
Advanced RAC troubleshooting

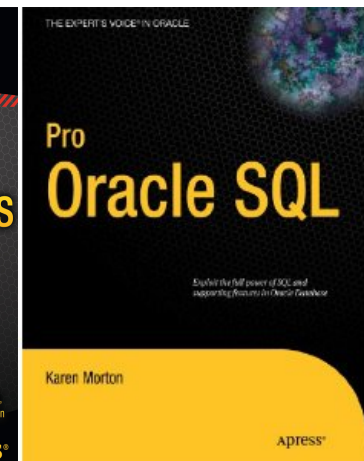
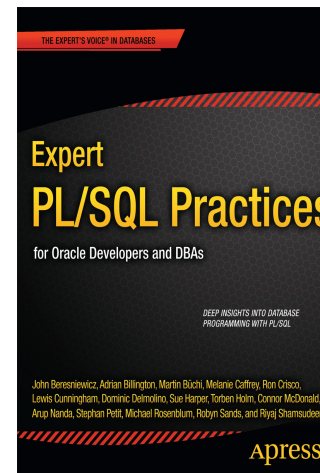
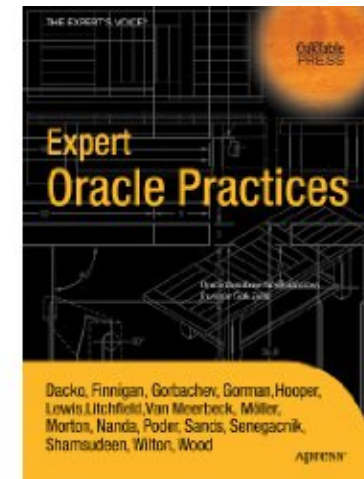
By
Riyaj Shamsudeen



Who am I?



- 18 years using Oracle products/DBA
- OakTable member
- Oracle ACE
- Certified DBA versions 7.0,7.3,8,8i,9i &10g
- Specializes in RAC, performance tuning, Internals and E-business suite
- Chief DBA with OraInternals
- Email: rshamsud@orainternals.com
- Blog : orainternals.wordpress.com
- URL: www.orainternals.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 be reproduced or used without the authors' written permission.

Concepts

Cache coherency

- There are multiple buffer caches in an instance and Oracle RAC uses shared everything architecture.
- Cache coherency is the method by which consistency of the database is maintained.
- Only one instance can hold a block in exclusive current mode and a block can be modified only if the block is held in exclusive current mode.
- There can be two pending transactions modifying the same block, but a block can only be held in exclusive mode in an instance.

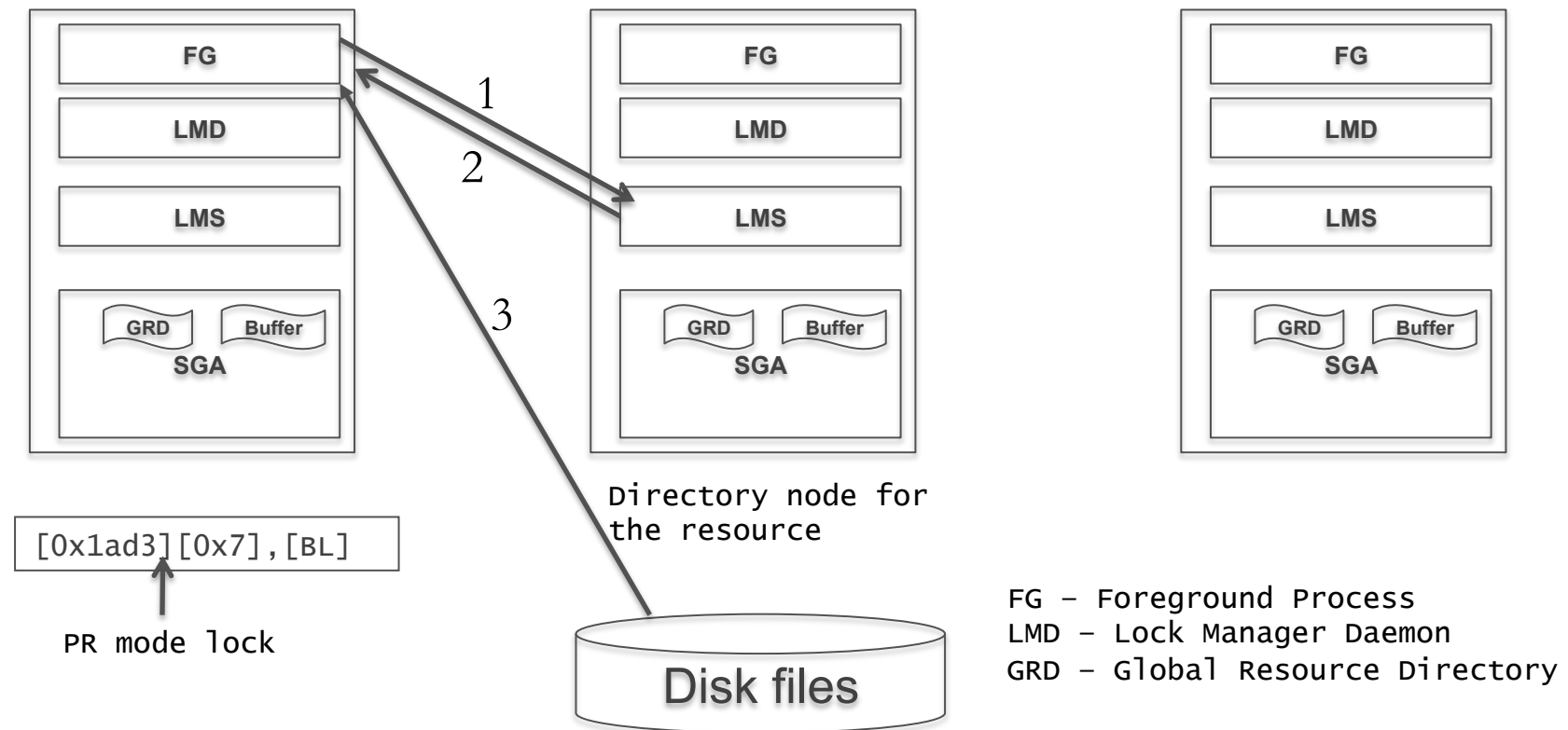
Single block read

- If the buffer is not in the Local buffer cache, process identifies the master node of that block.
- Then the process sends a request to a LMS process running in the master node over the interconnect.
- While sending the request, it is not known whether the block is in any instance buffer cache.
- Until LMS responds, User process waits for a place-holder wait event such as gc cr read, gc current read etc.
- Time is accounted to appropriate events after the response is received from the LMS process.

Demo: demo_01a.sql

Single block read

- Block is not in any buffer cache. LMS grants a PR mode lock on the resource and asks FG to read from the disk.



Trace lines

- Following trace lines shows that session waited for a 2-way grant, followed by a disk read.

```
WAIT #18446741324875049632: nam='gc cr grant 2-way' ela= 499 p1=7 p2=6867 p3=1  
obj#=76484 tim=4597940025
```

```
WAIT #18446741324875049632: nam='db file sequential read' ela= 758 file#=7 block#=6867  
blocks=1 obj#=76484 tim=4597941129
```

- Lock mode of PR (Protected Read) granted to the instance before reading the block from the disk.

KJBLNAME	KJBLNAME2	KJBLGRANT	KJBLROLE	KJBLREQUES
[0x1ad3][0x7],[BL][e 6867,7,BL xt 0x0,0x0		KJUSERPR		0 KJUSERNL

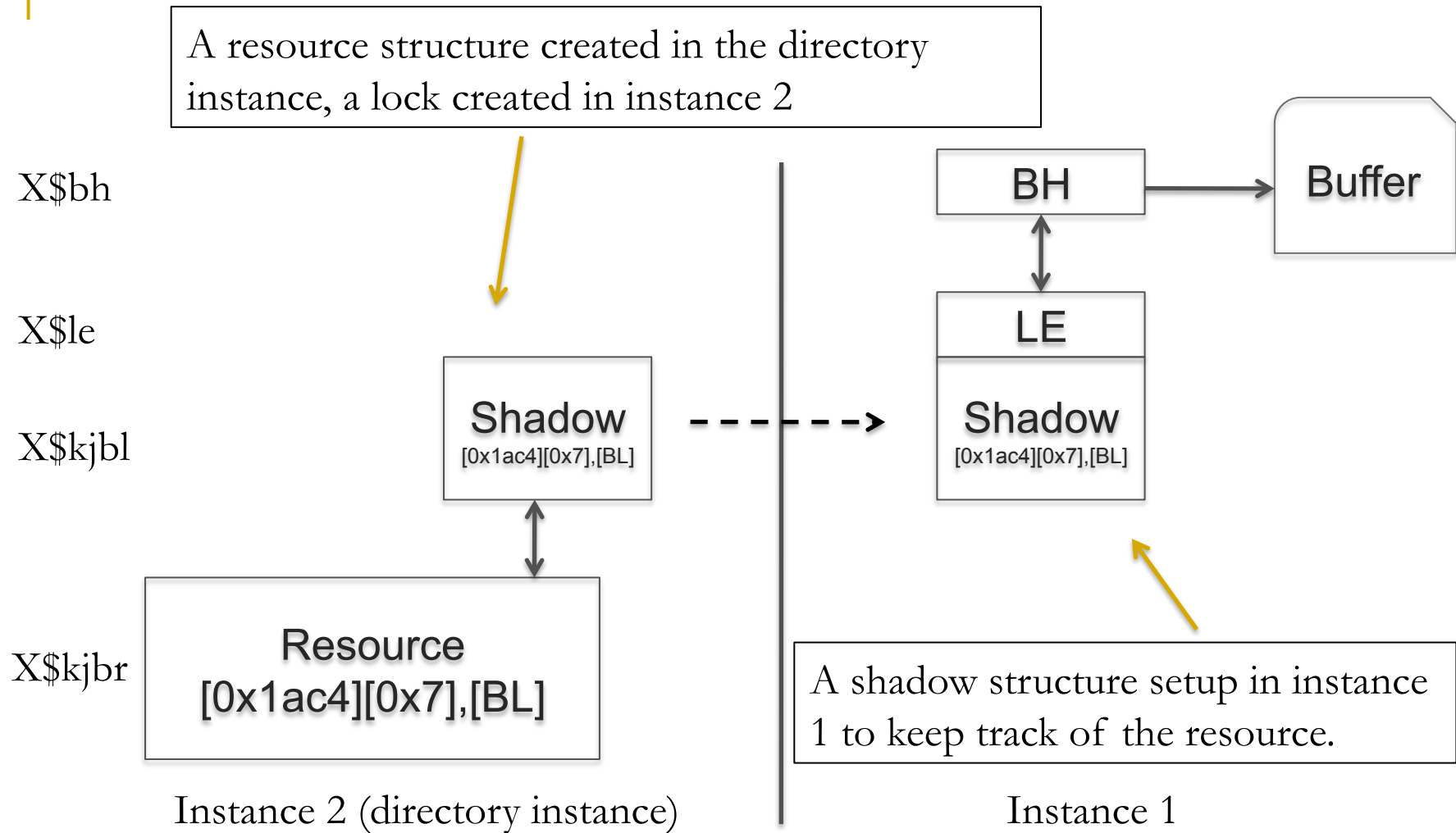
Single block transfer

- If the buffer is in the remote instance in a compatible mode, LMS process grants a lock.
- Remote LMS process transfers the block to the foreground process.
- Foreground process copies the buffer to the buffer cache.
- Instances with that block may acquire lock on that block (CR block transfer does not GRD updates).
- You can see gc events, but no disk events following the gc events.

```
WAIT #18446741324875056000: nam='gc current block 2-way' ela= 1453 p1=7 p2=6852  
p3=1 obj#=76483 tim=6688296584  
FETCH #18446741324875056000:
```

Demo: demo_01a.sql and demo_01b.sql

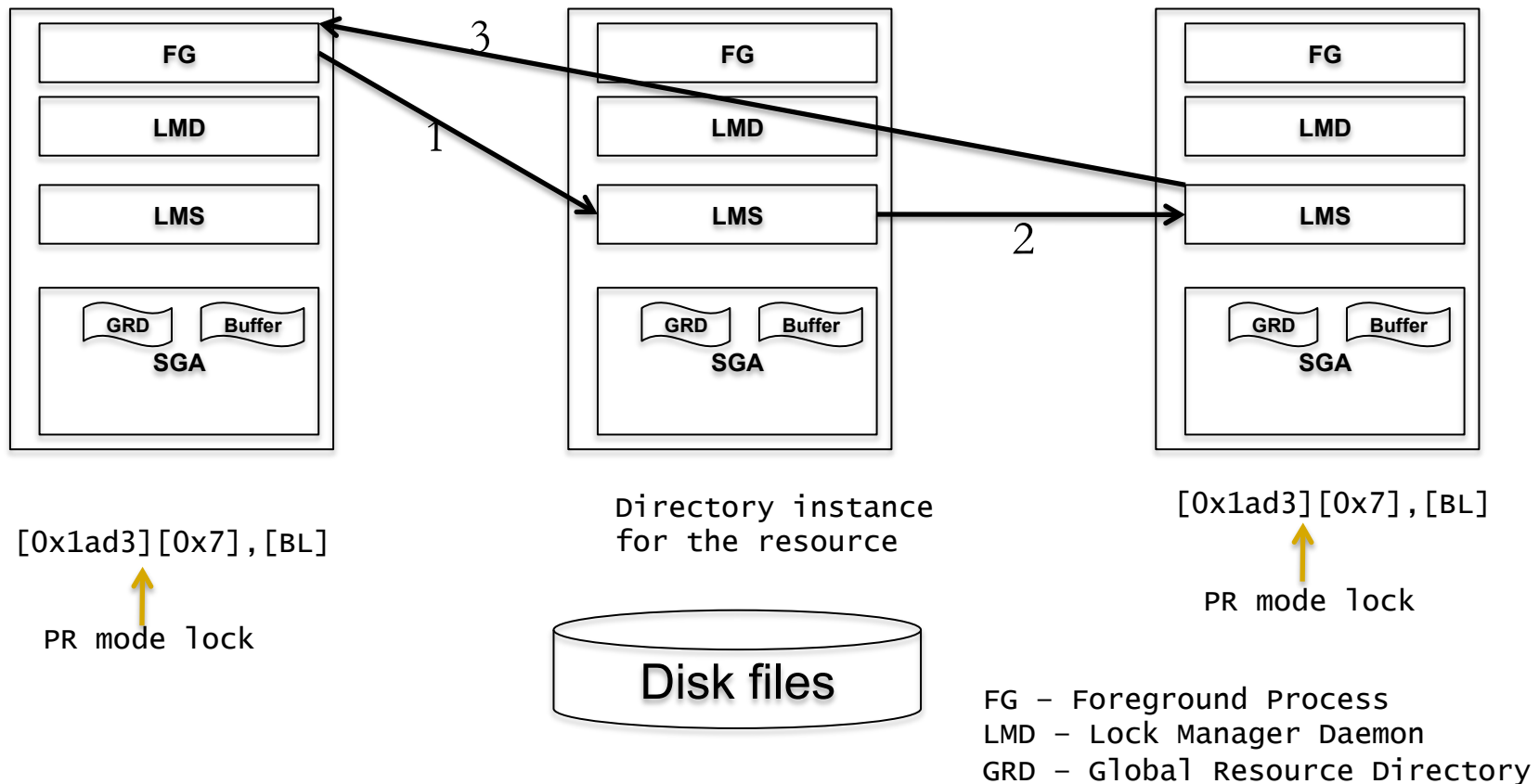
GCS structures



Demo: demo_01a.sql and demo_01b.sql

Single block transfer -3 way

- Block is in the buffer cache of instance 3. Instance 2 is the directory instance of the resource. LMS process transfers the blocks from instance 3 over the interconnect.



GRD

- After the transfer, GRD is updated with ownership changes. Both instances are the owner of the block.

KJBLNAME	KJBLNAME2	KJBLGRANT	KJBLROLE	KJBLREQUES
[0x1ad3][0x7], [BL] [e 6867, 7, BL xt 0x0, 0x0		KJUSERPR	0	KJUSERNL

- If the block is transferred from one instance to another instance in PR mode, then the block mode is considered current mode transfer.
- Subsequently, 'gc current blocks received' statistics incremented.

Buffer changes

- Before modifying a buffer, BL lock on that buffer must be acquired in Exclusive mode (EX).
- Other instances will downgrade or flush that buffer from their instance, if that buffer is already in their cache.
- Instance acquired the block in EX mode and other instance(s) flushed the buffer.

```
KJBLNAME                KJBLNAME2                KJBLGRANT  KJBLROLE  KJBLREQUES
-----
[0x1ac4][0x7],[BL][ext 0x0,0x0 6852,7,BL  KJUSEREX                0 KJUSERNL
```

Enter value for block: 6852

```
STATE  MODE_HELD  LE_ADDR                DBARFIL  DBABLK  CR_SCN_BAS  CR_SCN_WRP  CLASS
-----
1      0  000000006D3E3AB0      7        6852      0          0          1
```

Demo: demo_02a.sql ,demo_02b.sql demo_02c.sql

Busy

- Event gc cr block busy, gc current block busy indicates that those blocks were “busy”.
- In this case, that block was in EX mode in another instance and LMS process applied undo blocks to reconstruct a consistent mode buffer reconstructing a CR mode buffer.
- Excessive *busy events would indicate application affinity is not in play.
- Application affinity will reduce *busy events as the buffers will be modified in the same instance.

Gcs log flush sync

- But, if the instances crash right after the block is transferred to other node, how does RAC maintain consistency?
- Actually, before sending a current mode block LMS process will request LGWR for a log flush.
- Until LGWR sends a signal back to LMS process, LMS process will wait on 'gcs log flush' event.
- CR block transfer might need log flush if the block was considered "busy".
- One of the busy condition is that if the block was constructed by applying undo records.

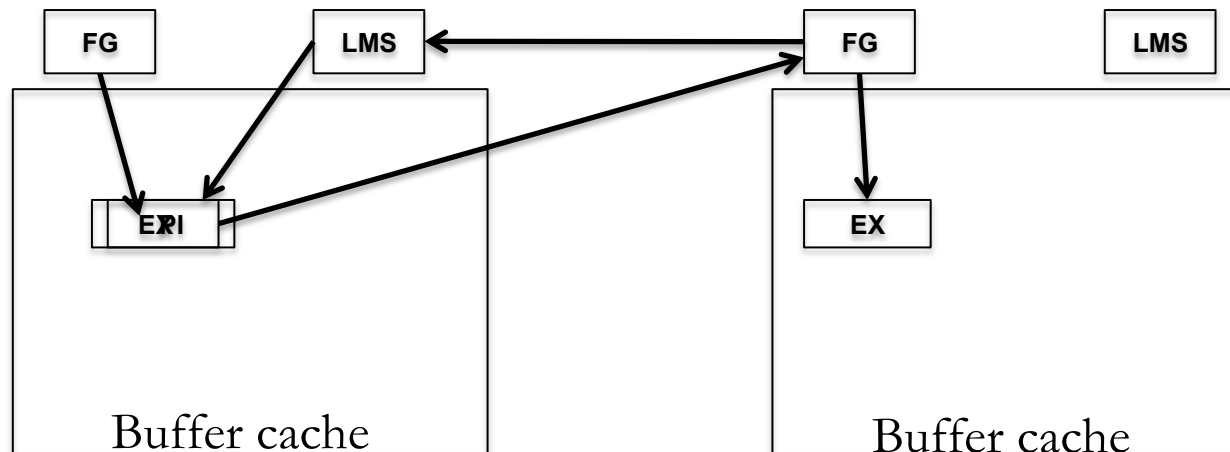
CUR mode

- What happens if two instances modify same block, but different rows?
- Row level locks prevent the same row being updated from two different instances.
- Before an instance can modify a block, the instance must acquire EX mode lock on the buffer.
- No two instances can hold the block in EX mode and a compatible buffer state.

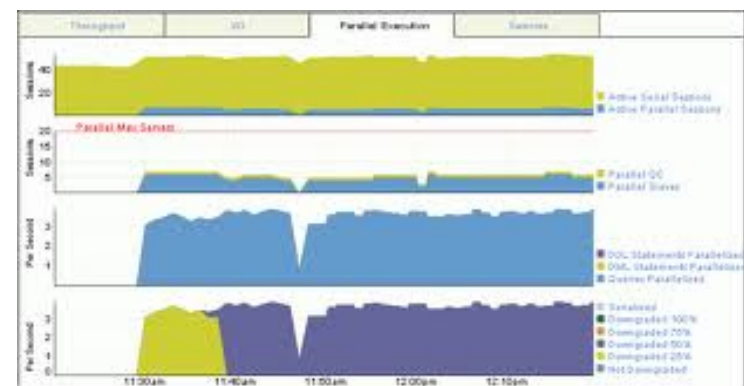
Demo: demo_04a.sql ,demo_04b.sql

CUR mode

- What happens if there are two pending transactions from two different instances in the same block? No two instances are allowed to hold XCUR mode buffers with EX mode GCS lock concurrently.



RAC Wait Events



Types of packets

- Block oriented packets
 - Consistent Read blocks
 - Current Read blocks
- Message oriented packets
 - Single block grants
 - Multi block grants
- Service oriented packets
 - SCN generation
 - Row cache updates
 - GES layer packets

CR Wait events

- Following are the top wait events associated with CR mode transfers:

gc cr block 2-way

gc cr block 3-way

gc cr multi block request

gc cr block busy

gc buffer busy (acquire/release)

gc cr grant 2-way

gc cr grant congested

gc cr block congested

Transfers without
congestion or concurrency.

Multi block read

Concurrency related

Grants

Congestion related

Gc cr block 2/3-way

- Time is accounted for 'gc cr block 2-way' if the block owner and master is an instance.
- If the owner and master instance are different than 3-way wait events are used.
- Time is accounted to these wait events if there was no need for additional work such as CR block creation or contention.

nam='gc cr block 2-way' e1a= 627 p1=7 p2=6852 p3=1 obj#=76483 tim=37221074057

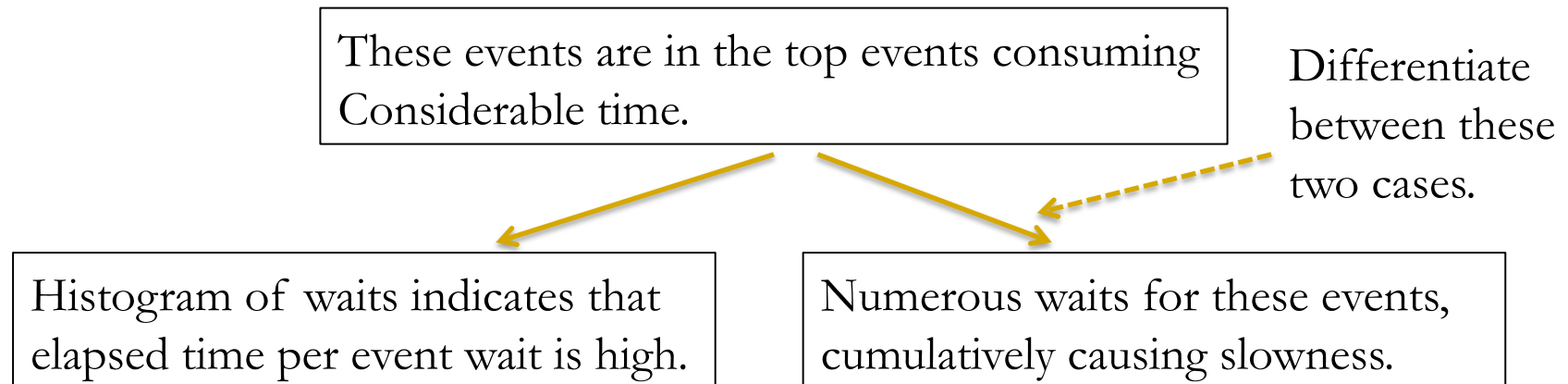
Db_objects.object_id or
data_object_id

Demo: demo_gc_cr_2wayb.sql, demo_gc_cr_2waya.sql

©OraInternals Riyaj Shamsudeen

Analysis

- These two events 'gc cr block 2-way' and 'gc cr block 3-way' can be considered as baseline events to calibrate cache fusion performance.



- Generally, concurrency or congestion issues are not factored in to these events.

Case 1 Average wait time is higher

- If the time_waited histogram indicated for this event is higher, it could be due to:
 - High CPU usage in the nodes, leading to processes not getting CPU quick enough.
 - Network performance or Network configuration issue.
 - Platform issues as SMP scaling or NUMA related.
- Since concurrency or congestion related waits are not factored in to these waits, these are good baseline indicators for cache fusion performance.

Diagnostics

- Review the histogram for this event using event_histogram.sql script.

41% of waits took between 2-4ms in this example below.

INST_ID	EVENT	WAIT_TIME_MILLI	WAIT_COUNT	PER
1	gc cr block 2-way	1	3720856	1.11
1	gc cr block 2-way	2	148016413	44.25
1	gc cr block 2-way	4	140006974	41.86
1	gc cr block 2-way	8	40140870	12
1	gc cr block 2-way	16	2491886	.74
1	gc cr block 2-way	32	43253	.01
...				
1	gc cr block 2-way	8192	9	0
1	gc cr block 2-way	16384	24	0

Demo: event_histogram.sql

Recommendations

- Keep CPU usage below 80-85%. Above 80% CPU usage, scheduling inefficiency kicks in and multiplies the cache fusion performance issues.
- Possibly consider jumbo frames. Jumbo frames reduces assembly and disassembly of packets, so will reduce CPU usage slightly.
- Review network performance using OS tools.
- Review if cache fusion traffic is using private interconnect.
- Review if the cache fusion traffic is mixed with other network traffic.

Case 2: Numerous waits for these two events

- If there are numerous waits for this wait event, identify the object and SQL causing these waits.
- SQL Trace or ASH data can be used to identify the object associated with these wait events.
- ASH data is a sampled data, so caution should be taken so that big enough samples are used.
- Object_id from the SQLTrace file can be used to identify the objects too.

Diagnostics

- Top objects leading to these waits are printed below.

```
@ash_gcwait_to_obj.sql
```

```
Enter value for event_name: gc cr block 2-way
```

INST_ID	OWNER	OBJECT_NAME	OBJECT_TYPE	CNT
...				
1	APPLSYS	FND_CONCURRENT_PROCESSES	TABLE	118
1	INV	MTL_SERIAL_NUMBERS	TABLE	144
1	INV	MTL_TRANSACTIONS_INTERFACE_N1	INDEX	176
1	APPLSYS	FND_CONCURRENT_REQUESTS	TABLE	184
1	INV	MTL_MATERIAL_TRANSACTIONS	TABLE	211
1	INV	MTL_TRANSACTIONS_INTERFACE	TABLE	216
1			Undo Header/Undo block?	18483

For undo header blocks/undo blocks, current_obj# is set to 0 and for undo blocks, current_obj# is set to -1.

Demo: ash_gcwait_to_obj.sql

Recommendations

- Consider application affinity. Huge number of blocks transferred back and forth between the instances are indicating that application affinity might help.
- SGA size might be smaller for the workload. Try to see if increasing SGA size is an option.
- Stretch clusters will suffer from longer latencies due to network latency between the end points.

Gc cr block congested/gc cr grants congested

- These wait events indicate that there were CPU resource starvation issues.
- For example, sudden spikes in PQ processing can increase CPU load average leading to CPU starvation.
- Reducing CPU usage by tuning costly SQL statement, scheduling jobs to run different times, or even adding new nodes is generally required.
- In a really busy and active environments, there will be few of these wait events; These events are concerns only if the AWR or SQLTrace indicates high amount of wait times for these events.

Gc cr grants 2-way

- Time is accounted to this wait event, if the block is not in any of the buffer cache.
- Trace file will indicate this wait event followed by a disk read.

```
nam='gc cr grant 2-way' e1a= 659 p1=1 p2=88252 p3=1 obj#=77779
```

```
nam='db file sequential read' e1a= 938 file#=1 block#=88252 blocks=1  
obj#=77779
```

- Typical latency is 1-2ms. Any thing above needs to be reviewed as these are light-wait events.
- Process sends a request to remote master LMS process and the LMS process simply responds with 'read from disk'.
- This is another base line wait event to measure interconnect response time, as LMS processing is limited.

RAC-Tuning objects

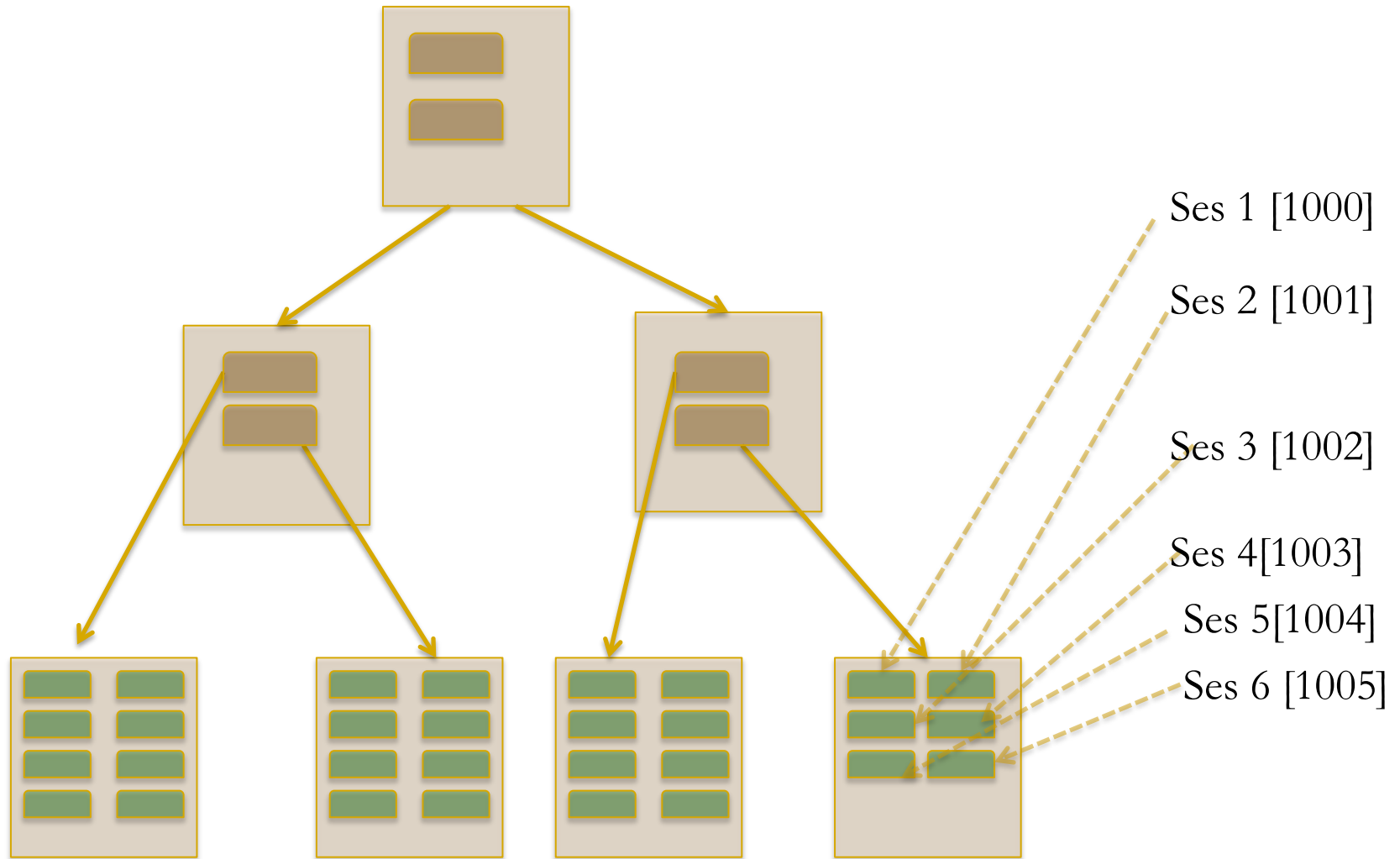
Partitioning

- Partitioning can be used to improve performance and scalability in RAC instances. Few Guidelines:
 - Range Partitioning : For physical segment segregation
 - Hash Partitioning Index : To improve insert concurrency
 - Hash Partitioning Index : To reduce GC traffic for Select
 - Hash Partitioning Table with local indexes: To improve insert concurrency
 - Hash Partitioning Table : To reduce GC traffic for Select
- From Version 10g onwards, partitioned indexes can be created on non-partitioned tables.

Right hand growth index contention

- Btree indexes store ordered (key, rowid) pair.
- If the key column values are generated using a sequence value or monotonically increasing values, then those values are stored in the right most leaf block of the index.
- If many sessions are concurrently inserting into the index, all those sessions will be trying to insert in to right most leaf block of the index.
- This leads to contention in right most leaf block and known as right hand index growth contention.

Non-partitioned indexes



In RAC...

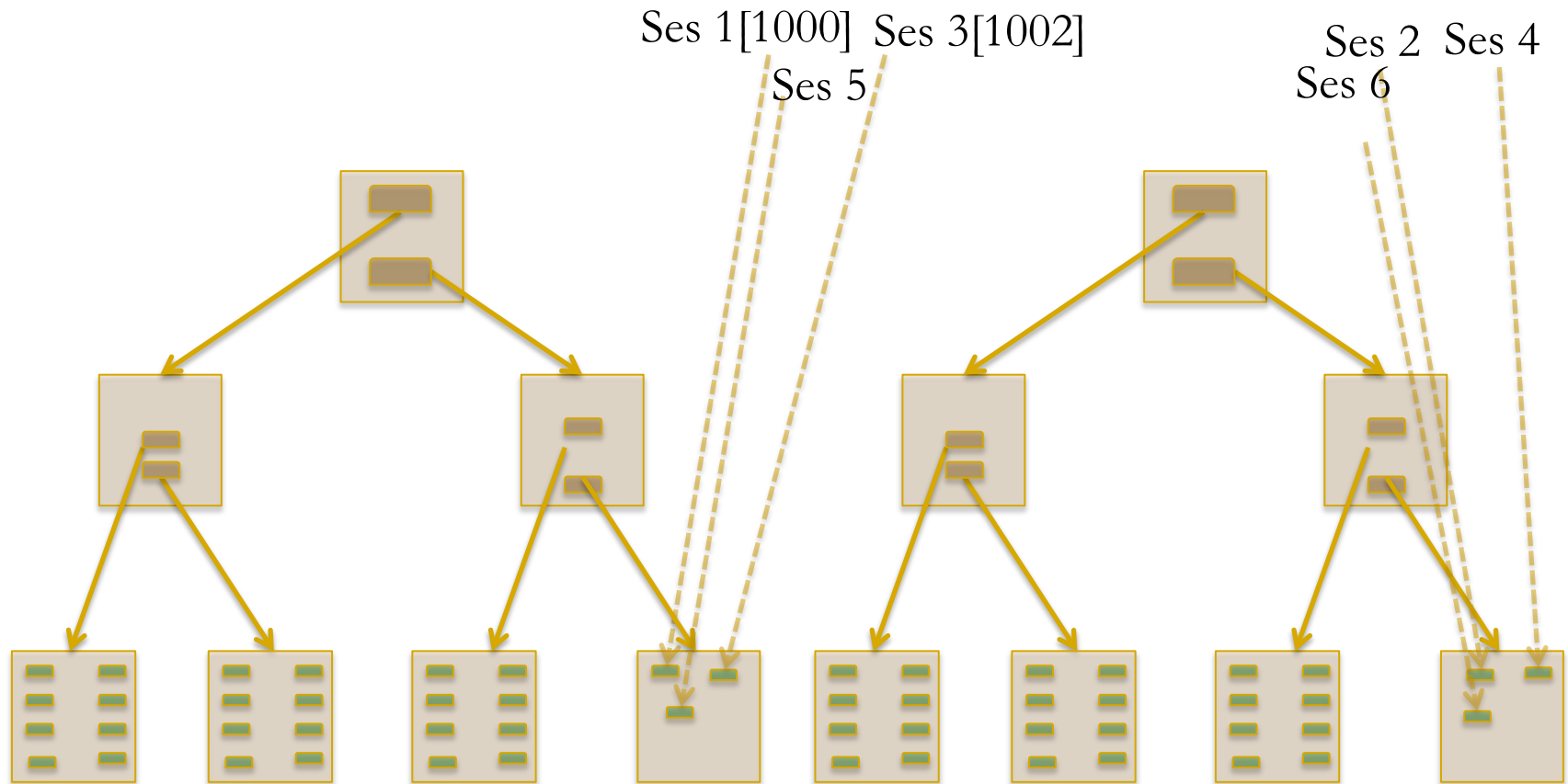
- Right hand growth indexes will suffer from buffer busy waits in single instance.
- In RAC, this problem is magnified with enormous waits on gc buffer busy events and other downstream events.

INST_ID	SQL_ID	EVENT	STATE	COUNT(*)
4	4jtbgawt37mcd	gc cr request	WAITING	9
3	4jtbgawt37mcd	gc cr request	WAITING	9
3	a1bp5ytvpfj48	gc buffer busy	WAITING	11
4	a1bp5ytvpfj48	gc buffer busy	WAITING	17
4	14t0wadn1t0us	gc buffer busy	WAITING	33
4	gt1rdqk2ub851	gc buffer busy	WAITING	34
4	a1bp5ytvpfj48	buffer busy waits	WAITING	35
2	a1bp5ytvpfj48	gc buffer busy	WAITING	65
1	a1bp5ytvpfj48	gc buffer busy	WAITING	102
2	7xzqcrdrnyw1j	gc buffer busy	WAITING	106
2	7xzqcrdrnyw1j	enq: TX - index c	WAITING	173
1	7xzqcrdrnyw1j	gc buffer busy	WAITING	198
3	7xzqcrdrnyw1j	gc buffer busy	WAITING	247
4	7xzqcrdrnyw1j	gc buffer busy	WAITING	247

Hash partitioning

- Hash partitioning is an option to resolve concurrency issues associated with right hand growth indexes.
- Hashing algorithm uniformly distributes values to various leaf blocks leading to increased concurrency.
- For example, by converting an unique non-partitioned index to a partitioned index with 2 partition, concurrency can be doubled.
- Conversion to 32 partition index will lead to a concurrency increase of near 32 fold.
- This action may be needed for even non-unique indexes if the data is almost unique, such as timestamp column.

Hash Partitioned indexes



Partition count

- Keep partition count to be a binary power of 2 as hash partitioning algorithm uses hashing algorithm.
- Err on caution: Use bigger number of partitions such as 32, 64, or 128 partitions if the concurrency is higher.
- Of course, this might induce more logical reads, but the effect of that increase is negligible.

Demo: generate_insert.ksh, generate_insert_setup.sql, generate_insert_setup_hash.sql

Hash partitioning tables

- Another option to resolve right-hand-growth index contention is to convert the table to partitioned table and create the indexes as a local indexes.
- Since the algorithm uses hashing techniques, keep the partition count as binary power of 2.
- Hash partitioning indexes or table is a proven way to scale the application concurrency in RAC.
- Index Organized Tables also can suffer from this insert concurrency, if the row is short.

ASSM

- ASSM avoids the need for manual freelist management.
- It is out of scope to go deeper in to ASSM, but in ASSM, L1 bitmaps are keeping the list of free blocks for insert.
- L2 bitmaps points to L1 bitmaps and L3 bitmaps in turn points to L2 bitmaps. L3 bitmaps is not common though.
- L1 bitmaps are searched to find free blocks.

Demo: generate_ins_freelist.ksh 1 10, freelist_blocks.sql

ASSM & RAC

- Each instance assumes ownership of few L1 bitmaps. Processes in that instance search L1 bitmaps owned by that instance.
- Essentially, ASSM avoids the need for freelist groups, by instance-owning L1 bitmaps and second/third level indirect bitmaps.
- `Dbms_space_admin` package can be used to dump segment header information in ASSM tablespace.

```
dbms_space_admin.segment_dump(  
    c1.tablespace_name,  
    c1.relative_fno,  
    c1.header_block);
```

L1 bitmaps

- When Instance 1 processes were inserting in to the segment, all L1 bitmaps were owned by that instance.
- When both instances were inserting in to the segment, new L1 bitmaps were owned by the second instance.

L1 Ranges :

0x01c02470	Free: 1	Inst: 1
0x01000b08	Free: 1	Inst: 1
0x01000b28	Free: 1	Inst: 1
0x01000b48	Free: 1	Inst: 1
0x01000b60	Free: 1	Inst: 1
0x01000b78	Free: 1	Inst: 1
0x01c02588	Free: 1	Inst: 1
0x01c025b0	Free: 1	Inst: 1
0x01000b80	Free: 1	Inst: 1
0x01000b81	Free: 5	Inst: 1

L1 Ranges :

0x01c02470	Free: 1	Inst: 1
0x01000b08	Free: 1	Inst: 1
0x01000b28	Free: 1	Inst: 1
0x01000b48	Free: 1	Inst: 1
0x01000b60	Free: 1	Inst: 1
0x01000b78	Free: 1	Inst: 1
0x01c02588	Free: 1	Inst: 1
0x01c025b0	Free: 1	Inst: 1
0x01000b80	Free: 1	Inst: 1
0x01000b81	Free: 1	Inst: 2
0x01c02600	Free: 1	Inst: 2
0x01c02601	Free: 1	Inst: 2
0x01000c80	Free: 5	Inst: 2
0x01000c81	Free: 5	Inst: 2

Instance ownership

- L1 bitmaps can change ownership too.

Dump of First Level Bitmap Block

```
-----  
nbits : 4 nranges: 1          parent dba: 0x01c02471  poffset: 9  
unformatted: 0          total: 64          first useful block: 0  
owning instance : 2  
instance ownership changed at 01/28/2011 22:43:06  
Last successful search 01/28/2011 22:43:06  
Freeness Status:  nf1 0          nf2 0          nf3 0          nf4 0
```

- Excessive deletes doesn't lead to ill effects similar to freelist blocks. Free blocks are correctly accounted to L1 bitmaps and instance ownership maintained.
- In a nutshell, consider using ASSM in RAC environment.

Sequences

- Incorrect configuration of sequences can be fatal to performance.
 - For cached sequences, each instance caches a range of sequence values.
 - Problem is that sessions running from different nodes can get non-sequential values.
 - For example, if the cache is 20, then session #1 in instance 1 will get a value of 1 and session #2 in instance 2 will retrieve a value starting at 21.
 - In normal operations, there is no loss of values, just possible gaps.
-

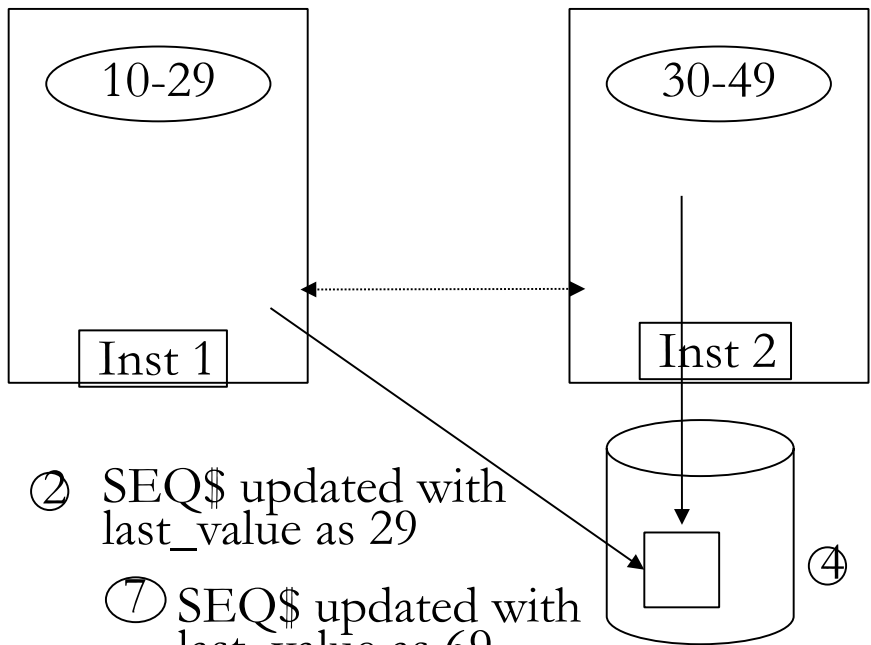
Sequence operation in RAC

emp_seq
cache 20
start with 10

⑥ After 29, values will be in 50-69 range.

⑤ Subsequent accesses return values until value reaches 29

① First access to sequence caches values from 10 to 29 ③ Second access caches value from 30-49



1. 60 access to sequence results in 3 changes to block.
2. These changes might not result in physical reads/writes.
3. Gaps in sequence values.
4. Still, log flush needed for cache transfer.

Sequence operation in RAC

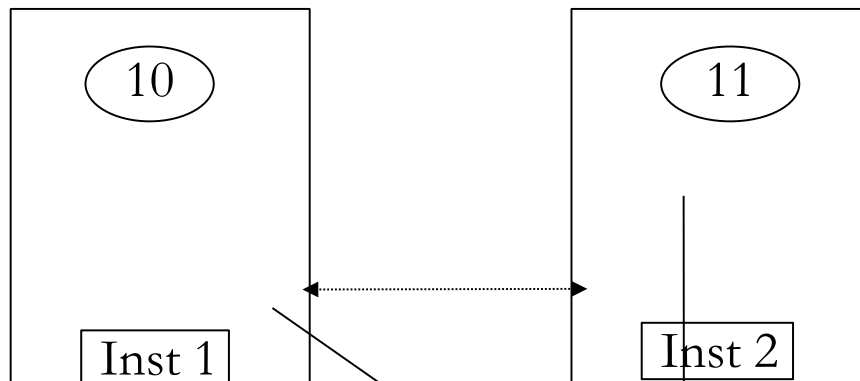
emp_seq
nocache
start with 10

⑥ Due to nocache values,
there will be no gaps.

⑤ Subsequent accesses returns
value 12

① First access to sequence
returns value 10

③ Second access returns
value of 11



② SEQ\$ updated with
last_value as 10

⑦ SEQ\$ updated with
last_value as 12

④ SEQ\$ updated with
last_value as 11

1. 3 access to sequence
results in 3 block changes.
2. No gaps in sequence
values.
3. But, SEQ\$ table blocks
transferred back and forth.

Sequences


- If ordered values are needed, consider “order, cache” attributes.
- “Order, cache” attribute provides better performance since the GES layer is used to maintain the order between the instances.
- Still, with “order, cache” it is possible to lose the values in case of instance crashes.
- You should consider order, nocache only for lightly used sequences such as control table sequences etc.

Code executions – two nodes (order,cache)

```
INSERT INTO RS.T_GEN_SEQ_02
VALUES
( RS.T_GEN_SEQ_02_SEQ.NEXTVAL, LPAD ('Gen',25,'DEADBEEF')
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.01	0	0	0	0
Execute	5001	0.94	12.60	0	910	16440	5001
Fetch	0	0.00	0.00	0	0	0	0
total	5002	0.94	12.62	0	910	16440	5001

Event waited on	Times waited	Max. Wait	Total waited
DFS lock handle	359	0.05	0.64
enq: HW - contention	6	0.03	0.09
buffer busy waits	130	0.06	0.50



- “Order, cache attribute is implemented using GES layer and interconnect.

Code executions – two nodes (cache)

```
INSERT INTO RS.T_GEN_SEQ_02
```

```
VALUES
```

```
( RS.T_GEN_SEQ_02_SEQ.NEXTVAL, LPAD ('Gen',25,'DEADBEEF')
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	5001	7.71	282.75	3	333	20670	5001
Fetch	0	0.00	0.00	0	0	0	0
total	5002	7.71	282.75	3	333	20670	5001

Event waited on	Times Waited	Max. wait	Total waited
row cache lock	4586	0.76	255.01
Disk file operations I/O	7	0.00	0.00
db file sequential read	3	0.01	0.03
gc current block busy	1064	0.46	7.08
gc current block 2-way	2660	0.05	3.36

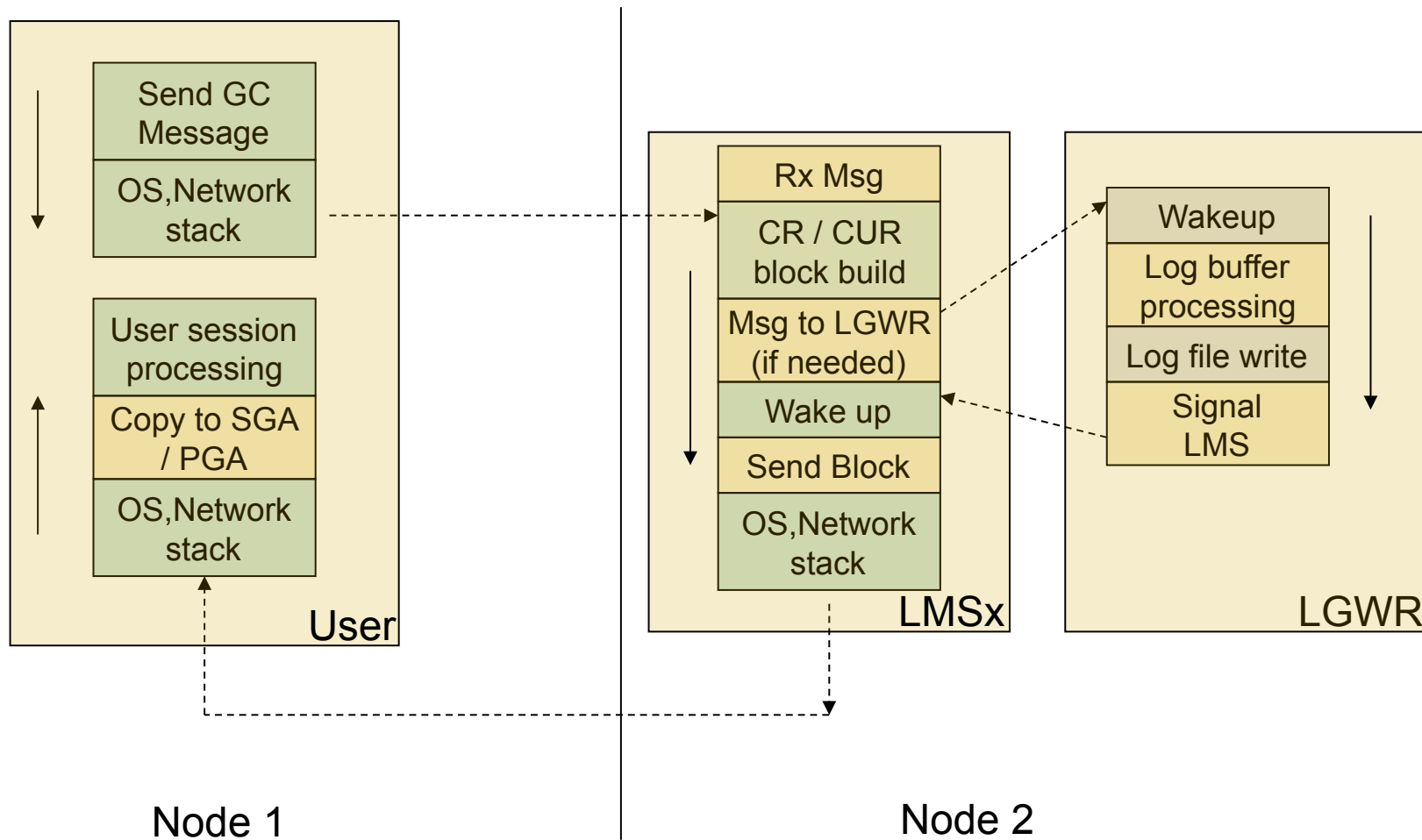
- Order,nocache causes excessive row cache lock waits.

Break it down..



Coutesy:npowersoftware.com

LMS Processing (over simplified)



GC CR latency

- GC CR latency \approx

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

Averages can be misleading. Always review both total time and average to understand the issue.

Breakdown latency

In this case, LGWR flush time
Need to be reduced to tune latency.

Avg global cache cr block receive time (ms): 6.2

Wait time	Node 1	Node 2	Node 3	Node 4	Total
gc cr block build time	402	199	100	227	1679
Gc cr block flush time	3016	870	978	2247	7111
Gc cr block send time	375	188	87	265	1290

GC CURRENT latency

- GC CUR latency $\sim =$

Time spent in sending message to LMS +
LMS processing : (Pin and build block) +
LGWR latency: Log flush +
Wire latency

Statistics : gc current block flush time
gc current block pin time
gc current block send time

GV\$ views

- GV\$ and V\$ views are implemented with an abstraction layer.
- GV\$ views: Fixed views, accessing x\$ tables. For example, gv\$database is accessing x\$kccdi and x\$kccdi2 fixed tables.

```
select di.inst_id,di.didbi,di.didbn,  
to_date(di.dicts,'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),  
to_number(di.dirls) ,  
to_date(di.dirlc,'MM/DD/RR HH24:MI:SS','NLS_CALENDAR=Gregor
```

....

```
f12,64), 64, 'YES', 'NO'),  
decode(di2.di2min_req_capture_scn,0, to_number(null),  
di2.di2min_req_capture_scn) from x$kccdi di, x$kccdi2 di2
```

- GV_\$ views: Traditional views accessing GV\$ fixed views.

```
create or replace view gv_$database as select * from gv$database;
```

- Gv\$database is a public synonym referring to sys.gv_\$ views.

```
create or replace public synonym gv$database for gv_$database;
```

V\$ views

- V\$ views access gv\$ views and filter data specific to current instance. This is true even in a single instance.

Create or replace v\$database as

```
select DBID, NAME, CREATED, RESETLOGS_CHANGE#, RESETLOGS_TIME,  
       PRIOR_RESETLOGS_CHANGE#, PRIOR_RESETLOGS_TIME, LOG_MODE, CHECKPOINT_CHANGE#,  
       ARCHIVE_CHANGE#, CONTROLFILE_TYPE, CONTROLFILE_CREATED, CONTROLFILE_SEQUENCE#,  
       CONTROLFILE_CHANGE#,  
  
...  
from  
GV$DATABASE where inst_id = USERENV('Instance')  
;
```


GV\$ implementation

- GV\$ views retrieves from all instances and merges them to produce final output.
- Specialized parallel query slaves are used to retrieve rows from different instances (10g and above).

Username	INST_ID	QC/Slave	Slave Set	SID	QC SID	Requested DOP	Actual DOP
SYS	1	QC		52	52		
- pz99	2	(Slave)	1	55	52	2	2
- pz99	1	(Slave)	1	62	52	2	2

- In 9i, normal PQ slaves were used to retrieve rows from remote instances.

Demo: gv_pq.sql , pxslaves_global.sql

Caution

- Don't use gv\$views to find averages. Use AWR reports or custom scripts.
- gv\$views are aggregated data and persistent from the instance restart.
- For example this query can be misleading:

```
select b1.inst_id, b2.value "RECEIVED",  
       b1.value "RECEIVE TIME",  
       ((b1.value / b2.value) * 10) "AVG RECEIVE TIME (ms)"  
from gv$sysstat b1, gv$sysstat b2  
where b1.name = 'gc cr block receive time' and  
       b2.name = 'gc cr blocks received' and b1.inst_id = b2.inst_id
```

gc_traffic_print.sql

- You can use my script to print global cache performance data for the past minute. Download from scripts archive:

http://www.orainternals.com/scripts_rac1.php

Inst	CR blocks Rx	CR time	CUR blocks Rx	CUR time	CR blocks Tx	CUR blocks Tx	Tot blocks
1	40999	13.82	7827	4.82	25070	17855	91751
2	12471	5.85	8389	5.28	31269	9772	61901
3	28795	4.11	18065	3.97	28946	4248	80054
4	33105	4.54	12136	4.68	29517	13645	88403

- During the same time frame, output of the script from prior slide:

INST_ID	RECEIVED	RECEIVE TIME	AVG RECEIVE TIME (ms)
4	165602481	104243160	6.2947825
2	123971820	82993393	6.69453695
3	215681074	103170166	4.7834594
1	134814176	66663093	4.9448133

Very misleading!

Review all nodes.

- It is important to review performance data from all the nodes.
- It is easy to create AWR reports from all nodes using my script:
Refer awrrpt_all_gen.sql.
 - [Don't forget that access to AWR report needs license]
- Or use my script gc_traffic_processing.sql from my script archive.

Default collection period is 60 seconds.... Please wait for at least 60 seconds...

Inst	CR blk Tx	CR bld	CR fls tm	CR snd tm	CUR blk TX	CUR pin tm	CUR fls tm	CUR blk TX
2	67061	.08	.88	.23	34909	1.62	.2	.23
3	38207	.17	2.19	.26	28303	.61	.08	.26
4	72820	.06	1.76	.2	40578	1.76	.24	.19
5	84355	.09	2.42	.23	30717	2.69	.44	.25

Place holder events

- Few events are place holder events such as:
 - gc cr request
 - gc cr multiblock request
 - gc current request
 - ...
- Sessions can be seen waiting for these wait events, but will not show up in AWR / ADDM reports.
- After sending the global cache block request, foreground process waits on these events.
- On receipt of the response, time is accounted for correct wait event.

Histogram

89.4% of these waits are Under 4ms.

- Averages can be misleading. Use `v$event_histogram` to understand true performance metrics.
- It is better to take snapshots of this data and compare the differences.

INST_ID	EVENT	WAIT_TIME_MILLI	WAIT_COUNT	THIS_PER	TOTAL_PER
1	gc cr block 2-way	1	466345	.92	.92
1	gc cr block 2-way	2	23863264	47.58	48.51
1	gc cr block 2-way	4	20543430	40.96	89.47
1	gc cr block 2-way	8	4921880	9.81	99.29
1	gc cr block 2-way	16	329769	.65	99.95
1	gc cr block 2-way	32	17267	.03	99.98
1	gc cr block 2-way	64	2876	0	99.99
1	gc cr block 2-way	128	1914	0	99.99
1	gc cr block 2-way	256	1483	0	99.99
1	gc cr block 2-way	512	618	0	99.99
1	gc cr block 2-way	1024	83	0	99.99
1	gc cr block 2-way	2048	4	0	99.99
1	gc cr block 2-way	4096	3	0	99.99
1	gc cr block 2-way	8192	5	0	99.99
1	gc cr block 2-way	16384	3	0	100

GC event histograms

- Better yet, use my script `gc_event_histogram.sql` to understand current performance metrics.

Default collection period is sleep seconds. Please wait..

Enter value for event: gc cr block 2-way

Enter value for sleep: 60

Inst id	Event	wait time milli	wait cnt
1	gc cr block 2-way	1	37
1	gc cr block 2-way	2	4277
1	gc cr block 2-way	4	5074
1	gc cr block 2-way	8	1410
1	gc cr block 2-way	16	89
1	gc cr block 2-way	32	1
1	gc cr block 2-way	64	0
1	gc cr block 2-way	128	0
1	gc cr block 2-way	256	0

Gc buffer busy waits

- GC buffer busy waits are usually symptoms. In many instances, this event can show up the top most waited event.
- GC Buffer busy simply means that buffer is pinned by another process and waiting for a different global cache event.
- Understand why that ‘buffer pin holder’ is waiting. Resolving that will resolve global cache buffer busy waits.
- Segment header changes dues to insufficient freelist groups also can lead to longer ‘gc buffer busy’ waits.

Example analysis

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

## 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

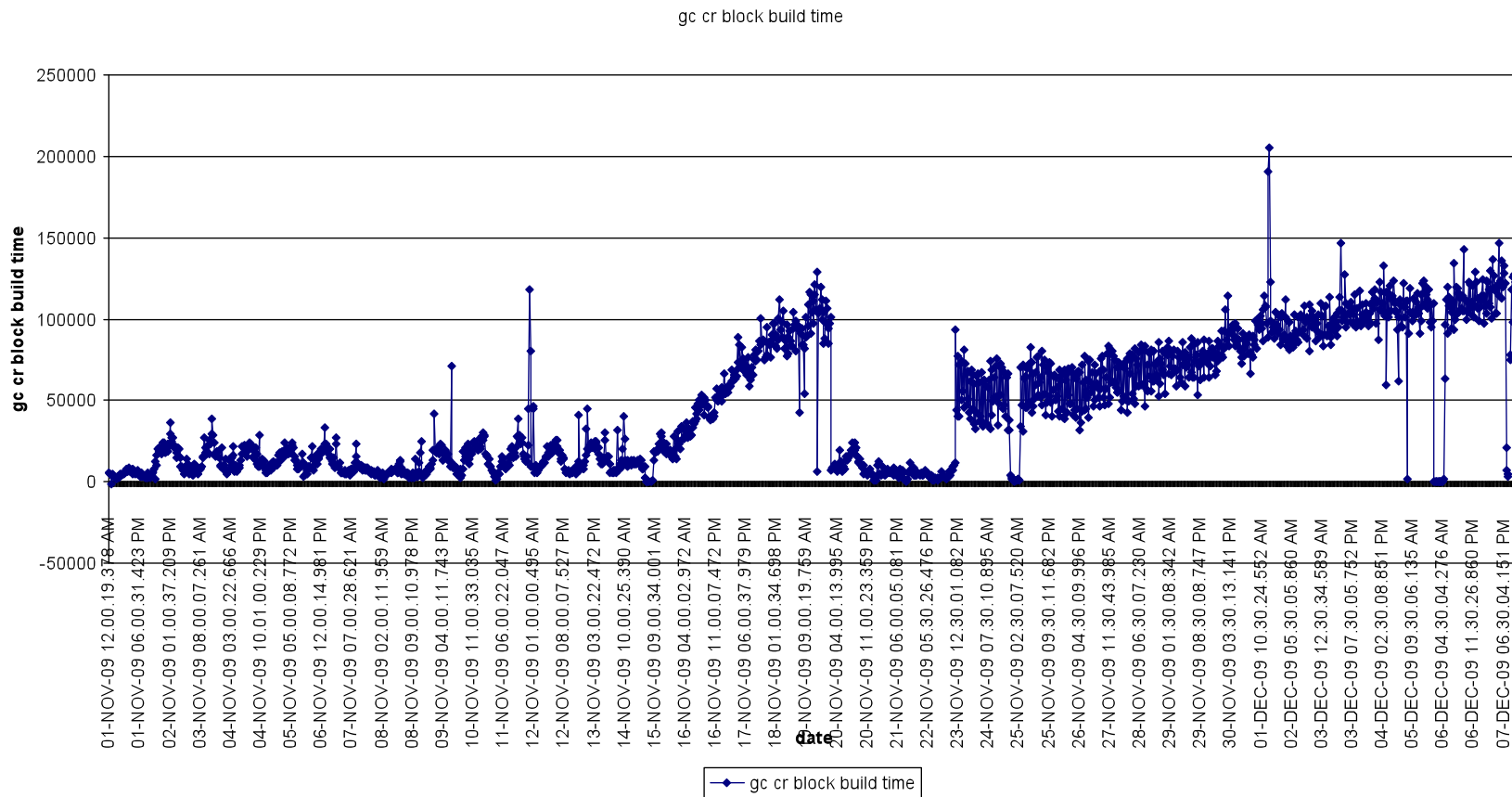
- For CR blocks, time is spent in building blocks, which indicates consistent block generation.

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

- Global cache performance
- **Undo, redo and more**
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Network layer tuning
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering



---

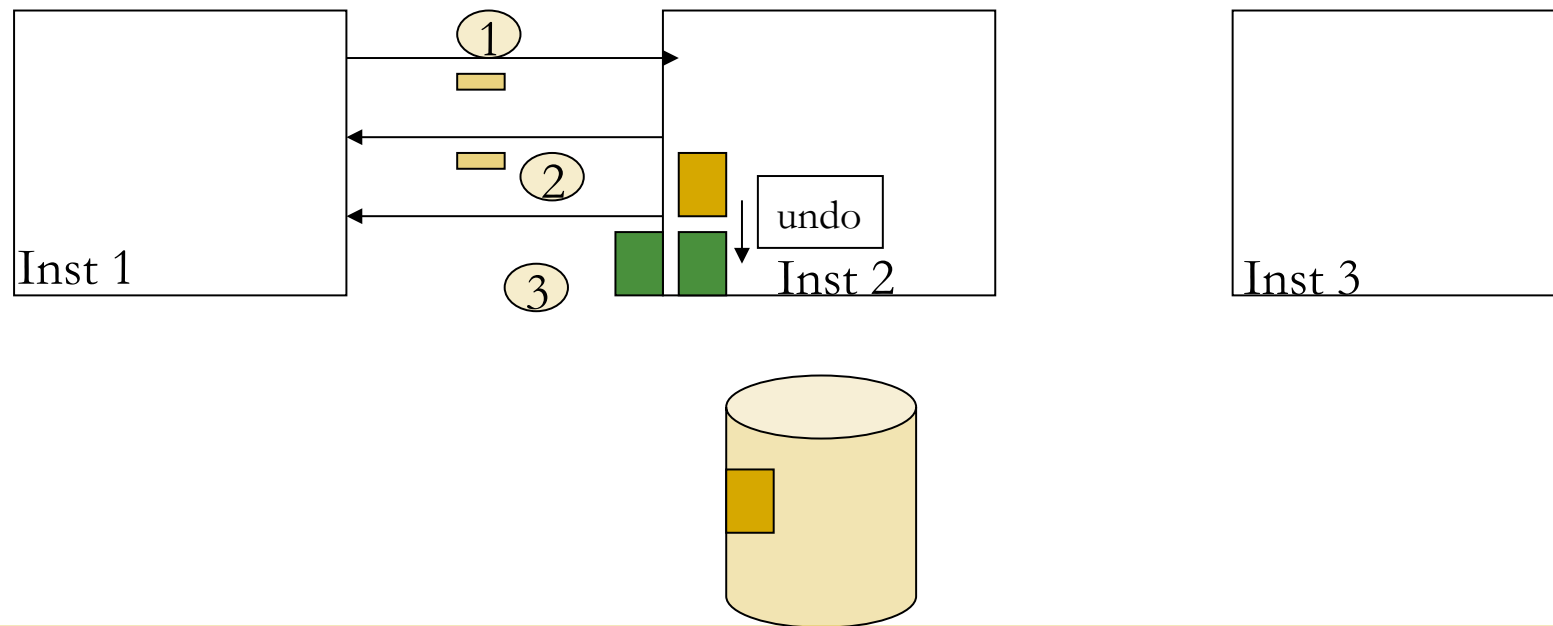
## Question

- Does an instance access undo blocks allocated to another instance?

## CR and undo

```
Select c1 from t1 where n1=:b1;
```

- ① User process in instance 1 requests master for the block in PR mode.
- ② Current owner (2) holds the block in Exclusive mode.
- ③ Instance 2 applies undo to create a version of the block consistent with SCN requested. Then ships the block to instance 1.



---

## Light-works rule/Fairness downconvert

- But, if one instance is a read only instance, then it might request another instance to generate CR copies applying undo blocks excessively.
- This is avoided by light-works rule. If an instance is excessively serving a block by applying undo blocks, it will downgrade the block mode.
- Requesting instance then will read from the disk and apply undo blocks (if needed) reducing load on the prior owner.

---

## Undo for CR

- A query can not read the block image with an SCN later than the query environment SCN.
- If a block is ahead of time, then undo blocks are applied to create a consistent version of the block.
- But, if the block was modified by a different instance, then undo blocks may need to be shipped from another instance.
- This can cause excessive cache transfers or excessive physical reads for undo blocks.
- Node affinity will be helpful to resolve this.

---

## Commit cleanout

- ITL entries in the blocks may not be cleaned out immediately.
- Session reading the block next time will check if the pending transaction is committed or not.
- If the transaction table is cycled through, then the session will apply undo to find a transaction table version with that transaction.
- This can lead to vicious cycle of rolling back the transaction table accessing undo blocks excessively.
- In RAC, this problem is magnified, since the transaction(s) could be in a different instance. So, transaction table blocks and undo blocks from different instance need to be shipped from other node or read from the disk.

## Redo and LGWR

- LGWR performance is important for global cache response time.
- Even for CR blocks LGWR must flush if the block is considered busy.

| Statistics             | Node 1  | Node 2 | Node 3 | Node 4 |
|------------------------|---------|--------|--------|--------|
| Gc cr block flush time | 129,970 | 12,289 | 11,556 | 27143  |

| Statistics               | Node 1 | Node 2 | Node 3 | Node 4 |
|--------------------------|--------|--------|--------|--------|
| Avg. gc cr block rx time | 4.2    | 22.7   | 21.5   | 11.0   |

---

# Agenda

- Global cache performance
- Undo, redo and more
- **RAC background process tuning**
- Interconnect issues, lost packets and network layer
- Network layer tuning
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering

---

## Global Cache waits

- Global Cache waits increases due to increase in LMS latency in the CPU starved node.
- Much of these GC waits are blamed on interconnect interface and hardware.
- In many cases, interconnect is performing fine, it is that GCS server processes are introducing latencies.



---

## LMS & 10.2.0.3

- In 9i, increasing priority of LMS processes to RT helps (more covered later).
- From Oracle release 10.2.0.3 LMS processes run in Real Time priority by default.
- Two parameters control this behaviour:
  - `_high_priority_processes`
  - `_os_sched_high_priority`

---

## Parameters in 10gR2

- `_high_priority_processes`:

Default value: LMS\* | VKTM\*

This parameter controls what background processes should get Real time priority. Default is all LMS processes and VKTM process.

- `_os_sched_high_priority` :

Default value: 1

This is a switch. If set to 0, no background process will run in high priority.

---

## oradism

- Of course, bumping priority needs higher privileges such as root in UNIX.
- Oradism utility is used to increase the priority class of these critical background process in UNIX.
- Verify that LMS processes are using Real time priority in UNIX and if not, oradism might not have been configured properly.
- In Windows, oradism service is used to increase the priority.

---

## More LMS processes?

- Typical response is to increase number of LMS processes adjusting `_lm_lms` (9i) or `gcs_server_processes`(10g).
- Increase in LMS processes without enough need increases `xcalls/migrates/tlb-misses` in massive servers.
- Further, LMS process runs in RT CPU priority and so, CPU usage will increase.

---

## LMS & CPU usage

- In huge servers, by default, number of LMS processes might be quite high. It is possible to get up to 26 LMS processes by default.
- Typically, same number of LMS processes as interconnect or remote nodes is a good starting point.
- If there is enormous amount of interconnect traffic, then configure LMS processes to be twice the interconnect.

---

## LGWR and CPU priority

- LGWR performance is akin to Global cache performance.
- If LGWR suffers from performance issues, it will reflect on Global cache performance.
- For example, If LGWR suffers from CPU latency issues, then LMS will have longer waits for 'gcs log flush sync' event
- This leads to poor GC performance in other nodes.

## LGWR priority

- Method to increase priority for LGWR and LMS in 9i (Example for Solaris). If you don't want to increase priority to RT for LGWR, at least, consider FX priority.

```
priocntl -e -c class -m userlimit -p priority
```

```
priocntl -e -c RT -p 59 `pgrep -f ora_lgwr_${ORACLE_SID}`
```

```
priocntl -e -c FX -m 60 -p 60 `pgrep -f ora_lms[0-9]*_${ORACLE_SID}`
```

- In 10g, parameter `_high_priority_processes` can be used (needs database restart though)

```
alter system set "_high_priority_processes"="LMS*|LGWR*" scope=spfile sid='*';
```

```
alter system set "_high_priority_processes"="LMS*|VKTM*|LGWR*" scope=spfile  
sid='*'; (11g)
```

- See note 759082.1 for HP-UX : `hpux_sched_noage` and other issues.

---

## Pitfalls of RT mode

- Of course, there are few! RT is kernel preemptive.
- LMS process can continuously consume CPU and can introduce CPU starvation in servers with few CPUs.
- A bug was opened to make LMS process sleep intermittently, but that causes LMS to be less active and can cause GC latency.
- Another undocumented parameter `_high_priority_process_num_yields_before_sleep` was introduced as a tunable. But, hardly a need to alter this parameter.
- So, LMS might wait for LGWR. If LGWR is not running in RT, then LMS can preempt LGWR leading to not-so-optimal wait graph. But, LGWR can block interrupt which LGWR might need!



---

## Binding..

- Another option is to bind LGWR/LMS to specific processors or processor sets.
- Still, interrupts can pre-empt LMS processors and LGWR. So, binding LMS to processor set without interrupts helps (see psradm in solaris).
- But, of course, processor binding is useful in servers with higher number of CPUs such as E25K / M9000 platforms.

---

## CSSD/CRSD

- CSSD is a critical process. Few CSSD processes must be running with RT priority.

`crsctl set css priority 4`

- CPU starvation in the server can lead to missed network or disk heart beat. This can lead to node reboots.
  - It is important to have good and consistent I/O performance to `ORA_CRS_HOME` directories.
  - If CSSD can't access those directories efficiently (i.e. due to NFS or other file system issues), then that can lead to node reboots too.
-

---

## Summary

- In summary,
  - Use optimal # of LMS processes
  - Use RT or FX high priority for LMS and LGWR processes.
  - Configure decent hardware for online redo log files.
  - Tune LGWR writes and Of course, avoid double buffering and double copy using optimal file systems.
  - Of course, tune SQL statement to reduce logical reads and reduce redo size.

---

# Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- **Interconnect issues, lost packets and network layer**
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering

---

## gc blocks lost

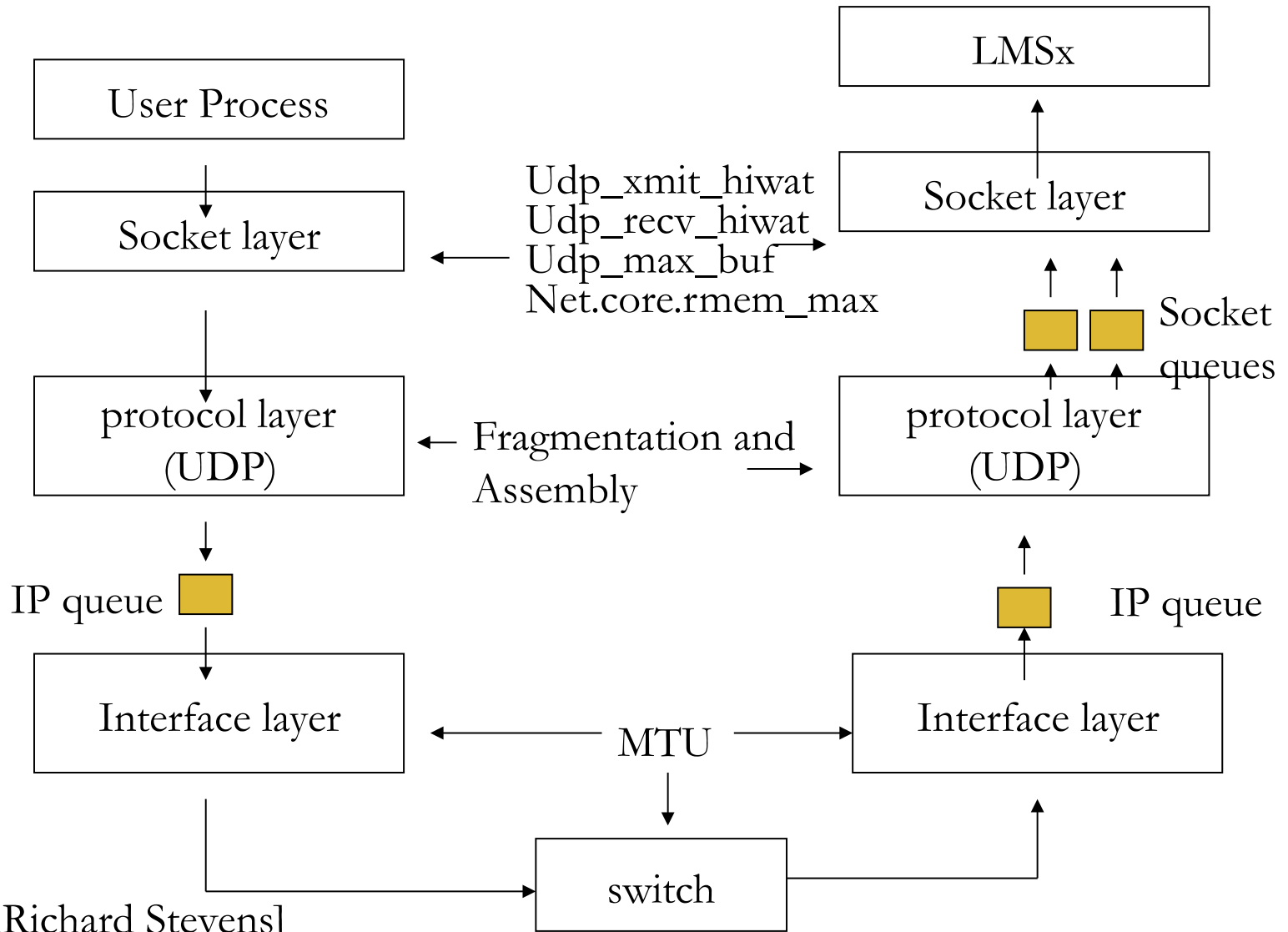
- Probably, the most direct statistics indicating interconnect issues.
- Consistent high amount of ‘gc blocks lost’ is an indication of problem with underlying network infrastructure. (Hardware, firmware, setup etc).
- Need to understand which specific component is an issue. Usually, this is an inter-disciplinary analysis.
- Ideal value is near zero. But, only worry about this, if there are consistently higher values.

---

## Effects of lost blocks

- Higher number of block loss can lead to timeouts in GC traffic wait events. Many processes will be waiting for place-holder events.
- Use `total_timeouts` column in `v$instance_event` to see if the timeouts are increasing.
- Percent of `total_timeouts` should be very small.

# Network layers



Source: [8, Richard Stevens]

---

## UDP buffer space

- UDP is a “send-and-forget” type protocol. Sending process does not get any acknowledgement.
- UDP Tx/Rx buffers are allocated per process.
- When the process executes CPU, it drains the UDP buffers. If the buffer is full, then incoming packets to that process are dropped.
- Default values for the UDP buffers are small for the bursty nature of interconnect traffic. Increase UDP buffer space to 128KB or 256KB.

---

Demo: wireshark in node2, tc\_one\_row



---

## CPU latency and UDP

- Due to CPU latency, process might not be able to acquire CPU quick enough.
- This can lead to buffer full conditions and lost packets.
- It is essential to keep CPU usage under 80% to avoid latencies and lost packets.

---

# Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- **Effective use of parallel query**
- Troubleshooting locking issues
- Object re-mastering

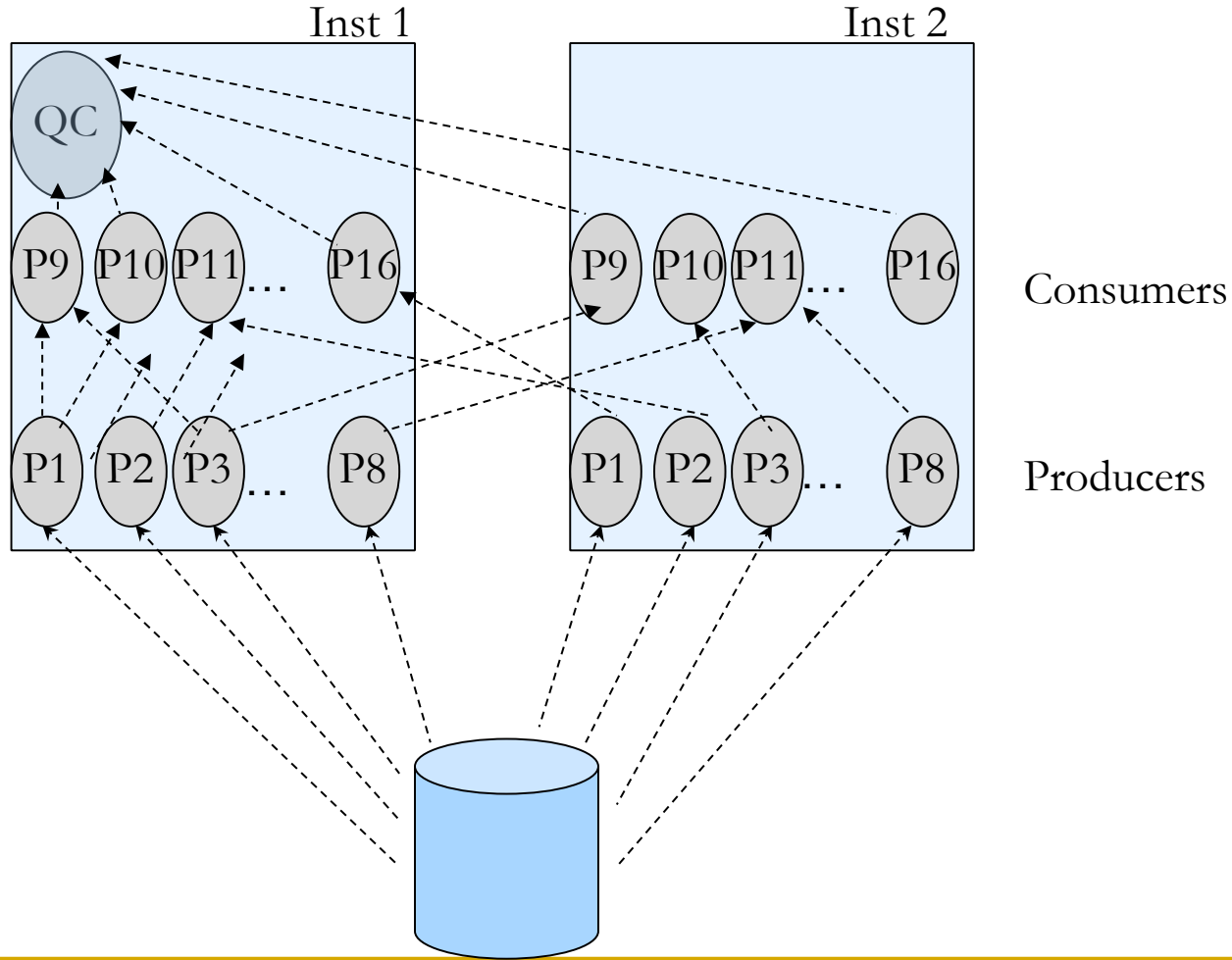
---

## Parallel Query Setup

- Parallel Query slaves can be allocated from multiple instances for a query.
- It is imperative that PQ messages are transmitted between producers and consumers.
- Insufficient network bandwidth with PQ storm can cause higher GC latency and possible packet loss.

# PQ Optimization

Communication between producers/consumers are Not limited to one node. Gigabytes of data flew Between node 1 and node 2.



---

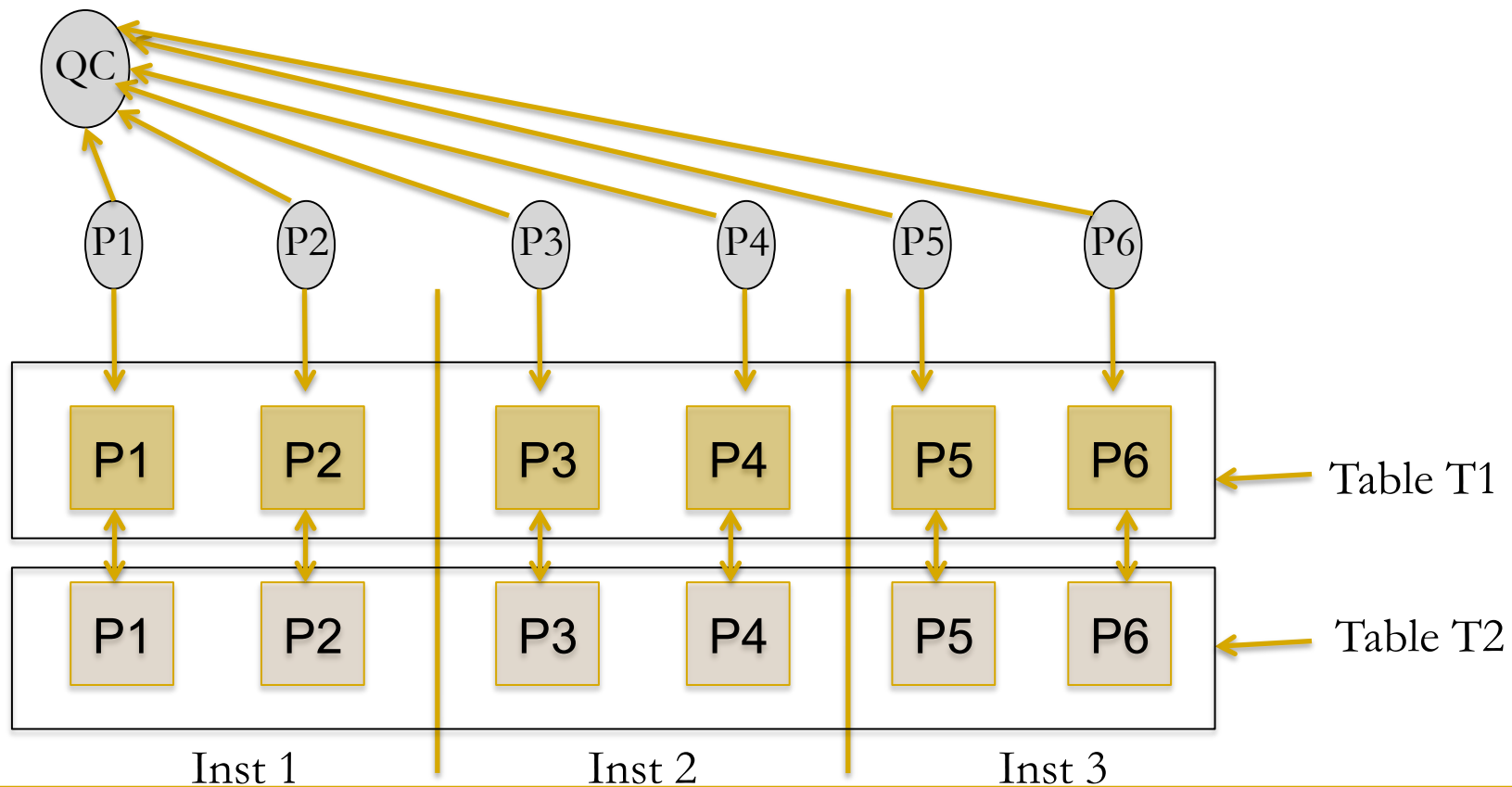
## Optimizations in 10g/11g

- PQ algorithms are optimized in Oracle versions 10g and 11g. Only few discussed here.
- In 11g, interconnect traffic due to PQ is also reported in the AWR reports.
- Oracle code tries to allocate all PQ slaves in one node, if possible. This minimizes PQ induced interconnect traffic.
- If it not possible to allocate all slaves from a node, then the least loaded node(s) are chosen for PQ slave allocation.

## Partition-wise joins

...2

- Interconnect traffic is kept minimal as the equivalent partitions are joined by a PQ process and final result is derived by Query Co-ordinator.



Demo: pq\_query\_range

---

## PQ-Summary

- Inter instance parallelism need to be carefully considered and measured.
- For partition based processing, when processing for a set of partitions is contained within a node, performance will be better.
- Excessive inter instance parallelism will increase interconnect traffic leading to performance issues.
- [http://www.oracle.com/technology/products/bi/db/11g/pdf/twp\\_bidw\\_parallel\\_execution\\_11gr1.pdf](http://www.oracle.com/technology/products/bi/db/11g/pdf/twp_bidw_parallel_execution_11gr1.pdf)

“..inter-node parallel execution will not scale with an undersized interconnect”

---

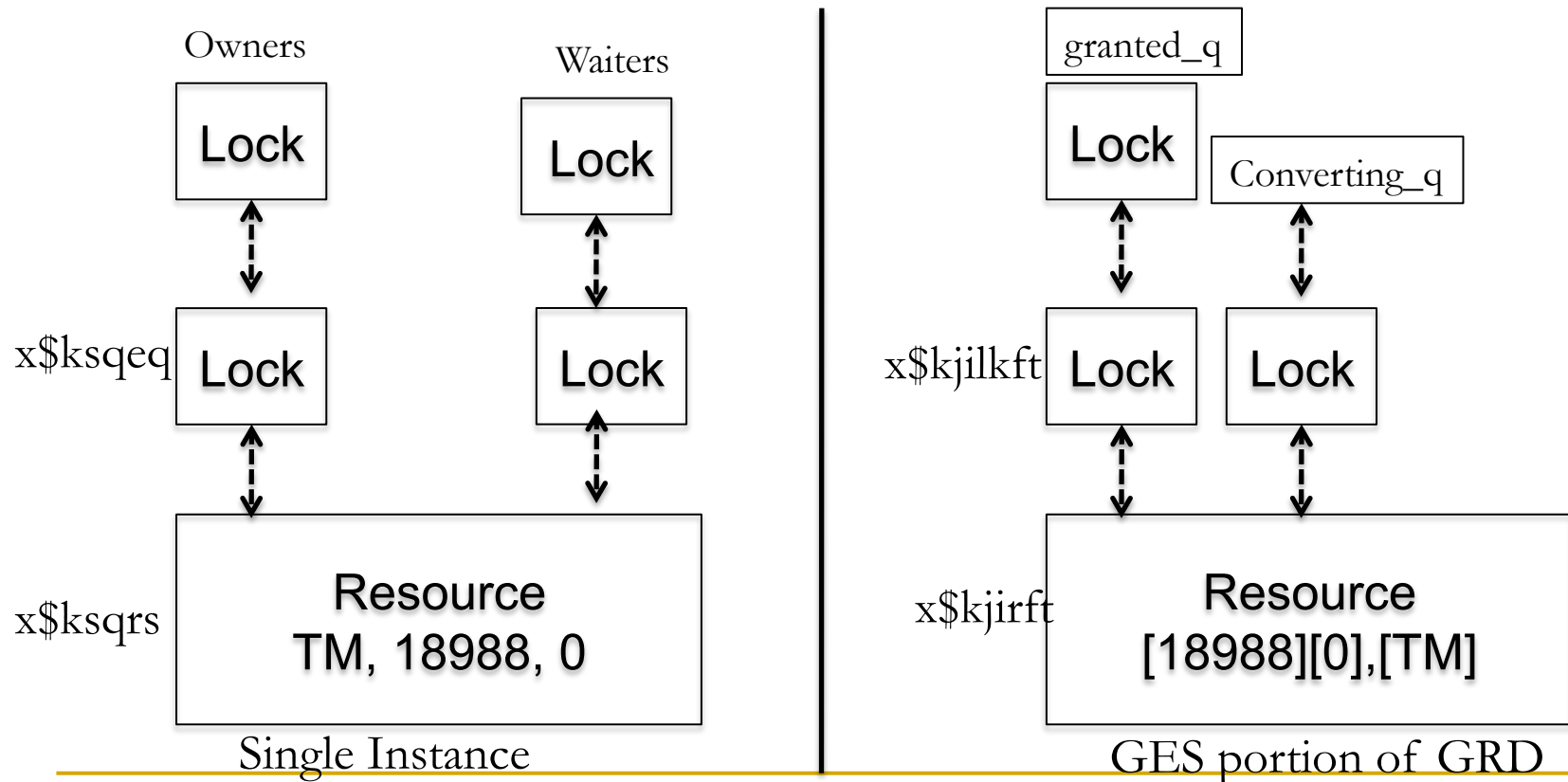
# Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Effective use of parallel query
- **Troubleshooting locking issues**
- Object re-mastering



## Globalization

- GES layer locks are externalized through x\$kjirft and x\$kjilkft tables.



## Demo:1 row update

- Updating 1 row creates a local transaction.

| INST_ID | SID | TY | ID1     | ID2 | CTIME | LMODE | REQUEST | BLOCK |
|---------|-----|----|---------|-----|-------|-------|---------|-------|
| 1       | 48  | TM | 76483   | 0   | 55    | 3     | 0       | 2     |
| 1       | 48  | TX | 3866640 | 49  | 55    | 6     | 0       | 2     |

- But, TX resource and lock is not globalized.

```
@ges_resource_tx.sql
```

```
-----  
Resource details...
```

```
-----  
-----
```

```
Lock details...
```

```
-----
```

## Demo:1 row update

...2

- Updating the same row in another instance, globalised the TX resource and created lock structures in the master instance.

-----  
Resource details...  
-----

Resource name [0x470001][0x159],[TX][ext 0x2, Master 0,Instance 1  
Resource name [0x470001][0x159],[TX][ext 0x2, Master 0,Instance 2

-----  
Lock details...  
-----

Res name [0x470001][0x159],[TX][ext 0x2, owner 0  
...Transaction\_id0 2097153,Level KJUSEREX ,State GRANTED  
...blocked 0,1

Res name [0x470001][0x159],[TX][ext 0x2, owner 1  
...Transaction\_id0 0,Level KJUSERNL ,State GRANTED  
...blocked 0,0

Res name [0x470001][0x159],[TX][ext 0x2, owner 1  
...Transaction\_id0 2228226,Level KJUSERNL ,State OPENING , Req. 1v1 KJUSEREX

For TX resource, instance creating the transaction is the master instance, starts with 0.

Waiting session

Demo: demo\_enq\_tx\_a.sql demo\_enq\_tx\_b.sql

---

# Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Effective use of parallel query
- Troubleshooting locking issues
- **Object re-mastering**

---

## Object re-mastering

- Before reading the block, an user process must request master node of the block to access that block.
- Typically, a batch process will access few objects aggressively.
- If an object is accessed excessively from a node then re-mastering the object to that node reduces Global cache grants.
- Local grants (affinity locks) are very efficient compared to remote grants avoiding global cache messaging traffic.

---

## Object based in 10gR2

- Dynamic remastering is file based in 10gR1. If a block need to be remastered, then every block in that data file must be remastered to an instance.
- In 10gR2, remastering is object based. If a block to be remastered, then all blocks associated with that object is remastered to an instance.
- Three background processes work together to implement dynamic remastering functionality.

---

## High level overview 10gR2

- LCK0 process maintains object level statistics and determines if remastering must be triggered.
- If an object is chosen, a request is queued. LMD0 reads the request queue and initiates GES freeze. LMD0 trace file

```
*** 2010-01-08 19:41:26.726
```

```
* kjdrchkdrm: found an RM request in the request queue  
  Dissolve pkey 6984390
```

```
*** 2010-01-08 19:41:26.727
```

```
Begin DRM(189) - dissolve pkey 6984390 from 2 oscan 1.1  
  ftd received from node 1 (8/0.30.0)  
  ftd received from node 0 (8/0.30.0)  
  ftd received from node 3 (8/0.30.0)  
  all ftds received
```

- LMON performs reconfiguration.

```
*** 2010-01-08 19:41:26.793
```

```
Begin DRM(189)  
  sent syncr inc 8 lvl 5577 to 0 (8,0/31/0)  
  synca inc 8 lvl 5577 rcvd (8.0)
```

---

## Parameters 10gR2

Three parameters control the behavior:

- `_gc_affinity_limit`
- `_gc_affinity_time`
- `_gc_affinity_minimum`
- `_gc_affinity_limit` default value is 50. Not documented well, but, it is number of times a node should open an object more than other nodes.
- `_gc_affinity_time` default value is 10. Frequency in seconds to check if remastering to be triggered or not.
- `_gc_affinity_minimum` determines minimum activity per minute to trigger remastering default to 2400.



---

## Defaults

- Default for these parameters may be too low in a very busy, high-end instances.
- If your database have higher waits for 'gc remaster' and 'gcs drm server freeze' then don't disable this feature completely. Instead tune it.
- Some good starting points (for a very busy environment) are:
  - [ YMMV]
  - `_gc_affinity_limit` to 250
  - `_gc_affinity_minimum` to 2500.

---

## 11g

- In 11g, these are few changes.
- Three new parameters are introduced:
  - `_gc_affinity_locking`
  - `_gc_affinity_locks`
  - `_gc_affinity_ratio`
- Two parameters are renamed:
  - `_gc_policy_minimum` ( default 1500 per minute)
  - `_gc_policy_time` (default 10 minutes)

## An example

### Top 5 Timed Events

~~~~~

Event	waits	Time (s)	Avg wait (ms)	%Total Call Time	wait Class
gc buffer busy	1,826,073	152,415	83	62.0	cluster
CPU time		30,192		12.3	
enq: TX - index contention	34,332	15,535	453	6.3	Concurrenc
gcs drm freeze in enter server	22,789	11,279	495	4.6	other
enq: TX - row lock contention	46,926	4,493	96	1.8	Applicatio

Global Cache and Enqueue Services - workload Characteristics

~~~~~

|                                                   |      |
|---------------------------------------------------|------|
| Avg global enqueue get time (ms):                 | 16.8 |
| Avg global cache cr block receive time (ms):      | 17.1 |
| Avg global cache current block receive time (ms): | 14.9 |

## Views

- View `v$gcspfmaster_info` provides remastering details. For example, you can identify the object with high remastering count.

| FILE_ID | OBJECT_ID | CURRENT_MASTER | PREVIOUS_MASTER | REMASTER_CNT |
|---------|-----------|----------------|-----------------|--------------|
| 0       | 6983606   | 0              | 32767           | 1            |
| 0       | 5384799   | 2              | 1               | 2            |
| 0       | 6561032   | 3              | 2               | 2            |
| 0       | 5734002   | 0              | 2               | 2            |
| 0       | 6944892   | 2              | 0               | 2            |
| 0       | 5734007   | 2              | 0               | 4            |
| 0       | 6944891   | 2              | 0               | 5            |
| 0       | 6795604   | 2              | 0               | 5            |
| 0       | 6944894   | 2              | 0               | 5            |
| 0       | 6795648   | 2              | 0               | 6            |
| 0       | 5734006   | 2              | 0               | 6            |
| 0       | 4023250   | 2              | 0               | 6            |
| 0       | 5734003   | 0              | 2               | 7            |

## Views

- View `x$object_object_affinity_statistics` provides current object affinity statistics.

```
select * from x$object_affinity_statistics order by opens
```

| ADDR             | INDX  | INST_ID | OBJECT  | NODE  | OPENS |
|------------------|-------|---------|---------|-------|-------|
| -----            | ----- | -----   | -----   | ----- | ----- |
| ...              |       |         |         |       |       |
| FFFFFFFF7C04CB40 | 8     | 3       | 4740170 | 1     | 113   |
| FFFFFFFF7C04CB40 | 109   | 3       | 1297745 | 1     | 127   |
| FFFFFFFF7C04CB40 | 21    | 3       | 1341531 | 1     | 128   |
| FFFFFFFF7C04CB40 | 2     | 3       | 2177393 | 1     | 135   |
| FFFFFFFF7C04CB40 | 153   | 3       | 6942171 | 2     | 174   |
| FFFFFFFF7C04CB40 | 108   | 3       | 1297724 | 1     | 237   |
| FFFFFFFF7C04CB40 | 3     | 3       | 2177593 | 1     | 239   |
| FFFFFFFF7C04CB40 | 106   | 3       | 1297685 | 1     | 337   |
| FFFFFFFF7C04CB40 | 53    | 3       | 6984154 | 3     | 1162  |

---

## Oradebug

- You can manually remaster an object with oradebug command

```
oradebug lkdebug -m pkey <object_id>
```

- This enqueues an object remaster request. LMD0 and LMON completes this request

```
*** 2010-01-08 23:25:54.948
```

```
* received DRM start msg from 1 (cnt 1, last 1, rmno 191)
```

```
Rcvd DRM(191) Transfer pkey 6984154 from 0 to 1 oscan 0.0
```

```
ftd received from node 1 (8/0.30.0)
```

```
ftd received from node 0 (8/0.30.0)
```

```
ftd received from node 3 (8/0.30.0)
```

## Oradebug

- You can manually remaster an object with oradebug command. Current\_master starts from 0.

```
1* select * from v$gcspfmaster_info where object_id=6984154
SQL> /
```

| FILE_ID | OBJECT_ID | CURRENT_MASTER | PREVIOUS_MASTER | REMASTER_CNT |
|---------|-----------|----------------|-----------------|--------------|
| 0       | 6984154   | 1              | 0               | 2            |

```
SQL> oradebug lkdebug -m pkey 6984154
Statement processed.
```

```
SQL> select * from v$gcspfmaster_info where object_id=6984154
2 /
```

| FILE_ID | OBJECT_ID | CURRENT_MASTER | PREVIOUS_MASTER | REMASTER_CNT |
|---------|-----------|----------------|-----------------|--------------|
| 0       | 6984154   | 2              | 1               | 3            |

Thank you for attending!

If you like this presentation, you will love our upcoming intensive RAC webinar in August '11.

Watch for updates in:

[www.tanelpoder.com](http://www.tanelpoder.com)

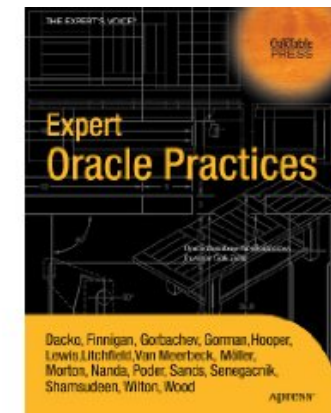
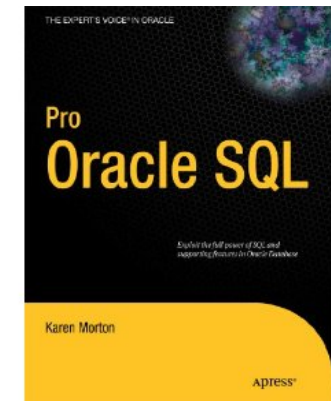
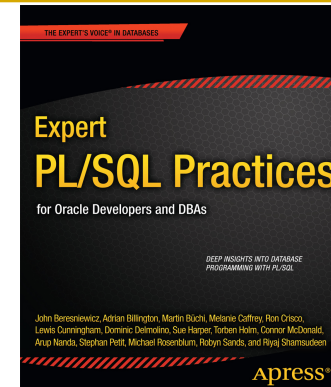
[orainternals.wordpress.com](http://orainternals.wordpress.com)

Contact info:

Email: [rshamsud@gmail.com](mailto:rshamsud@gmail.com)

Blog : [orainternals.wordpress.com](http://orainternals.wordpress.com)

URL : [www.orainternals.com](http://www.orainternals.com)





---

# References

1. Oracle support site. Metalink.oracle.com. Various documents
2. Internal's guru Steve Adam's website  
[www.ixora.com.au](http://www.ixora.com.au)
3. Jonathan Lewis' website  
[www.jlcomp.daemon.co.uk](http://www.jlcomp.daemon.co.uk)
4. Julian Dyke's website  
[www.julian-dyke.com](http://www.julian-dyke.com)
5. 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'  
by Steve Adams
6. Randolph Geist : <http://oracle-randolf.blogspot.com>
7. Tom Kyte's website  
[Asktom.oracle.com](http://asktom.oracle.com)
8. Richard Stevens, Gary R Wright: TCP/IP Illustrated