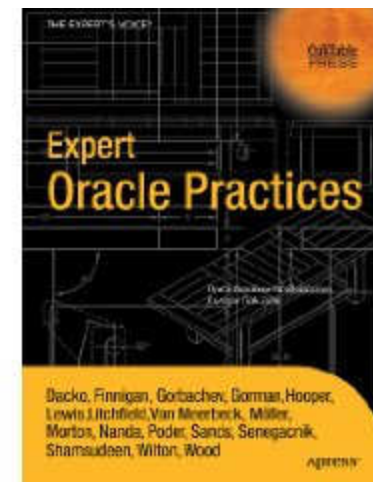

Few close encounters with real world performance issues

By
Riyaj Shamsudeen



Who am I?

- 17 years using Oracle products/DBA
- OakTable member
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in RAC, performance tuning, Internals and E-business suite
- Chief DBA with OraInternals
- Co-author of “Expert Oracle Practices” ‘2010
- Email: rshamsud at gmail.com
- Blog : orainternals.wordpress.com



Disclaimer

These slides and materials represent the work and opinions of the author and do not constitute official positions of my current or past employer or any other organization. This material has been peer reviewed, but author assume no responsibility whatsoever for the test cases.

If you corrupt your databases by running my scripts, you are solely responsible for that.

This material should not should not be reproduced or used without the authors' written permission.

This presentation

Idea behind this presentation is, to show how scientific techniques can be used to troubleshoot the performance issues, however bizarre those issues may be.

Agenda

- **Issue: High Kernel mode CPU usage**
- **Issue: High Shared pool latch contention and ORA-4031**
- **Issue: Hung RAC cluster**
- **Issue: High Kernel mode CPU usage – 2**
- **Issue: Excessive updates and RAC**
- **Issue: ASMM (Skipped due to time constraints)**

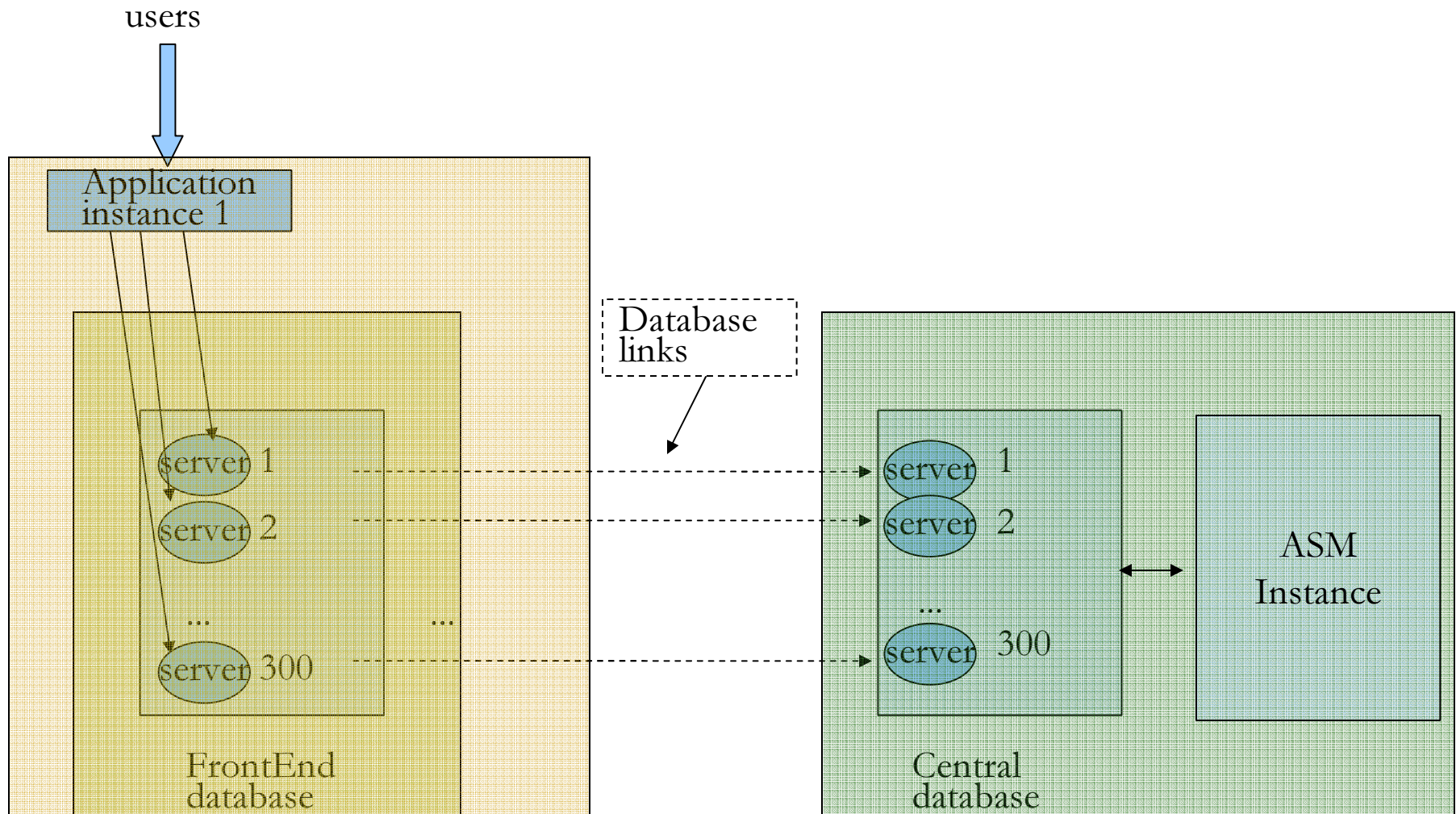
Issue

There are many application instances each servicing a disjoint group of users. As per the design, any application instance can be shutdown with no user impact and traffic will be automatically rerouted to surviving instances (almost similar to RAC).

Database version :10.2.0.4

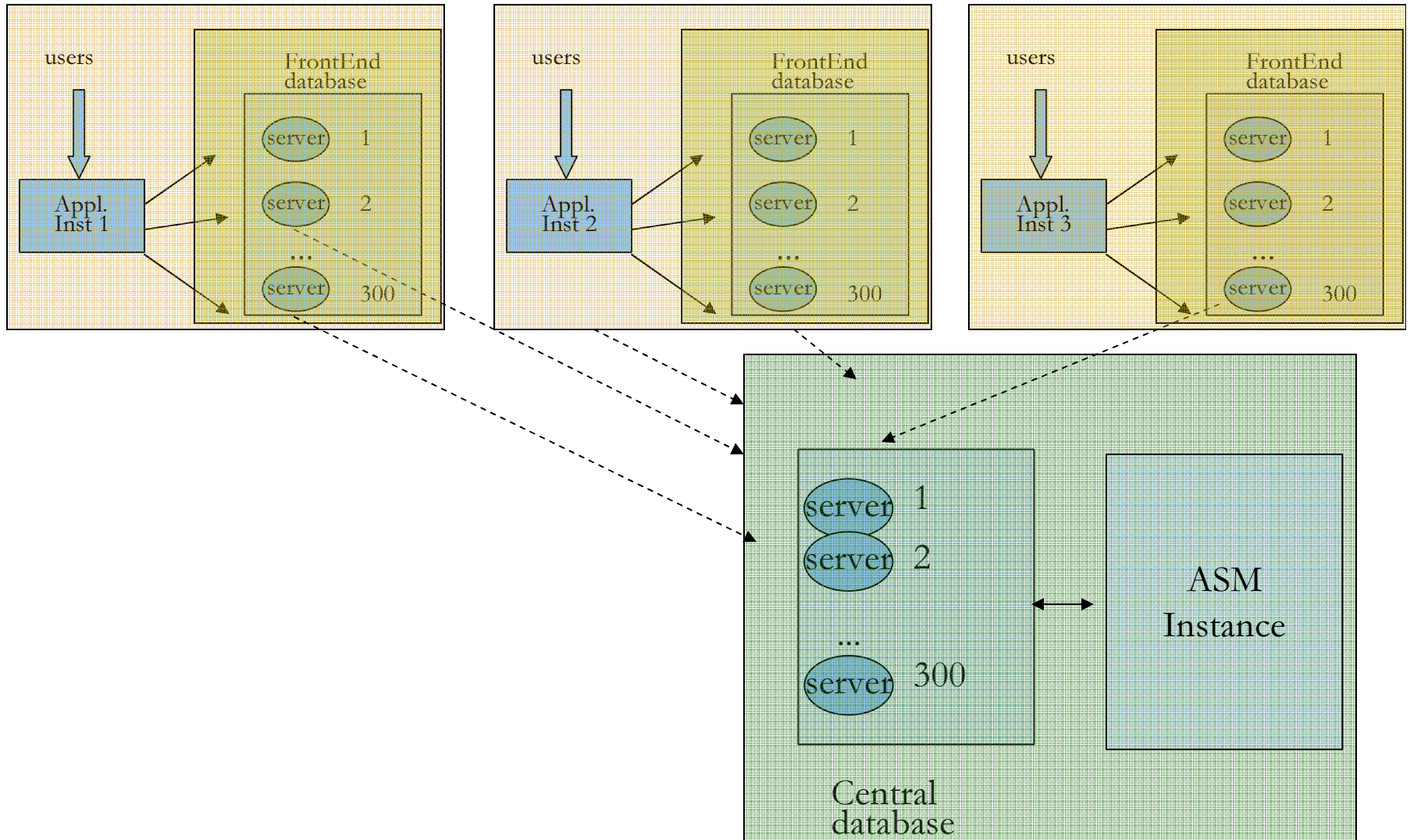
Operating system: Solaris 10

Architecture overview



Note, only partial architecture shown due to client confidentiality.

Modular

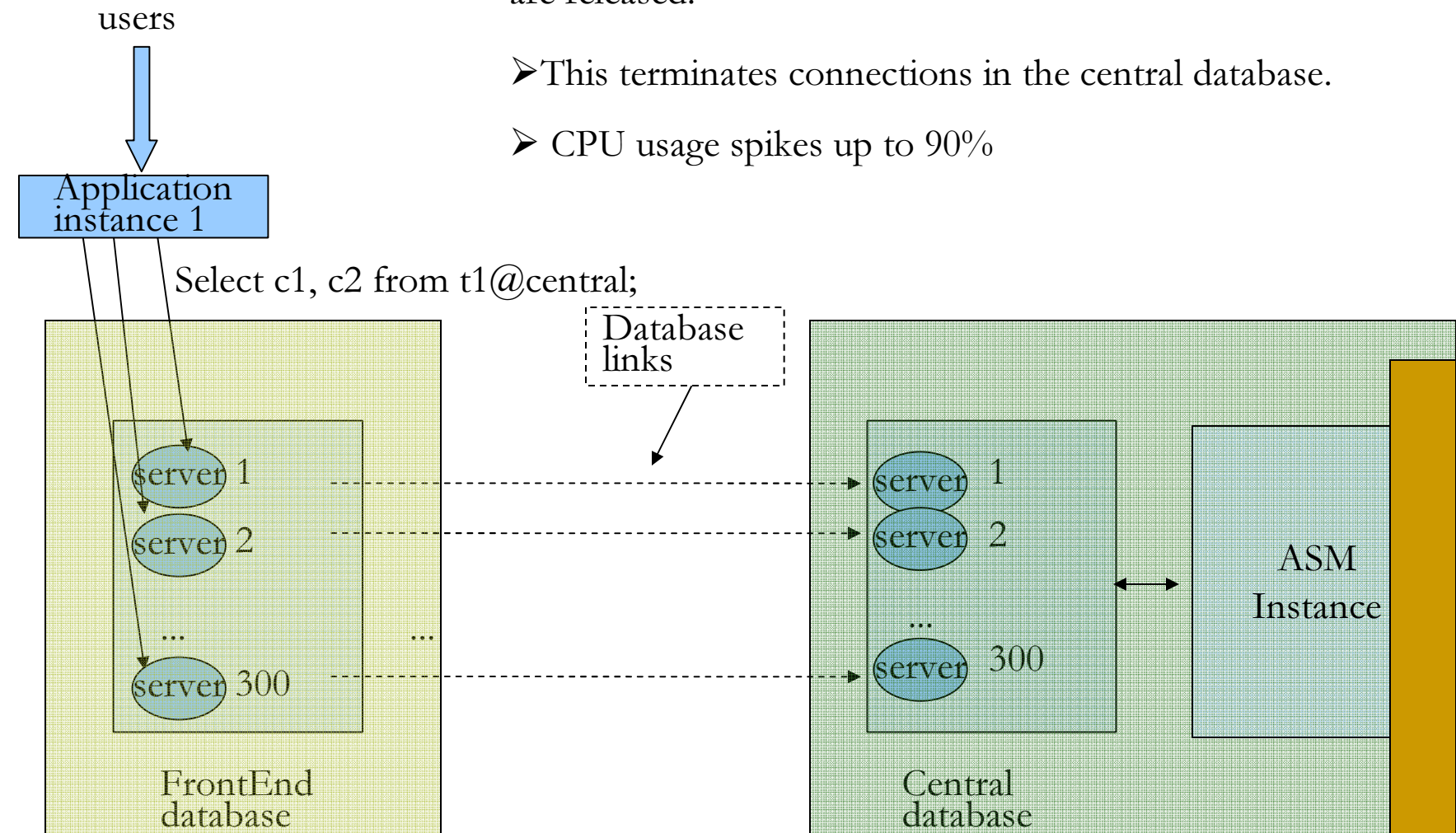


Note, only partial architecture shown due to client confidentiality.

Unfortunately, shutdown of *any* application instance shuts down *every* application instance leading to a site wide downtime.

Details

- After an application restart database link connections are released.
- This terminates connections in the central database.
- CPU usage spikes up to 90%



Symptoms

- Very high CPU usage in the kernel mode in the central database.
- ASM instance in that central database server times out and crashes.
- Why is there such an high CPU usage?

Who is using my CPU?

- In Solaris (and in many UNIX platforms) mpstat utility can be used to understand the per-processor statistics.
- Mpstat output shows that almost all the CPUs are used in kernel mode.

Tue Sep 9 17:46:34 2008	CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
Tue Sep 9 17:46:34 2008	0	561	0	9	554	237	651	219	87	491	0	4349	9	91	0	0
Tue Sep 9 17:46:34 2008	1	1197	1	34	911	0	2412	591	353	630	0	15210	30	63	0	7
Tue Sep 9 17:46:34 2008	2	58	0	9	313	0	613	106	190	105	0	3562	8	90	0	2
Tue Sep 9 17:46:34 2008	3	161	0	26	255	0	492	92	161	530	0	2914	6	92	0	2
Tue Sep 9 17:46:34 2008	4	0	0	0	86	1	2	3	1	63	0	8	0	100	0	0
Tue Sep 9 17:46:34 2008	5	283	0	34	662	0	1269	153	326	211	0	6753	13	77	0	10
Tue Sep 9 17:46:34 2008	6	434	0	43	349	0	589	54	170	1534	0	3002	7	88	0	5
...																
Tue Sep 9 17:46:34 2008	12	30	0	0	195	0	279	110	31	80	0	1590	3	97	0	0
Tue Sep 9 17:46:34 2008	13	288	0	9	449	0	844	117	158	127	0	4486	7	85	0	8
Tue Sep 9 17:46:34 2008	14	155	0	0	430	0	744	102	160	83	0	3875	7	80	0	13
Tue Sep 9 17:46:34 2008	15	16	0	0	237	0	359	115	31	124	0	2074	3	91	0	6

Statspack report

Out of 655 seconds total elapsed time
455 seconds spent on latch free waits.

	Snap Id	Snap Time	Sessions	Curs/Sess
Begin Snap:	3131	09-Sep-08 17:46:17	5,030	1.9
End Snap:	3132	09-Sep-08 17:47:16	4,995	2.0
Elapsed:		0.98 (mins)↑		
DB Time:		10.55 (mins)↑		

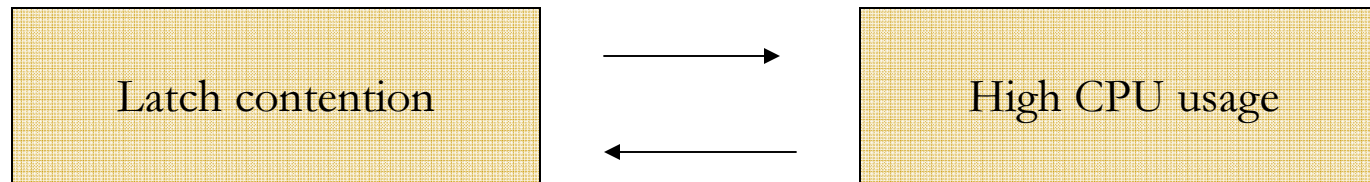
Almost all latch waits are
For enqueues latches.

Event	waits	Time (s)	(ms)	Time wait Class
latch free	868	455	525	71.9 Other
latch: row cache objects	103	189	1833	29.8 Concurrenc
log file sync	885	92	103	14.5 Commit
CPU time		85		13.4
db file parallel write	3,868	10	3	1.6 System I/O

latch contention is for enqueues latches:

Latch Name	Get Requests	Pct Get Miss	Avg Slps /Miss	wait Time (s)	NOwait Requests	Pct NOwait Miss
enqueues	1,355,722	3.3	0.0	452	0	N/A

Chicken or Egg?



- But, usual symptoms of latch contention is high CPU usage in *user* mode.
- In this case, We will ignore latch contention for now.

Let's reproduce the issue

- To reproduce kernel mode CPU issue, we will create a database link from an application schema, connecting to a test schema in the central database.
- Then, we will execute a select over the database link creating a new connection in the central database:

```
select * from dual@central;
```

Truss – trace system calls

- Identified the connection from test schema @ front-end database to the test schema @ central database.

```
select sid, serial#, LOGON_TIME, LAST_CALL_ET from v$session where  
logon_time > sysdate-(1/24)*(1/60)↑
```

SID	SERIAL#	LOGON_TIME	LAST_CALL_ET
1371	35028	12-SEP-2008 20:47:30	0
4306	51273	12-SEP-2008 20:47:29	1 <---

- Starting truss on that process in central database

```
truss -p <pid> -d -o /tmp/truss.log
```

Better yet.. Truss `-d -D -E -p <pid> -o /tmp/truss.log`

- Logged off from content database and this should trigger a log-off from remote central database.

Truss output

- Reading truss output in central database connection, we can see ten shmdt calls are consuming time.

```
18.4630 close(10)           = 0
18.4807 shmdt(0x380000000)  = 0
18.5053 shmdt(0x440000000)  = 0
18.5295 shmdt(0x640000000)  = 0
18.5541 shmdt(0x840000000)  = 0
18.5784 shmdt(0xA40000000)  = 0
18.6026 shmdt(0xC40000000)  = 0
18.6273 shmdt(0xE40000000)  = 0
18.6512 shmdt(0x1040000000) = 0
18.6752 shmdt(0x1240000000) = 0
18.6753 shmdt(0x1440000000) = 0
```

- Each shmdt call consumed approximately 0.024 seconds or 24ms. $18.5295 - 18.5053 = 0.0242$

Call: shmdt

- **shmdt calls** are used to detach from shared memory segments.
- Every database disconnect must detach from shared memory segments.

Shmctl calls

- There are 10 shared memory segments for this SGA. So, there are 10 shmctl calls.

```
ipcs -ma|grep 14382
m 1241514024 0x97e45100 --rw-r----- oracle orainvtr ...
m 1241514023 0 --rw-r----- oracle orainvtr ...
m 1241514022 0 --rw-r----- oracle orainvtr ...
.....
m 1241514016 0 --rw-r----- oracle orainvtr ...
m 889192479 0 --rw-r----- oracle orainvtr ...
```



8GB

Fun with numbers

- Each shmdt call consumes 0.024 seconds
- For one session (10 calls) = 0.24 seconds
- For 300 connections = $300 * 0.24 = 72$ seconds.
- At best case, with 12 concurrent processes, this would last for 6 seconds or so.
- This is matching with our observation of 6 seconds high kernel mode CPU usage.

Reducing shmdt calls

- To reduce shmdt calls we need to reduce shared memory segments.
- Database engine tries to create biggest segment possible at initial startup and slowly reduces segment size, until segments can be created successfully.
- SHMMAX kernel parameter was set lower, and so, we decided to increase that parameter first and reboot the server.

Well, that was embarrassing..

- After increasing SHMMAX size we expected to have just one shared memory segment.
- This will reduce the impact 10 times.
- Surprise! There were still 10 shared memory segments.

After SHMMAX change

- Started to truss on database startup to see why the instance is creating multiple shared memory segments.

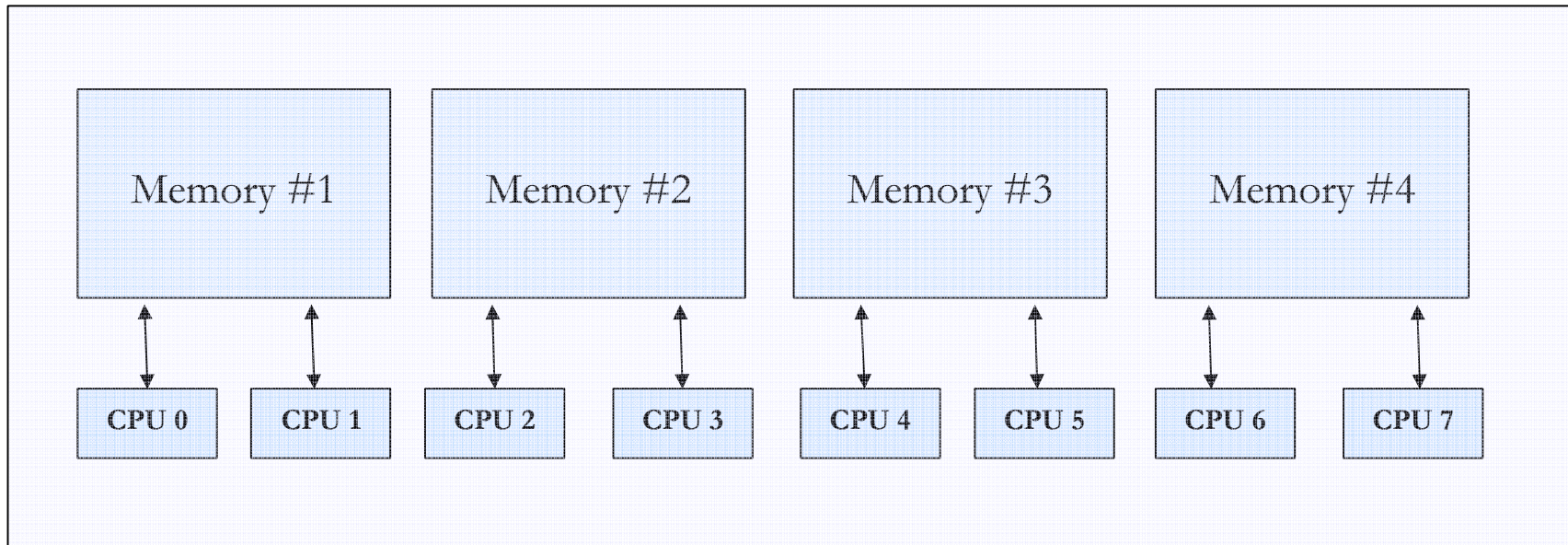
```
17252: 4.5957 munmap(0xFFFFFD7FFDAE0000, 32768) = 0
17252: 4.5958 lgrp_version(1, ) = 1
17252: 4.5958 _lgrpsys(1, 0, ) = 42
17252: 4.5958 _lgrpsys(3, 0x00000000, 0x00000000) = 19108
17252: 4.5959 _lgrpsys(3, 0x00004AA4, 0x06399D60) = 19108
17252: 4.5959 _lgrpsys(1, 0, ) = 42
17252: 4.5960 pset_bind(PS_QUERY, P_LWPID, 4294967295, 0xFFFFFD7FFFDFB11C) = 0
17252: 4.5960 pset_info(PS_MYID, 0x00000000, 0xFFFFFD7FFFDFB0D4, 0x00000000) = 0
17252: 4.5961 pset_info(PS_MYID, 0x00000000, 0xFFFFFD7FFFDFB0D4, 0x061AA2B0) = 0
```

pset_bind and _lgrpsys

➤ Calls _lgrpsys and pset_bind are new and googling these function calls showed that there may be related to Non Uniform Memory Access (NUMA) architecture.

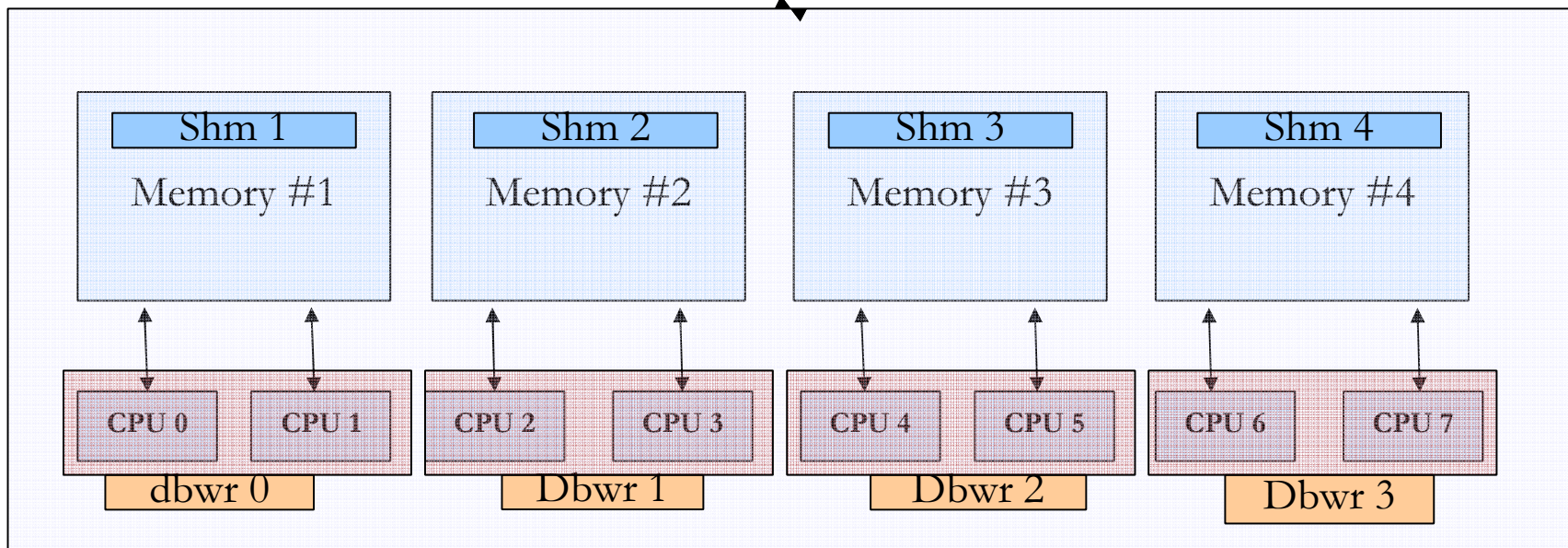
```
17252: 4.5957 munmap (0xFFFFFD7FFDAE0000, 32768) = 0
17252: 4.5958 lgrp_version (1, ) = 1
17252: 4.5958 _lgrpsys (1, 0, ) = 42
17252: 4.5958 _lgrpsys (3, 0x00000000, 0x00000000) = 19108
17252: 4.5959 _lgrpsys (3, 0x00004AA4, 0x06399D60) = 19108
17252: 4.5959 _lgrpsys (1, 0, ) = 42
17252: 4.5960 pset_bind (PS_QUERY, P_LWPID, 4294967295, 0xFFFFFD7FFDFB11C) = 0
17252: 4.5960 pset_info (PS_MYID, 0x00000000, 0xFFFFFD7FFDFB0D4, 0x00000000) = 0
17252: 4.5961 pset_info (PS_MYID, 0x00000000, 0xFFFFFD7FFDFB0D4, 0x061AA2B0) = 0
```

NUMA architecture (overview)



- For `cpu0` and `cpu1`, memory board 1 is local. Other memory areas are remote for `cpu0` and `cpu1`.
- Access to local memory is faster compared to remote memory access.

NUMA architecture (Overview)



- To make use of NUMA technology, Oracle spreads SGA across NUMA nodes.

NUMA optimization

- Binds DBWR to a CPU set. That DBWR handles all writes from that shared memory segment.
- User processes also tries to use free buffers from the working set of buffers from that NUMA node process is running from.
(Update: This turned out to be a Oracle database bug 5173642).
- LGWR also seems to have some code optimization to better use NUMA technology, but my test cases are not conclusive enough.

Locality groups

- In Solaris, NUMA technology is implemented as locality groups.
- `_lgrpsys` and `pset_bind` calls are to get current locality group information and bind processes to a processor set.
- Now, we can understand why SGA was split into multiple segments.
- But, Do we really have that many locality groups in this server?

Locality groups

- Lgrpinfo tool can provide NUMA node details

```
:~#/usr/local/bin/lgrpinfo
```

```
lgroup 0 (root):
```

```
  Children: 10 12 14 15 17 19 21 23
```

```
  CPUs: 0-15
```

```
  Memory: installed 65024 Mb, allocated 2548 Mb, free 62476 Mb
```

```
  Lgroup resources: 1-8 (CPU); 1-8 (memory)↑
```

```
  Latency: 146
```

```
lgroup 1 (leaf):
```

```
  Children: none, Parent: 9
```

```
  CPUs: 0 1
```

```
  Memory: installed 7680 Mb, allocated 1964 Mb, free 5716 Mb
```

```
  Lgroup resources: 1 (CPU); 1 (memory)↑
```

```
  Load: 0.105
```

```
  Latency: 51
```

```
...
```

There were many locality groups defined and seven of them were leaf node locality groups in this server.

Latency?

Local access to memory from CPU 1 to memory in the same NUMA node as the CPU has a reference number of 51.

Remote access to memory in a remote node has a reference number of 146.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
0	146	146	113	113	113	113	113	113	146	146	146	146	146	146	146	113	146	113	146	146	146	146	146	146	146	
1	146	51	81	81	113	113	113	113	146	81	113	113	146	113	146	146	113	146	113	146	146	146	146	146	146	146
2	113	81	51	113	81	113	113	81	113	113	113	81	113	113	113	113	113	113	113	113	113	113	113	113	113	113
3	113	81	113	51	113	81	81	113	113	113	113	113	113	81	113	113	113	113	113	113	113	113	113	113	113	113
4	113	113	81	113	51	81	81	113	113	113	113	113	113	113	113	113	81	113	113	113	113	113	113	113	113	113
...																										
15	146	146	113	113	113	113	113	113	146	146	146	146	146	146	146	113	113	146	113	146	146	146	146	146	146	146
...																										

Is NUMA bad?

- Indeed 10 shared memory segments were created, one for a locality groups.
- We disabled NUMA to resolve this problem temporarily.
- NUMA is a great technology. Sequent Dynix/ptx has implemented NUMA technology successfully a decade ago.
- It's just that we are encountering an unfortunate side effect of NUMA.

Solution

- We can disable NUMA or reduce number of NUMA nodes.
 - *._enable_NUMA_optimization=FALSE
 - *._db_block_numa = 1
 - Use patch for bug 819953 disable NUMA instead of underscore parameters (only if needed to disable NUMA)
 - Note 399261.1 and 759565.1 describes these issues.
 - It looks like, there is one shared memory segment per locality group and one segment encompassing all locality groups. One small bootstrap segment is also created.
-

Solution – contd.

- Another option is to control logout rate.
- Jonathan Lewis mentioned these parameters to control logout storm rate later.

Parameter	Meaning	Value
<code>_logout_storm_rate</code>	number of processes that can logout in a second	0
<code>_logout_storm_retrycnt</code>	maximum retry count for logouts	600
<code>_logout_storm_timeout</code>	timeout in centi-seconds for wait between retries	5

Agenda

- Issue: High Kernel mode CPU usage
- Issue: High Shared pool latch contention and ORA-4031
- Issue: Hung RAC cluster
- Issue: High Kernel mode CPU usage – 2
- Issue: Excessive updates and RAC
- Issue: ASMM (Skipped due to time constraints)

Problem

- Client was encountering high shared pool latch contention and, more importantly ORA-4031 errors.
- Client DBA queried v\$sgastat and found that ‘free memory’ is over few GBs in that 6GB shared pool.
- Of course, frequent ORA-4031 errors are dumping the SGA causing enormous disk I/O leading to more performance issues.

Heapdump

- Took heapdump of the SGA level 2, which is for shared pool dump.

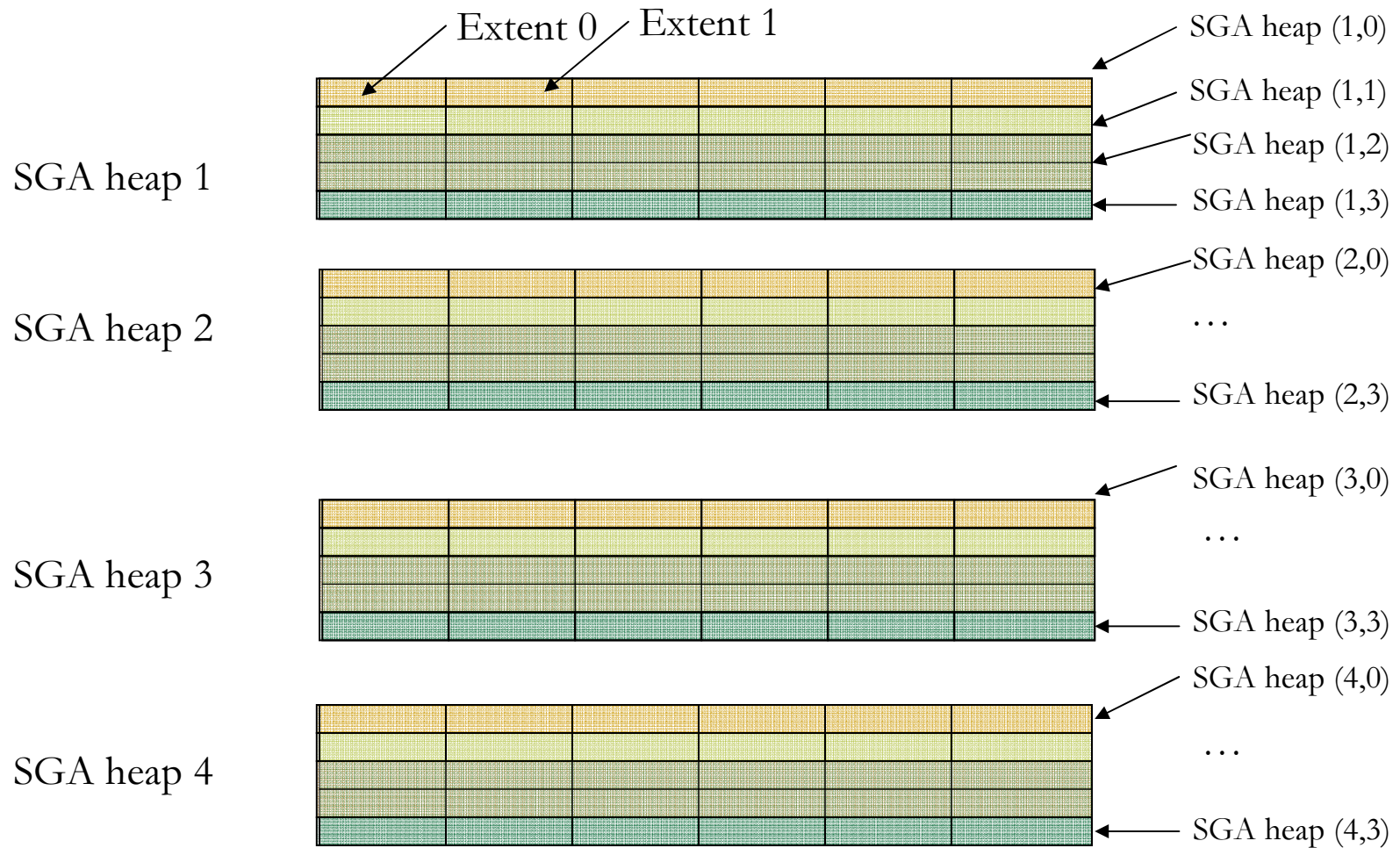
```
oradebug setmypid
```

```
oradebug dump heapdump 2
```

```
oradebug tracefile_name
```

- We know that shared pool is divided in to multiple smaller sub-heaps. But, what we saw much more different.

Heapdump analysis



In a nutshell..

- SGA heap is split in to multiple sub-heaps. Parameter `_ksmdsidx_count` controls number of sub-heaps.
- Each sub-heap is divided in to multiple mini-heaps or durations (4 in version 10g). `_enable_shared_pool_durations` parameter controls this behavior.
- Extents are allocated to the durations as needed basis. Parameter `_ksmg_granule_size` controls the size of these extents.

Extents & chunks

HEAP DUMP heap name="sga heap(1,0)" desc=0967F1D0

...

durations enabled for this heap

reserved granules for root 12 (granule size 4194304)

EXTENT 0 addr=26000000

Chunk 26000038 sz= 24 R-freeable "reserved stoppe"
Chunk 26000050 sz= 212888 R-free " "
Chunk 26033fe8 sz= 24 R-freeable "reserved stoppe"
Chunk 26034000 sz= 3979368 perm "perm" alo=3979368
Chunk 263ff868 sz= 1944 free " "

Reserved Area

EXTENT 1 addr=26400000

Chunk 26400038 sz= 24 R-freeable "reserved stoppe"
Chunk 26400050 sz= 212888 R-free " "
Chunk 26433fe8 sz= 24 R-freeable "reserved stoppe"

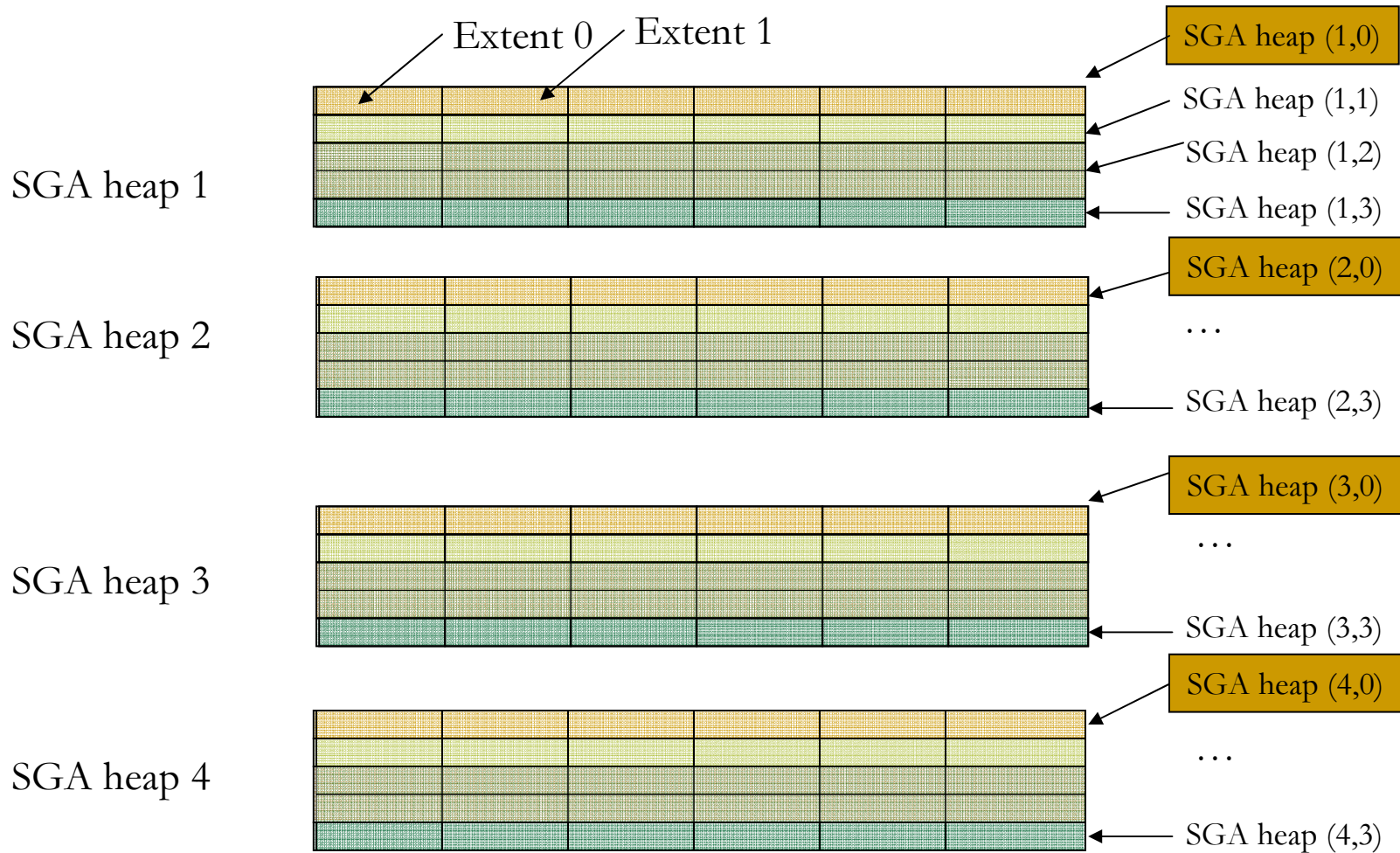
...

Durations

- A specific type of chunk is allocated in one of these durations.
- For example, all PERM chunks (permanent duration) or allocated in “SGA heap(1,0)”, “SGA heap (2,0)”, “SGA heap(3,0)” etc..
- These PERM chunks are allocated only from the first duration. In essence, each duration is reserved for specific type of allocation.

PERM

PERM chunks are allocated from Zeroth duration SGA heap(1,0), SGA heap (2, 0), SGA heap (3,0), SGA heap (4,0).



ORA-4031

- If the first duration can not allocate more extents to grow, then ORA-4031 is thrown.
- It is perfectly possible for other durations within the same sub-heaps to have ample amount of free space.
- Due to a code issue, there was a sudden increase “enqueue resources” chunk. This lead to allocation of PERM chunks from sga heap(1,0).
- But, there was not enough free memory to allocate a new extent leading to ORA-4031 error.

Solution

- Recommended solution would be to use Automatic Shared Memory Management feature, but it has its own perils.
- As an immediate work-around, we disabled the duration using `_enable_shared_pool_durations` parameter.
- Client was also working with a vendor to modify the code to avoid accessing enqueues excessively.
- There was a debate as to whether this is a bug or a feature. The decision is left to the readers as an exercise.

Agenda

- Issue: High Kernel mode CPU usage
- Issue: High Shared pool latch contention and ORA-4031
- Issue: Hung RAC cluster
- Issue: High Kernel mode CPU usage – 2
- Issue: Excessive updates and RAC
- Issue: ASMM (Skipped due to time constraints)

Problem

- All RAC instances are stuck for 10-15 minutes intermittently. Application is not responsive during that time period.
- This happens randomly and no specific correlation with time of day.

AWR analysis

- AWR analysis indicates gc buffer busy waits, many sessions were waiting for GC events.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
gc buffer busy acquire	Cluster	46.61	6.42
CPU + Wait for CPU	CPU	21.91	3.02
gc cr block busy	Cluster	9.14	1.26
enq: CF - contention	Other	4.44	0.61
gc current block busy	Cluster	2.50	0.35

ASH analysis

- ASH report confirms the issue too.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
gc buffer busy acquire	Cluster	43.68	11.48
gc cr block busy	Cluster	10.92	2.87
CPU + Wait for CPU	CPU	9.84	2.59
row cache lock	Concurrency	5.12	1.35
gc cr multi block request	Cluster	4.90	1.29

Why gc buffer busy?

- GC buffer busy waits indicates that buffer is busy waiting for some sort of Global event.

↳ ■ Another session is working on that buffer and that session is waiting for a global cache event.

↳ ■ We need to understand why that session 2 is waiting for global cache event.

Another node...

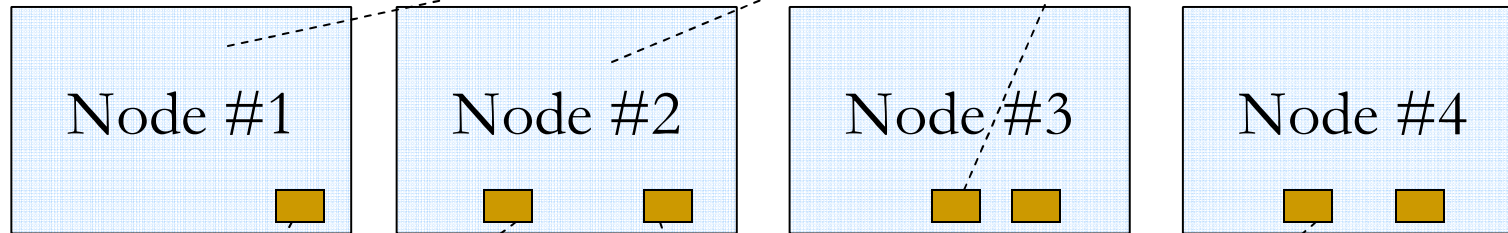
- In RAC, it is essential to review performance statistics from all nodes.
- One node performance can bring down entire cluster performance or even lead to hung cluster.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
buffer busy waits	Concurrency	30.92	12.00
log file sync	Commit	23.61	9.16
log buffer space	Configuration	16.88	6.55
CPU + Wait for CPU	CPU	7.48	2.90
row cache lock	Concurrency	3.31	1.29

Review of waits

User sessions are waiting for
Global buffer busy waits.



Background sessions are
Waiting for CF locks.

User sessions are waiting for buffer
busy waits and log file sync waits.

Event	% Activity
enq: CF - contention	5.52
CPU + Wait for CPU	2.62
enq: TM - contention	1.28
gc cr block busy	1.27

Event	% Event
buffer busy waits	30.92
log file sync	23.61
log buffer space	16.88

Buffer busy waits

- Buffer busy waits indicates that buffers are not available and busy undergoing a short change.
- Buffer busy waits can be caused by DBW trying to write buffers too.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
buffer busy waits	Concurrency	30.92	12.00
log file sync	Commit	23.61	9.16
log buffer space	Configuration	16.88	6.55
CPU + Wait for CPU	CPU	7.48	2.90
row cache lock	Concurrency	3.31	1.29

Log file sync waits

- Log file sync waits indicates that log file write mechanism is not fast enough.
- This could be due to problem with LGWR, Log file I/O performance issue or even OS CPU scheduling issues.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
buffer busy waits	Concurrency	30.92	12.00
log file sync	Commit	23.61	9.16
log buffer space	Configuration	16.88	6.55
CPU + Wait for CPU	CPU	7.48	2.90
row cache lock	Concurrency	3.31	1.29

Background waits

- Further review of ASH report indicates that there were waits for background processes too.
- Few enq: CF contention waits. %Activity is 5.5, but that can be misleading.

Top Background Events

Event	Event Class	% Activity	Avg Active Sessions
enq: CF - contention	Other	5.52	0.63
CPU + Wait for CPU	CPU	2.62	0.30
enq: TM - contention	Application	1.28	0.15
gc cr block busy	Cluster	1.27	0.15

Review

- User processes in node 3 & 4 are suffering from global buffer busy waits.
- User processes in node 2 are suffering from buffer busy waits and log file sync waits.
- Background processes in node 2 are suffering from CF enqueue waits and buffer busy waits.
- If there are background processes waiting for locking contention, then that must be resolved first. Every thing else could be just a symptom.

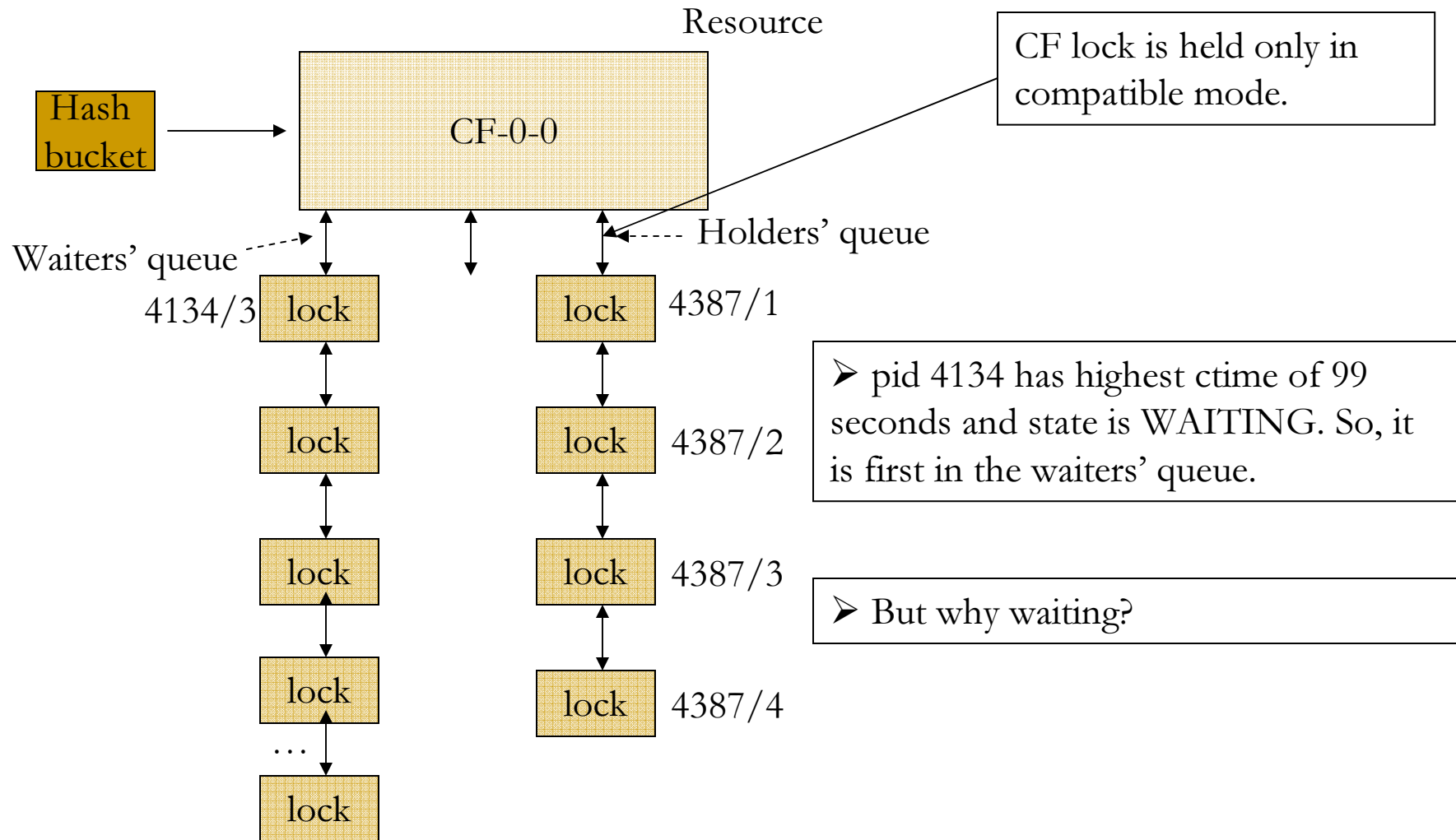
gv\$lock

Notice that no process holding CF lock in an incompatible mode.

INST	ADDR	KADDR	SID	TY	ID1	ID2	LMODE	REQ	CTIME	BLOCK
4	0000001022E12398	0000001022E123F0	4368	CF	0	0	0	4	8	0
4	0000001022E12FE0	0000001022E13038	4369	CF	0	0	0	4	39	0
...										
4	0000001022E15588	0000001022E155E0	4374	CF	0	0	0	4	34	0
4	0000001022E12BD0	0000001022E12C28	4375	CF	0	0	0	4	39	0
4	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	120592	2
4	0000001022E13E98	0000001022E13EF0	4388	CF	0	0	0	5	49	0
...										
1	0000001022E12058	0000001022E120B0	4372	CF	0	0	0	4	41	0
1	0000001022E121F8	0000001022E12250	4373	CF	0	0	0	4	41	0
1	0000001022E12E40	0000001022E12E98	4374	CF	0	0	0	4	41	0
1	0000001022E133F0	0000001022E13448	4376	CF	0	0	0	4	41	0
1	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	121783	2
3	0000001022E09BC8	0000001022E09C20	4134	CF	0	0	0	4	99	0
3	0000001022E15A68	0000001022E15AC0	4368	CF	0	0	0	4	39	0
3	0000001022E15658	0000001022E156B0	4369	CF	0	0	0	4	39	0
3	0000001022E15C08	0000001022E15C60	4370	CF	0	0	0	4	39	0
...										
3	0000001022E154B8	0000001022E15510	4376	CF	0	0	0	4	39	0
3	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	120855	2
2	0000001022E15318	0000001022E15370	4368	CF	0	0	0	4	40	0
...										
2	0000001022E14EF0	0000001022E14F48	4373	CF	0	0	0	5	81	0
2	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	121231	2

38 rows selected.

Locking scenario



Problem continued..

Process is waiting for 283
Seconds

INST_ID	ADDR	KADDR	SID	TY	ID1	ID2	LMODE	REQUEST	CTIME	BLOCK
4	0000001022E12398	0000001022E123F0	4368	CF	0	0	0	4	193	0
4	0000001022E12FE0	0000001022E13038	4369	CF	0	0	0	4	224	0
4	0000001022E13CF8	0000001022E13D50	4370	CF	0	0	0	5	266	0
4	0000001022E0FD20	0000001022E0FD78	4371	CF	0	0	0	4	224	0
4	0000001022E12E40	0000001022E12E98	4372	CF	0	0	0	4	224	0
4	0000001022E126D8	0000001022E12730	4373	CF	0	0	0	4	224	0
4	0000001022E15588	0000001022E155E0	4374	CF	0	0	0	4	219	0
4	0000001022E12BD0	0000001022E12C28	4375	CF	0	0	0	4	224	0
4	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	120777	2
4	0000001022E13E98	0000001022E13EF0	4388	CF	0	0	0	5	234	0
2	0000001022E15318	0000001022E15370	4368	CF	0	0	0	4	224	0
2	0000001022E15CD8	0000001022E15D30	4369	CF	0	0	0	4	224	0
2	0000001022E14108	0000001022E14160	4370	CF	0	0	0	4	224	0
2	0000001022E15E90	0000001022E15EE8	4371	CF	0	0	0	4	223	0
2	0000001022E15B38	0000001022E15B90	4372	CF	0	0	0	4	224	0
2	0000001022E14EF0	0000001022E14F48	4373	CF	0	0	0	5	265	0
2	0000001022E154B8	0000001022E15510	4374	CF	0	0	0	4	224	0
2	0000001022E15DA8	0000001022E15E00	4375	CF	0	0	0	4	224	0
2	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	121415	2
1	0000001022E13660	0000001022E136B8	4368	CF	0	0	0	4	225	0
1	0000001022E12128	0000001022E12180	4369	CF	0	0	0	4	225	0
1	0000001022E13250	0000001022E132A8	4370	CF	0	0	0	4	225	0
1	0000001022E10CA8	0000001022E10D00	4371	CF	0	0	0	4	249	0
1	0000001022E12058	0000001022E120B0	4372	CF	0	0	0	4	225	0
1	0000001022E121F8	0000001022E12250	4373	CF	0	0	0	4	225	0
1	0000001022E12E40	0000001022E12E98	4374	CF	0	0	0	4	225	0
1	0000001022E133F0	0000001022E13448	4376	CF	0	0	0	4	225	0
1	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	121967	2
3	0000001022E09BC8	0000001022E09C20	4134	CF	0	0	0	4	283	0
3	0000001022E0EE68	0000001022E0EEC0	4190	CF	0	0	0	4	18	0
3	0000001022E15A68	0000001022E15AC0	4368	CF	0	0	0	4	223	0
3	0000001022E15658	0000001022E156B0	4369	CF	0	0	0	4	223	0
3	0000001022E15C08	0000001022E15C60	4370	CF	0	0	0	4	223	0
3	0000001022E13590	0000001022E135E8	4371	CF	0	0	0	4	238	0
3	0000001022E13F68	0000001022E13FC0	4372	CF	0	0	0	4	225	0
3	0000001022E15998	0000001022E159F0	4373	CF	0	0	0	4	223	0
3	0000001022E15318	0000001022E15370	4374	CF	0	0	0	4	223	0
3	0000001022E154B8	0000001022E15510	4376	CF	0	0	0	4	223	0
3	0000001022DFBD30	0000001022DFBD88	4387	CF	0	0	2	0	121039	2

Pstack

Pstack 4134

```
#0 0x00000030364cb053 in __select_nocancel () from /lib64/libc.so.6
#1 0x000000001d92111 in skgpnep ()
#2 0x00000000752d9b6 in kslwat ()
#3 0x00000000752b668 in kslwait ()
....
#9 0x00000000753ed1b in ksqgtlctx ()
#10 0x00000000753db0b in ksqgelctx ()
#11 0x0000000076f5bb8 in kcc_get_enqueue ()
#12 0x0000000076f329b in kccocx ()
#13 0x0000000076f3140 in kccbcx ()
#14 0x000000005563b0c in kcra_scan_redo ()
#15 0x00000000556334d in kcra_dump_redo ()
#16 0x000000005561fcc in kcra_dump_redo_internal ()
```

Usually called if the process dumping
due to errors or exceptions.

Is there a process dumping errors?

Alert log

- At the same time, alert log had entries for that PID 4134
- At the end of the trace file it was hung in 'PINNED BUFFER HISTORY'.

*** 2009-05-21 10:46:04.109

*** SESSION ID:(**4134.4598**) 2009-05-21 10:46:04.109

*** CLIENT ID:() 2009-05-21 10:46:04.109

*** SERVICE NAME:(PROD) 2009-05-21 10:46:04.109

*** MODULE NAME:() 2009-05-21 10:46:04.109

*** ACTION NAME:() 2009-05-21 10:46:04.109

Dump continued from file:

/plogs/PROD/dump/diag/rdbms/prod/PROD3/trace/PROD3_ora_1004.trc

ORA-07445: exception encountered: core dump [ksxpmprrp()+42] [SIGSEGV] [ADDR:0x14]

[PC:0x33BB5BE] [Address not mapped to object] []

That's a bug!

➤ Of course, that's a bug we were encountering.

➤ Bug 8318486: CF ENQUEUE CONTENTION WHILE DUMPING REDO RECORDS IN PINNED BUFFER HISTORY

➤ As per the bug, process requests for CF locks, but hangs until cleaned up by pmon.

➤ Of course, easy fix is to kill the processes encountering ORA-7445 errors immediately and the long term fix was to fix these bugs (Both ORA-7445 errors and bug 8318486).

CF enqueue wierdness

INST_ID	SID	TY	ID1	ID2	LMODE	REQUEST	CTIME	BLOCK
2	4387	CF	0	0	2	0	122115	2
1	4113	CF	0	4	4	0	77	0
1	4113	CF	0	0	4	0	77	2
1	4387	CF	0	0	2	0	122667	2
4	4387	CF	0	0	2	0	121476	2
3	4387	CF	0	0	2	0	121739	2

Agenda

- Issue: High Kernel mode CPU usage
- Issue: High Shared pool latch contention and ORA-4031
- Issue: Hung RAC cluster
- Issue: High Kernel mode CPU usage – 2
- Issue: Excessive updates and RAC
- Issue: ASMM (Skipped due to time constraints)

Problem

- High Kernel mode usage in an high end servers (Different client from discussion 1).
- Database was upgraded from 9i to 10gR2 recently.
- But, the node was running for fine for few weeks with no issues!

Mpstat – per processor stats

Mpstat indicates many processes using CPU in %sys mode.

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
0	25	0	107	294	6	1711	232	661	265	0	3702	33	40	0	26
1	21	0	454	205	9	1399	161	547	227	0	3422	44	33	0	23
2	23	0	91	146	5	1301	123	502	255	0	3503	44	34	0	22
3	123	0	1772	161	5	1383	127	544	264	0	3438	44	33	0	23
4	163	0	1651	144	5	1308	109	448	208	0	2984	45	31	0	23
...															
37	94	0	1295	4652	4600	450	44	174	376	0	1554	51	43	0	5
38	418	0	217	128	24	1039	79	376	127	1	3307	49	32	0	20
39	41	0	1310	4904	4863	495	35	174	428	0	2218	51	40	0	9
64	4	0	45	171	18	887	121	340	299	0	5802	31	49	0	20
65	73	0	1188	148	9	1219	116	453	231	0	5418	38	42	0	21
66	171	0	5809	133	27	1247	78	452	220	0	4524	41	38	0	22
67	5	0	278	204	57	1583	110	567	254	0	4898	35	40	0	25
68	0	0	5	41	27	8	9	5	5	0	7	99	1	0	0
69	1	0	79	128	5	1465	87	495	279	0	6097	29	45	0	26
70	6	0	1173	4342	4277	789	63	275	653	0	5596	23	64	0	13

AWR report

AWR report for a 30 minute period showed nothing obvious

Top 5 Timed Events

Event	Waits	Time (s)	Avg wait (ms)	%Total Call Time	Wait Class
CPU time		67,946		48.1	
db file sequential read	7,007,846	38,738	6	27.4	User I/O
gc buffer busy	1,705,205	12,142	7	8.6	Cluster
gc cr grant 2-way	2,804,825	6,538	2	4.6	Cluster
db file scattered read	761,202	6,330	8	4.5	User I/O

- I/O wait times are not too abnormal.
- We can theorize that if there is high I/O, it can result in high kernel mode CPU usage.
- But, this workload is quite normal for this node.

Dtrace and Solaris 10

- Dtrace is a great tool to do root cause analysis and available in Solaris 10.
- Dtrace can be used to see what calls are executed by peeking at CPUs.
- For example, following dtrace one-liner can break down the system calls executing in the server.

```
dtrace -n 'syscall:::entry { @Calls[probefunc] = count(); }'
```

Dtrace output

Dtrace output for kernel mode CPU usage shows CPUs
Are spending time in mutex. Not much help here.

unix`lock_set_spl_spin	1845	0.6%
unix`utl0	1873	0.6%
unix`atomic_add_32	2036	0.7%
unix`page_exists	2105	0.7%
unix`lock_set	2165	0.7%
SUNW,UltraSPARC-IV`send_mondo_set	2716	0.9%
genunix`fsflush	2765	0.9%
genunix`avl_walk	3622	1.2%
unix`disp_lowpri_cpu	3722	1.2%
unix`default_lock_delay	6790	2.2%
unix`kphysm_del_span_query	6790	2.2%
genunix`rm_assize	6830	2.2%
unix`_resume_from_idle	11758	3.8%
unix`mutex_enter	16469	5.4%
unix`disp_getwork	18809	6.1%
unix`mutex_delay_default	69892	22.7%

Mpstat again

One interesting to notice in the mpstat output is that CPUs with high Kernel mode usage also has high amount Of xcalls.

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
...															
453	33	1	615	76	6	1052	54	290	265	0	3243	38	23	0	39
454	0	0	35	58	6	807	37	227	197	0	3090	51	16	0	33
455	37	0	39	74	6	993	37	244	220	0	3812	36	18	0	46
480	0	0	105694	107	6	926	80	292	299	0	4319	32	35	0	32
481	51	0	214	81	6	842	59	255	233	0	3217	52	19	0	28
482	41	0	43	92	6	1105	64	325	318	0	3377	41	22	0	37
483	68	1	95373	104	6	1060	80	313	355	0	4238	33	28	0	39
484	0	0	23	56	6	746	36	231	156	0	2131	46	16	0	38
485	10	0	1931	64	6	703	43	193	151	0	3659	53	14	0	33
486	0	0	52	39	6	564	17	145	137	0	1513	17	16	0	67
487	1	1	420	30	6	225	14	68	44	0	778	85	5	0	10

xcalls

- Cross calls are CPU-to-CPU interrupts used for Memory consistency.
- In a server with many memory boards and huge SGA, cross calls are necessary evil.
- But, excessive and continuous cross calls are not optimal.
- If many processes are accessing SGA buffers aggressively then that can lead to increased cross calls.
- Increased cross calls = increased %sys mode CPU usage.

AWR report again

- So, excessive database activity can lead to higher xcalls and can lead to a symptom of high Kernel mode CPU usage.
- Interestingly, there is just one SQL with very high elapsed and cpu time. One session can't cause this issue!

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id
20,813	20,784	1	20812.8	14.7	d3zaxrb127axc

```
INSERT INTO ACCOUNTS_HISTORY ( SID, ACT, PARENT, LINEAGE_STRING, DESCRIPTION,  
LEVEL_NUMBER, PARENT_CHILD_FLAG) SELECT SESSIONID, FLEX_VALUE, ACT, PARENT, DESCR,  
:B1, RANGE_ATTRIBUTE FROM ACCOUNT...
```

But..

- AWR reports does not show SQL statements if the statements are still executing!
- So, decided to review few hours report instead of a 30 minutes AWR report.
- 32.1% of DB time spent on one SQL? That can cause problems.

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id
60,400	61,156	22	2745.5	32.1	d3zaxrb127axc

INSERT INTO ACCOUNTS_HISTORY (SID, ACT, PARENT, LINEAGE_STRING, DESCRIPTION,
LEVEL_NUMBER, PARENT_CHILD_FLAG) SELECT SESSIONID, FLEX_VALUE, ACT, PARENT, DESCR,
:B1, RANGE_ATTRIBUTE FROM ACCOUNT..

If you don't succeed first, try again?

- Apparently, a report were not completing in time.
- Front end program timed out and users submitted more reports out of frustration.
- Client has an alert: If there are many sessions executing the same SQL statement for prolonged period.
- Unfortunately, these sessions had no sql_id populated in v\$session and so alert failed.

Opened cursors?

But, v\$open_cursor shows that there are many sessions with opened cursors on that sql_id.

```
select sid, serial#, module, osuser from v$session where sid in (  
  select sid from v$open_cursor where sql_id=' d3zaxrb127axc ' );
```

SID	SERIAL#	MODULE	OSUSER
10365	65389		prod
10162	31712		prod
9979	13788		prod
10763	1819		prod
10007	46806		prod
9576	33605		prod

...

40 rows selected.

Execution plan

Merge join cartesian step 2 caused the issue. Cardinality estimates at step 3 is 1 and so, CBO chose cartesian join at step 2.

Id	Operation	Name	E-Rows	OMem	lMem	Used-Mem
1	NESTED LOOPS		1			
2	MERGE JOIN CARTESIAN		1			
3	NESTED LOOPS		1			
4	NESTED LOOPS		1			
5	NESTED LOOPS		1			
6	NESTED LOOPS		1			
7	NESTED LOOPS		1			
8	NESTED LOOPS		1			
9	NESTED LOOPS		1			
* 10	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
* 11	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
* 12	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
* 13	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
14	TABLE ACCESS BY INDEX ROWID	VALUE_SETS	1			
* 15	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
* 16	INDEX UNIQUE SCAN	VALUE_SETS_U2	1			
17	TABLE ACCESS BY INDEX ROWID	ACCT_HIER_ALL	1			
* 18	INDEX RANGE SCAN	ACCT_HIER_N1	1			
* 19	INDEX RANGE SCAN	FLEX_VALUE_NM_HIER_U1	1			
20	BUFFER SORT		97323	67M	2842K	59M (0)
* 21	TABLE ACCESS FULL	FLEX_VALUES_TL	97323			
* 22	TABLE ACCESS BY INDEX ROWID	FLEX_VALUES	1			
* 23	INDEX UNIQUE SCAN	FLEX_VALUES_U1	1			

Solution

- Cardinality estimates on table step 18 and 17 were totally incorrect.
- An index was added to the table.
- From 10g onwards, compute statistics is default. This threw away histograms on those columns in the index leading to incorrect cardinality calculations.
- Recollecting stats with proper histograms completely resolved the issue.

Agenda

- Issue: High Kernel mode CPU usage
- Issue: High Shared pool latch contention and ORA-4031
- Issue: Hung RAC cluster
- Issue: High Kernel mode CPU usage – 2
- Issue: Excessive updates and RAC
- Issue: ASMM (Skipped due to time constraints)

Problem

Client had high Global Cache response time waits.

Global Cache and Enqueue Services - Workload Characteristics

```
~~~~~
```

Avg global enqueue get time (ms):	2.5
Avg global cache cr block receive time (ms):	18.2
Avg global cache current block receive time (ms):	14.6
Avg global cache cr block build time (ms):	0.3
Avg global cache cr block send time (ms):	0.2
Global cache log flushes for cr blocks served %:	25.1
Avg global cache cr block flush time (ms):	5.2
Avg global cache current block pin time (ms):	0.4
Avg global cache current block send time (ms):	0.2
Global cache log flushes for current blocks served %:	1.7
Avg global cache current block flush time (ms):	5.2

GC CR latency

- GC CR latency $\sim =$

Time spent in sending message to LMS +
LMS processing (building blocks etc) +
LGWR latency (if any) +
LMS send time +
Wire latency

Statistics : gc cr block flush time
gc cr block build time
gc cr block send time

CR latency

- Three instances are suffering from CR latency, except instance 2!

Wait time	Node 1	Node 2	Node 3	Node 4
Avg. CR block receive time	18.2	6.7	20.0	17.3
Avg CUR block receive time	14.6	5.0	11.6	17.3

- In RAC, node suffering from chronic issues causes GC performance issues in other nodes. With that logic in mind, node 2 should be suffering from chronic issues.

Breakdown of latency

- Sum of flush time is higher, but it is comparable across the cluster.

But, notice the build time in node 2.

Statistics	Node 1	Node 2	Node 3	Node 4	Total
gc cr block build time	11,392	148,666	5,267	6,632	171,957
Gc cr block flush time	56,634	75,751	34,406	53,031	219,822
Gc cr block send time	9,153	7,779	4,018	7,905	28,855

Consistent reads

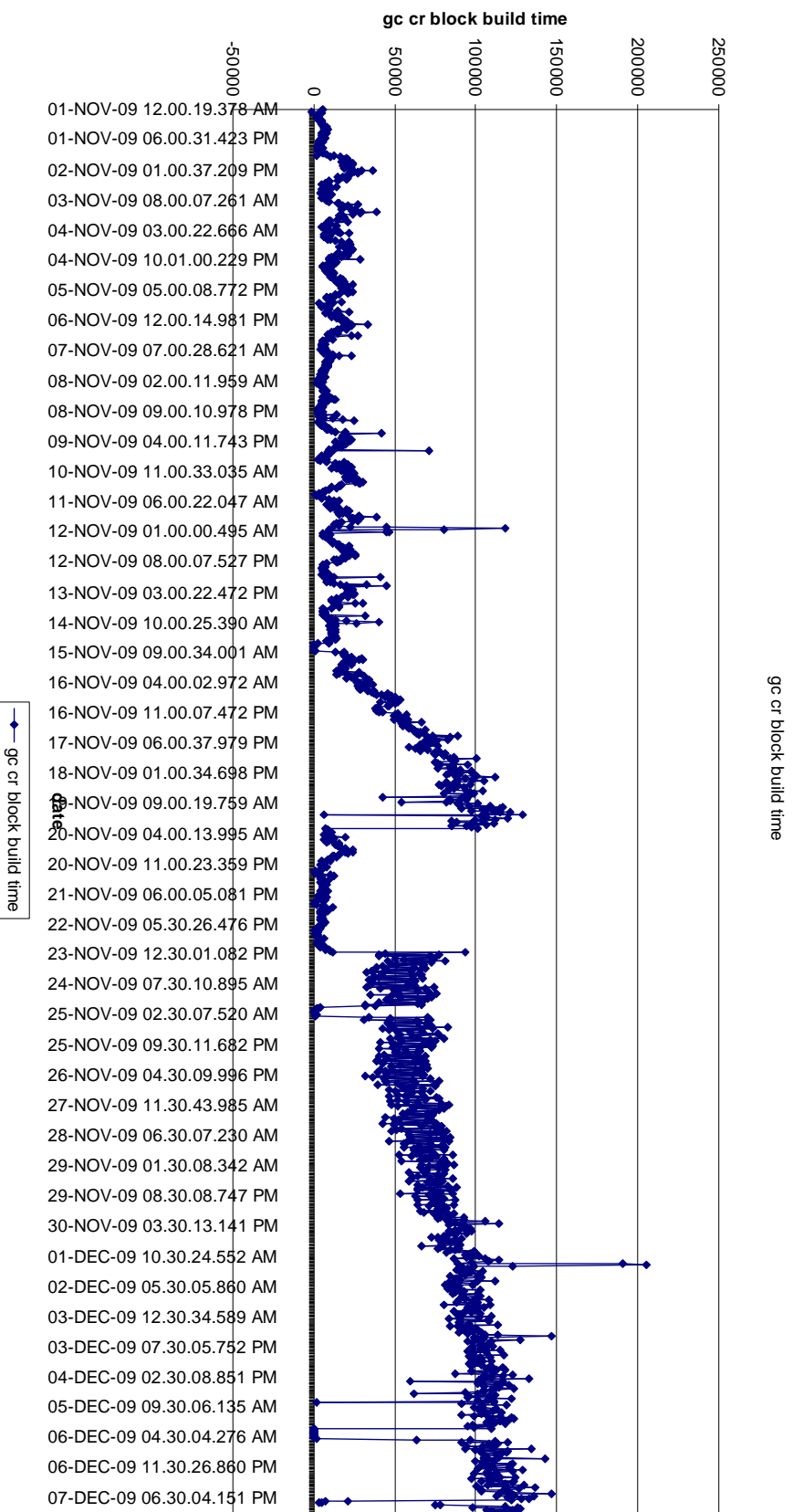
- For CR blocks, time is spent in building blocks, which indicates consistent block generation.

Very high value compared to other nodes.

Statistics	Node 1	Node 2	Node 3	Node 4
data blocks consistent Reads – undo records applied	2,493,242	86,988,512	3,090,308	7,208,575
db block changes	6,276,149	43,898,418	20,698,189	14,259,340

Time line

- We wanted to see when this problem started. Surprisingly, instance 2 had a pattern of increasing flush time.



Db block changes

Unfortunately, AWR report does not capture segments with high 'db block changes'.

```
with segstats as (  
  select * from (  
    select inst_id, owner, object_name, object_type , value ,  
          rank() over (partition by inst_id, statistic_name order by value desc  
    ) rnk , statistic_name  
    from gv$segment_statistics  
    where value >0  
  ) where rnk <11  
),  
sumstats as ( select inst_id, statistic_name, sum(value) sum_value from  
gv$segment_statistics group by statistic_name, inst_id)  
select a.inst_id, a.statistic_name, a.owner, a.object_name,  
a.object_type,a.value,(a.value/b.sum_value)*100 perc  
from segstats a , sumstats b  
where a.statistic_name = b.statistic_name  
and a.inst_id=b.inst_id  
and a.statistic_name ='db block changes'  
order by a.statistic_name, a.value desc  
/
```

INST_ID	STATISTIC_NAME	OWNER	OBJECT_NAME	TYPE	VALUE	PERC
2	db block changes	AR	CUSTOM_TABLE	TABLE	122949282400	81.39
4		INV	MTL_MATERIAL_TRANS_TEMP_N1	INDEX	1348827648	16.59
3		AR	RA_INTERFACE_LINES_N2	INDEX	791733296	9.77
3		AR	RA_CUSTOMER_TRX_LINES_N2	INDEX	715855840	8.83
1		INV	MTL_MATERIAL_TRANS_TEMP_N1	INDEX	652495808	12.44

...

Solution

- Finally, it boiled down to a custom code bug which was updating almost all rows in a table unnecessarily.
- Unfortunately, number of rows that fall in to that criteria was slowly increasing.
- So, GC CR response time was slowly creeping up and it wasn't easy to identify the root cause.
- After the code fix, GC CR time came down to normal range.

Agenda

- Issue: High Kernel mode CPU usage
- Issue: High Shared pool latch contention and ORA-4031
- Issue: Hung RAC cluster
- Issue: High Kernel mode CPU usage – 2
- Issue: Excessive updates and RAC
- Issue: ASMM (Skipped due to time constraints)

Problem

- Problem: Application was very slow. Database was recently upgraded to 10g. Management was quite unhappy and blamed it on 10g.
- Even rollback to 9i was on the table as an option. Following report is for 3 minutes duration.

Top 5 Timed Events ~~~~~ Event	waits	Time (s)	Avg wait (ms)	%Total Call Time
db file sequential read	96,535	704	7	30.1
SGA: allocation forcing component growth	50,809	498	10	21.3
library cache pin	180	219	1218	9.4
latch: shared pool	2,767	217	78	9.3
log file switch completion	225	216	958	9.2

Statspack report

- From top 5 timed events, high amount of SGA activity, waits for library cache pin and latch contention etc.
- Application opens new connections if it “detects” that SQL is hung.

Top 5 Timed Events ~~~~~ Event	waits	Time (s)	Avg wait (ms)	%Total Call Time
db file sequential read	96,535	704	7	30.1
SGA: allocation forcing component growth	50,809	498	10	21.3
library cache pin	180	219	1218	9.4
latch: shared pool	2,767	217	78	9.3
log file switch completion	225	216	958	9.2

SGA resize ?

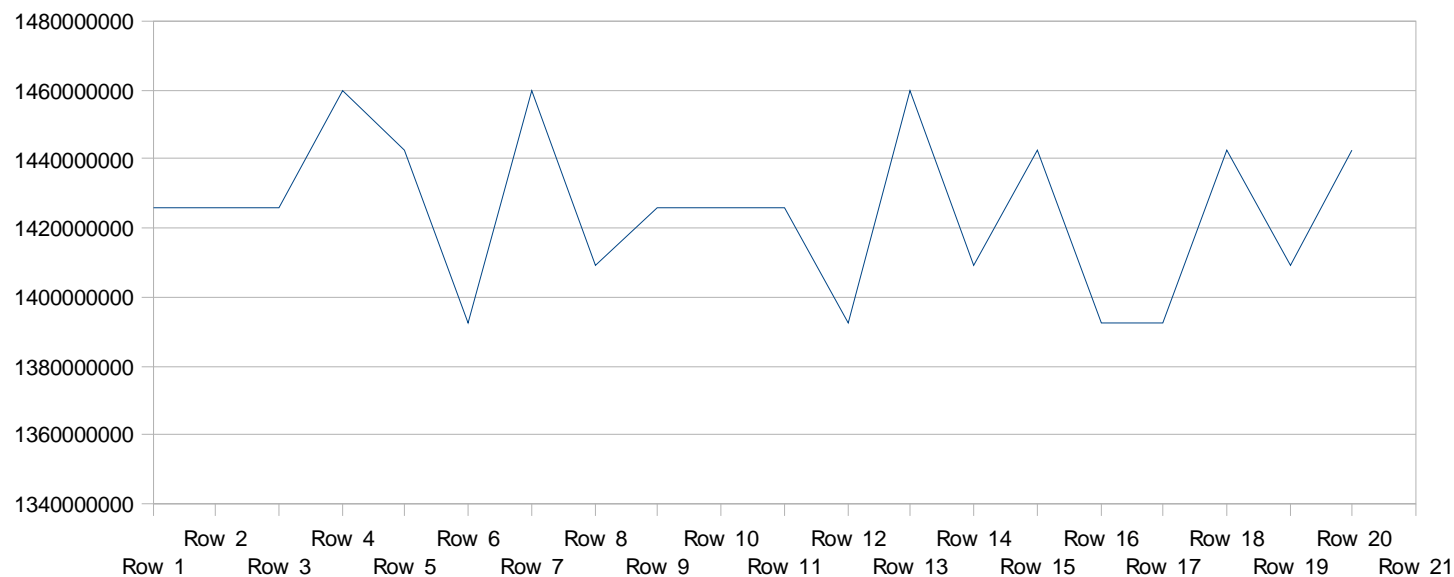
- 'SGA: allocation forcing component growth' gives a clue that there is SGA re-alignment occurring.
- Looking at SGA area section of statspack:
Buffer cache increased by 32 MB and
Shared pool decreased by 32 MB

Snap Id	Cache	Prior Size (MB)	New Size (MB)	Difference (MB) ↑
181	Buffer Cache	1,376	1,344	-32
	Shared Pool	288	320	32

Buffer_cache

- Plotting buffer_cache size from statspack table shows that buffer cache is underwent constant reorganization.

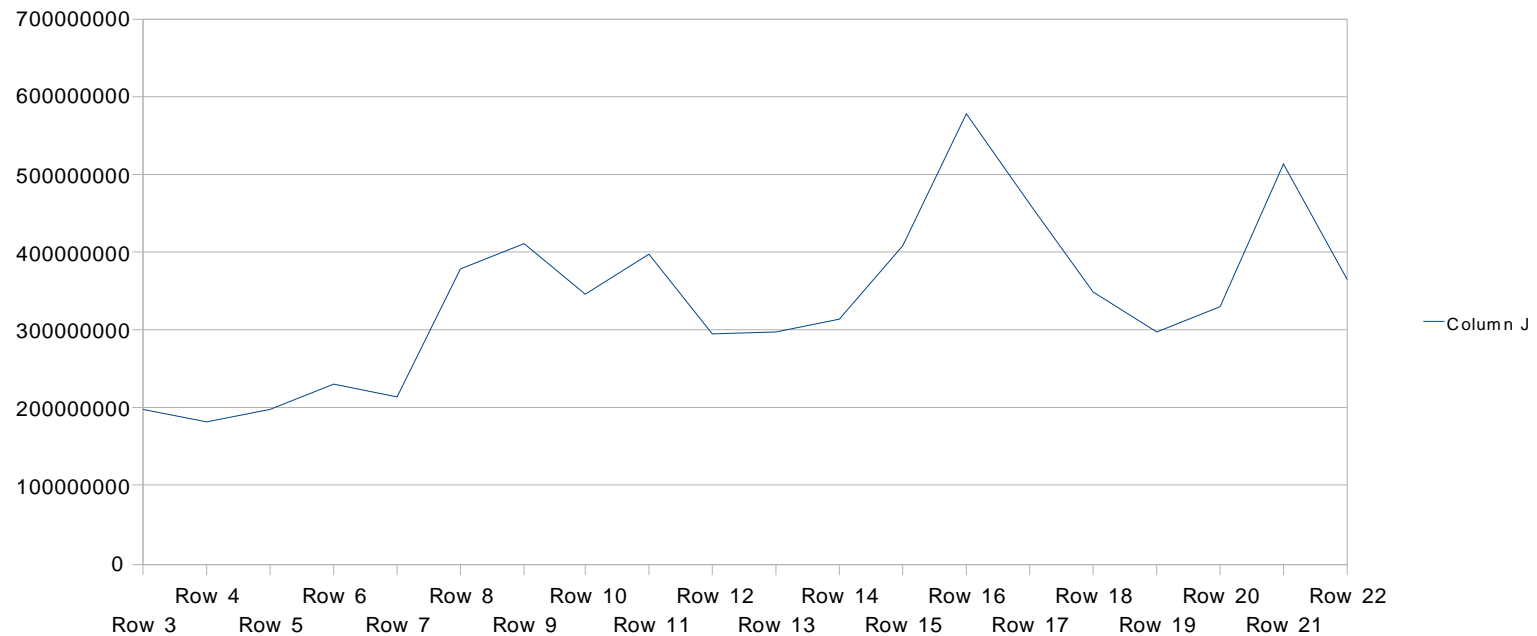
```
select * from perfstat.stats$sgastat where name='buffer_cache'  
order by snap_id;
```



Shared_pool

- Plotting heaps with KGH:NO ACCESS tag shows turbulent reorganization.

```
select * from perfstat.stats$sgastat where name='KGH: NO ACCESS'  
order by snap_id
```



Review

- ASMM algorithms were detecting need for more buffer cache memory.
- This forced de-allocation from shared pool to buffer cache.
- This created more library cache latching and parsing issue. So, algorithm detected need for more shared pool memory.
- Algorithm de-allocated memory from buffer cache and allocated to shared pool, inducing artificial disk reads.
- Of course, this vicious cycle was continuous and causing performance issues.

ASMM

- In summary, constant reorganization of SGA areas caused this issue.
- DBA has setup `sga_target` and commented out all other memory parameters.
- Memory was allocated and deallocated constantly from shared pool. `V$sga_resize_ops` showing constant reorganization of memory areas.

Solution

- There are many ways this can be resolved.
 - Disabling ASMM completely.
 - Providing a minimum size for all SGA components and leaving little bit for ASMM for automatic resize.
 - Increasing the underscore parameter `_memory_broker_stat_interval` to higher value like 3600 or higher vlue.
 - or applying few patches

References

- Oracle support site. Metalink.oracle.com. Various documents
- Internal's guru Steve Adam's website

www.ixora.com.au

- Jonathan Lewis' website

www.jlcomp.daemon.co.uk

- Julian Dyke's website

www.julian-dyke.com

- 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'
by Steve Adams

- Tom Kyte's website

Asktom.oracle.com

- Blog: <http://orainternals.wordpress.com>