Latch and Mutex Contention Troubleshooting in Oracle

Tanel Põder
http://blog.tanelpoder.com
If you like this session...

...then you're probably an Oracle geek like me ;)

And you will like my tech-blog:
  * http://blog.tanelpoder.com

All my scripts freely downloadable
  * http://blog.tanelpoder.com/seminar/seminar-files/
  * Look for TPT scripts

Oh... and I do seminars with much more stuff like this ;-)  
  * And I also fix these kinds of problems as a consultant
  * http://blog.tanelpoder.com/seminar/
  * tanel@tanelpoder.com
What is a latch?

- Oracle's low-level mechanism for serializing concurrent access to very shortly accessed memory structures such as cache buffer headers etc...

Yeah, but what is a latch?

- Latch is just a simple memory structure
- Usually around 100-200B in size (dependent on version, platform)
- Is wrapped into a latch state object structure (since v8.0 I think)
- Can reside in fixed SGA (parent latches) or shared pool
- Is set using hardware-atomic compare-and-swap (CAS) instructions
  - LOCK CMPXCHG on intel
- Latches are taken and released without any OS interaction
  - Actually the OS doesn't even have a clue that Oracle latches exist!
- Can be shared (since Oracle 8.0)
  - Used for some AQ ops
  - For example, used for cache buffers chains latch gets if examining a buffer chain
Latch is a lock period.
Latch contention

What is latch contention?

I want to get a latch...
but someone is already holding it!

To troubleshoot, find out:

1) Who and why tries to get the latch (waiter)
2) Who and why are holding the latch (blocker)

In other words - find who and why is blocking us! Just like with regular enqueue locks...

Remember, latches are locks!
Latch contention troubleshooting

**V$SESSION_WAIT / V$ACTIVE_SESSION_HISTORY**
- PARAMETER1 = latch address for any latch wait event
- sw, WaitProf to see latch addresses of latch wait events
- Addresses map to v$latch_parent/v$latch_children.ADDR column

**SQL Trace**
- you need to convert the P1/address number to hex:

```
WAIT #3: nam='latch free' ela= 3749 address=15709454624 number=202 tries=1
    obj#=-1 tim=5650189627

SQL> select to_char(15709454624, 'XXXXXXXXXXXXXXXX') addr_hex from dual;
ADDR_HEX -----------------
    3A85B4120

SQL> @la 3A85B4120
ADDR LATCH#  CHLD NAME                  GETS
---------------- ---------- ------ --------------- ----------
000000003A85B4120 202      2 kks stats           494562
```
Identifying latch holders

**V$LATCHHOLDER**

- Shows top-level latch holders from process state objects
  - See the *Session ID* of latch holder!!
- Any latch held by a process state object is shown there
- Works in 99.9% of cases - enough for practical troubleshooting
- Additional latches held (while the top-level one is held) are not reported by v$latchholder :-(

- In the 0.01% of cases a fallback to old latch contention troubleshooting method could be used
  - Alternatively its possible to traverse state object trees via direct SGA attach

```sql
SQL> desc v$latchholder
Name                        Null?    Type ------------------------------- -------- ------------
---------------------------- -------- ------------
1      PID                                      NUMBER
2      SID                                      NUMBER
3      LADDR                                    RAW(8)
4      NAME                                    VARCHAR2 (64)
5      GETS                                     NUMBER
```
Latch contention troubleshooting approach

1. Identify the session(s) experiencing problems
   - Remember, databases don't have problems, only users, through database sessions

2. Quantify for which latch that session is waiting for the most
   - ...and whether the wait time is significant enough

3. Identify the child latch involved in contention
   - Is the contention concentrated on a particular child latch or is it spread across many?

4. Identify where in kernel code (why) the latch is held from
   - V$LATCHHOLDER
   - LatchProfX / X$KSUPRLAT / X$KSUPR if problem with some sessions
Latch contention troubleshooting with LatchProfX

Sample V$LATCHHOLDER stats with LatchProfX, fast
- 10-100k samples per second!
- @latchprofx <columns> <sid> <latches> <#samples>
- @latchprofx sid,name,func,hmode % % 1000000

```sql
SQL> @latchprofx sid,name,func,hmode &sid % 100000

-- LatchProfX 1.07 by Tanel Poder ( http://www.tanelpoder.com )

<table>
<thead>
<tr>
<th>SID</th>
<th>NAME</th>
<th>FUNC</th>
<th>HMODE</th>
<th>Held</th>
</tr>
</thead>
<tbody>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kghalo</td>
<td>exclusive</td>
<td>3174</td>
</tr>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kghalp</td>
<td>exclusive</td>
<td>1294</td>
</tr>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kghuprl</td>
<td>exclusive</td>
<td>704</td>
</tr>
<tr>
<td>139</td>
<td>shared pool simulator</td>
<td>kglsim_unpin_simhp</td>
<td>exclusive</td>
<td>581</td>
</tr>
<tr>
<td>139</td>
<td>kks stats</td>
<td>kksAllocChildStat</td>
<td>exclusive</td>
<td>489</td>
</tr>
<tr>
<td>139</td>
<td>shared pool simulator</td>
<td>kglsim_upd_newhp</td>
<td>exclusive</td>
<td>240</td>
</tr>
<tr>
<td>139</td>
<td>row cache objects</td>
<td>kqrpre: find obj</td>
<td>exclusive</td>
<td>158</td>
</tr>
<tr>
<td>139</td>
<td>enqueues</td>
<td>ksqdel</td>
<td>exclusive</td>
<td>116</td>
</tr>
<tr>
<td>139</td>
<td>enqueues</td>
<td>ksqgel: create enqueue</td>
<td>exclusive</td>
<td>91</td>
</tr>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kgh_heap_sizes</td>
<td>exclusive</td>
<td>57</td>
</tr>
<tr>
<td>139</td>
<td>row cache objects</td>
<td>kqrreqd: reget</td>
<td>exclusive</td>
<td>58</td>
</tr>
<tr>
<td>139</td>
<td>enqueue hash chains</td>
<td>ksggtl3</td>
<td>exclusive</td>
<td>57</td>
</tr>
<tr>
<td>139</td>
<td>shared pool simulator</td>
<td>kglsim_scan_lru: scan</td>
<td>exclusive</td>
<td>53</td>
</tr>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kghfre</td>
<td>exclusive</td>
<td>49</td>
</tr>
<tr>
<td>139</td>
<td>row cache objects</td>
<td>kqrreqd</td>
<td>exclusive</td>
<td>41</td>
</tr>
<tr>
<td>139</td>
<td>enqueue hash chains</td>
<td>ksqrcrl</td>
<td>exclusive</td>
<td>36</td>
</tr>
<tr>
<td>139</td>
<td>shared pool simulator</td>
<td>kglsim_chg_simhp_free</td>
<td>exclusive</td>
<td>22</td>
</tr>
<tr>
<td>139</td>
<td>shared pool</td>
<td>kghasp</td>
<td>exclusive</td>
<td>18</td>
</tr>
<tr>
<td>139</td>
<td>MinActiveScn Latch</td>
<td>ktucloGetGlobalMinScn</td>
<td>shared</td>
<td>14</td>
</tr>
</tbody>
</table>
```
KGX mutexes

KGX = Kernel Generic mutexX module
- Kernel functions managing mutexes start with kgx

Introduced in Oracle 10.2
- Physically like a latch (a piece of memory)
  - only more lightweight
  - and smaller
- Can be embedded inside other structures (lib cache object handle)
- Can have flexible spin/yield/wait strategy defined by "client"
- Do not account SPINGETS,YIELDS, only WAITS
  - GETS are accounted internally, but not externalized in any view

KGX mutexes are \textit{not} OS mutexes!!!
Mutexes for Library Cache

Used for protecting V$SQLSTATS buckets
- oradebug dump cursor_stats 1

Used for pinning library cache cursors and parent examination
- If _kks_use_mutex_pin=true (default from 10.2.0.2)
- oradebug dump librarycache level 10

In 11g+ mutexes are used instead of most library cache latches
- Instead of up to 67 library cache latches there's 131072 mutexes!
- Each library cache mutex protects one library cache hash bucket

Known mutex types in 11g:
- Cursor Parent
- Cursor Pin
- Cursor Stat
- Library Cache
- hash table
- ...

Tanel Põder  http://www.tanelpoder.com
Mutex troubleshooting

V$SESSION_WAIT
- Shows wait events such:
  - cursor: mutex S
  - cursor: mutex X
  - library cache: mutex S
  - library cache: mutex X

V$MUTEX_SLEEP
- Shows the wait time, and the number of sleeps for each combination of mutex type and location. Somewhat useless.

V$MUTEX_SLEEP_HISTORY
- Shows last individual occurrences of mutex sleeps
- Based on a circular buffer, has most detail
- @mutexprof script

Systemstate dumps
Mutex waits and their meaning - 1

**cursor: mutex S**
- We try to get a mutex on Parent cursor or V$SQLSTAT bucket in shared mode.
- The mutex is "in flux" (someone is in progress of taking it in shared mode) so we have to wait until the holder finishes its shared get.
- Used when:
  - Examining parent cursor, Querying V$SQLSTATS bucket

**cursor: mutex X**
- We try to get a mutex on Parent cursor or V$SQLSTAT bucket in exclusive mode.
- Someone is already holding the mutex in incompatible mode
- ...Either there's someone already holding the mutex in X mode
- ...Or there may be multiple holders in S mode
- Used when:
  - Loading new child cursor under parent, Modifying V$SQLSTATS bucket, Updating bind capture data
Mutex waits and their meaning - 2

cursor: pin S
- We try to pin the cursor in shared mode (for execution for example)
- Mutex for child cursor pinning is "in flux", someone is in process of pinning that same cursor already.
- We have to wait until the other session completes their pin request

cursor: pin X
- We try to pin a cursor in exclusive mode, but someone already has pinned it in a non-compatible mode
- Either one session has pinned it in X mode or multiple sessions in S mode

cursor: pin S wait on X
- We try to pin a cursor in shared mode, but someone already has pinned it in X mode
- Other session is currently loading that child cursor (parsing)
Mutex waits and their meaning - 3

In 11g, most library cache latches have been replaced by mutexes directly on library cache hash buckets

- 131072 KGL hash buckets
- Each is protected by a separate mutex
- Less room for false contention

**library cache: mutex S**

- Trying to get a mutex on library cache hash bucket in S mode
- The mutex is already held in incompatible mode or is "in flux"

**library cache: mutex X**

- Trying to get a mutex on library cache hash bucket in X mode
- The mutex is already held in incompatible mode or is "in flux"
## Mutex wait event parameters - 1

### SQL> @sed mutex  <-- this script queries v$event_name

<table>
<thead>
<tr>
<th>EVENT_NAME</th>
<th>PARAMETER1</th>
<th>PARAMETER2</th>
<th>PARAMETER3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor: mutex S</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>cursor: mutex X</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>cursor: pin S</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>cursor: pin S wait on X</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>cursor: pin X</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>library cache: mutex S</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
<tr>
<td>library cache: mutex X</td>
<td>idn</td>
<td>value</td>
<td>where</td>
</tr>
</tbody>
</table>

As with any wait event, the parameters (P1,P2,P3) provide additional detail, context info about the wait:

- Parameter1 (idn) can be used for finding the cursor related to mutex
- Parameter 2 (value) can be used for finding the session holding the mutex in exclusive mode
  - For all shared mode holders a systemstate dump would be needed
Mutex wait event parameters - 2

PARAMETER1 - \textit{idn}:

- cursor:* wait events
  - idn = hash value of the library cache object under protection

- library cache: mutex* wait events
  1) library cache hash bucket number (if idn <= 131072)
  2) idn = hash value of the library cache object under protection (if idn > 131072)

Find SQL by hash\_value:
\[
\text{SELECT sql\_text FROM v\$sql WHERE hash\_value = \&idn;}
\]

Find SQL by library cache hash bucket (idn <= 131072):
\[
\text{SELECT sql\_text FROM v\$sql WHERE MOD(hash\_value, 131072) = \&idn;}
\]

Find SQL from AWR by SQL\_ID (hash\_value is the lower half of SQLID):
\[
\text{SELECT sql\_text FROM dba\_hist\_sqlstat WHERE tpt.sqlid\_to\_sqlhash( sql\_id ) = \&idn;}
\]

You'll find the TPT package in setup directory in TPT\_public.zip
Mutex wait event parameters - 3

PARAMETER 2 - **value**:  
- low bytes of word (2 or 4 bytes) - number of mutex shared references  
- high bytes of word (2 or 4 bytes) - SID of exclusive holder

SQL> select session_id, event, blocking_session,  
2         to_char(p2, '0XXXXXXX') value_hex  
3  from v$active_session_history  
4  where event like 'library cache: mutex%';

<table>
<thead>
<tr>
<th>SESSION_ID</th>
<th>EVENT</th>
<th>BLOCKING_SESSION</th>
<th>VALUE_HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>157</td>
<td>library cache: mutex X</td>
<td></td>
<td>00830000</td>
</tr>
</tbody>
</table>

PARAMETER 3 - **where**:  
- where = maps to x$mutex_sleep.location_id  
- Useful for understanding from which kernel function the mutex get operation was done. Used for advanced diagnostics.
Questions?
Thank you !!!

Further questions are welcome:
- tanel@tanelpoder.com

Slides, scripts and my blog:
- http://blog.tanelpoder.com

If you liked this session, you will also like my **Advanced Oracle Troubleshooting** seminar!
- I do both public and private corporate seminars
- http://blog.tanelpoder.com/seminar/