

# Latch, Lock and Mutex Contention Troubleshooting in Oracle

**Tanel Põder**

**<http://www.tanelpoder.com>**



# Intro to latching

## 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
- 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 contention

What is latch contention?

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

- If get was in no-wait mode, return to caller with failure
- If get was in willing-to-wait mode, continue trying:

*So, I will try to get it again immediately! And again! And again!*

- This is spinning (busywaiting)

*Still can't get it so I go to sleep for a very long time... ..*

- 10ms is very long time in latching world
- `_max_exponential_sleep`
- `_max_sleep_holding_latch`

*I might be woken up by the process who releases that latch...*

- Not used for all latches (as it requires latch waiter list scanning)
- This used to be controlled using `_latch_wait_posting` parameter

# Latch contention troubleshooting

## V\$SESSION\_EVENT

- Snapper

## V\$SESSION\_WAIT

- SW
- WaitProf

## V\$LATCH / V\$LATCH\_MISSES (systemwide)

- Snapper
- Statspack
- awrrpt
- ashrpt

## V\$LATCHHOLDER

- LatchProf
- LatchProfX (based on X\$KSUPRLAT and X\$KSLLW)

# 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 the waiting latch gets were done
  - Statspack, AWR or V\$LATCH\_MISSES if problem systemwide
  - 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,ksllwnam,ksllwbl,hmode % % 1000000

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

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

SID	NAME	KSLLWNAM	HMODE	Held
139	shared pool	kghalo	exclusive	3174
139	shared pool	kghalp	exclusive	1294
139	shared pool	kghupr1	exclusive	704
139	shared pool simulator	kglsim_unpin_simhp	exclusive	581
139	kks stats	kksAllocChildStat	exclusive	489
139	shared pool simulator	kglsim_upd_newhp	exclusive	240
139	row cache objects	kqrpre: find obj	exclusive	158
139	enqueuees	ksqdel	exclusive	116
139	enqueuees	ksqgel: create enqueue	exclusive	91
139	shared pool	kgh_heap_sizes	exclusive	90
139	row cache objects	kqreqd: reget	exclusive	58
139	enqueue hash chains	ksqgtl3	exclusive	57
139	shared pool simulator	kglsim_scan_lru: scan	exclusive	53
139	shared pool	kghfre	exclusive	49
139	row cache objects	kqreqd	exclusive	41
139	enqueue hash chains	ksqrcl	exclusive	36
139	shared pool simulator	kglsim_chg_simhp_free	exclusive	22
139	shared pool	kghasp	exclusive	18
139	MinActiveScn Latch	ktucloGetGlobalMinScn	shared	14

# KGX mutexes

KGX = Kernel Generic muteX

Introduced in Oracle 10.2

- Physically like a latch (a piece of memory)
  - only more lightweight
  - and smaller
- Can be embedded inside other structures (KGL HD)
- Can have flexible spin/yield/wait strategy defined by client
- Do not account GETS,SPINGETS,YIELDS, only WAITS

KGX mutexes are *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 library cache latches

- Instead of up to 67 library cache latches there's 131072 mutexes!

Known mutex types in 11g:

- Cursor Parent
- Cursor Pin
- Cursor Stat
- Library Cache
- hash table
- ...

# Mutex troubleshooting

## V\$SESSION\_WAIT

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

The mutex sleeps are well instrumented in wait interface. P1,P2,P3 values show what is the hash value of library cache objects under contention, the session holding the mutex etc. v\$event\_name and v\$session\_wait "text" columns document the meaning of P1,P2,P3

## V\$MUTEX\_SLEEP

- Shows the wait time, and the number of sleeps for each combination of mutex type and location

## V\$MUTEX\_SLEEP\_HISTORY

- Shows last individual occurrences of mutex sleeps
- Based on a circular buffer, has most detail

## Systemstate dumps

- <http://el-caro.blogspot.com/2007/10/identifying-mutex-holder.html>

# 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
- <http://blog.tanelpoder.com/2008/08/03/library-cache-latches-gone-in-oracle-11g/>

### **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

```
SQL> @sed mutex
```

<b>EVENT_NAME</b>	<b>PARAMETER1</b>	<b>PARAMETER2</b>	<b>PARAMETER3</b>
cursor: mutex S	idn	value	where sleeps
cursor: mutex X	idn	value	where sleeps
cursor: pin S	idn	value	where sleeps
cursor: pin S wait on X	idn	value	where sleeps
cursor: pin X	idn	value	where sleeps
library cache: mutex S	idn	value	where
library cache: mutex X	idn	value	where

idn = library cache object hash value (cursor:\* events)

idn = library cache hash bucket number (library cache: events)

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

where = maps to x\$mutex\_sleep.location\_id

## \_spin\_count - good or bad?

### \_spin\_count

- Spin count for latches - how many cycles of *busy waiting* before yield or sleep
- **Often abused!**
- Should be increased only if all other problems are fixed *and* there's sufficient CPU

### \_latch\_classes and \_latch\_class\_x parameters

- Allow setting different spin counts for different latches
- `select indx,spin,yield,waittime from x$ksllclass;`

### \_kgx\_spin\_count

- spin count for KGX mutexes (11g+)

*Fix the root cause and you don't need to tune spin count!!!*

# Intro to enqueues - why do we need locks?

Imagine a hypothetical single user database engine

- We would not need any locks at all - at least for concurrency purposes

Let say we want to "scale" this database to more users...

- What would be the *simplest* way for implementing concurrency?
- A "single big lock" protecting access to *any* shared resource
- This would not scale well at all...

It makes sense to break the lock down into multiple pieces:

1) ...each piece protecting a different class of resources, for example:

- a TM table lock
- a TX locks protecting rows changed by transactions
- a MR lock protecting changes to datafile status and layout
- ...

2) Further breakdown is needed by individual objects to avoid unnecessary contention

- And this is where enqueue resource identifiers come into play

# Oracle enqueue locking mechanism

## Resources

- Slots in a hash table identifying some resource
- Resources are not locks, they are just placeholders referencing the object which can be locked
- The unique identifier (primary key) to a resource consists of:
  - Resource type (TX, TM, MR, etc)
  - Resource identifier 1 (ID1)
  - Resource identifier 2 (ID2)
  - For example:
    - **TM-00015802-00000000**

The resource identifiers are needed for breaking down a resource class into individual objects.

For example if two tables both have a TM resource allocated for them in memory, then the ID1 specifies the object\_id of the table (v\$lock\_type shows the meaning of ID1/2 for each resource type)

## Locks

- Locks are what tie the lock holder to the resource locked
- If there are many lock holders on a single resource (in compatible mode) then there will still be one resource for that object, but multiple lock structures pointing to that resource

# Terminology and views

## Why are locks called enqueues?

- It's because the locking infrastructure implements an ordered queueing mechanism for lock waiters
- Latches and mutexes don't have sophisticated queueing mechanisms

## Enqueue Resources

- V\$RESOURCE (X\$KSQRS)

## Enqueue Locks

- V\$LOCK (X\$KSQEQ)

## Enqueue resource types:

- V\$LOCK\_TYPE - has the meanings of ID1 and ID2 documented!
- @!t TX

# Troubleshooting enqueue activity

Troubleshooting is simple due good instrumentation

- V\$SESSION\_WAIT , @sw <SID>
- V\$LOCK

## Enqueue tracing events:

```
10704, 00000, "Print out information about what enqueues are being obtained"
// *Cause: When enabled, prints out arguments to calls to ksqcmi and
//          ksqlrl and the return values.
// *Action: Level indicates details:
//          Level: 1-4: print out basic info for ksqlrl, ksqcmi
//                  5-9: also print out stuff in callbacks: ksqlac, ksqlop
//                  10+: also print out time for each line
10706, 00000, "Print out information about global enqueue manipulation"
// *Cause: When enabled, prints out activity in ksi routines.
// *Action: INTERNAL system parameter (_ksi_trace) can be used in addition
//          to setting this event to limit output information on selected
//          set of global enqueue server enqueue type(s)
//          For example:
//              _ksi_trace = RTXA
//          output information only for RT and XA lock types.
//
// *Action: Level indicates details:
//          0-4: show args for each main call
//          5-9: also indicate callbacks
//          10+: also printout time for each line
```

# Reading enqueue deadlock trace

DEADLOCK DETECTED

[Transaction Deadlock]

## Current SQL statement for this session:

*update t set object\_id = 2 where owner = 'SH'*

The following deadlock is not an ORACLE error. It is a deadlock due to user error in the design of an application or from issuing incorrect ad-hoc SQL. The following information may aid in determining the deadlock:

Deadlock graph:

Resource Name	-----Blocker(s)-----			-----Waiter(s)-----		
	process	session	holds waits	process	session	holds waits
TX-00070003-00000a4a	19	<b>150</b>	X	16	<b>147</b>	X
TX-0005002e-00000d63	16	<b>147</b>	X	19	<b>150</b>	X

session 150: DID 0001-0013-00000032                      session 147: DID 0001-0010-00000452  
session 147: DID 0001-0010-00000452                      session 150: DID 0001-0013-00000032

Rows waited on:

Session 147: obj - rowid = 0001593C - **AAAVk8AABAAARJFAAC**

(dictionary objn - **88380**, file - 1, block - 70213, slot - 2)

Session 150: obj - rowid = 0001593C - AA AVk8AABAAASG8AAj

(dictionary objn - 88380, file - 1, block - 74172, slot - 35)

## Information on the OTHER waiting sessions:

Session **147**:

pid=16 serial=625 audsid=-1 user: 0/SYS

O/S info: user: PORGAND\Tanel, term: PORGAND, ospid: 5900:2388, machine: MSHOME\P  
program: sqlplus.exe

application name: sqlplus.exe, hash value=0

## Current SQL Statement:

*update t set object\_id = 3 where object\_type = 'TABLE'*

End of information on OTHER waiting sessions.

# Thank you !!!

## Further questions welcome:

- [tanel@tanelpoder.com](mailto:tanel@tanelpoder.com)

## Slides, scripts and my blog:

- <http://blog.tanelpoder.com>
- ...I will post this session's sqlplus output there after the conference

## ***Advanced Oracle Troubleshooting*** seminar:

- 3-5. June - Dallas, TX
- 10-12. June - Denver, CO
- 15-17. June - Salt Lake City, UT

## More dates and seminar information:

- <http://blog.tanelpoder.com/seminar/>