Analyzing Application Performance in RAC

Arup Nanda
Starwood Hotels
Why Analyze

• “The Database is Slow”!
  – Storage, CPU, memory, runqueues all affect the performance
  – Know what specifically is causing them to be slow
• To build a profile of the application
• To check scalability
  – You have developed against non-RAC
    • Will it scale up in RAC?
  – Currently it runs with 100 users
    • What will happen if we have 1000?
• Effective Tuning
  – take a baseline before some tuning exercise
  – re-measure to see if the tuning was effective
  – check the resource usage of applications
What to Measure

• Timing of Events
  – An Oracle session is in any of these three states
    • Doing something useful (consuming CPU) $U$
    • Waiting for some resource (a block from disk, a latch) $W$
    • Idle (Waiting for some work from the user) $I$
  – Total Time = $U+W+I$
  – Accurately measure each component

• Resource Usage
  – Latches, Locks
  – Redo, Undo
  – Logical I/O
Inter-instance Round Trip Times

<table>
<thead>
<tr>
<th>Requesting Instance</th>
<th>Network</th>
<th>Receiving Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>process for block</td>
<td>xtf + n/w latency</td>
<td>processing msg</td>
</tr>
<tr>
<td>wait for block</td>
<td>xtf + n/w latency</td>
<td>process block</td>
</tr>
<tr>
<td>receive block</td>
<td>send</td>
<td></td>
</tr>
</tbody>
</table>

process for block
xtfr + n/w latency
processing msg
process block
send
How to Get the Times

• You can get these times by examining the session in real time

\[
\text{select state, seconds_in_wait, wait_time, event}
\]

\[
\text{from v$session}
\]

\[
\text{where sid = <sessionid>}
\]

• There are several issues
  – You should be watching in real time
  – You will miss the times when these events are past
  – How will you know the sessionID in advance?

• Other Option – Tracing

• There is an event called 10046 which allows you to enable tracing in sessions
Enabling Trace

• SQL Trace can be enabled by
  
  \texttt{alter session set sql_trace = true;}

• You can set the event at the session level so that it can capture the wait events.
  
  \texttt{alter session set events '10046 trace name context forever, level 12'}

• It produces a trace file similar to sql_trace, but with extended trace data
  – With information on how much time was spent where

• It creates a trace file in the user_dump_dir
  – In 11g, the udump dir is inside the diag structure
Different Session

- To set SQL Trace in a different session:
  
  \[\text{dbms\_system\.set\_sql\_trace\_in\_session}(\langle sid\rangle, \langle serial\#\rangle, \text{true})\;\]

- To set 10046 Trace in a different session:
  
  \[\text{dbms\_system\.set\_ev}(\langle sid\rangle, \langle ser\#\rangle, 10046, \langle level\#\rangle, \text{null})\;\]

  - The same effect as
    
    \[\text{after\_session\_set\_events\('10046\_trace\_name\_context\_forever,\_level\langle level\#\rangle'\)}\]
• From 10g onwards, you can enable it any other session by:

```sql
begin
    dbms_monitor.session_trace_enable (
        session_id       => 131,
        serial_num      => 5879,
        waits            => true,
        binds            => true
    );
end;
/
```

To capture wait events

To capture bind variables
Analyzing

• Tracefiles are not quite readable
• To analyze the tracefile (SQL Trace or the 10046 Trace)
  – A tool called **tkprof**
    
    ```
    # tkprof D11_D_ora_9204.trc
    D11_D_ora_9204.out
    explain=arup/arup waits=yes sys=no
    ```

• Other Analyzers
  – Trace Analyzer (downloadable from MetaLink)
  – Third party analyzers
    • Hotsos Profiler
    • Trivadis TVD$XSTAT analyzer
Trace Analyzer

• A much better tool to analyze trace files.
• Refer to MetaLink Doc 224270.1 for download and instructions on use
• A small zip file, with bunch of directories
• Connect as SYS and run tacreate.sql to create the Trace Analyzer schema (TRCANLZR)
• Run it

    cd trca/run

    sqlplus trcanlzar/trcanlzar

    @trcanlzar <tracefile name in udump dir>
value passed to transzr.sql:

---

RACE_FILENAME  D11D_ora_205.trc

analyzing D11D_ora_205.trc

race Analyzer completed.

eview first transzr_error.log file for possible fatal errors.

eview next transzr_22881.log for parsing messages and totals.

. . . copying new generated files into local directory

KPRF: Release 11.1.0.7.0 - Production on Wed Oct 28 11:45:05 2009

copyright (c) 1982, 2007, Oracle. All rights reserved.

adding: transzr_22881_c.html (deflated 90%)

adding: transzr_22881_c.log (deflated 82%)

adding: transzr_22881_c.txt (deflated 84%)

adding: transzr_22881_tkprof (deflated 85%)

These files are produced in the local directory
Trace Analyzer

- It generates
  - The log file of the run. Scan for errors.
  - The tkprof output of the trace file
  - The analysis in text format
  - The analysis in html format
The Connection Pool Effect

• Most applications use connection pool
• A “pool” of connections connected to the database
• When the demand on the connection from the pool grows, the pool creates new database sessions
• When the demand lessens, the sessions are disconnected
• The SID is not known
Enabling Tracing in Future Sessions

- Service Names start tracing when any session connected with that service name will be traced.

```sql
BEGIN

dbms_monitor.serv_mod_act_trace_enable(
    service_name => 'APP',
    action_name => dbms_monitor.all_actions,
    waits => true,
    binds => true
);

END;
```

Warning: This is case sensitive; so “app” and “APP” are different.
What’s Special About RAC

- Multiple Instances $\rightarrow$ multiple hosts
- The tracefiles are on different hosts
- Application connect through a connection pool
Multiple Tracefiles

• Tracefiles are generated for each Oracle session
• So, a single user’s action can potentially go to many sessions → many tracefiles
• Workaround: create only one session in the connection pool
Mixed Activities

- But that does not solve the problem
- The single Oracle session will service activities of many users
- So the tracefile will have activities of all users; not just the user you are interested in.
Consolidation of Tracefiles

• The trcsess utility comes handy in that case
  – It combines all tracefiles into one!

  trcsess output=alltraces.trc service=app
  * . trc

  – It creates the tracefile alltraces.trc from all the tracefiles in that directory where activities by all sessions connected with the **app** service

• Now you can treat this new tracefile as a regular tracefile.

  $ tkprof alltraces.trc alltraces.out sys=no ...
TRCSESS

- The utility has many options

\[ \text{trc} \text{sess } [\text{output}=<\text{output file name}> ] \\
[\text{session}<\text{session ID}> ] \\
[\text{client id}<\text{client id}> ] [\text{service}<\text{service name}> ] [\text{action}<\text{action name}> ] \\
[\text{module}<\text{module name}> ] <\text{trace file names}> \]

\text{output}=<\text{output file name}> \text{output destination default being standard output.} \\
\text{session}<\text{session ID}> \text{session to be traced.} \\
\text{Session id is a combination of SID and Serial # e.g. 8.13.} \\
\text{client id}<\text{client id}> \text{client id to be traced.} \\
\text{service}<\text{service name}> \text{service to be traced.} \\
\text{action}<\text{action name}> \text{action to be traced.} \\
\text{module}<\text{module name}> \text{module to be traced.}
Other Profiles

• So far we talked about timings of various activities
• Applications consume resources
  – Buffers (consistent gets)
    • Which in turn drives the I/O up
  – Latches (cache buffer chains, library cache, etc.)
  – Locks
  – CPU
  – Redo Generation
• All these resources affect the scalability of the applications
  – Especially in RAC
• You need to measure these resource stats as well
Many develop apps against a database running on their laptops; deem the performance acceptable and **assume** that the performance will be similar in a multi-user system!
Source of Resource Stats

• The best source is V$SESSTAT

```sql
SELECT name, value
FROM v$sesstats s, v$statname n
WHERE n.statistic# = s.statistic#
AND
    n.name IN ('CPU used by this session', 'redo size')
AND sid = 149;
```

• Take measurement before and after the application run
• Measure the difference; it’s the resource utilized
Runstats Package

• Tom Kyte has an excellent package that can automate this for you.

• This allows you to build a test harness

  1. SQL: `EXEC RUNSTATS_PKG.RS_START;`
  2. Run the application
  3. SQL: `EXEC RUNSTATS_PKG.RS_MIDDLE;`
  4. Run the application (changed)
  5. SQL: `EXEC RUNSTATS_PKG.RS_STOP;`

• It shows the difference between the two runs for latches and statistics
Output

- Shows the resources have been consumed – latches and other stats.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATCH enqueue hash chains</td>
<td>1,579</td>
</tr>
<tr>
<td>LATCH row cache objects</td>
<td>1,678</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>1,935</td>
</tr>
<tr>
<td>LATCH cache buffers chains</td>
<td>3,688</td>
</tr>
<tr>
<td>undo change vector size</td>
<td>4,420</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>4,560</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>6,900</td>
</tr>
<tr>
<td>table scan rows gotten</td>
<td>8,002</td>
</tr>
<tr>
<td>redo size</td>
<td>70,944</td>
</tr>
<tr>
<td>session uga memory max</td>
<td>131,036</td>
</tr>
<tr>
<td>session pga memory max</td>
<td>131,072</td>
</tr>
</tbody>
</table>
What about Future Sessions

- Another procedure in DBMS_MONITOR

```sql
BEGIN

  dbms_monitor.client_id_stat_enable('CLIENT1');

END;
```

- It enables statistics collection for all client calls with client identifier CLIENT1

- You set the client identifier by

```sql
BEGIN

  dbms_session.set_identifier('CLIENT1');

END;
```
Recording of Stats

- The stats are exposed through V$CLIENT_STATS

```
SQL> desc v$client_stats
Name                         Null?  Type
------------------------------------------
CLIENT_IDENTIFIER            VARCHAR2(64)  
STAT_ID                      NUMBER      
STAT_NAME                    VARCHAR2(64)  
VALUE                        NUMBER      
```

- The stats are aggregated, i.e. all the stats are for a specific client_identifier; not individual sessions
- A subset of the stats; not all
V$CLIENT_STATS

SQL: `SELECT stat_name, value FROM v$client_stats WHERE client_identifier = 'CLIENT1';`

<table>
<thead>
<tr>
<th>STAT NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>user calls</td>
<td>4</td>
</tr>
<tr>
<td>DB time</td>
<td>2614</td>
</tr>
<tr>
<td>DB CPU</td>
<td>4000</td>
</tr>
<tr>
<td>parse count (total)</td>
<td>5</td>
</tr>
</tbody>
</table>

Only 27 stats were captured; not all.
Other Stats Collection

• On Service Name and/or Module Name and Actions
• Here we want to capture sessions starting with

```sql
begin
    dbms_monitor.serv_mod_act_stat_enable(
        service_name => 'APP',
        module_name  => 'SQL*Plus',
        action_name  => 'UPDATE'
    );
end;
```

Default is all actions
Checking Stats Collection

- To find out which type of aggregation is enabled

```
SQL> desc DBA_ENABLED_AGGREGATIONS
Name                      Null?     Type
-------------------- --------------- -------------------
AGGREGATION_TYPE           VARCHAR2(21)
PRIMARY_ID                 VARCHAR2(64)
QUALIFIER_ID1              VARCHAR2(48)
QUALIFIER_ID2              VARCHAR2(32)
```

This table provides information about enabled aggregations and their associated types.
Other Sessions

• How do you start measuring when the session is not yet connected?
  – When the stats on individual sessions is desirable
  – When the client id, service, etc. are not alterable

• BYOT - Build your own tool
  – Create a post-login trigger to write the stats at the beginning of a session to a table
  – Write the values at the end of the session using a pre-logoff trigger
  – Measure the resource usage (the difference)

• Download the scripts to build the complete tool from my blog.
  – http://arup.blogspot.com/2010/09/other-day-i-was-putting-together-my.html
Inference from Resource Usage

• Watch out for stats that increase with load
  – Redo size
    • More the redo, more time for I/O and redo latches
  – Session Logical Reads
    • More I/O, indicates more buffers
    • More inter-instance locking, messaging
    • DW environment: more buffer flush
  – Cache Buffer Chain Latch
    • More latching $\rightarrow$ more CPU usage
  – If these stats and latches are high, the application will scale negatively
  – If you test in a small environment, you must measure it to test its scalability on a much bigger system.
Putting it all Together

• Profile Components
  – 10046 Tracing
  – Combining Traces to a Single File
  – Getting the time spent at different components
  – Gather Resource Usage

• Strategy
  – Capture all the profile components
  – Make changes to your app
  – Capture all the profile components

• Decision
  – Better, worse?
  – How much?
  – Decide on the next course of action – the scientific way.
Thank You!

Questions?

Email: arup@proligence.com
Blog: arup@blogspot.com