQL and exact ROWID being waited on by issuing the following query:

```
select s.sid, s.serial#, s.username, s.module, s.ROW_WAIT_OBJ# object_id,
 dbms_rowid.rowid_create(1, s.row_wait_obj#, s.row_wait_file#,
 s.row_wait_block#, s.row_wait_row#) my_rowid
 s.sql_hash_value, s.sql_address, sq.SQL_TEXT,
 from v$session_wait sw, v$session s, v$sql sq, v$lock l
 where sw.event = 'enqueue'
 and sw.sid = s.sid
 and l.type = 'TX' and l.request = 6
 and l.sid = s.sid
 and s.sql_hash_value = sq.hash_value and s.sql_address = sq.address
```

Note: "object_id" can be used to query DBA_OBJECTS.

Documentation

Concepts: Data Concurrency and Consistency

How Oracle Locks Data

Concepts: Row Locks (TX)

Notes

Tracing sessions: waiting on an enqueue

TX Transaction locks - Example wait scenarios

TX Lock "Transaction Enqueue"

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Long running statement while locks are held

A long running statement may delay the time between a transaction starts (via some DML) and commits. This statement may need to be tuned.

M ***Effort Details***

Medium effort; requires access and examination of application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. ***Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).***
2. ***For the session identified in step 1 (lock holder), check if it is currently executing SQL. For example, assuming the session ID is 12:***

```
SELECT s.sid, s.status, sq.sql_text
FROM v$session s, v$sql sq
WHERE s.sid = 12
and s.status = 'ACTIVE'
and s.sql_hash_value = sq.hash_value
and s.sql_address = sq.address
```

3. ***Investigate the performance of this cursor. At this point, the problem becomes a query tuning problem. You can use this guide for help or the SQL Tuning Advisor (10g or higher with EM Tuning Pack license).***
4. ***Tune the query and evaluate whether the locking problem has been resolved. If it hasn't been resolved, examine the application in more detail to see if the application should be changed.***

See the documents below for more information.

10g+: SQL Tuning Advisor

How to use the Sql Tuning Advisor

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

TBD

Manual Tuning

SQL Tuning Overview

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX row lock contention

In 10g, enq: TX - row lock contention in mode 6 (exclusive mode) usually occurs when one session is updating or deleting a row, while another session wishes to update or delete the same row.

Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq: TX row lock contention waits.*
  - *optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 6*
- *AWR or statspack:*
  - *enq: TX - row lock contention is among the top timed events*

Cause Identified: Waiting for a row level lock due to faulty application design

Flaws in application design are often the reason for locks being held for a long time. A couple of scenarios to illustrate this are:

1) A user navigates to a certain row on a page and makes a change without committing it. The user then leaves the page for a time while the row is locked. If another user wants to update the same row, he or she will have to wait. This type of situation can be detected by identifying the blocking session (either through V$LOCK or V$SESSION.BLOCKING_SESSION in 10g) and finding out how long it has been idle using the column V$SESSION.LAST_CALL_ET.

2) The application starts a transaction and locks or updates rows then executes one or more long running queries before it commits the changes. This has the effect of holding the row locks a long time; the solution is to tune the SQL in between the row lock and the final commit.

Cause Justification

- *Use the utllockt.sql script to identify locking problems. Focus on locks where the lock type is TX*

and LMode is 6 (Exclusive) and check if locks are being held for a long time.

- *Trace the lock HOLDER shown in the output of the utllockt.sql script using the 10046 event to see what its doing*
- *Look for long running queries that cause row locks to be held a long time or other problems in the application*

Solution Identified: SELECT FOR UPDATE locks too many rows

Sometimes a "pessimistic" locking strategy is implemented with SELECT FOR UPDATE statements that are missing predicates and are too "greedy" with their locking. Examine these statements to see if they are locking more rows than they actually need to lock.

M ***Effort Details***

Medium effort; requires access and examination of application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. *Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).*
2. *Look for recent cursors that the session has executed and are still open by querying V $OPEN_CURSOR for the session identified in step 1. For example:*

```
SELECT sql_text FROM v$open_cursor WHERE sid = 1234
```

3. *See if any cursors involve FOR UPDATE, UPDATE, or DELETE*
4. *Examine these cursors to see if they are selecting too many rows*
5. *Change the application to lock fewer rows at time. Sometimes this may require splitting up the work into a SELECT statement that finds candidate rows to lock and then a SELECT FOR UPDATE to lock an individual row. There are many ways to implement this kind of change - it all depends on the application.*

Sometimes, the cursor that locked the rows is no longer open and other cursors have executed since then. In these cases it is difficult to find the exact cause of the blocking without looking at the application in depth. One clue that may help is knowing the SQL for the waiting session and then examining the application code for other places and situations where the tables in the SQL statement may be locked. You can find the SQL and exact ROWID being waited on by issuing the following query:

```
select s.sid, s.serial#, s.username, s.module, s.ROW_WAIT_OBJ# object_id,
 dbms_rowid.rowid_create(1, s.row_wait_obj#, s.row_wait_file#,
 s.row_wait_block#, s.row_wait_row#) my_rowid
 s.sql_hash_value, s.sql_address, sq.SQL_TEXT,
 from v$session_wait sw, v$session s, v$sql sq, v$lock l
 where sw.event = 'enqueue'
 and sw.sid = s.sid
 and l.type = 'TX' and l.request = 6
 and l.sid = s.sid
 and s.sql_hash_value = sq.hash_value and s.sql_address = sq.address
```

Note: "object_id" can be used to query DBA_OBJECTS.

Documentation

Concepts: Data Concurrency and Consistency

How Oracle Locks Data

Concepts: Row Locks (TX)

Notes

Tracing sessions: waiting on an enqueue

TX Transaction locks - Example wait scenarios

TX Lock "Transaction Enqueue"

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Long running statement while locks are held

A long running statement may delay the time between a transaction starts (via some DML) and commits. This statement may need to be tuned.

M *Effort Details*

Medium effort; requires access and examination of application code.

L *Risk Details*

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. *Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).*
2. *For the session identified in step 1 (lock holder), check if it is currently executing SQL. For example, assuming the session ID is 12:*

```
SELECT s.sid, s.status, sq.sql_text
FROM v$session s, v$sql sq
WHERE s.sid = 12
```

```
and s.status = 'ACTIVE'
and s.sql_hash_value = sq.hash_value
and s.sql_address = sq.address
```

3. ***Investigate the performance of this cursor. At this point, the problem becomes a query tuning problem. You can use this guide for help or the SQL Tuning Advisor (10g or higher with EM Tuning Pack license).***
4. ***Tune the query and evaluate whether the locking problem has been resolved. If it hasn't been resolved, examine the application in more detail to see if the application should be changed.***

See the documents below for more information.

10g+: SQL Tuning Advisor

How to use the Sql Tuning Advisor

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

TBD

Manual Tuning

SQL Tuning Overview

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX allocate ITL entry

Waits for enq:TX allocate ITL entry (in mode 4) can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.
Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq:TX - allocate ITL entry waits.*
  - *optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 4*
- *AWR or statspack:*
  - *enq:TX - allocate ITL entry is among the top timed events*

Cause Identified: Insufficient ITLs in a block

Waits for the TX enqueue in mode 4 can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.

Cause Justification
Prior to 9.2.x:
In versions prior to 9.2.x, it is difficult to pinpoint an ITL wait exactly. Consider this justified if you have examined other causes for TX, mode 4 waits and none are justifiable.

9.2.x:
Using statspack snapshots taken at level 7, look in the segment statistics section to see which segments have the highest ITL waits (e.g., Segments by ITL Waits). If these are a significant portion of the TX enqueue waits (see the Enqueue Activities section), then this cause is justified.

In 10g, the wait event itself tells you this is an ITL wait, so it is justified from the wait event.

Solution Identified: Increase the table's INITRANS setting

Increase the table's INITRANS setting to account for the number of concurrent sessions changing an individual block.

L *Effort Details*

Medium effort; may require rebuilding the table.

L *Risk Details*

Low risk; no risky side effects except if INITRANS is set too large and the block size is small (this will waste a lot of block space).

Solution Implementation

See the documents below.

Documentation
SQL Ref: INITRANS

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX index contention

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).
Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq:TX - allocate ITL entry waits.*
  - *optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 4*
- *AWR or statspack:*
  - *enq:TX - allocate ITL entry is among the top timed events*

Cause Identified: Index contention due to block splits

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).

Cause Justification
To identify which segment is involved:

- *Look in the TKProf of one or more sessions that experience the most of this kind of wait.*
- *Find the statement that waited the longest amount of time on the event with long TX, mode 4 waits. This is generally an insert statement.*
- *Examine the statement to find the indexes involved.*

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L ***Effort Details***

Low effort; will require rebuilding an index.

M ***Risk Details***

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M ***Effort Details***

Medium effort; requires rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: library cache pin

Library cache pins are used to manage library cache concurrency. Pinning an object causes the heaps to be loaded into memory (if not already loaded). PINS can be acquired in NULL, SHARE or EXCLUSIVE modes and can be considered like a special form of lock. A wait for a "library cache pin" implies some other session holds that PIN in an incompatible mode.

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for library cache pin waits.***
- ***AWR or statspack:***
  - ***Significant waits for library cache pin***

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed parse time***
- ***Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses***
- ***Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).***

AWR or statspack reports:

- ***Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen***
- ***The "% SQL with executions>1" is over 60%, meaning statements are being shared***

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; increasing the size of the shared pool is not risky unless:

- ***There are many unshared statements due to literals***
  For more details, see: Understanding and Tuning the Shared Pool
- ***The machine doesn't have enough physical memory and starts swapping***
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M ***Effort Details***

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M ***Risk Details***

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation
GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes
Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Objects being compiled across sessions

One or more sessions are compiling objects (typically PL/SQL) while another session wants to pin the same object prior to executing or compiling it. One or more sessions will wait on library cache pin in Share mode (if it just wants to execute it) or eXclusive mode (if it want to compile/change the object).

Cause Justification
TKProf:

- *library cache pin waits and / or library cache pin waits*
- *Statement is compiling or executing PL/SQL*

Solution Identified: Avoid compiling objects in different sessions at the same time or during busy times

Do not compile interdependent objects across concurrent sessions or during peak usage.
The HangAnalyze command can usually help identify the blockers, waiters, and the SQL which is causing the waits (see the "Hang / Locking tab > Issue Identification > Data Collection" for more information).

L *Effort Details*

Low effort; requires some thought on how and when to recompile objects.

L *Risk Details*

Low risk.

Solution Implementation

Schedule and/or sequence the recompilation to avoid conflicts.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M ***Effort Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M ***Risk Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference
CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting
Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Contention caused by changing object privileges

Changing object privilges causes contention in the library cache since the object will need to be invalidated and reparsed with the new privileges. Any type of privilege change using GRANT or REVOKE on an object may cause dependent objects to become invalidated too thereby amplifying the effect of the change and causing contention if the system is busy.

Cause Justification
This cause is likely if there are:

- *waits on the library cache, shared pool latches, mutexes, and/or library cache pins*
- *High invalidations*
- *DDL and other causes have been eliminated*

Solution Identified: Avoid making grants during periods of high activity or concurrency

Schedule the privilege changes when the system is quiet to avoid impacting users.

L *Effort Details*

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M *Risk Details*

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: library cache lock

The library cache lock controls the concurrency between clients of the library cache by acquiring a lock on the object handle so that either:

- *one client can prevent other clients from accessing the same object*
- *The client can maintain a dependency for a long time (no other client can change the object).*

This lock is also obtained as part of the operation to locate an object in the library cache (a library cache child latch is obtained to scan a list of handles, then the lock is placed on the handle once the object has been found).

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for library cache lock waits.*
- *AWR or statspack:*
  - *Significant waits for library cache lock*

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the presence of literal values.*

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M *Effort Details*

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

*Pro*C/C++ Precompiler Programmer's Guide*

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- *EXACT: Leave the statement as it was written with literals (default value)*
- *FORCE: Substitute all literals with binds (as much as possible)*
- *SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)*

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L *Effort Details*

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; increasing the size of the shared pool is not risky unless:

- ***There are many unshared statements due to literals***
  For more details, see: Understanding and Tuning the Shared Pool
- ***The machine doesn't have enough physical memory and starts swapping***
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M ***Effort Details***

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M ***Risk Details***

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation

GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes

Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Objects being compiled across sessions

One or more sessions are compiling objects (typically PL/SQL) while another session wants to pin the same object prior to executing or compiling it. One or more sessions will wait on library cache pin in Share mode (if it just wants to execute it) or eXclusive mode (if it want to compile/change the object).

Cause Justification
TKProf:

- *library cache pin waits and / or library cache pin waits*
- *Statement is compiling or executing PL/SQL*

Solution Identified: Avoid compiling objects in different sessions at the same time or during busy times

Do not compile interdependent objects across concurrent sessions or during peak usage.
The HangAnalyze command can usually help identify the blockers, waiters, and the SQL which is causing the waits (see the "Hang / Locking tab > Issue Identification > Data Collection" for more information).

L *Effort Details*

Low effort; requires some thought on how and when to recompile objects.

L *Risk Details*

Low risk.

Solution Implementation

Schedule and/or sequence the recompilation to avoid conflicts.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Auditing is turned on

Auditing will increase the need to acquire library cache locks and potentially increase contention for them. This is especially true in a RAC environment where the library cache locks become database-wide (across all instances).

Cause Justification
AWR / Statspack:

- ***library cache lock waits***
- ***audit_trail parameter is set to something other than "none"***

Solution Identified: Evaluate the need to audit

Consider disabling auditing if it is not absolutely necessary.

L ***Effort Details***

Low effort; initialization parameter change

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

Documentation
Keeping Audited Information Manageable

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Unshared SQL in a RAC environment

Library cache locks waits may occur in RAC environments when applications are not sharing SQL. In single-instance environments, library cache and shared pool latch contention is typically the symptom for unshared SQL. However, in RAC, the main symptom may be library cache lock contention.

Cause Justification
RAC environment

TKProf:
- ***Many statements are hard parsed***
- ***library cache lock waits occur as part of a hard parse***

AWR / Statspack:
- ***library cache lock waits***
- ***Low percentage for "% SQL with executions>1" (less than 60%)***
- ***soft parse ratio is below 80%***

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M ***Effort Details***

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M ***Risk Details***

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting
Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation
7.3.1.3 SQL Sharing Criteria

Searches
Pro*C/C++ Precompiler Programmer's Guide

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- *EXACT: Leave the statement as it was written with literals (default value)*
- *FORCE: Substitute all literals with binds (as much as possible)*
- *SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)*

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L *Effort Details*

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*

- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Extensive use of row level triggers

When row level triggers are fired frequently, higher than usual library cache activity may occur, because of the need to check if mutating tables are being read. During trigger execution, it is possible that the application tries to read mutating tables, i.e., tables that are in the process of being modified by the statement that caused the trigger to fire. As this may lead to inconsistencies, it is not allowed, and the application should receive the error ORA-4091. The mechanism to detect this error involves one library cache lock acquisition per table referenced in each select statement executed.

The extent of the problem depends on how many times the row triggers fire rather than on the number of row triggers have been created (i.e., one trigger that fires 10000 times will cause more problems than 100 triggers that fire once).

Cause Justification
TKProf:

- *Many statements are hard parsed*
- *library cache lock waits*
- *evidence of a row level trigger firing (maybe some recursive SQL related to a trigger)*

Solution Identified: Evaluate the need for the row trigger

Sometimes row triggers aren't needed to accomplish the functionality. Consider if there is an alternative.

M *Effort Details*

Medium effort; may require application and schema changes

M *Risk Details*

Medium risk. If the application and schema changes, there is a possibility that some adverse effect will be introduced. Thorough testing will be needed.

Solution Implementation

Requires understanding the application and how row-level triggers are used. See the documents below for reference information.

Documentation

App Dev Guide: Coding Triggers

App Dev Guide: Coding Triggers

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M *Effort Details*

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M *Risk Details*

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference

CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: row cache lock

The row cache lock is used primarily to serialize changes to the data dictionary. Waits on this event usually indicate some form of DDL occurring, or possibly recursive operations such as storage management and sequence numbers incrementing frequently.

What to look for

In Statspack, AWR, or TKProf, significant waits for row cache lock are seen.

Cause Identified: Sequence Cache Management Causing Contention

Sequences are being incremented quickly and exhausting their caches; this is leading to contention in the row cache as multiple sessions attempt to acquire the row cache lock when they update the row cache with new values.

Cause Justification
In AWR, ASH, statspack, TRCANLZR, or TKProf, observe:

1. *Significant row cache lock contention*
2. *row cache lock waits are identified as being due to "dc_sequences". This may be done by observing in AWR or Statspack in the "Dictionary Cache Stats" section, that "dc_sequences" are responsible for most of the modification requests. ASH will show you the row cache lock wait's P3 values which correspond to the cache_id and may be correlated to the "parameter column" in V $ROWCACHE to see if most waits are for dc_sequences.*

Solution Identified: Increase the cache size for sequences incrementing rapidly

Identify the sequences under contention by obtaining a 10046, level 8 trace of sessions that are experiencing the high row cache lock waits. Find the statements with the highest row cache lock waits on the dc_sequences cache and then look for the sequences they are using. Increase the cache size of these sequences by using the "ALTER SEQUENCE CACHE " command so they don't need to be updated as frequently.

L **Effort Details**

simple command to change cache size.

L **Risk Details**

Larger sized caches will improve performance. Be aware that if the database is restarted or the sequence is aged out of the shared pool, there will be a gap in the sequences up to CACHE size large.

Solution Implementation

Use the syntax listed in the SQL reference guide for ALTER SEQUENCE.

Reference

ALTER SEQUENCE syntax

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Deeper Investigation Needed

No specific cause was found for the high row cache lock waits - deeper investigation techniques are needed.

Cause Justification
High row cache lock waits were found but other causes have been ruled out.

Solution Identified: Contact Oracle Support Services for additional help.

This problem requires assistance from Oracle Support Services.

L ***Effort Details***

N/A

L ***Risk Details***

N/A

Solution Implementation

N/A

Implementation Verification

N/A

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

6. Concurrency - Latches and Mutexes

Waits for latches and mutexes that are used to coordinate operations. Typical events:

- ***cursor: pin S wait on X***
- ***latch free***
- ***latch: cache buffers chains***
- ***latch: library cache***
- ***latch: library cache lock***
- ***latch: library cache pin***
- ***latch: row cache objects***
- ***latch: shared pool***

Facts Required for Analysis:

The key is to determine which latch is causing the performance problem.

Please read the note, How to Identify Which Latch is Associated with a "latch free" wait for more details.

Examine the table below for common causes of the wait events you found.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: latch: cache buffers chains

Block headers in the buffer cache are placed on linked lists (cache buffer chains) which are accessed through a hash table. One or more hash chains are protected by one child of this latch. Processes need to get the child latch to scan for a buffer. This prevents the linked list from changing while scanning.

Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall summary for non-recursive and recursive statements shows significant amount of time for latch: or latch free waits.*
  - *The cache buffers chains latch is a significant part of the total waits*
- *AWR or statspack:*
  - *10g or higher: latch: cache buffers chains waits is among the top timed events*
  - *Prior to 10g: latch free waits is among the top timed events and the cache buffers chains latch is among the more prominent latches (high wait times or sleeps)*

Cause Identified: Hot blocks due to inefficient execution plan

Hot blocks refer to block headers that are accessed very frequently (via logical reads) and this frequent access leads to contention on the cache buffers chains latch.

Inefficient execution plans may perform many logical reads while they visit many blocks. If this query is executed by many sessions concurrently (or other similar queries against the same blocks), then there will be contention on these blocks.

Cause Justification

TKProf:
- *Use the report sorted by elapsed fetch time*
- *Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.*

AWR or statspack reports:
- *10g: waits on latch: cache buffers chains*
- *Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch*
- *Examine the Top SQL sections; look for statements with the highest elapsed time. You will see some of these statements performing a large number of buffer gets (logical reads) per execution*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Hot blocks due to concurrently accessing a popular block

Hot blocks refer to block headers that are accessed very frequently (via logical reads) and this frequent access leads to contention on the cache buffers chains latch.

This particular kind of hot block contention occurs when a query only reads a few blocks but this same query (or other ones that access the same blocks) are executed by many sessions at the same time. The problem can't be solved by tuning the query because the execution plan is already efficient. Either the query must be executed less often or the rows need to be spread out among more blocks.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed fetch time***
- ***Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.***
- ***Check if these statements only access a few blocks per execution ([query + current] / executions is low)***

AWR or statspack reports:

- *10g: waits on latch: cache buffers chains*
- *Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch*
- *Examine the Top SQL sections; look for statements with the highest elapsed time. You will see some of these statements performing a small number of buffer gets (logical reads) per execution*

Solution Identified: Spread out the rows over more blocks

Alter (or even rebuild) tables listed above to use a higher PCTFREE setting. This will reduce the number of rows per block and hopefully, spread out contention for the blocks (at the expense of wasting space).

M *Effort Details*

Medium effort; will require rebuilding a table.

M *Risk Details*

Medium risk; some queries may run slower because they will need to access more blocks to obtain the same number of rows. Review how this table is accessed before implementing this solution.

Solution Implementation

Rows can be spread out by rebuilding the table using a larger value for PCTFREE. Another way to spread rows out is to make use of the table option, MINIMIZE RECORDS_PER_BLOCK as follows:

1. *Export the table*
2. *Truncate the table*
3. *Insert the desired number of rows per block. E.g., if you only want 10 rows per block, insert just 10 rows.*
4. *Alter the table to set minimize records_per_block setting. E.g.,*

```
Alter table stock_prices minimize records_per_block;
```

5. *Delete the rows you inserted in step 3*
6. *Import the table*

See the documents below for additional information.

Documentation

The PCTFREE Parameter

SQL Ref: Minimize records_per_block Clause

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L *Effort Details*

Low effort; will require rebuilding an index.

M *Risk Details*

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread values across blocks

Hash partitioning will distribute rows evenly for a given column in a table.

M ***Effort Details***

Medium effort; requires recreating the table and importing rows into it.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

SQL Ref: Create Table, hash partitioning clause

Creating a Hash-Partitioned Table: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Bug 4742607 - "cache buffer chains" latch contention from concurrent index range scans

Concurrent index range-scan initializations can lead to contention on the "cache buffers chains" hash latches due to latch upgrades.

Bug 4742607 - "cache buffer chains" latch contention from concurrent index range scans

Cause Justification

TKProf:

- ***Use the report sorted by elapsed fetch time***
- ***Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.***
- ***The execution plans for these statements make use of index scans***

AWR or statspack reports:

- ***10g: waits on latch: cache buffers chains***
- ***Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch***
- ***In the Instance Statistics, you will see high "shared hash latch upgrade" statistic counts***

Solution Identified: Bug 4742607

Bug 4742607

M ***Effort Details***

Medium effort; requires a patch.

L ***Risk Details***

Low risk; this patch has been well proven.

Solution Implementation

See the documents below.

Additional bug information:
Bug 4742607

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Wait: latch: library cache
Wait: latch: shared pool

Library Cache Latch
The library cache latches serialize access to the objects in the library cache. Every time that a SQL statement, a PL/SQL block or a stored object (Procedures, packages, functions, triggers) is parsed or executed, this latch is acquired to ensure the object doesn't change while it is locked in the library cache.

Shared Pool Latch
Free memory in the shared pool is tracked on a number of freelists. The shared pool latch is typically acquired when a chunk of memory is requested, and lasts while scanning the relevant freelists for a chunk of the required size. The latch may also be acquired for other operations such as coalescing memory or releasing memory back to a freelist.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall summary for non-recursive and recursive statements shows significant amount of time for latch: or latch free waits.***
  - ***The library cache or shared pool latch is a significant part of the total waits***

- ***AWR or statspack:***

- 10g or higher: latch: library cache waits is among the top timed events
- Prior to 10g: latch free waits is among the top timed events and the library cache or shared pool latch is among the more prominent latches (high wait times or sleeps)

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :

- Use the report sorted by elapsed parse time
- Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses
- Examine the statements that are being hard parsed and look for the presence of literal values.

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M Effort Details

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M Risk Details

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

Pro*C/C++ Precompiler Programmer's Guide

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- *EXACT: Leave the statement as it was written with literals (default value)*
- *FORCE: Substitute all literals with binds (as much as possible)*
- *SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)*

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L *Effort Details*

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:
- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:
- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; increasing the size of the shared pool is not risky unless:
- *There are many unshared statements due to literals*
  *For more details, see: **Understanding and Tuning the Shared Pool***
- *The machine doesn't have enough physical memory and starts swapping*
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M *Effort Details*

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M *Risk Details*

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation
GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes
Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive soft parsing

Soft parsing occurs when Oracle looks in the library cache for a cursor or object it hopes to share. If it finds the cursor and it is sharable (same optimizer mode, etc), then it will consider this as a soft parse. Soft parsing is more efficient than hard parsing but still impacts latches to a degree.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being soft parsed; these will have "Misses in the library cache" close to zero*

AWR or statspack reports:

- *The Instance Efficiency Percentages will report high values (usually over 60%) for Soft Parse %*

Solution Identified: Avoid unnecessary soft parsing in the application

Application code will sometimes needlessly force a soft parse when it could have simply used an open cursor handle and re-executed the cursor with new bind values. Look through the application code and determine whether the soft parse is really needed.

M *Effort Details*

Medium effort; will require coordination with developers to review and change code.

L *Risk Details*

Low risk; the change should be very localized.

Solution Implementation

Ensure your application doesn't perform unnecessary soft parsing. Typically this occurs when a parse statement is placed in the middle of a loop that iterates over a set of rows. Consider this pseudo-pseudo-code:

```
list_of_rows = Retrieve some rows()
FOR each row in list_of_rows LOOP
        cursor_handle = PARSE(sql)  # parse for each loop iteration
        EXECUTE(cursor_handle, bind1, bind2)
        CLOSE(cursor_handle)
END LOOP
```

To avoid the repeated soft parses:

```
list_of_rows = Retrieve some rows()
cursor_handle = PARSE(sql)  # parse once
FOR each row in list_of_rows LOOP
        EXECUTE(cursor_handle, bind1, bind2)
END LOOP
CLOSE(cursor_handle)
```

It's also a good idea to make sure the application leaves cursors open and doesn't re-open them unnecessarily (see references below for best-practice information on this and compensating this using the SESSION_CACHED_CURSORS parameter).

Documentation

Performance Tuning Guide: Using the Shared Pool Effectively

Performance Tuning Guide: Cursor Access and Management

Programmer's Guide to the Oracle Precompilers: Eliminating Unnecessary Parsing

Pro*C/C++ Programmer's Guide: Eliminating Unnecessary Parsing

JDBC Developer's Guide and Reference : Statement Caching

Reference

SQL Parsing Flow Diagram

How to work out how many of the parse count are hard/soft?

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Ensure session cached cursors are used

The session cached cursors parameter will allow Oracle to maintain a small cache of handles to cursors in the library cache. During a parse, the cache will be examined and if a match is found, the soft parse will be avoided.

Review the value of this parameter and consider increasing it (although it should be increased slowly and not above 200 to avoid locking too many statements in the library cache).

L ***Effort Details***

Low effort; a parameter change

L ***Risk Details***

Low risk; the change can be localized to a session. Not risky as long as the values is not increased over 200.

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Caching Session Cursors

Reference: SESSION_CACHED_CURSORS parameter

Performance Tuning Guide: Using the Shared Pool Effectively

Performance Tuning Guide: Cursor Access and Management

Notes

Understanding and Tuning the Shared Pool, see SESSION_CACHED_CURSORS parameter

Reference Note for Init.Ora Parameter "SESSION_CACHED_CURSORS"

SCRIPT - to Gauge the Impact of the SESSION_CACHED_CURSORS Parameter

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M ***Effort Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M ***Risk Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference
CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting
Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Contention caused by changing object privileges

Changing object privilges causes contention in the library cache since the object will need to be invalidated and reparsed with the new privileges. Any type of privilege change using GRANT or REVOKE on an object may cause dependent objects to become invalidated too thereby amplifying the effect of the change and causing contention if the system is busy.

Cause Justification
This cause is likely if there are:

- *waits on the library cache, shared pool latches, mutexes, and/or library cache pins*
- *High invalidations*
- *DDL and other causes have been eliminated*

Solution Identified: Avoid making grants during periods of high activity or concurrency

Schedule the privilege changes when the system is quiet to avoid impacting users.

L *Effort Details*

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M *Risk Details*

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

7. ***Configuration***

Waits caused by inadequate configuration of database or instance resources (for example, undersized log file sizes, shared pool size). Typical events:

- ***free buffer waits***
- ***log buffer space***
- ***log file switch (archiving needed)***
- ***log file switch (checkpoint incomplete)***
- ***log file switch completion***
- ***write complete waits***

Facts Required for Analysis:

- ***TKProf, elapsed times for events (Overall Totals, recursive and non-recursive):***
  - ***Total wait time for the event***
  - ***Average wait for the event = total wait time / total waits***

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: free buffer waits

This wait event indicates that a server process was unable to find a free buffer and has posted the database writer to make free buffers by writing out dirty buffers (buffers w/unwritten changes). Once DBWR finishes writing the dirty buffers to disk, they are free to be reused.

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for free buffer waits waits.***
- ***AWR or statspack:***
  - ***Significant waits for free buffer waits***

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L *Effort Details*

Low effort

L *Risk Details*

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To
How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference
Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Poor file write performance in some filesystems

Some filesystems have poor write performance (writes take too long) and is impacting DBwriter's ability to keep enough clean buffers in the buffer cache.

Cause Justification
AWR / Statspack:

- ***free buffer waits***
- ***db file parallel write waits have an average wait time LARGER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter)***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation
I/O Configuration and Design

Wait Event: db file scattered read

Notes
Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***

- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

If the buffer cache is too small and filled with hot blocks, then sessions will be starved for free buffers (clean, cold blocks) and will need to spend too much time looking for free buffers and/or posting DBWR to write dirty blocks and make them free. Increase the parameter DB_BLOCK_BUFFERS(Oracle8+) or DB_CACHE_SIZE (Oracle9+) and monitor the effect of the change.

Cause Justification
AWR / Statspack:

- *free buffer waits*
- *DBWriter is not seeing a performance problem in writing the files. Specifically, db file parallel write waits have an average wait time SMALLER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter*
- *You may see high values (compared to a baseline) for statistics write clones, hot blocks moved to the head of the LRU, and free buffers inspected*

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: DBWriter is not using asynchronous I/O

The DBWriter will achieve optimal throughput when asynchronous I/O is available to it. DBWriter may not be able to keep up with buffer demands if asynch I/O is not available.

Cause Justification
AWR / Statspack:

- ***free buffer waits***
- ***DBWriter is seeing a performance problem in writing the files. Specifically, db file parallel write waits have an average wait time LARGER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter***
- ***Asynchronous I/O is disabled via the initialization parameter disk_asynch_io or filesystemio_options***

Solution Identified: Enable asynchronous I/O

Enable asynchronous I/O If the platform supports it. This is preferred over adding multiple DBwriters or I/O slaves.

L ***Effort Details***

Low effort; initialization parameter change

L ***Risk Details***

Low risk. Ensure your platform supports it and is up-to-date on patches

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Choosing Between Multiple DBWR Processes and I/O Slaves

Performance Tuning Guide: Asynchronous I/O

AIX: Using Asynchronous I/O

HPUX: Using Asynchronous I/O

Linux: Using Asynchronous I/O

Reference: DISK_ASYNCH_IO Parameter

Notes

How To Check If Asynchronous I/O Is Working On Linux

Asynchronous I/O (aio) on RedHat Advanced Server 2.1 and RedHat Enterprise Linux 3

Understanding and Tuning Buffer Cache and DBWR

Database Writer and Buffer Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use multiple DBwriters

Enable asynchronous I/O If the platform supports it. However, if your platform doesn't support it, then adding multiple DBWriters can help divide the workload.

L ***Effort Details***

Low effort; initialization parameter change

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Choosing Between Multiple DBWR Processes and I/O Slaves

Understanding and Tuning Buffer Cache and DBWR

Database Writer and Buffer Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: log buffer space

This event occurs when server processes are writing redo records to the log buffer faster than LGWR can write them out; eventually, the log buffer fills up and the processes wait for free space. After LGWR writes some buffers out, then those buffers may be reused by other processes.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for log buffer space waits associated with DML statements.*
- *AWR or statspack:*
  - *Significant waits for log buffer space*

Cause Identified: The log buffer is too small

If the log buffer is too small, then the demand for redolog buffer space will overtake the supply of buffers and cause these waits.

Cause Justification
AWR / Statspack:

- *log buffer space waits*
- *initialization parameter, log buffer is smaller than:*
  *statistic: redo size per sec * 600 (10 min worth of redo)*
- *The average time for log file parallel write is less than 20mSec*

Solution Identified: Increase the size of the log buffer

Increase the parameter LOG_BUFFER to increase the redo log buffer size. Values of LOG_BUFFER larger than 32 MB (and even around 3 MB) will usually not have any effect (and will just waste memory).

M *Effort Details*

Medium effort; easy to change but requires the database to be restarted.

L *Risk Details*

Low risk; larger size log buffers could waste memory but will not adversely affect performance (unless there is a memory shortage on the machine).

Solution Implementation

See the documents below.

Documentation

Concepts: Redolog Buffer

Concepts: Log Writer Process (LGWR)

Reference: LOG_BUFFER parameter

Reference: log buffer space wait event

Performance Tuning Guide: Configuring and Using the Redo Log Buffer

Performance Tuning Guide: log buffer space wait event

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Logwriter is writing too slow

If the size of the log buffer is already large (more than 3 MB), speed up the LGWR background process write operations by ensuring that the I/O devices where the redolog files are stored are not suffering from I/O contention.

Cause Justification
AWR / Statspack:

- *log buffer space waits*
- *The average time for log file parallel write is MORE than 20mSec*

OS disk performance data on the filesystems where redologs are placed show disk response times greater than 20mSec.

Additional Information:

Concepts: Redolog Buffer

Concepts: Log Writer Process (LGWR)

Wait Event "log file parallel write" Reference Note

Tuning I/O-related waits, see 'log file parallel write' wait event section

Checkpoint Tuning and Troubleshooting Guide

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- *Significant waits for the read by other session event*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

8. ***Network***

Waits caused by network related activity. Typical events:

- ***SQL*Net message from dblink***
- ***SQL*Net more data from dblink***
- ***SQL*Net more data to client***
- ***SQL*Net more data to dblink***

Facts Required for Analysis:

- ***TKProf, elapsed times for events (Overall Totals, recursive and non-recursive):***
  - ***Total wait time for the event***
  - ***Average wait for the event = total wait time / total waits***

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: SQL*Net message from dblink

The Oracle shadow process is waiting for a message over a database link from a remote process. Note that this wait is also used when waiting for data from "extproc" or from a remote gateway process.

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from dblink waits.***
- ***AWR or statspack:***
  - ***Significant waits for SQL*Net message from dblink***

Cause Identified: A remote database is not executing the query fast enough

If the local database is waiting for this event on a distributed query, the remote node(s) may be taking too long to execute the query and return results back to the local node.

Cause Justification
TKProf:

1. ***Focus attention on the remote database***
2. ***On the "remote" database, find the session corresponding to the "local" database (it will look like a typical database client)***
3. ***Determine how long it takes to execute the query sent over the dblink (best if you can trace this session with the 10046 event)***
4. ***If most of the time is spent executing the "remote" query, this issue is justified***

Solution Identified: Tune the remote query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail on the remote site using the information in the Performance Diagnostic Guide's Query Tuning section.

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

M ***Effort Details***

Medium; tracing distributed queries is more challenging than local queries.

L ***Risk Details***

Not applicable

Solution Implementation

In addition to using the Performance Diagnostic Guide's Query Tuning section, see the documents below for specific issues with distributed queries.

Documentation

Concepts: Distributed Database Concepts

Admin Guide: Tuning Distributed Queries

Notes

Distributed Queries

Determining the execution plan for a distributed query

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

9. ***Reads / Writes***

Waits for I/O (for example 'db file sequential read'). Typical events:

- ***db file parallel write***
- ***db file sequential read***
- ***db file scattered read***
- ***direct path read***
- ***direct path write***
- ***log file parallel write***
- ***io done***
- ***read by other session***

Facts Required for Analysis:

- ***TKProf, elapsed times for events (Overall Totals, recursive and non-recursive):***
  - ***Total wait time for the event***
  - ***Average wait for the event = total wait time / total waits***

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: db file sequential read

The session waits while a sequential read from the database is performed. This event is also used for rebuilding the control file, dumping datafile headers, and getting the database file headers.

Wait class: User I/O, typically foreground

What to look for

- ***TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for db file sequential read waits.***
- ***AWR or statspack: db file sequential read waits is among the top timed events***

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Wait: db file scattered read

The session waits while a multiblock read from the database is performed. Similar to db file sequential read, except that the session is reading multiple data blocks and scattering them around the buffer cache.

Wait class: User I/O, typically foreground

What to look for

- ***TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for db file scattered read waits.***
- ***AWR or statspack: db file scattered read waits is among the top timed events***

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: direct path read

Direct Path operations (parallel execution, hash joins, sorts to disk) read data from datafiles directly into the PGA (opposed to the buffer cache in SGA). When the process attempts to access a block in the PGA that has not yet been read from disk, it then issues a wait call and updates the statistics for this event.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for direct path read waits.*
- *AWR or statspack:*
  - *Significant waits for direct path read*

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- *Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file*
- *Determine why the filesystems are performing poorly. Some common causes are:*
  - *"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth*
  - *hardware problem*
  - *In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.*

M *Effort Details*

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Incorrect manual workarea sizing

Oracle uses in-memory workareas in the PGA for performing sorts, hash joins, and other operations. These can be manually controlled by parameters such as sort_area_size and hash_area_size.

When these parameters are sized below what Oracle needs do an operation in memory, then some of the data will need to be written in temp segments causing direct path write waits. Eventually, this data will be read back and will cause direct path read waits.

Cause Justification
TKProf:

- *Significant waits on direct path read or direct path writes*
- *Execution plan shows sorts or hash join operations*
- *Average wait time is less than 20 mSec*

Solution Identified: Use automatic PGA memory management

*When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.*

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L *Effort Details*

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L *Risk Details*

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Parallel execution is occurring but not expected or desired

Parallel execution is occurring and causing CPU or I/O problems (typically direct path read/write waits) due to the degree of parallelism. PX is not expected; the CBO will attempt to use parallel operations if the following are set or used:

- *Parallel hint: parallel(t1, 4)*
- *ALTER SESSION FORCE PARALLEL*
- *Setting a degree of parallel and/or the number of instances on a table or index in a query*

Cause Justification

- *The process with very high direct path read waits is a parallel execution slave process.*
- *There are many more PX slave processes than expected or desired*
- *The filesystems where the I/O is occurring were never meant to handle the I/O bandwidth required by the number of PX processes*

Additional Information:

Summary of Parallelization Rules

Solution Identified: Remove parallel hints

The statement is executing in parallel due to parallel hints. Removing these hints may allow the statement to run serially.

L ***Effort Details***

Low effort; simply remove the hint from the statement.

L ***Risk Details***

Low risk, only affects the statement.

Solution Implementation

Remove one or more hints of the type:

- ***PARALLEL***
- ***PARALLEL_INDEX***
- ***PQ_DISTRIBUTE***

If one of the tables has a degree greater than 1, the query may still run in parallel.

Hint information:
Hints for Parallel Execution

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Alter a table or index's degree of parallelism

A table or index in the query has its degree (of parallelism) set higher than 1. This may be one factor causing the query to execute in parallel. If the parallel plan is not performing well, a serial plan may be obtained by changing the degree.

L ***Effort Details***

Low effort; the object may be changed with an ALTER command.

M ***Risk Details***

Medium risk; other queries may be running in parallel due to the degree setting and will revert to a serial plan. An impact analysis should be performed to determine the effect of this change on other queries.

The ALTER command will invalidate cursors that depend on the table or index and may cause a spike in library cache contention - the change should be done during a period of low activity.

Solution Implementation

See the documents below.

Parallel clause for the CREATE and ALTER TABLE / INDEX statements

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: direct path write

When a process is writing buffers directly from the PGA (as opposed to the DBWR writing them from the buffer cache), the process waits on this event to ensure that all outstanding write requests are completed. Example of "direct path writes" operations are: sorts that go to disk, parallel DML operations, direct-path INSERTs, parallel create table as select, and some LOB operations.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for direct path write waits.*
- *AWR or statspack:*
  - *Significant waits for direct path write*

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Incorrect manual workarea sizing

Oracle uses in-memory workareas in the PGA for performing sorts, hash joins, and other operations. These can be manually controlled by parameters such as sort_area_size and hash_area_size.

When these parameters are sized below what Oracle needs do an operation in memory, then some of the data will need to be written in temp segments causing direct path write waits. Eventually, this data will be read back and will cause direct path read waits.

Cause Justification
TKProf:

- *Significant waits on direct path read or direct path writes*
- *Execution plan shows sorts or hash join operations*
- *Average wait time is less than 20 mSec*

Solution Identified: Use automatic PGA memory management

When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L *Effort Details*

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L *Risk Details*

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- *Significant waits for the read by other session event*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- *Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file*
- *Determine why the filesystems are performing poorly. Some common causes are:*
  - *"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth*
  - *hardware problem*
  - *In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.*

M *Effort Details*

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- *Significant waits on waits , and/or for RAC, global cache CR request*
- *SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution*
- *Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event*
- *The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).*

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Reduce Client Bottlenecks

A client bottleneck in the context of a slow database is another way to say that most of the time for sessions is being spent outside of the database. This could be due to a truly slow client or a slow network (and related components).

1. ***Observations and Causes***

Examine the table below for common observations and causes:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

High Wait Time due to Client Events Before Any Type of Call

The Oracle shadow process is spending a significant amount of time waiting for messages from clients. The waits occur between FETCH and PARSE calls or before EXECUTE calls. There are few FETCH calls for the same cursor.

What to look for

TKProf:

- ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from client waits compared to the total elapsed time in the database***
- ***Each FETCH call typically returns 5 or more rows (indicating that array fetches are occurring)***

Cause Identified: Slow client is unable to respond to the database quickly

The client is running slowly and is taking time to make requests of the database.

Cause Justification
TKProf:

1. ***SQL*Net message from client waits are a large part of the overall time (see the overall summary section)***
2. ***There are more than 5 rows per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls). When array operations are used, you'll see 5 to 10 rows per execution.***

You may also observe that performance is good when the same queries that the client sends are executed via a different client (on another node).

Solution Identified: Investigate the client

Its possible that the client or middle-tier is saturated (not enough CPU or memory) and is simply unable to send requests to the database fast enough.

You will need to check the client for sufficient resources or application bugs that may be delaying database calls.

M ***Effort Details***

Medium effort; It is easy to check clients or mid-tiers for OS resource saturation. Bugs in application code are more difficult to find.

L ***Risk Details***

Low risk.

Solution Implementation

It may help to use a tool like OSWatcher to capture OS performance metrics on the client.

To identify a specific client associated with a database session, see the V$SESSION view under the columns, CLIENT_INFO, PROCESS, MACHINE, PROGRAM.

Documentation

Reference: V$SESSION

Notes

The OS Watcher (OSW) User Guide

The OS Watcher For Windows (OSWFW) User Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Slow network limiting the response time between client and database

The network is saturated and this is limiting the ability of the client and database to communicate with each other.

Cause Justification
TKProf:

1. ***SQL*Net message from client waits are a large part of the overall time (see the overall summary section)***
2. ***Array operations are used. This is seen when there are more than 5 rows per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls)***
3. ***The average time for a ping is about equal to twice the average time for a SQL*Net message from client wait and this time is more than a few milliseconds. This indicates that most of the client time is spent in the network.***

You may also observe that performance is good when the same queries that the client sends are executed via a different client on a different subnet (especially one very close to the database server).

Solution Identified: Investigate the network

Check the responsiveness of the network from different subnets and interface cards. The netstat, ping and traceroute utilities can be used to check network performance.

M ***Effort Details***

Medium effort; Network problems are relatively easy to check but sometimes difficult to solve.

L ***Risk Details***

Low risk.

Solution Implementation

Consult your system documentation for utilities such as ping, netstat, and traceroute

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

High Wait Time due to Client Events Between FETCH Calls

The Oracle shadow process is spending a significant amount of time waiting for messages from clients between FETCH calls for the same cursor.

What to look for

10046 / TKProf:

- *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from client waits compared to the total elapsed time in the database*
- *The client waits occur between many fetch calls for the same cursor (as seen in the cursor #).*
- *On average, there are less than 5 (and usually 1) row returned per execution*

Cause Identified: Lack of Array Operations Causing Excess Calls to the Database

The client is not using array operations to process multiple rows in the database. This means that many more calls are performed against the database. Each call incurs a wait while the database waits for the next call. The time accumulates over many calls and will impact performance.

Cause Justification
TKProf:

1. *SQL*Net message from client waits are a large part of the overall time (see the overall summary section)*
2. *There is nearly 1 row per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls). When array operations are used, you'll see 5 to 10 rows per execution.*
3. *In some cases, most of the time is for a few SQL statements; you may need to examine the whole TKProf to find where the client waits were highest and examine those for the use of array operations*

Solution Identified: Use array operations to avoid calls

Array operations will operate on several rows at a time (either fetch, update, or insert). A single fetch or execute call will do the work of many more. Usually, the benefits of array operations diminish after an arraysize of 10 to 20, but this depends on what the application is doing and should be determined through benchmarking.

Since fewer calls are needed, there are savings in waiting for client messages, network traffic, and database work such as logical reads and block pins.

M *Effort Details*

Medium effort; Depending on the client, it may be easy or difficult to change the application and use array operations.

L *Risk Details*

Very low risk; it is risky when enormous array sizes are used in OLTP operations and many rows are expected. This is due to waiting for the entire array to be filled until the first row is returned.

Solution Implementation

The implementation of array operations will vary by the type of programming language being used. See

the documents below for some common ways to implement array operations.

Documentation

PL/SQL User's Guide and Reference : Reducing Loop Overhead for DML Statements and Queries with Bulk SQL

Programmer's Guide to the Oracle Precompilers : Using Host Arrays

JDBC Developer's Guide and Reference: Update Batching

JDBC Developer's Guide and Reference: Oracle Row Prefetching

Notes

Bulk Binding - What it is, Advantages, and How to use it

How To Fetch Data into a Table of Records using Bulk Collect and FOR All

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Reduce Oracle Memory Consumption

Oracle uses memory for the SGA and PGAs. Examine the size of the SGAs and PGAs to determine what is using the system's memory.

1. ***Observations and Causes***

Examine the table below for common observations and causes:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Oracle Memory Consumption due to large SGA

One or more SGAs on the machine are leaving very little memory left for PGAs and other use on the machine.

What to look for

RDA:

- *A large portion of the memory on the machine is used by one or more SGAs (see the total size of the buffer cache and shared pool), see the following:*
  1. *Overview > System Information > Total Physical Memory*
  2. *RDBMS > SGA Information, add up all components*
  3. *Repeat for all other instances on the machine*
  4. *Compare total size of all SGAs to physical memory*

Cause Identified: Oversized buffer cache

The buffer cache is very large and is using more memory than is needed.

Cause Justification
AWR or Statspack report:

- *Not using automatic shared memory management (ASMM), i.e., SGA_TARGET=0*
- *Buffer cache hit ratio is around 99%*

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Oversized shared pool

The pool is very large and is using more memory than is needed.

Cause Justification
AWR or Statspack report:

- *Not using automatic shared memory management (ASMM), i.e., SGA_TARGET=0*
- *shared pool free memory is more than 30%*

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: The Large, Java and/or Streams Pool are Oversized

A significant amount of free space is present in the large, java and/or streams pool when there is evidence of memory pressure on the instance and/or host.

Cause Justification
If the large, java or streams pool individual free space is greater than 20%, and there is evidence of memory pressure, then this cause is likely. Memory pressure is generally detected when the system is paging out memory.

Solution Identified: Reduce the size of the Large, Java or Streams pool

Reduce the large, java and streams pool so they typically have 5% of free space during peak memory usage.

L *Effort Details*

Initialization parameter change.

M *Risk Details*

If the values are set too low, then certain operations may fail; values should be adjusted cautiously, over time if possible.

Solution Implementation

See documents below:

Reference

LARGE_POOL_SIZE parameter

JAVA_POOL_SIZE parameter

STREAMS_POOL_SIZE parameter

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Oracle Memory Consumption due to large PGAs

One or more Oracle processes are using large amounts of PGA memory on the machine.

What to look for

A large portion of the memory on the machine is used by one or more PGAs , see the following:

1. *RDA: Overview > System Information > Total Physical Memory*
2. *Run this query to see total PGA memory used by the instance:*

```
select sn.name, sum(s.value)
from v$sesstat s, v$statname sn
where s.statistic# = sn.statistic#
and sn.name like '%pga%'
group by sn.name
```

3. *Repeat for all other instances on the machine*
4. *Determine which instance uses the most PGA memory and which sessions account for the memory usage.*

Cause Identified: Manually sized private workareas are too large

Private workareas are too large for the total number of Oracle processes and the amount of memory available.

Cause Justification
AWR or Statspack report:

- *Not using automatic PGA memory management i.e., PGA_AGGREGATE_TARGET=0 or WORKAREA_SIZE_POLICY = MANUAL*
- *Parameters like sort_area_size and hash_area_size are very large and when multiplied by the number of active sessions will use up most of the system's physical memory*

Solution Identified: Use automatic PGA memory management

When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L ***Effort Details***

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L ***Risk Details***

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Automatically sized private workareas are too large

Private workareas are too large for the total number of Oracle processes and the amount of memory available.

Cause Justification
AWR or Statspack report:

- ***Using automatic PGA memory management i.e., PGA_AGGREGATE_TARGET=some large value and WORKAREA_SIZE_POLICY = AUTO***

Solution Identified: Reduce the amount of PGA_AGGREGATE_TARGET memory

PGA_AGGREGATE_TARGET may be set too large for the memory capacity of the machine. If memory is constrained, then a balance may be found where some queries run slower but the overall system runs faster since memory is available for critical operations.

L ***Effort Details***

Low effort; parameter change.

H ***Risk Details***

High risk. Some execution plans may change and some queries may perform worse.

Solution Implementation

See the documents below for guidance on the proper use of the automatic PGA memory feature.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Increase the amount of physical memory on the machine

Adding memory instead of reducing the size of the PGA target will give memory to the processes and reduce the possibility that executions plans will change.

M ***Effort Details***

Medium effort; simple hardware change but downtime involved if non-RAC.

L ***Risk Details***

Low risk. Low chance of execution plans changing.

Solution Implementation

Not applicable.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Open a Service Request with Oracle Support Services

If you would like to stop at this point and receive assistance from Oracle Support Services, please do the following:

- Please copy and paste the following into the SR:

  Last Diagnostic Step = Performance_Diagnostic_Guide.SLow_DB.Cause_Determination.
  Data_Analysis

- Enter the problem statement and how the issue has been verified
- Upload into the SR:
  - Any data you have collected up to this point (esp. good and bad statspack / AWR / TKProfs)
  - Observations, causes, and solutions you have examined and dismissed or don't understand

The more data you collect ahead of time and upload to Oracle, the fewer round trips will be required for this data and the quicker the problem will be resolved.

Click here to log your service request

Give Us Your Feedback

Your feedback is very valuable to us - please email your comments to: Vickie.Carbonneau@oracle.com

Slow Database > Reference

This section contains a summary of useful information to help diagnose and solve performance problems.

Causes and Solutions

This section contains a summary of common causes and solutions to slow database problems.

CPU

CPU consumption in the database can be due to parsing operations or non-parsing operations. The causes for each type are listed below.

1. ***Parse CPU***

Common causes for parse CPU consumption are described in this section.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

One or a few queries with High CPU usage during HARD parse

High CPU usage during hard parses are often seen with large statements involving many objects or partitioned objects.

What to look for

1. ***Check if the statement was hard parsed***
2. ***Compare parse cpu time to parse elapsed time to see if parse cpu time is more than 50%***

Cause Identified: Dynamic sampling is being used for the query and impacting the parse time

Dynamic sampling is performed by the CBO (naturally at parse time) when it is either requested via hint or parameter, or by default because statistics are missing. Depending on the level of the dynamic sampling, it may take some time to complete - this time is reflected in the parse time for the statement.

Cause Justification

- ***The parse time is responsible for most of the query's overall elapsed time***
- ***The execution plan output of SQLTXPLAIN, the UTLXPLS script, or a 10053 trace will show if dynamic sampling was used while optimizing the query.***

Solution Identified: Alternatives to Dynamic Sampling

If the parse time is high due to dynamic sampling, alternatives may be needed to obtain the desired plan without using dynamic sampling.

M ***Effort Details***

Medium effort; some alternatives are easy to implement (add a hint), whereas others are more difficult (determine the hint required by comparing plans)

L ***Risk Details***

Low risk; in general, the solution will affect only the query.

Solution Implementation

Some alternatives to dynamic sampling are:

1. ***In 10g or higher, use the SQL Tuning Advisor (STA) to generate a profile for the query (in fact, its unlikely you'll even set dynamic sampling on a query that has been tuned by the STA)***
2. ***Find the hints needed to implement the plan normally generated with dynamic sampling and modify the query with the hints***
3. ***Use a stored outline to capture the plan generated with dynamic sampling***

For very volatile data (in which dynamic sampling was helping obtain a good plan), an approach can be used where an application will choose one of several hinted queries depending on the state of the data (i.e., if data recently deleted use query #1, else query #2).

Documents for hints:

Using Optimizer Hints

Forcing a Known Plan Using Hints

How to Specify an Index Hint

QREF: SQL Statement HINTS

Documents for stored outlines / plan stability:

Using Plan Stability

Stored Outline Quick Reference

How to Tune a Query that Cannot be Modified

How to Move Stored Outlines for One Application from One Database to Another

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Query has many IN LIST parameters / OR statements

The CBO may take a long time to cost a statement with dozens of IN LIST / OR clauses.

Cause Justification

- *The parse time is responsible for most of the query's overall elapsed time*
- *The query has a large set of IN LIST values or OR clauses.*

Solution Identified: Implement the NO_EXPAND hint to avoid transforming the query block

In versions 8.x and higher, this will avoid the transformation to separate query blocks with UNION ALL (and save parse time) while still allowing indexes to be used with the IN-LIST ITERATOR operation. By avoiding a large number of query blocks, the CBO will save time (and hence the parse time will be shorter) since it doesn't have to optimize each block.

L *Effort Details*

Low effort; hint applied to a query.

L *Risk Details*

Low risk; hint applied only to the query and will not affect other queries.

Solution Implementation

See the reference documents.

Optimization of large inlists/multiple OR`s

NO_EXPAND Hint

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Partitioned table with many partitions

The use of partitioned tables with many partitions (more than 1,000) may cause high parse CPU times while the CBO determines an execution plan.

Cause Justification

1. ***The parse time is responsible for most of the query's overall elapsed time***
2. ***Determine total number of partitions for all tables used in the query.***
3. ***If the number is over 1,000, this cause is likely***

Solution Identified: 9.2.0.x, 10.0.0: Bug 2785102 - Query involving many partitions (>1000) has high CPU/ memory use

A query involving a table with a large number of partitions takes a long time to parse, causes rowcache contention, and high CPU consumption. The case of this bug involved a table with greater than 10000 partitions and global statistics ere not gathered.

M ***Effort Details***

Medium effort; application of a patchset.

L ***Risk Details***

Low risk; patchsets generally are low risk because they have been regression tested.

Solution Implementation

Apply patchset 9.2.0.4

Workaround:
Set "_improved_row_length_enabled"=false

Additional bug information:
Bug 2785102

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Many queries being HARD parsed

Hard parsing is costly for the database since it has to create various memory structures in the library cache and also optimize the SQL statement. If many queries are being hard parsed, parse CPU will be high.

What to look for

1. *Check if many statements were hard parsed*

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the presence of literal values.*

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M *Effort Details*

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

Pro*C/C++ Precompiler Programmer's Guide

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- ***EXACT: Leave the statement as it was written with literals (default value)***
- ***FORCE: Substitute all literals with binds (as much as possible)***
- ***SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)***

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L ***Effort Details***

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M ***Risk Details***

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; increasing the size of the shared pool is not risky unless:

- *There are many unshared statements due to literals*
  For more details, see: Understanding and Tuning the Shared Pool
- *The machine doesn't have enough physical memory and starts swapping*
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M *Effort Details*

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M *Risk Details*

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*

- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

2. Non-Parse CPU

Common causes for non-parse CPU consumption are described in this section.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

One or a few queries use most non-parse CPU

One or a few queries stand out as the heaviest users of non-parse CPU time. This signifies that those particular queries need to be tuned.

What to look for

- *TKProf: Only a few statements consume most of the total CPU usage (top statements when TKProf is sorted by fetch and execute CPU time)*
- *AWR or statspack: Only a few SQL statements are reported to have the highest CPU usage, and these statements' CPU usage is responsible for most of the database's CPU time (as reported in the Top 5 Timed Events section)*

Cause Identified: SQL tuning required

If one or a few statements use most of the fetch or execute time, then these statements need to be tuned.

Cause Justification
Most of the CPU time either in the entire instance (shown in AWR or statspack) or within a session (shown in TKProf) is consumed by one or a few statements.

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Waits

Common causes for wait events are described in this section.

1. ***Cluster Waits***

Waits related to Real Application Cluster resources (for example, global cache resources such as 'gc buffer busy').

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: global cache CR request

The event is waited for when a session is looking for a consistent read version of a block but cannot find it in its local cache. It also implies that the current block is not cached locally. The wait ends when either a block or a grant arrives. Depending on whether the remote instance has the block cached or not, the requesting instance receives

- ***A CR block, resulting in the statistic global cache cr block received to be incremented***
- ***A grant, resulting in the statistic global cache gets to be incremented***
- ***(9i RAC Only) A current block, resulting in the statistic global cache current blocks received to be incremented.***

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for global cache CR request waits.***
- ***AWR or statspack:***
  - ***Significant waits for global cache CR request***

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L ***Effort Details***

Low effort

L ***Risk Details***

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To

How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference

Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Inefficient SQL causing too many block reads across nodes

A poorly performing SQL statement will require an excessive amount of reads. In a RAC database those reads may require bringing blocks from other nodes and waiting for those blocks to arrive.

Cause Justification
TKProf:

- ***Significant waits on global cache CR request***
- ***SQL statements perform 100 or more logical reads (query + current) per row per execution***
- ***Full table scans (in a RAC database) may be seen in the execution plan for a statement that is waiting on this event***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To
How to use the Sql Tuning Advisor

Documentation
Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference
SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: More lock manager processes are needed

The database may require more lock manager processes to meet the demands of the database. When the lock managers are too busy, block transfers will take longer and cause waits for these blocks.

Cause Justification
TKProf:

- ***Significant waits on global cache CR request***
- ***SQL statements perform 100 or fewer logical reads (query + current) per row per execution, meaning that the statement is reasonably tuned***
- ***Full table scans (in a RAC database) are not seen in the execution plan for a statement that is waiting on this event***

OS data:

- ***The LMD process is very busy for the instance, possibly using as much as one CPU on a consistent basis***

Solution Identified: Increase the number of lock manager processes

Increase the number of Lock Manager processes for the instance by altering the value of the init.ora parameter _LM_DLMD_PROCS

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

TBD

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

2. Commit Waits

This wait class only comprises one wait event - wait for redo log write confirmation after a commit (that is, 'log file sync').

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: log file sync

When a user session commits (or rolls back), the session's redo information must be flushed to the redo logfile by the LGWR background process. This event shows the time that it takes for the LGWR to complete the write and then post the requester. The server process performing the COMMIT or ROLLBACK waits under this event for the write to the redo log to complete.

Wait class: Commit, typically foreground

What to look for

- *TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for log file sync waits.*
- *AWR or statspack: log file sync waits is among the top timed events*

Cause Identified: Frequent commits by the application

The application is committing frequently (and possibly unnecessarily

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *In the AWR or Statspack report, the average wait time for log file sync is much higher than the average wait time for log file parallel write - meaning that most of the wait for log writer is NOT due to waiting for the redo to be written*
- *In the AWR or Statspack report, the average user commits / user call is less than 30 - meaning that commits are happening frequently*

Solution Identified: Reduce the rate of commits or rollbacks

Look into the application and determine if more rows can be processed per commit. Sometimes a developer will allow the underlying language to "auto-commit" by default; this is suboptimal and should be controlled by the developer.

If the ratio of rollbacks to commits is more than 10 percent, investigate if this is unexpected or can be avoided. Rollback operations will cause the logwriter to flush redo and induce waits on log file sync waits just as commits would.

M ***Effort Details***

Medium effort; this will require some work and coordination with developers to examine their code.

L ***Risk Details***

Low risk; however, the business needs must be well understood to commit at the right times.

Solution Implementation

See the documents below.

Reference

WAITEVENT: "log file sync" Reference Note

WAITEVENT: "log file parallel write" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Redolog file write performance problems

Logwriter is not able to write to the redo log files efficiently; writes are taking too long.

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***In the AWR or Statspack report, the average wait time for log file sync is very similar to the average wait time for log file parallel write - meaning that most of the wait for log writer is due to waiting for the redo to be written***
- ***The average time for the log file parallel write event is more than 20msec***
- ***In the AWR or Statspack report, the average user commits / user call is more than 30 - meaning that commits are NOT happening frequently***

Solution Identified: Investigate redolog file write performance

Work with the system administrator to examine the filesystems where the redologs are located. Look for other processes that may be writing to that same location or a capacity problem.

M ***Effort Details***

Medium effort; this will require some work and coordination with system administrators to examine the filesystems. The redolog files may need to be moved.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See the documents below.

Reference

WAITEVENT: "log file sync" Reference Note

WAITEVENT: "log file parallel write" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L ***Effort Details***

Low effort

L ***Risk Details***

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To

How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference

Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

3. Concurrency - Buffer Busy Waits

Waits for miscellaneous internal database resources used to coordinate operations.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: Buffer busy waits

Buffer busy waits indicate that there are some buffers in the buffer cache that multiple processes are attempting to either access concurrently while its being read from disk or waiting for another session's block change to complete. In this case (buffer busy wait > data block), the contention is on the actual block where the data is stored, and can be either a table or an index.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for buffer busy waits.***
- ***AWR or statspack:***
  - ***Oracle 9.2 or higher: buffer busy waits is among the top timed events***

Cause Identified: Heavy insert activity with poor freelist configuration

Concurrent INSERTs with a suboptimal freelist configuration can lead to buffer busy wait contention as multiple sessions attempt to insert data into the same block (because it appears on the freelist to them).

Cause Justification

TKProf:
- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are INSERT statements.***

AWR or statspack reports:
- ***buffer busy wait event is among the top ones***
- ***SQL with highest wait time (derive as elapsed time - cpu time) are INSERT statements***

Solution Identified: Use ASSM or add additional freelists and/or freelist groups

Heavy INSERT activity by concurrent sessions can cause multiple sessions to attempt their insert into the same blocks because automatic segment space management (ASSM) is NOT used AND there is only a single freelist, too few process freelists, and/or no freelist groups.

The best solution is to use ASSM since it is sometimes tricky to arrive at a correct freelist or freelist group setting.

Adding process freelists will help remove contention as each process will map to separate blocks. Freelists can be added at any time without rebuilding the table.

Adding freelist groups will also remove contention by mapping processes to other freelists. This is of greatest benefit in RAC environments where the freelist group block itself will be associated with an instance, but will still help in single instance environments as well. The table must be rebuilt to change the freelist group setting.

L ***Effort Details***

Medium effort; may require rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Freelists

Performance Tuning: Buffer Busy Waits, Segment Header Contention

Admin Guide: Specifying Segment Space Management in Locally Managed Tablespaces

Reference

WAITEVENT: "buffer busy waits" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Heavy insert activity affecting an index segment

Concurrent INSERTs or updates may see contention when a related index has a key that is constantly increasing (e.g., a key based on a sequence number)

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are DML statements.***
- ***The raw trace shows the buffer busy wait's file (P1) and block (P2) values resolve to an index segment***

AWR or statspack reports:

- ***buffer busy wait event is among the top ones***
- ***SQL with highest wait time (derive as elapsed time - cpu time) are INSERT statements***
- ***9.2+ : the segments with the most buffer busy waits are indexes (as shown on the Top Buffer Busy Waits per Segment section***

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L ***Effort Details***

Low effort; will require rebuilding an index.

M ***Risk Details***

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***

- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M *Effort Details*

Medium effort; requires rebuilding the table.

L *Risk Details*

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Many concurrent SQL statements performing physical reads

Many concurrent physical reads against the same blocks will result in buffer busy waits as one session gets to do the actual physical read and the others will be blocked by the buffer busy wait event until the read completes.

This is usually an indication that the SQL statement must be tuned.

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are SELECT statements or DDL with a SELECT subquery.***
- ***The SQL statement performs many physical reads (i.e., disk on the TKProf); you see events like db file scattered reads or db file sequential reads taking significant amounts of time***

AWR or statspack reports:

- ***buffer busy wait event is among the top ones***
- ***SQL with highest wait time (derive as elapsed time - cpu time) are SELECT statements***
- ***Events like db file scattered reads or db file sequential reads are prominent in the top events lists***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Many concurrent SQL statements performing physical reads and I/O performance is poor

Many concurrent physical reads against the same blocks will result in buffer busy waits as one session gets to do the actual physical read and the others will be blocked by the buffer busy wait event until the read completes.

This is usually an indication that the SQL statement must be tuned. The waits can be amplified greatly when physical reads are slow due to poor I/O subsystem performance.

Cause Justification

TKProf:

- ***Use the TKProf reports sorted by elapsed execute time***
- ***Look at the top statements and determine if they are seeing buffer busy waits and are SELECT statements or DDL with a SELECT subquery.***
- ***The SQL statement performs many physical reads (i.e., disk on the TKProf); you see events like db file scattered read or db file sequential read taking significant amounts of time***
- ***The average time for db file scattered read or db file sequential read is around 20 mSec or higher***

(derive from: total wait time / waits).

AWR or statspack reports:

- *buffer busy wait event is among the top ones*
- *SQL with highest wait time (derive as elapsed time - cpu time) are SELECT statements*
- *Events like db file scattered reads or db file sequential reads are prominent in the top events lists*
- *The average time for db file scattered read or db file sequential read is around 20 mSec or higher.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- *Significant waits for the read by other session event*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M ***Effort Details***

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L ***Risk Details***

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

4. Concurrency - Enqueues / Locks / Pins

Waits related to enqueues or locks; these usually result from user application code (for example, lock waits caused by row level locking or explicit lock commands).

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

9.2 or prior: wait: enqueue, type TM
10g+: wait: enq: TM - contention

This could be for various reasons and is identified in pre-10g versions as waits for enqueue for the TM enqueue.
In 10g, the wait is enq: TM - contention
Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enqueue waits for TM enqueue or enq:TM - contention waits.***
  - ***In the raw 10046 trace file, most wait's P1 field decodes to TM***
- ***AWR or statspack:***
  - ***pre-10g: enqueue waits; Enqueue Activities show most waits for TM enqueue***
  - ***10g: enq:TM - contention is among the top timed events***

Cause Identified: Foreign key columns missing an index

Foreign key columns should be indexed to avoid locking issues with the parent or child tables. The exact behavior varies by version but in all versions, Oracle will use indexes to avoid locks or use a more permissive lock mode.

Cause Justification
TKProf:
- ***Lock wait is for TM enqueue, generally in mode 3 or 4***
- ***Statement involves an update to a parent or child table that has an FK constraint on a column being changed***

Solution Identified: Create indexes on the child table's foreign key columns

An index on a foreign key column will permit Oracle to either avoid or minimize lock waits when rows in the parent or child table are changed.

L ***Effort Details***

Low effort; requires creation of an index.

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

Documentation

Concepts (10gR2): No Index on the Foreign Key

Concepts (9iR2): No Index on the Foreign Key

Notes

Referential Integrity and Locking

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: ANALYZE INDEX is blocking DML statements

An index is being analyzed using the ANALYZE INDEX VALIDATE STRUCTURE command while a DML operation on the underlying table is being attempted (requiring a TM lock to be placed).

Cause Justification
The ANALYZE INDEX command acquires a TM enqueue in share mode on the underlying table; this will block other sessions when they attempt to place a TM lock that is incompatible with a share-mode lock.

The following query shows the command type for the session currently blocking another session with the TM enqueue:

```
select s.command
from v$lock l, v$session s
where l.sid = s.sid
and l.block = 1
and l.type='TM'
```

If the command type is 63 (versions 9.2 - 11.x), then an analyze index command is responsible for the blocking.

Solution Identified: Run the ANALYZE INDEX command during a maintenance window or quiet time

There is no workaround such as an "ONLINE" option for the ANALYZE INDEX VALIDATE STRUCTURE command. You will simply need to avoid the contention by scheduling the command when there is no contention likely.

L ***Effort Details***

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

L ***Risk Details***

Low risk; the analyze index command may be interrupted if necessary. The index statistics that it populates do not directly affect execution plans.

Solution Implementation

Not Applicable - solution is trivial.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Parallel DML Being Used While Other DML Performed on Same Objects

Parallel DML will acquire TM enqueues on the partitions involved (share mode) as well as the entire table (row exclusive). No other DML against affected partitions will be allowed until the PDML transaction completes.

Cause Justification
This cause is likely if there are:

- ***waits on the TM enqueue***
- ***sessions waiting are either attempting to perform PDML or are waiting for another session performing PDML***

Solution Identified: Schedule the PDML to occur during a quiet time

Schedule the PDML activity when the system is quiet to avoid impacting users.

L ***Effort Details***

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M ***Risk Details***

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use a custom parallel DML script

Sometimes its possible to avoid contention by controlling which partitions are going to concurrently receive DML through individual sessions rather than a single PDML command. This involves splitting the workload in some way and performing the DML across several sessions.

H ***Effort Details***

It could take some time to split the workload properly and script the job to run across sessions.

L ***Risk Details***

Contention can be stopped by stopping the jobs.

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

9.2 or prior: Wait: "enqueue"; TX Contention, Mode 4

TX enqueue contention in mode 4 (share mode) may be due to a variety of causes. They are not due to specific row locks but for operations related to transaction management like:

- *Lack of ITLs in a block*
- *Foreign key constraints without an index on a child table's key column*
- *Index block splits*

Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for the enqueue event*
  - *In the raw 10046 trace file, most wait's P1 field decodes to type TX, mode 4*
- *AWR or statspack:*
  - *Oracle 9.2 or prior: enqueue is among the top timed events*
  - *Enqueue Activities section shows that TX enqueues account for a significant amount of the enqueue times*
  - *Its not possible to know the modes requested without looking at the raw 10046 trace file or by looking at V$SESSION_WAIT, V$LOCK, or similar during the wait.*

Cause Identified: Insufficient ITLs in a block

Waits for the TX enqueue in mode 4 can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.

Cause Justification
Prior to 9.2.x:
In versions prior to 9.2.x, it is difficult to pinpoint an ITL wait exactly. Consider this justified if you have examined other causes for TX, mode 4 waits and none are justifiable.

9.2.x:
Using statspack snapshots taken at level 7, look in the segment statistics section to see which segments have the highest ITL waits (e.g., Segments by ITL Waits). If these are a significant portion of the TX enqueue waits (see the Enqueue Activities section), then this cause is justified.

In 10g, the wait event itself tells you this is an ITL wait, so it is justified from the wait event.

Solution Identified: Increase the table's INITRANS setting

Increase the table's INITRANS setting to account for the number of concurrent sessions changing an individual block.

L ***Effort Details***

Medium effort; may require rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects except if INITRANS is set too large and the block size is small (this will waste a lot of block space).

Solution Implementation

See the documents below.

Documentation

SQL Ref: INITRANS

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Index contention due to block splits

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).

Cause Justification
To identify which segment is involved:

- ***Look in the TKProf of one or more sessions that experience the most of this kind of wait.***
- ***Find the statement that waited the longest amount of time on the event with long TX, mode 4 waits. This is generally an insert statement.***
- ***Examine the statement to find the indexes involved.***

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L ***Effort Details***

Low effort; will require rebuilding an index.

M ***Risk Details***

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M ***Effort Details***

Medium effort; requires rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

9.2 or prior: Wait: "enqueue"; TX Contention, Mode 6

TX enqueue contention in mode 6 (exclusive mode) usually occurs when one session is updating or deleting a row, while another session wishes to update or delete the same row.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enqueue waits.***
  - ***In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 6***
- ***AWR or statspack:***
  - ***Oracle 9.2 or prior: enqueue is among the top timed events***
  - ***Enqueue Activities section shows that TX enqueues account for a significant amount of the enqueue times***
  - ***Its not possible to know the modes requested***

Cause Identified: Waiting for a row level lock due to faulty application design

Flaws in application design are often the reason for locks being held for a long time. A couple of scenarios to illustrate this are:

1) A user navigates to a certain row on a page and makes a change without committing it. The user then leaves the page for a time while the row is locked. If another user wants to update the same row, he or she will have to wait. This type of situation can be detected by identifying the blocking session (either through V$LOCK or V$SESSION.BLOCKING_SESSION in 10g) and finding out how long it has been idle using the column V$SESSION.LAST_CALL_ET.

2) The application starts a transaction and locks or updates rows then executes one or more long running queries before it commits the changes. This has the effect of holding the row locks a long time; the solution is to tune the SQL in between the row lock and the final commit.

Cause Justification

- *Use the utllockt.sql script to identify locking problems. Focus on locks where the lock type is TX and LMode is 6 (Exclusive) and check if locks are being held for a long time.*
- *Trace the lock HOLDER shown in the output of the utllockt.sql script using the 10046 event to see what its doing*
- *Look for long running queries that cause row locks to be held a long time or other problems in the application*

Solution Identified: SELECT FOR UPDATE locks too many rows

Sometimes a "pessimistic" locking strategy is implemented with SELECT FOR UPDATE statements that are missing predicates and are too "greedy" with their locking. Examine these statements to see if they are locking more rows than they actually need to lock.

M *Effort Details*

Medium effort; requires access and examination of application code.

L *Risk Details*

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. ***Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).***
2. ***Look for recent cursors that the session has executed and are still open by querying V$OPEN_CURSOR for the session identified in step 1. For example:***

```
SELECT sql_text FROM v$open_cursor WHERE sid = 1234
```

3. ***See if any cursors involve FOR UPDATE, UPDATE, or DELETE***
4. ***Examine these cursors to see if they are selecting too many rows***
5. ***Change the application to lock fewer rows at time. Sometimes this may require splitting up the work into a SELECT statement that finds candidate rows to lock and then a SELECT FOR UPDATE to lock an individual row. There are many ways to implement this kind of change - it all depends on the application.***

Sometimes, the cursor that locked the rows is no longer open and other cursors have executed since then. In these cases it is difficult to find the exact cause of the blocking without looking at the application in depth. One clue that may help is knowing the SQL for the waiting session and then examining the application code for other places and situations where the tables in the SQL statement may be locked. You can find the SQL and exact ROWID being waited on by issuing the following query:

```
select s.sid, s.serial#, s.username, s.module, s.ROW_WAIT_OBJ# object_id,
 dbms_rowid.rowid_create(1, s.row_wait_obj#, s.row_wait_file#,
 s.row_wait_block#, s.row_wait_row#) my_rowid
 s.sql_hash_value, s.sql_address, sq.SQL_TEXT,
 from v$session_wait sw, v$session s, v$sql sq, v$lock l
 where sw.event = 'enqueue'
 and sw.sid = s.sid
 and l.type = 'TX' and l.request = 6
 and l.sid = s.sid
 and s.sql_hash_value = sq.hash_value and s.sql_address = sq.address
```

Note: "object_id" can be used to query DBA_OBJECTS.

Documentation

Concepts: Data Concurrency and Consistency

How Oracle Locks Data

Concepts: Row Locks (TX)

Notes

Tracing sessions: waiting on an enqueue

TX Transaction locks - Example wait scenarios

TX Lock "Transaction Enqueue"

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Long running statement while locks are held

A long running statement may delay the time between a transaction starts (via some DML) and commits. This statement may need to be tuned.

M ***Effort Details***

Medium effort; requires access and examination of application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. ***Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).***
2. ***For the session identified in step 1 (lock holder), check if it is currently executing SQL. For example, assuming the session ID is 12:***

```
SELECT s.sid, s.status, sq.sql_text
FROM v$session s, v$sql sq
WHERE s.sid = 12
and s.status = 'ACTIVE'
and s.sql_hash_value = sq.hash_value
and s.sql_address = sq.address
```

3. ***Investigate the performance of this cursor. At this point, the problem becomes a query tuning problem. You can use this guide for help or the SQL Tuning Advisor (10g or higher with EM Tuning Pack license).***
4. ***Tune the query and evaluate whether the locking problem has been resolved. If it hasn't been resolved, examine the application in more detail to see if the application should be changed.***

See the documents below for more information.

10g+: SQL Tuning Advisor

How to use the Sql Tuning Advisor

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

TBD

Manual Tuning

SQL Tuning Overview

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX row lock contention

In 10g, enq: TX - row lock contention in mode 6 (exclusive mode) usually occurs when one session is updating or deleting a row, while another session wishes to update or delete the same row.

Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq: TX row lock contention waits.*
  - *optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 6*
- *AWR or statspack:*
  - *enq: TX - row lock contention is among the top timed events*

Cause Identified: Waiting for a row level lock due to faulty application design

Flaws in application design are often the reason for locks being held for a long time. A couple of scenarios to illustrate this are:

1) A user navigates to a certain row on a page and makes a change without committing it. The user then leaves the page for a time while the row is locked. If another user wants to update the same row, he or she will have to wait. This type of situation can be detected by identifying the blocking session (either through V$LOCK or V$SESSION.BLOCKING_SESSION in 10g) and finding out how long it has been idle using the column V$SESSION.LAST_CALL_ET.

2) The application starts a transaction and locks or updates rows then executes one or more long running queries before it commits the changes. This has the effect of holding the row locks a long time; the solution is to tune the SQL in between the row lock and the final commit.

Cause Justification

- *Use the utllockt.sql script to identify locking problems. Focus on locks where the lock type is TX*

and LMode is 6 (Exclusive) and check if locks are being held for a long time.

- *Trace the lock HOLDER shown in the output of the utllockt.sql script using the 10046 event to see what its doing*
- *Look for long running queries that cause row locks to be held a long time or other problems in the application*

Solution Identified: SELECT FOR UPDATE locks too many rows

Sometimes a "pessimistic" locking strategy is implemented with SELECT FOR UPDATE statements that are missing predicates and are too "greedy" with their locking. Examine these statements to see if they are locking more rows than they actually need to lock.

M ***Effort Details***

Medium effort; requires access and examination of application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. *Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).*
2. *Look for recent cursors that the session has executed and are still open by querying V $OPEN_CURSOR for the session identified in step 1. For example:*

```
SELECT sql_text FROM v$open_cursor WHERE sid = 1234
```

3. *See if any cursors involve FOR UPDATE, UPDATE, or DELETE*
4. *Examine these cursors to see if they are selecting too many rows*
5. *Change the application to lock fewer rows at time. Sometimes this may require splitting up the work into a SELECT statement that finds candidate rows to lock and then a SELECT FOR UPDATE to lock an individual row. There are many ways to implement this kind of change - it all depends on the application.*

Sometimes, the cursor that locked the rows is no longer open and other cursors have executed since then. In these cases it is difficult to find the exact cause of the blocking without looking at the application in depth. One clue that may help is knowing the SQL for the waiting session and then examining the application code for other places and situations where the tables in the SQL statement may be locked. You can find the SQL and exact ROWID being waited on by issuing the following query:

```
select s.sid, s.serial#, s.username, s.module, s.ROW_WAIT_OBJ# object_id,
 dbms_rowid.rowid_create(1, s.row_wait_obj#, s.row_wait_file#,
 s.row_wait_block#, s.row_wait_row#) my_rowid
 s.sql_hash_value, s.sql_address, sq.SQL_TEXT,
 from v$session_wait sw, v$session s, v$sql sq, v$lock l
 where sw.event = 'enqueue'
 and sw.sid = s.sid
 and l.type = 'TX' and l.request = 6
 and l.sid = s.sid
 and s.sql_hash_value = sq.hash_value and s.sql_address = sq.address
```

Note: "object_id" can be used to query DBA_OBJECTS.

Documentation

Concepts: Data Concurrency and Consistency

How Oracle Locks Data

Concepts: Row Locks (TX)

Notes

Tracing sessions: waiting on an enqueue

TX Transaction locks - Example wait scenarios

TX Lock "Transaction Enqueue"

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Long running statement while locks are held

A long running statement may delay the time between a transaction starts (via some DML) and commits. This statement may need to be tuned.

M *Effort Details*

Medium effort; requires access and examination of application code.

L *Risk Details*

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

If the locking situation is in progress (not at some point in the past), the following steps may help identify the reason for it:

1. *Identify the session holding the TX enqueue that is being waited on. You can use utllockt.sql (see Note ID 166534.1).*
2. *For the session identified in step 1 (lock holder), check if it is currently executing SQL. For example, assuming the session ID is 12:*

```
SELECT s.sid, s.status, sq.sql_text
FROM v$session s, v$sql sq
WHERE s.sid = 12
```

```
and s.status = 'ACTIVE'
and s.sql_hash_value = sq.hash_value
and s.sql_address = sq.address
```

3. ***Investigate the performance of this cursor. At this point, the problem becomes a query tuning problem. You can use this guide for help or the SQL Tuning Advisor (10g or higher with EM Tuning Pack license).***
4. ***Tune the query and evaluate whether the locking problem has been resolved. If it hasn't been resolved, examine the application in more detail to see if the application should be changed.***

See the documents below for more information.

10g+: SQL Tuning Advisor

How to use the Sql Tuning Advisor

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

TBD

Manual Tuning

SQL Tuning Overview

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX allocate ITL entry

Waits for enq:TX allocate ITL entry (in mode 4) can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.
Wait class: Concurrency, typically foreground

What to look for

- TKProf:
  - Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq:TX - allocate ITL entry waits.
  - optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 4
- AWR or statspack:
  - enq:TX - allocate ITL entry is among the top timed events

Cause Identified: Insufficient ITLs in a block

Waits for the TX enqueue in mode 4 can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4.

Cause Justification
Prior to 9.2.x:
In versions prior to 9.2.x, it is difficult to pinpoint an ITL wait exactly. Consider this justified if you have examined other causes for TX, mode 4 waits and none are justifiable.

9.2.x:
Using statspack snapshots taken at level 7, look in the segment statistics section to see which segments have the highest ITL waits (e.g., Segments by ITL Waits). If these are a significant portion of the TX enqueue waits (see the Enqueue Activities section), then this cause is justified.

In 10g, the wait event itself tells you this is an ITL wait, so it is justified from the wait event.

Solution Identified: Increase the table's INITRANS setting

Increase the table's INITRANS setting to account for the number of concurrent sessions changing an individual block.

L Effort Details

Medium effort; may require rebuilding the table.

L Risk Details

Low risk; no risky side effects except if INITRANS is set too large and the block size is small (this will waste a lot of block space).

Solution Implementation

See the documents below.

Documentation
SQL Ref: INITRANS

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

10g+: Wait: enq - TX index contention

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).
Wait class: Concurrency, typically foreground

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for enq:TX - allocate ITL entry waits.*
  - *optional: In the raw 10046 trace file, most wait's P1 field decodes to TX in mode 4*
- *AWR or statspack:*
  - *enq:TX - allocate ITL entry is among the top timed events*

Cause Identified: Index contention due to block splits

This wait can occur when a transaction is inserting a row in an index and has to wait for the end of an index block split being done by another transaction. In this case the session is waiting for the TX enqueue in mode 4 (share).

Cause Justification
To identify which segment is involved:

- *Look in the TKProf of one or more sessions that experience the most of this kind of wait.*
- *Find the statement that waited the longest amount of time on the event with long TX, mode 4 waits. This is generally an insert statement.*
- *Examine the statement to find the indexes involved.*

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L ***Effort Details***

Low effort; will require rebuilding an index.

M ***Risk Details***

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread index values around

Hash partitions will randomly spread out the data and reduce contention on leaf blocks.

M ***Effort Details***

Medium effort; requires rebuilding the table.

L ***Risk Details***

Low risk; no risky side effects.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

Performance Tuning Guide: Using Partitioned Indexes for Performance:

SQL Ref: Create Index, Index Partitioning Clauses, GLOBAL PARTITION BY HASH:

Creating a Hash-Partitioned Global Index: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use natural keys instead of sequence numbers

Change the application to use a different key (i.e., a composite natural key) that does not increase monotonically. Obviously, this kind of change may take some time to implement because it may require many changes to application as well as changes to the physical data model.

H ***Effort Details***

High effort; may require changes to the data model and application code.

L ***Risk Details***

Low risk if implemented properly. But, changes of this magnitude are risky if not designed or tested properly.

Solution Implementation

See the documents below.

Documentation

Performance Tuning: Serializing within Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: library cache pin

Library cache pins are used to manage library cache concurrency. Pinning an object causes the heaps to be loaded into memory (if not already loaded). PINS can be acquired in NULL, SHARE or EXCLUSIVE modes and can be considered like a special form of lock. A wait for a "library cache pin" implies some other session holds that PIN in an incompatible mode.

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for library cache pin waits.***
- ***AWR or statspack:***
  - ***Significant waits for library cache pin***

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed parse time***
- ***Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses***
- ***Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).***

AWR or statspack reports:

- ***Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen***
- ***The "% SQL with executions>1" is over 60%, meaning statements are being shared***

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; increasing the size of the shared pool is not risky unless:

- ***There are many unshared statements due to literals***
  For more details, see: Understanding and Tuning the Shared Pool
- ***The machine doesn't have enough physical memory and starts swapping***
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M ***Effort Details***

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M ***Risk Details***

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation
GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes
Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Objects being compiled across sessions

One or more sessions are compiling objects (typically PL/SQL) while another session wants to pin the same object prior to executing or compiling it. One or more sessions will wait on library cache pin in Share mode (if it just wants to execute it) or eXclusive mode (if it want to compile/change the object).

Cause Justification
TKProf:

- *library cache pin waits and / or library cache pin waits*
- *Statement is compiling or executing PL/SQL*

Solution Identified: Avoid compiling objects in different sessions at the same time or during busy times

Do not compile interdependent objects across concurrent sessions or during peak usage.
The HangAnalyze command can usually help identify the blockers, waiters, and the SQL which is causing the waits (see the "Hang / Locking tab > Issue Identification > Data Collection" for more information).

L *Effort Details*

Low effort; requires some thought on how and when to recompile objects.

L *Risk Details*

Low risk.

Solution Implementation

Schedule and/or sequence the recompilation to avoid conflicts.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M ***Effort Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M ***Risk Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference
CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting
Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Contention caused by changing object privileges

Changing object privilges causes contention in the library cache since the object will need to be invalidated and reparsed with the new privileges. Any type of privilege change using GRANT or REVOKE on an object may cause dependent objects to become invalidated too thereby amplifying the effect of the change and causing contention if the system is busy.

Cause Justification
This cause is likely if there are:

- *waits on the library cache, shared pool latches, mutexes, and/or library cache pins*
- *High invalidations*
- *DDL and other causes have been eliminated*

Solution Identified: Avoid making grants during periods of high activity or concurrency

Schedule the privilege changes when the system is quiet to avoid impacting users.

L *Effort Details*

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M *Risk Details*

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: library cache lock

The library cache lock controls the concurrency between clients of the library cache by acquiring a lock on the object handle so that either:

- *one client can prevent other clients from accessing the same object*
- *The client can maintain a dependency for a long time (no other client can change the object).*

This lock is also obtained as part of the operation to locate an object in the library cache (a library cache child latch is obtained to scan a list of handles, then the lock is placed on the handle once the object has been found).

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for library cache lock waits.*
- *AWR or statspack:*
  - *Significant waits for library cache lock*

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the presence of literal values.*

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M *Effort Details*

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

*Pro*C/C++ Precompiler Programmer's Guide*

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- *EXACT: Leave the statement as it was written with literals (default value)*
- *FORCE: Substitute all literals with binds (as much as possible)*
- *SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)*

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L *Effort Details*

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; increasing the size of the shared pool is not risky unless:

- ***There are many unshared statements due to literals***
  For more details, see: Understanding and Tuning the Shared Pool
- ***The machine doesn't have enough physical memory and starts swapping***
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M ***Effort Details***

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M ***Risk Details***

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation
GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes
Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Objects being compiled across sessions

One or more sessions are compiling objects (typically PL/SQL) while another session wants to pin the same object prior to executing or compiling it. One or more sessions will wait on library cache pin in Share mode (if it just wants to execute it) or eXclusive mode (if it want to compile/change the object).

Cause Justification
TKProf:

- *library cache pin waits and / or library cache pin waits*
- *Statement is compiling or executing PL/SQL*

Solution Identified: Avoid compiling objects in different sessions at the same time or during busy times

Do not compile interdependent objects across concurrent sessions or during peak usage.
The HangAnalyze command can usually help identify the blockers, waiters, and the SQL which is causing the waits (see the "Hang / Locking tab > Issue Identification > Data Collection" for more information).

L *Effort Details*

Low effort; requires some thought on how and when to recompile objects.

L *Risk Details*

Low risk.

Solution Implementation

Schedule and/or sequence the recompilation to avoid conflicts.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Auditing is turned on

Auditing will increase the need to acquire library cache locks and potentially increase contention for them. This is especially true in a RAC environment where the library cache locks become database-wide (across all instances).

Cause Justification
AWR / Statspack:

- ***library cache lock waits***
- ***audit_trail parameter is set to something other than "none"***

Solution Identified: Evaluate the need to audit

Consider disabling auditing if it is not absolutely necessary.

L ***Effort Details***

Low effort; initialization parameter change

L ***Risk Details***

Low risk.

Solution Implementation

See the documents below.

Documentation
Keeping Audited Information Manageable

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Unshared SQL in a RAC environment

Library cache locks waits may occur in RAC environments when applications are not sharing SQL. In single-instance environments, library cache and shared pool latch contention is typically the symptom for unshared SQL. However, in RAC, the main symptom may be library cache lock contention.

Cause Justification
RAC environment

TKProf:
- *Many statements are hard parsed*
- *library cache lock waits occur as part of a hard parse*

AWR / Statspack:
- *library cache lock waits*
- *Low percentage for "% SQL with executions>1" (less than 60%)*
- *soft parse ratio is below 80%*

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M *Effort Details*

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting
Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation
7.3.1.3 SQL Sharing Criteria

Searches
*Pro*C/C++ Precompiler Programmer's Guide*

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- *EXACT: Leave the statement as it was written with literals (default value)*
- *FORCE: Substitute all literals with binds (as much as possible)*
- *SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)*

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L *Effort Details*

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M *Risk Details*

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*

- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Extensive use of row level triggers

When row level triggers are fired frequently, higher than usual library cache activity may occur, because of the need to check if mutating tables are being read. During trigger execution, it is possible that the application tries to read mutating tables, i.e., tables that are in the process of being modified by the statement that caused the trigger to fire. As this may lead to inconsistencies, it is not allowed, and the application should receive the error ORA-4091. The mechanism to detect this error involves one library cache lock acquisition per table referenced in each select statement executed.

The extent of the problem depends on how many times the row triggers fire rather than on the number of row triggers have been created (i.e., one trigger that fires 10000 times will cause more problems than 100 triggers that fire once).

Cause Justification
TKProf:

- *Many statements are hard parsed*
- *library cache lock waits*
- *evidence of a row level trigger firing (maybe some recursive SQL related to a trigger)*

Solution Identified: Evaluate the need for the row trigger

Sometimes row triggers aren't needed to accomplish the functionality. Consider if there is an alternative.

M *Effort Details*

Medium effort; may require application and schema changes

M *Risk Details*

Medium risk. If the application and schema changes, there is a possibility that some adverse effect will be introduced. Thorough testing will be needed.

Solution Implementation

Requires understanding the application and how row-level triggers are used. See the documents below for reference information.

Documentation

App Dev Guide: Coding Triggers

App Dev Guide: Coding Triggers

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M *Effort Details*

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M *Risk Details*

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference

CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

5. Concurrency - Latches and Mutexes

Waits for latches and mutexes that are used to coordinate operations.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: latch: cache buffers chains

Block headers in the buffer cache are placed on linked lists (cache buffer chains) which are accessed through a hash table. One or more hash chains are protected by one child of this latch. Processes need to get the child latch to scan for a buffer. This prevents the linked list from changing while scanning.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall summary for non-recursive and recursive statements shows significant amount of time for latch: or latch free waits.***
  - ***The cache buffers chains latch is a significant part of the total waits***
- ***AWR or statspack:***
  - ***10g or higher: latch: cache buffers chains waits is among the top timed events***
  - ***Prior to 10g: latch free waits is among the top timed events and the cache buffers chains latch is among the more prominent latches (high wait times or sleeps)***

Cause Identified: Hot blocks due to inefficient execution plan

Hot blocks refer to block headers that are accessed very frequently (via logical reads) and this frequent access leads to contention on the cache buffers chains latch.

Inefficient execution plans may perform many logical reads while they visit many blocks. If this query is executed by many sessions concurrently (or other similar queries against the same blocks), then there will be contention on these blocks.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed fetch time***
- ***Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.***

AWR or statspack reports:

- ***10g: waits on latch: cache buffers chains***
- ***Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch***
- ***Examine the Top SQL sections; look for statements with the highest elapsed time. You will see some of these statements performing a large number of buffer gets (logical reads) per execution***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Hot blocks due to concurrently accessing a popular block

Hot blocks refer to block headers that are accessed very frequently (via logical reads) and this frequent access leads to contention on the cache buffers chains latch.

This particular kind of hot block contention occurs when a query only reads a few blocks but this same query (or other ones that access the same blocks) are executed by many sessions at the same time. The problem can't be solved by tuning the query because the execution plan is already efficient. Either the query must be executed less often or the rows need to be spread out among more blocks.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed fetch time***
- ***Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.***
- ***Check if these statements only access a few blocks per execution ([query + current] / executions is low)***

AWR or statspack reports:

- ***10g: waits on latch: cache buffers chains***
- ***Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch***
- ***Examine the Top SQL sections; look for statements with the highest elapsed time. You will see some of these statements performing a small number of buffer gets (logical reads) per execution***

Solution Identified: Spread out the rows over more blocks

Alter (or even rebuild) tables listed above to use a higher PCTFREE setting. This will reduce the number of rows per block and hopefully, spread out contention for the blocks (at the expense of wasting space).

M ***Effort Details***

Medium effort; will require rebuilding a table.

M ***Risk Details***

Medium risk; some queries may run slower because they will need to access more blocks to obtain the same number of rows. Review how this table is accessed before implementing this solution.

Solution Implementation

Rows can be spread out by rebuilding the table using a larger value for PCTFREE. Another way to spread rows out is to make use of the table option, MINIMIZE RECORDS_PER_BLOCK as follows:

1. ***Export the table***
2. ***Truncate the table***
3. ***Insert the desired number of rows per block. E.g., if you only want 10 rows per block, insert just 10 rows.***
4. ***Alter the table to set minimize records_per_block setting. E.g.,***

```
Alter table stock_prices minimize records_per_block;
```

5. ***Delete the rows you inserted in step 3***
6. ***Import the table***

See the documents below for additional information.

Documentation

The PCTFREE Parameter

SQL Ref: Minimize records_per_block Clause

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use reverse key indexes

Index leaf blocks may see contention due to key values that are increasing steadily (using a sequence) and concentrated in a leaf block on the "right-hand side" of the index. Look at using reverse key indexes (if range scans aren't commonly used against the segment).

A reverse key index will spread keys around evenly and avoid creating these hot leaf blocks. However, the reverse key index will not be usable for index range scans, so care must be taken to ensure that access is normally done via equality predicates.

L *Effort Details*

Low effort; will require rebuilding an index.

M *Risk Details*

Medium risk; some queries may run much slower because they will not be able to use an index for range scans and may resort to full table scans. Determine if range scans are needed before implementing this.

Solution Implementation

See the documents below.

Documentation

Concepts: Reverse Key Indexes

SQL Reference: Create index syntax:

Performance Tuning Guide: Reverse Key Indexes

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*

- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use hash partitioning to spread values across blocks

Hash partitioning will distribute rows evenly for a given column in a table.

M *Effort Details*

Medium effort; requires recreating the table and importing rows into it.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Documentation

Concepts: Overview of Hash Partitioning

When to Use Hash Partitioning

SQL Ref: Create Table, hash partitioning clause

Creating a Hash-Partitioned Table: Example

Notes

Boosting Performance by Hash and Composite Partitions

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Bug 4742607 - "cache buffer chains" latch contention from concurrent index range scans

Concurrent index range-scan initializations can lead to contention on the "cache buffers chains" hash latches due to latch upgrades.

Bug 4742607 - "cache buffer chains" latch contention from concurrent index range scans

Cause Justification

TKProf:

- ***Use the report sorted by elapsed fetch time***
- ***Look at the top statements and determine if they are seeing latch contention on the cache buffers chains latch.***
- ***The execution plans for these statements make use of index scans***

AWR or statspack reports:

- ***10g: waits on latch: cache buffers chains***
- ***Pre-10g: waits on latch free and highest latch time or sleeps is on the cache buffers chains latch***
- ***In the Instance Statistics, you will see high "shared hash latch upgrade" statistic counts***

Solution Identified: Bug 4742607

Bug 4742607

M ***Effort Details***

Medium effort; requires a patch.

L ***Risk Details***

Low risk; this patch has been well proven.

Solution Implementation

See the documents below.

Additional bug information:
Bug 4742607

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Wait: latch: library cache
Wait: latch: shared pool

Library Cache Latch
The library cache latches serialize access to the objects in the library cache. Every time that a SQL statement, a PL/SQL block or a stored object (Procedures, packages, functions, triggers) is parsed or executed, this latch is acquired to ensure the object doesn't change while it is locked in the library cache.

Shared Pool Latch
Free memory in the shared pool is tracked on a number of freelists. The shared pool latch is typically acquired when a chunk of memory is requested, and lasts while scanning the relevant freelists for a chunk of the required size. The latch may also be acquired for other operations such as coalescing memory or releasing memory back to a freelist.

Wait class: Concurrency, typically foreground

What to look for

- ***TKProf:***
  - ***Overall summary for non-recursive and recursive statements shows significant amount of time for latch: or latch free waits.***
  - ***The library cache or shared pool latch is a significant part of the total waits***

- ***AWR or statspack:***
  - ***10g or higher: latch: library cache waits is among the top timed events***
  - ***Prior to 10g: latch free waits is among the top timed events and the library cache or shared pool latch is among the more prominent latches (high wait times or sleeps)***

Cause Identified: Unshared SQL Due to Literals

SQL statements are using literal values where a bind value could have been used. The literal values cause the statement to be unshared and will force a hard parse.

Cause Justification

TKProf :
- ***Use the report sorted by elapsed parse time***
- ***Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses***
- ***Examine the statements that are being hard parsed and look for the presence of literal values.***

Solution Identified: Rewrite the SQL to use bind values

Rewriting the SQL to use bind values will allow the statement to be reused when specific values in the statement change but the overall statement is the same. This is the best way to promote sharing of SQL statements in the library cache.

M ***Effort Details***

Medium or high effort; rewriting statements requires a change to the application but the change is rather trivial.

M ***Risk Details***

Medium risk; the use of bind values could lead to worse execution plans for some statements. The statements modified to use binds values should be thoroughly tested to avoid regressing the statement's performance.

Solution Implementation

See the documents below.

Troubleshooting

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Searches

Pro*C/C++ Precompiler Programmer's Guide

Performance Tuning Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use the CURSOR_SHARING initialization parameter

The CURSOR_SHARING parameter will substitute literal values with bind values in a statement automatically. The settings for this parameter are:

- ***EXACT: Leave the statement as it was written with literals (default value)***
- ***FORCE: Substitute all literals with binds (as much as possible)***
- ***SIMILAR: Substitute literals with binds only if the query's execution plan won't change (i.e., safe literal replacement)***

In general, most OLTP apps that use equality predicates will see little change to their execution plans, but the effects of these parameters should be tested in your application.

These parameters can be set at the session level to further contain their effects - this is the preferred way to use them to minimize widespread changes.

L ***Effort Details***

Low effort; an init.ora / spfile change. In the worst case it may require a LOGON trigger to set it for a session.

M ***Risk Details***

Medium risk; the use of bind values could lead to worse execution plans for some statements. Risk can be mitigated by using SIMILAR instead of FORCE but this may not make enough statements shareable.

Solution Implementation

See the documents below.

Reference

Reference: CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting

CURSOR_SHARING for Existing Applications

Understanding and Tuning the Shared Pool

Handling and resolving unshared cursors/large version_counts

Documentation

7.3.1.3 SQL Sharing Criteria

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Shared SQL being aged out

The shared pool is too small and is causing many statements that could be shared to age out of the library cache and later reloaded. Each reload requires a hard parse and impacts the CPU and latches.

Cause Justification

TKProf:

- ***Use the report sorted by elapsed parse time***
- ***Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses***
- ***Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).***

AWR or statspack reports:

- ***Library Cache statistics section shows that reloads are high (usually several thousand per hour) and little or no invalidations are seen***
- ***The "% SQL with executions>1" is over 60%, meaning statements are being shared***

Solution Identified: Increase the size of the shared pool

Increasing the shared pool size will reduce the need to age out statements that could be shared.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; increasing the size of the shared pool is not risky unless:

- ***There are many unshared statements due to literals***
  For more details, see: Understanding and Tuning the Shared Pool
- ***The machine doesn't have enough physical memory and starts swapping***
  TECH: Unix Virtual Memory, Paging & Swapping explained

Verify the above points before changing the size of the shared pool.

Solution Implementation

See the documents below.

Documentation

Admin: Using Manual Shared Memory Management, see Specifying the Shared Pool Size

Reference: SHARED_POOL_SIZE Parameter

Reference: SHARED_POOL_SIZE and Automatic Storage Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L ***Effort Details***

Low effort; an init.ora / spfile change.

L ***Risk Details***

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Keep ("pin") frequently used large PL/SQL and cursor objects in the shared pool

Use the DBMS_SHARED_POOL.KEEP() procedure to mark large, frequently used PL/SQL and SQL objects in the shared pool and avoid them being aged out. This will reduce reloads and fragmentation since the object doesn't need to keep reentering the shared pool over and over.

M ***Effort Details***

Medium effort; need to identify which objects should be kept and then run a procedure to keep them.

M ***Risk Details***

Medium risk; if you aren't careful in keeping these objects, you may keep too many of them and cause ORA-4031 errors.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Performance Tuning: Keeping Large Objects to Prevent Aging

PL/SQL DBMS_SHARED_POOL

How-To

How To Pin Objects in Your Shared Pool

How to Automate Pinning Objects in Shared Pool at Database Startup

How To Use SYS.DBMS_SHARED_POOL In a PL/SQL Stored procedure To Pin objects in Oracle's Shared Pool

Reference

Using the Oracle DBMS_SHARED_POOL Package

Understanding and Tuning the Shared Pool

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Library cache object Invalidations

When objects (like tables or views) are altered via DDL or collecting statistics, the cursors that depend on them are invalidated. This will cause the cursor to be hard parsed when it is executed again and will impact CPU and latches.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being hard parsed; these will have "Misses in the library cache" equal or close to the total number of parses*
- *Examine the statements that are being hard parsed and look for the ABSENCE of literal values, this means these statements could have been shared but weren't (this is not entirely reliable since you could have statements that use binds but will not be executed again).*

AWR or statspack reports:

- *Library Cache statistics section shows that reloads are high (usually several thousand per hour) and invalidations are high*
- *The "% SQL with executions>1" is over 60%, meaning statements are being shared*
- *Check the Dictionary Statistics section of the report and look for non-zero values in the Modification Requests column, meaning that DDL occurred on some objects.*

Solution Identified: Do not perform DDL operations during busy periods

DDL will often cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

L *Effort Details*

Low effort; defer the DDL to a quiet time.

L *Risk Details*

Low risk; may involve some downtime.

Solution Implementation

Not Applicable. Simply schedule DDL during maintenance or low activity periods.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not collect optimizer statistics during busy periods

Collecting statistics (using ANALYZE or DBMS_STATS) will cause library cache objects to be invalidated and this could cascade to many different dependent objects like cursors. Invalidations have a large impact on the library cache, shared pool, row cache, and CPU since they will likely require many hard parses to occur at the same time.

For some database versions, the DBMS_STATS procedure allows give you the option of not invalidating objects (see the "no_invalidate" option).

L ***Effort Details***

Low effort; defer the gathering of statistics to a quiet time. In 10g, you have a choice of whether or not to invalidate objects after gathering statistics.

L ***Risk Details***

Low risk; defer the gathering of statistics to a quiet time.

Solution Implementation

The document links below shows how to specify statistics collection without causing invalidations.

Documentation
GATHER_TABLE_STATS Procedure, see the "no_invalidate" option

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Do not perform TRUNCATE operations during busy periods

See the document below:

L ***Effort Details***

Low effort; defer the DDL to a quiet time.

L ***Risk Details***

Low risk; may involve some downtime.

Solution Implementation

See documents below:

Notes
Truncate - Causes Invalidations in the LIBRARY CACHE

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive soft parsing

Soft parsing occurs when Oracle looks in the library cache for a cursor or object it hopes to share. If it finds the cursor and it is sharable (same optimizer mode, etc), then it will consider this as a soft parse. Soft parsing is more efficient than hard parsing but still impacts latches to a degree.

Cause Justification

TKProf:

- *Use the report sorted by elapsed parse time*
- *Look at the top statements and determine if they are being soft parsed; these will have "Misses in the library cache" close to zero*

AWR or statspack reports:

- *The Instance Efficiency Percentages will report high values (usually over 60%) for Soft Parse %*

Solution Identified: Avoid unnecessary soft parsing in the application

Application code will sometimes needlessly force a soft parse when it could have simply used an open cursor handle and re-executed the cursor with new bind values. Look through the application code and determine whether the soft parse is really needed.

M *Effort Details*

Medium effort; will require coordination with developers to review and change code.

L *Risk Details*

Low risk; the change should be very localized.

Solution Implementation

Ensure your application doesn't perform unnecessary soft parsing. Typically this occurs when a parse statement is placed in the middle of a loop that iterates over a set of rows. Consider this pseudo-pseudo-code:

```
list_of_rows = Retrieve some rows()
FOR each row in list_of_rows LOOP
        cursor_handle = PARSE(sql)  # parse for each loop iteration
        EXECUTE(cursor_handle, bind1, bind2)
        CLOSE(cursor_handle)
END LOOP
```

To avoid the repeated soft parses:

```
list_of_rows = Retrieve some rows()
cursor_handle = PARSE(sql)  # parse once
FOR each row in list_of_rows LOOP
        EXECUTE(cursor_handle, bind1, bind2)
END LOOP
CLOSE(cursor_handle)
```

It's also a good idea to make sure the application leaves cursors open and doesn't re-open them unnecessarily (see references below for best-practice information on this and compensating this using the SESSION_CACHED_CURSORS parameter).

Documentation

Performance Tuning Guide: Using the Shared Pool Effectively

Performance Tuning Guide: Cursor Access and Management

Programmer's Guide to the Oracle Precompilers: Eliminating Unnecessary Parsing

Pro*C/C++ Programmer's Guide: Eliminating Unnecessary Parsing

JDBC Developer's Guide and Reference : Statement Caching

Reference

SQL Parsing Flow Diagram

How to work out how many of the parse count are hard/soft?

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Ensure session cached cursors are used

The session cached cursors parameter will allow Oracle to maintain a small cache of handles to cursors in the library cache. During a parse, the cache will be examined and if a match is found, the soft parse will be avoided.

Review the value of this parameter and consider increasing it (although it should be increased slowly and not above 200 to avoid locking too many statements in the library cache).

L ***Effort Details***

Low effort; a parameter change

L ***Risk Details***

Low risk; the change can be localized to a session. Not risky as long as the values is not increased over 200.

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Caching Session Cursors

Reference: SESSION_CACHED_CURSORS parameter

Performance Tuning Guide: Using the Shared Pool Effectively

Performance Tuning Guide: Cursor Access and Management

Notes

Understanding and Tuning the Shared Pool, see SESSION_CACHED_CURSORS parameter

Reference Note for Init.Ora Parameter "SESSION_CACHED_CURSORS"

SCRIPT - to Gauge the Impact of the SESSION_CACHED_CURSORS Parameter

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Excessive Amount of Child Cursors

A large number of child cursors are being created for some SQL statements. This activity is causing contention among various sessions that are creating child cursors concurrently or with other sessions that also need similar resources (latches and mutexes).

Cause Justification
AWR / Statspack reports; look in the "SQL ordered by Version Count" section. If there are any SQL statements with more than 500 versions, then this problem is likely to be occurring. Alternatively, you can query V$SQLAREA to look for any SQL with version_count greater than 500.

Query V$SQL_SHARED_CURSOR to see the reasons why SQL isn't being shared.

Solution Identified: Inappropriate use of parameter CURSOR_SHARING set to SIMILAR

The difference between SIMILAR and FORCE is that SIMILAR forces similar statements to share the SQL area without deteriorating execution plans. Setting CURSOR_SHARING to FORCE forces similar statements to share the SQL area potentially deteriorating execution plans.

One of the cursor sharing criteria when literal replacement is enabled with CURSOR_SHARING as SIMILAR is that bind value should match initial bind value if the execution plan is going to change depending on the value of the literal. The reason for this is we might get a sub-optimal plan if we use the same cursor. This would typically happen when, depending on the value of the literal, the optimizer is going to chose a different plan. For example, if we have a predicate with " > ", then each execution with different bind values would result in a new child cursor because that would ensure that the plan didn't change (a range predicate influences cost and plans), if this was an equality predicate, we would always share the same child cursor.

Avoiding the use of CURSOR_SHARING set to SIMILAR entails either rewriting the SQL in the application so that it uses bind values and still gets a good plan (hints, profiles, or outlines may be needed), or using CURSOR_SHARING set to FORCE which will avoid generating child cursors but can cause plans to be sub-optimal.

M ***Effort Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is easy; changing the application to use binds will take more effort.

M ***Risk Details***

Depends on the change made. Changing the CURSOR_SHARING initialization parameter to FORCE is risky if done at the database instance level, but less risky at the session level. Changing the application SQL is not as risky since only the single statement is affected.

Solution Implementation

See documents below:

Reference
CURSOR_SHARING Parameter

Init.ora Parameter "CURSOR_SHARING" Reference Note

Troubleshooting
Handling and resolving unshared cursors/large version_counts

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Contention caused by changing object privileges

Changing object privilges causes contention in the library cache since the object will need to be invalidated and reparsed with the new privileges. Any type of privilege change using GRANT or REVOKE on an object may cause dependent objects to become invalidated too thereby amplifying the effect of the change and causing contention if the system is busy.

Cause Justification
This cause is likely if there are:

- *waits on the library cache, shared pool latches, mutexes, and/or library cache pins*
- *High invalidations*
- *DDL and other causes have been eliminated*

Solution Identified: Avoid making grants during periods of high activity or concurrency

Schedule the privilege changes when the system is quiet to avoid impacting users.

L *Effort Details*

Depends on the availability requirements of the system; no extra effort is involved - just rescheduling.

M *Risk Details*

Low risk; some contention is possible if the time period was not quiet enough

Solution Implementation

N/A

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

6. Configuration

Waits caused by inadequate configuration of database or instance resources (for example, undersized log file sizes, shared pool size).

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: free buffer waits

This wait event indicates that a server process was unable to find a free buffer and has posted the database writer to make free buffers by writing out dirty buffers (buffers w/unwritten changes). Once DBWR finishes writing the dirty buffers to disk, they are free to be reused.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for free buffer waits waits.*
- *AWR or statspack:*
  - *Significant waits for free buffer waits*

Cause Identified: CPU saturation

CPU saturation can induce certain wait events like latch contention, log file sync, or cluster-related events.

In some cases, a foreground process depends on a background process for an operation (e.g., a foreground's commit waits for logwriter to flush redo to disk). If the background process has to wait for CPU, then any dependent foreground processes will also wait.

Cause Justification
OS Data shows that CPU utilization is at or near 100% and the run queue size per CPU is greater than 4. This condition should have been caught earlier in the diagnostic process when OS data was being analyzed.

Solution Identified: Investigate the reasons for CPU saturation

See this guide's "Issue Identification > Analysis > Verify Oracle OS Resource Usage" section for more details.

L ***Effort Details***

Low effort

L ***Risk Details***

Low risk

Solution Implementation

Determine which processes are using most of the CPU on the machine. They could be Oracle processes (including more than one instance) or non-Oracle processes. If they are Oracle processes, then you should have detected this already in a previous step and investigated the reasons for Oracle's CPU consumption (of course, better late than never). Otherwise, you will need to find out how to handle the non-Oracle CPU consumption (outside of our scope).
You can use various OS tools and Oracle EM to investigate this.

For example, use the top utility or the ps command, ps -ef -o pid,pcpu,comm | sort -k 2 (this will give you a sorted list of processes using CPU - look at the 2nd column, "% CPU").

See the documents below for additional details.

How-To

How to use OS commands to diagnose Database Performance issues?

Diagnosing High CPU Utilization

Reference

Enterprise Manager: Host Performance page

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Poor file write performance in some filesystems

Some filesystems have poor write performance (writes take too long) and is impacting DBwriter's ability to keep enough clean buffers in the buffer cache.

Cause Justification
AWR / Statspack:

- ***free buffer waits***
- ***db file parallel write waits have an average wait time LARGER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter)***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation
I/O Configuration and Design

Wait Event: db file scattered read

Notes
Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

If the buffer cache is too small and filled with hot blocks, then sessions will be starved for free buffers (clean, cold blocks) and will need to spend too much time looking for free buffers and/or posting DBWR to write dirty blocks and make them free. Increase the parameter DB_BLOCK_BUFFERS(Oracle8+) or DB_CACHE_SIZE (Oracle9+) and monitor the effect of the change.

Cause Justification
AWR / Statspack:

- ***free buffer waits***
- ***DBWriter is not seeing a performance problem in writing the files. Specifically, db file parallel write waits have an average wait time SMALLER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter***
- ***You may see high values (compared to a baseline) for statistics write clones, hot blocks moved to the head of the LRU, and free buffers inspected***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: DBWriter is not using asynchronous I/O

The DBWriter will achieve optimal throughput when asynchronous I/O is available to it. DBWriter may not be able to keep up with buffer demands if asynch I/O is not available.

Cause Justification
AWR / Statspack:

- ***free buffer waits***
- ***DBWriter is seeing a performance problem in writing the files. Specifically, db file parallel write waits have an average wait time LARGER than several hundred milliseconds (DBwriter writes in batches so the rule of them is higher than 20mSec / write for DBWriter***
- ***Asynchronous I/O is disabled via the initialization parameter disk_asynch_io or filesystemio_options***

Solution Identified: Enable asynchronous I/O

Enable asynchronous I/O If the platform supports it. This is preferred over adding multiple DBwriters or I/O slaves.

L ***Effort Details***

Low effort; initialization parameter change

L ***Risk Details***

Low risk. Ensure your platform supports it and is up-to-date on patches

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Choosing Between Multiple DBWR Processes and I/O Slaves

Performance Tuning Guide: Asynchronous I/O

AIX: Using Asynchronous I/O

HPUX: Using Asynchronous I/O

Linux: Using Asynchronous I/O

Reference: DISK_ASYNCH_IO Parameter

Notes

How To Check If Asynchronous I/O Is Working On Linux

Asynchronous I/O (aio) on RedHat Advanced Server 2.1 and RedHat Enterprise Linux 3

Understanding and Tuning Buffer Cache and DBWR

Database Writer and Buffer Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Use multiple DBwriters

Enable asynchronous I/O If the platform supports it. However, if your platform doesn't support it, then adding multiple DBWriters can help divide the workload.

L *Effort Details*

Low effort; initialization parameter change

L *Risk Details*

Low risk.

Solution Implementation

See the documents below.

Documentation

Performance Tuning Guide: Choosing Between Multiple DBWR Processes and I/O Slaves

Understanding and Tuning Buffer Cache and DBWR

Database Writer and Buffer Management

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: log buffer space

This event occurs when server processes are writing redo records to the log buffer faster than LGWR can write them out; eventually, the log buffer fills up and the processes wait for free space. After LGWR writes some buffers out, then those buffers may be reused by other processes.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for log buffer space waits associated with DML statements.*
- *AWR or statspack:*
  - *Significant waits for log buffer space*

Cause Identified: The log buffer is too small

If the log buffer is too small, then the demand for redolog buffer space will overtake the supply of buffers and cause these waits.

Cause Justification
AWR / Statspack:

- *log buffer space waits*
- *initialization parameter, log buffer is smaller than:*
  *statistic: redo size per sec * 600 (10 min worth of redo)*
- *The average time for log file parallel write is less than 20mSec*

Solution Identified: Increase the size of the log buffer

Increase the parameter LOG_BUFFER to increase the redo log buffer size. Values of LOG_BUFFER larger than 32 MB (and even around 3 MB) will usually not have any effect (and will just waste memory).

M *Effort Details*

Medium effort; easy to change but requires the database to be restarted.

L *Risk Details*

Low risk; larger size log buffers could waste memory but will not adversely affect performance (unless there is a memory shortage on the machine).

Solution Implementation

See the documents below.

Documentation

Concepts: Redolog Buffer

Concepts: Log Writer Process (LGWR)

Reference: LOG_BUFFER parameter

Reference: log buffer space wait event

Performance Tuning Guide: Configuring and Using the Redo Log Buffer

Performance Tuning Guide: log buffer space wait event

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Logwriter is writing too slow

If the size of the log buffer is already large (more than 3 MB), speed up the LGWR background process write operations by ensuring that the I/O devices where the redolog files are stored are not suffering from I/O contention.

Cause Justification
AWR / Statspack:

- *log buffer space waits*
- *The average time for log file parallel write is MORE than 20mSec*

OS disk performance data on the filesystems where redologs are placed show disk response times greater than 20mSec.

Additional Information:

Concepts: Redolog Buffer

Concepts: Log Writer Process (LGWR)

Wait Event "log file parallel write" Reference Note

Tuning I/O-related waits, see 'log file parallel write' wait event section

Checkpoint Tuning and Troubleshooting Guide

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- ***Significant waits for the read by other session event***

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- ***Significant waits on waits , and/or for RAC, global cache CR request***
- ***SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution***
- ***Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event***
- ***The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).***

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L ***Effort Details***

Low effort; change an initialization parameter

L ***Risk Details***

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

7. Network

Waits caused by network related activity.

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

wait: SQL*Net message from dblink

The Oracle shadow process is waiting for a message over a database link from a remote process. Note that this wait is also used when waiting for data from "extproc" or from a remote gateway process.

What to look for

- ***TKProf:***
  - ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from dblink waits.***
- ***AWR or statspack:***
  - ***Significant waits for SQL*Net message from dblink***

Cause Identified: A remote database is not executing the query fast enough

If the local database is waiting for this event on a distributed query, the remote node(s) may be taking too long to execute the query and return results back to the local node.

Cause Justification
TKProf:

1. ***Focus attention on the remote database***
2. ***On the "remote" database, find the session corresponding to the "local" database (it will look like a typical database client)***
3. ***Determine how long it takes to execute the query sent over the dblink (best if you can trace this session with the 10046 event)***
4. ***If most of the time is spent executing the "remote" query, this issue is justified***

Solution Identified: Tune the remote query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail on the remote site using the information in the Performance Diagnostic Guide's Query Tuning section.

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

M ***Effort Details***

Medium; tracing distributed queries is more challenging than local queries.

L ***Risk Details***

Not applicable

Solution Implementation

In addition to using the Performance Diagnostic Guide's Query Tuning section, see the documents below for specific issues with distributed queries.

Documentation

Concepts: Distributed Database Concepts

Admin Guide: Tuning Distributed Queries

Notes

Distributed Queries

Determining the execution plan for a distributed query

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

8. Reads / Writes

Waits for I/O (for example 'db file sequential read').

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Wait: db file sequential read

The session waits while a sequential read from the database is performed. This event is also used for rebuilding the control file, dumping datafile headers, and getting the database file headers.

Wait class: User I/O, typically foreground

What to look for

- *TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for db file sequential read waits.*
- *AWR or statspack: db file sequential read waits is among the top timed events*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- *Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file*
- *Determine why the filesystems are performing poorly. Some common causes are:*
  - *"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth*
  - *hardware problem*
  - *In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.*

M *Effort Details*

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- *Significant waits on waits , and/or for RAC, global cache CR request*
- *SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution*
- *Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event*
- *The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).*

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*

- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Wait: db file scattered read

The session waits while a multiblock read from the database is performed. Similar to db file sequential read, except that the session is reading multiple data blocks and scattering them around the buffer cache.

Wait class: User I/O, typically foreground

What to look for

- ***TKProf: Overall summary for non-recursive and recursive statements shows significant amount of time for db file scattered read waits.***
- ***AWR or statspack: db file scattered read waits is among the top timed events***

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.***

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L ***Effort Details***

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L ***Risk Details***

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- ***Significant amount of the total time in TKProf is due to this wait event***
- ***The average time for this event (total time / wait count) is more than 20 mSec***

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

wait: direct path read

Direct Path operations (parallel execution, hash joins, sorts to disk) read data from datafiles directly into the PGA (opposed to the buffer cache in SGA). When the process attempts to access a block in the PGA that has not yet been read from disk, it then issues a wait call and updates the statistics for this event.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for direct path read waits.*
- *AWR or statspack:*
  - *Significant waits for direct path read*

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- *Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file*
- *Determine why the filesystems are performing poorly. Some common causes are:*
  - *"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth*
  - *hardware problem*
  - *In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.*

M *Effort Details*

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Incorrect manual workarea sizing

Oracle uses in-memory workareas in the PGA for performing sorts, hash joins, and other operations. These can be manually controlled by parameters such as sort_area_size and hash_area_size.

When these parameters are sized below what Oracle needs do an operation in memory, then some of the data will need to be written in temp segments causing direct path write waits. Eventually, this data will be read back and will cause direct path read waits.

Cause Justification
TKProf:

- *Significant waits on direct path read or direct path writes*
- *Execution plan shows sorts or hash join operations*
- *Average wait time is less than 20 mSec*

Solution Identified: Use automatic PGA memory management

*When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.*

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L *Effort Details*

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L *Risk Details*

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Parallel execution is occurring but not expected or desired

Parallel execution is occurring and causing CPU or I/O problems (typically direct path read/write waits) due to the degree of parallelism. PX is not expected; the CBO will attempt to use parallel operations if the following are set or used:

- *Parallel hint: parallel(t1, 4)*
- *ALTER SESSION FORCE PARALLEL*
- *Setting a degree of parallel and/or the number of instances on a table or index in a query*

Cause Justification

- *The process with very high direct path read waits is a parallel execution slave process.*
- *There are many more PX slave processes than expected or desired*
- *The filesystems where the I/O is occurring were never meant to handle the I/O bandwidth required by the number of PX processes*

Additional Information:

Summary of Parallelization Rules

Solution Identified: Remove parallel hints

The statement is executing in parallel due to parallel hints. Removing these hints may allow the statement to run serially.

L ***Effort Details***

Low effort; simply remove the hint from the statement.

L ***Risk Details***

Low risk, only affects the statement.

Solution Implementation

Remove one or more hints of the type:

- ***PARALLEL***
- ***PARALLEL_INDEX***
- ***PQ_DISTRIBUTE***

If one of the tables has a degree greater than 1, the query may still run in parallel.

Hint information:
Hints for Parallel Execution

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Alter a table or index's degree of parallelism

A table or index in the query has its degree (of parallelism) set higher than 1. This may be one factor causing the query to execute in parallel. If the parallel plan is not performing well, a serial plan may be obtained by changing the degree.

L ***Effort Details***

Low effort; the object may be changed with an ALTER command.

M ***Risk Details***

Medium risk; other queries may be running in parallel due to the degree setting and will revert to a serial plan. An impact analysis should be performed to determine the effect of this change on other queries.

The ALTER command will invalidate cursors that depend on the table or index and may cause a spike in library cache contention - the change should be done during a period of low activity.

Solution Implementation

See the documents below.

Parallel clause for the CREATE and ALTER TABLE / INDEX statements

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: direct path write

When a process is writing buffers directly from the PGA (as opposed to the DBWR writing them from the buffer cache), the process waits on this event to ensure that all outstanding write requests are completed. Example of "direct path writes" operations are: sorts that go to disk, parallel DML operations, direct-path INSERTs, parallel create table as select, and some LOB operations.

What to look for

- *TKProf:*
  - *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for direct path write waits.*
- *AWR or statspack:*
  - *Significant waits for direct path write*

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- ***Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file***
- ***Determine why the filesystems are performing poorly. Some common causes are:***
  - ***"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth***
  - ***hardware problem***
  - ***In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.***

M ***Effort Details***

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M ***Risk Details***

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Incorrect manual workarea sizing

Oracle uses in-memory workareas in the PGA for performing sorts, hash joins, and other operations. These can be manually controlled by parameters such as sort_area_size and hash_area_size.

When these parameters are sized below what Oracle needs do an operation in memory, then some of the data will need to be written in temp segments causing direct path write waits. Eventually, this data will be read back and will cause direct path read waits.

Cause Justification
TKProf:

- *Significant waits on direct path read or direct path writes*
- *Execution plan shows sorts or hash join operations*
- *Average wait time is less than 20 mSec*

Solution Identified: Use automatic PGA memory management

When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L *Effort Details*

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L *Risk Details*

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

wait: read by other session

A session wants to pin a block that is currently being read from disk into the buffer cache by another session.

What to look for

TKProf or AWR

- *Significant waits for the read by other session event*

Cause Identified: SQL tuning required; no I/O problems

If performance time is dominated by this wait event, then SQL tuning may reduce the number of reads and speed up queries.

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) should be less than 20 mSec to discount an I/O problem.*

Solution Identified: 10g+: Tune the query using the SQL Tuning Advisor

Oracle's SQL Tuning Advisor can help tune specific SQL statements quickly and easily if you are licensed to use the Enterprise Manager Tuning Pack.

L *Effort Details*

Low effort; the SQL Tuning Advisor is easy to use and requires little user effort to tune a statement.

L *Risk Details*

The tuning action will generally be to create a statement profile. The profile affects only a single statement. Other recommendations may have wide ranging effects and should be tested thoroughly.

Solution Implementation

See the documents below.

How-To

How to use the Sql Tuning Advisor

Documentation

Automatic SQL Tuning

Using Advisors to Optimize Database Performance

Using SQL Tuning Advisor with Oracle Enterprise Manager

Reference

SQL Tuning Advisor Subprograms

Using SQL Tuning Advisor APIs

Automatic SQL Tuning - SQL Profiles

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Tune the query using the Performance Diagnostic Guide's Query Tuning Section

This is a query tuning problem that needs to be addressed in detail using the information in the Performance Diagnostic Guide's Query Tuning section.

M *Effort Details*

Medium effort; manual query tuning can be easy or difficult depending on the query and application.

L *Risk Details*

Low risk; generally query tuning actions will affect only a single query. Of course this will depend on the ultimate actions taken and some of them can affect an entire instance.

Solution Implementation

Click on the Query Tuning tab, then skip to the Determine a Cause > Data Collection step.

Other helpful documents are listed below:

Documentation

SQL Tuning Overview

How-To

Diagnosing Query Tuning Problems

Diagnosing Why a Query is Not Using an Index

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: I/O performance problems

The average time for a an I/O is exceeds typical standards for I/O performance (less than 20 mSec).

Cause Justification

- *Significant amount of the total time in TKProf is due to this wait event*
- *The average time for this event (total time / wait count) is more than 20 mSec*

Solution Identified: Investigate possible I/O performance problems

To investigate further you must:

- *Find out which file numbers are causing the highest average waits and then determine which filesystem contains the file*
- *Determine why the filesystems are performing poorly. Some common causes are:*
  - *"hot filesystems" - too many active files on the same filesystem exhausting the I/O bandwidth*
  - *hardware problem*
  - *In Parallel Execution (PX) is being used, determine if the I/O subsystem is saturated by having too many slaves in use.*

M *Effort Details*

Medium effort; depends on the skill level of the system administrators. Correcting a problem can involve major effort to move files to a new destination.

M *Risk Details*

Medium risk; hardware changes and structural database changes carry risk that may require a restore. Backups should be taken and restoring procedures should be tested before attempting changes.

Solution Implementation

See the documents below.

Documentation

I/O Configuration and Design

Wait Event: db file scattered read

Notes

Tuning I/O-related waits

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve,

examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Buffer cache is too small

A small buffer cache will cause more physical reads or, for a RAC database, additional block transfers than would otherwise be required.

Cause Justification
TKProf:

- *Significant waits on waits , and/or for RAC, global cache CR request*
- *SQL statements perform 10 or fewer logical reads (query + current) per row per table per execution, meaning that the statement is reasonably tuned (i.e., if a query joins 2 tables and returns 10 rows, one would expect less than 10*2*3 = 60 logical reads per execution*
- *Full table scans (in a RAC database) are NOT seen in the execution plan for a statement that is waiting on this event*
- *The application is an OLTP type of application and in the overall section of the report, physical reads ("disk") are equal or close to the number of logical reads (query + current).*

Solution Identified: Manually Increase the size of the buffer cache using the db cache size parameter

Increase the size of the buffer cache and monitor the effects of the change.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. However, there must be sufficient memory on the machine to avoid memory shortage problems.

Solution Implementation

See the documents below.

Documentation

Concepts: Oracle Memory Architecture

Configuring and Using the Buffer Cache

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L *Effort Details*

Low effort; change an initialization parameter

L *Risk Details*

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Reduce Client Bottlenecks

A client bottleneck in the context of a slow database is another way to say that most of the time for sessions is being spent outside of the database. This could be due to a truly slow client or a slow network (and related components).

1. ***Observations and Causes***

Observations and causes are listed below:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

High Wait Time due to Client Events Before Any Type of Call

The Oracle shadow process is spending a significant amount of time waiting for messages from clients. The waits occur between FETCH and PARSE calls or before EXECUTE calls. There are few FETCH calls for the same cursor.

What to look for

TKProf:

- ***Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from client waits compared to the total elapsed time in the database***
- ***Each FETCH call typically returns 5 or more rows (indicating that array fetches are occurring)***

Cause Identified: Slow client is unable to respond to the database quickly

The client is running slowly and is taking time to make requests of the database.

Cause Justification
TKProf:

1. ***SQL*Net message from client waits are a large part of the overall time (see the overall summary section)***
2. ***There are more than 5 rows per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls). When array operations are used, you'll see 5 to 10 rows per execution.***

You may also observe that performance is good when the same queries that the client sends are executed via a different client (on another node).

Solution Identified: Investigate the client

Its possible that the client or middle-tier is saturated (not enough CPU or memory) and is simply unable to send requests to the database fast enough.

You will need to check the client for sufficient resources or application bugs that may be delaying database calls.

M ***Effort Details***

Medium effort; It is easy to check clients or mid-tiers for OS resource saturation. Bugs in application code are more difficult to find.

L ***Risk Details***

Low risk.

Solution Implementation

It may help to use a tool like OSWatcher to capture OS performance metrics on the client.

To identify a specific client associated with a database session, see the V$SESSION view under the columns, CLIENT_INFO, PROCESS, MACHINE, PROGRAM.

Documentation

Reference: V$SESSION

Notes

The OS Watcher (OSW) User Guide

The OS Watcher For Windows (OSWFW) User Guide

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Slow network limiting the response time between client and database

The network is saturated and this is limiting the ability of the client and database to communicate with each other.

Cause Justification
TKProf:

1. ***SQL*Net message from client waits are a large part of the overall time (see the overall summary section)***
2. ***Array operations are used. This is seen when there are more than 5 rows per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls)***
3. ***The average time for a ping is about equal to twice the average time for a SQL*Net message from client wait and this time is more than a few milliseconds. This indicates that most of the client time is spent in the network.***

You may also observe that performance is good when the same queries that the client sends are executed via a different client on a different subnet (especially one very close to the database server).

Solution Identified: Investigate the network

Check the responsiveness of the network from different subnets and interface cards. The netstat, ping and traceroute utilities can be used to check network performance.

M ***Effort Details***

Medium effort; Network problems are relatively easy to check but sometimes difficult to solve.

L ***Risk Details***

Low risk.

Solution Implementation

Consult your system documentation for utilities such as ping, netstat, and traceroute

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

High Wait Time due to Client Events Between FETCH Calls

The Oracle shadow process is spending a significant amount of time waiting for messages from clients between FETCH calls for the same cursor.

What to look for

10046 / TKProf:

- *Overall wait event summary for non-recursive and recursive statements shows significant amount of time for SQL*Net message from client waits compared to the total elapsed time in the database*
- *The client waits occur between many fetch calls for the same cursor (as seen in the cursor #).*
- *On average, there are less than 5 (and usually 1) row returned per execution*

Cause Identified: Lack of Array Operations Causing Excess Calls to the Database

The client is not using array operations to process multiple rows in the database. This means that many more calls are performed against the database. Each call incurs a wait while the database waits for the next call. The time accumulates over many calls and will impact performance.

Cause Justification
TKProf:

1. *SQL*Net message from client waits are a large part of the overall time (see the overall summary section)*
2. *There is nearly 1 row per execution on average (divide total rows by total execution calls for both recursive and non-recursive calls). When array operations are used, you'll see 5 to 10 rows per execution.*
3. *In some cases, most of the time is for a few SQL statements; you may need to examine the whole TKProf to find where the client waits were highest and examine those for the use of array operations*

Solution Identified: Use array operations to avoid calls

Array operations will operate on several rows at a time (either fetch, update, or insert). A single fetch or execute call will do the work of many more. Usually, the benefits of array operations diminish after an arraysize of 10 to 20, but this depends on what the application is doing and should be determined through benchmarking.

Since fewer calls are needed, there are savings in waiting for client messages, network traffic, and database work such as logical reads and block pins.

M *Effort Details*

Medium effort; Depending on the client, it may be easy or difficult to change the application and use array operations.

L *Risk Details*

Very low risk; it is risky when enormous array sizes are used in OLTP operations and many rows are expected. This is due to waiting for the entire array to be filled until the first row is returned.

Solution Implementation

The implementation of array operations will vary by the type of programming language being used. See

the documents below for some common ways to implement array operations.

Documentation

PL/SQL User's Guide and Reference : Reducing Loop Overhead for DML Statements and Queries with Bulk SQL

Programmer's Guide to the Oracle Precompilers : Using Host Arrays

JDBC Developer's Guide and Reference: Update Batching

JDBC Developer's Guide and Reference: Oracle Row Prefetching

Notes

Bulk Binding - What it is, Advantages, and How to use it

How To Fetch Data into a Table of Records using Bulk Collect and FOR All

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Reduce Oracle Memory Consumption

Oracle uses memory for the SGA and PGAs. Examine the size of the SGAs and PGAs to determine what is using the system's memory.

1. ***Observations and Causes***

Examine the table below for common observations and causes:

Note: This list shows some common observations and causes but is not a complete list. If you do not find a possible cause in this list, you can always open a service request with Oracle to investigate other possible causes. Please see the section below called, "Open a Service Request with Oracle Support Services".

Oracle Memory Consumption due to large SGA

One or more SGAs on the machine are leaving very little memory left for PGAs and other use on the machine.

What to look for

RDA:

- A large portion of the memory on the machine is used by one or more SGAs (see the total size of the buffer cache and shared pool), see the following:
  1. Overview > System Information > Total Physical Memory
  2. RDBMS > SGA Information, add up all components
  3. Repeat for all other instances on the machine
  4. Compare total size of all SGAs to physical memory

Cause Identified: Oversized buffer cache

The buffer cache is very large and is using more memory than is needed.

Cause Justification
AWR or Statspack report:

- Not using automatic shared memory management (ASMM), i.e., SGA_TARGET=0
- Buffer cache hit ratio is around 99%

Solution Identified: 10g+: Use automatic shared memory management (ASMM)

ASMM will seek to optimize the size of the buffer cache without human intervention.

L Effort Details

Low effort; change an initialization parameter

L Risk Details

Low risk. Be sure to set SGA_TARGET to a reasonable value.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Oversized shared pool

The pool is very large and is using more memory than is needed.

Cause Justification
AWR or Statspack report:

- *Not using automatic shared memory management (ASMM), i.e., SGA_TARGET=0*
- *shared pool free memory is more than 30%*

Solution Identified: 10g+: Use the Automatic Shared Memory Manager (ASMM) to adjust the shared pool size

ASMM will automate memory sizing for the shared pool to ensure an optimal amount is available. You will need to set a reasonable value for SGA_MAX_SIZE and SGA_TARGET to enable ASMM.

L *Effort Details*

Low effort; an init.ora / spfile change.

L *Risk Details*

Low risk; ASMM will ensure sufficient memory is available.

Solution Implementation

See the documents below.

Documentation

Concepts: Memory Architecture

Concepts: Automatic Shared Memory Management

Admin: Using Automatic Shared Memory Management

Performance Tuning: Configuring and Using the Shared Pool and Large Pool

Notes

Understanding and Tuning the Shared Pool

Oracle Database 10g Automated SGA Memory Tuning

How-To

How To Use Automatic Shared Memory Management (ASMM) In Oracle10g

Shared pool sizing in 10g

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: The Large, Java and/or Streams Pool are Oversized

A significant amount of free space is present in the large, java and/or streams pool when there is evidence of memory pressure on the instance and/or host.

Cause Justification
If the large, java or streams pool individual free space is greater than 20%, and there is evidence of memory pressure, then this cause is likely. Memory pressure is generally detected when the system is paging out memory.

Solution Identified: Reduce the size of the Large, Java or Streams pool

Reduce the large, java and streams pool so they typically have 5% of free space during peak memory usage.

L *Effort Details*

Initialization parameter change.

M *Risk Details*

If the values are set too low, then certain operations may fail; values should be adjusted cautiously, over time if possible.

Solution Implementation

See documents below:

Reference

LARGE_POOL_SIZE parameter

JAVA_POOL_SIZE parameter

STREAMS_POOL_SIZE parameter

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- *Review other possible reasons*
- *Verify that the data collection was done properly*
- *Verify the problem statement*

If you would like to log a service request, a test case would be helpful at this stage.

Oracle Memory Consumption due to large PGAs

One or more Oracle processes are using large amounts of PGA memory on the machine.

What to look for

A large portion of the memory on the machine is used by one or more PGAs , see the following:

1. *RDA: Overview > System Information > Total Physical Memory*
2. *Run this query to see total PGA memory used by the instance:*

```
select sn.name, sum(s.value)
from v$sesstat s, v$statname sn
where s.statistic# = sn.statistic#
and sn.name like '%pga%'
group by sn.name
```

3. *Repeat for all other instances on the machine*
4. *Determine which instance uses the most PGA memory and which sessions account for the memory usage.*

Cause Identified: Manually sized private workareas are too large

Private workareas are too large for the total number of Oracle processes and the amount of memory available.

Cause Justification
AWR or Statspack report:

- *Not using automatic PGA memory management i.e., PGA_AGGREGATE_TARGET=0 or WORKAREA_SIZE_POLICY = MANUAL*
- *Parameters like sort_area_size and hash_area_size are very large and when multiplied by the number of active sessions will use up most of the system's physical memory*

Solution Identified: Use automatic PGA memory management

When running under the automatic PGA memory management mode, sizing of work areas for all sessions becomes automatic and the *_AREA_SIZE parameters are ignored by all sessions running in that mode. At any given time, the total amount of PGA memory available to active work areas in the instance is automatically derived from the PGA_AGGREGATE_TARGET initialization parameter.

Under automatic PGA memory management mode, the main goal of Oracle is to honor the PGA_AGGREGATE_TARGET limit set by the DBA, by controlling dynamically the amount of PGA memory allotted to SQL work areas. At the same time, Oracle tries to maximize the performance of all the memory-intensive SQL operations, by maximizing the number of work areas that are using an optimal amount of PGA memory (cache memory).

L ***Effort Details***

Low effort; initialization parameter change. Some effort has to be made initially to set the proper target size and adjust it to ensure optimal performance.

L ***Risk Details***

Low risk. Initially, some effort should be made to ensure the PGA settings are reasonable and don't regress performance.

Solution Implementation

See the documents below.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Cause Identified: Automatically sized private workareas are too large

Private workareas are too large for the total number of Oracle processes and the amount of memory available.

Cause Justification
AWR or Statspack report:

- ***Using automatic PGA memory management i.e., PGA_AGGREGATE_TARGET=some large value and WORKAREA_SIZE_POLICY = AUTO***

Solution Identified: Reduce the amount of PGA_AGGREGATE_TARGET memory

PGA_AGGREGATE_TARGET may be set too large for the memory capacity of the machine. If memory is constrained, then a balance may be found where some queries run slower but the overall system runs faster since memory is available for critical operations.

L ***Effort Details***

Low effort; parameter change.

H ***Risk Details***

High risk. Some execution plans may change and some queries may perform worse.

Solution Implementation

See the documents below for guidance on the proper use of the automatic PGA memory feature.

Documentation

Concepts: Overview of the Program Global Areas

Performance Tuning Guide: PGA Memory Management

Reference: Initialization Parameter PGA_AGGREGATE_TARGET

Reference: Initialization Parameter WORKAREA_SIZE_POLICY

Automatic PGA Memory Management in 9i and 10g

Init.ora Parameter "PGA_AGGREGATE_TARGET" Reference Note

Init.ora Parameter "WORKAREA_SIZE_POLICY" Reference Note

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Solution Identified: Increase the amount of physical memory on the machine

Adding memory instead of reducing the size of the PGA target will give memory to the processes and reduce the possibility that executions plans will change.

M ***Effort Details***

Medium effort; simple hardware change but downtime involved if non-RAC.

L ***Risk Details***

Low risk. Low chance of execution plans changing.

Solution Implementation

Not applicable.

Implementation Verification

Implement the solution and determine if the performance improves. If performance does not improve, examine the following:

- ***Review other possible reasons***
- ***Verify that the data collection was done properly***
- ***Verify the problem statement***

If you would like to log a service request, a test case would be helpful at this stage.

Tools

The tools listed below are useful for diagnosing slow database problems.

Active Session History (ASH) Report

Active Session History Reports

Automatic Workload Repository (AWR) Report

Automatic Workload Repository Reports

10046 and TKProf

Understanding SQL Trace and TKProf

Recommended Method for Obtaining 10046 trace for Tuning

LTOM

LTOM - The On-Board Monitor User's Guide

OS Watcher

OS Watcher User Guide

Statspack Report

Statspack Complete Reference