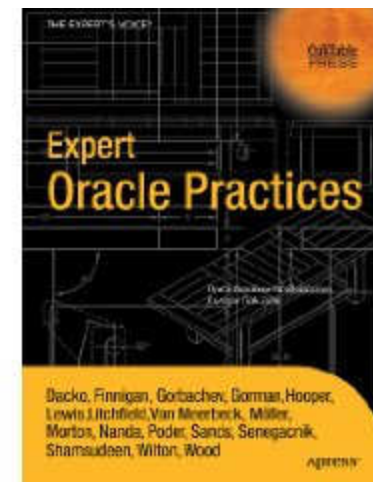

RAC Performance myths - Battle of the nodes

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.

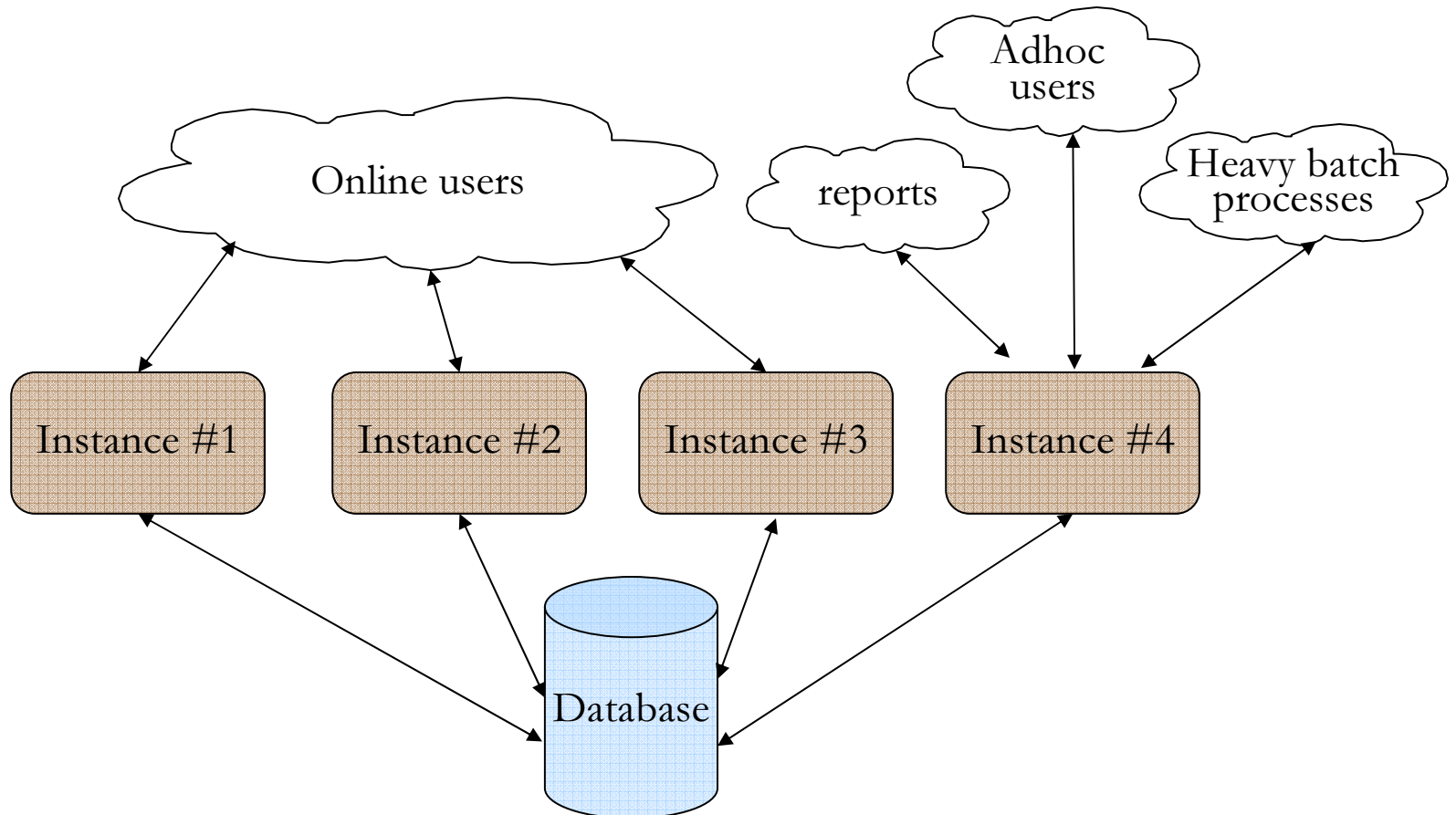
If you corrupt your databases by running my scripts, you are solely responsible for that.

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

Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

Typical RAC node setup

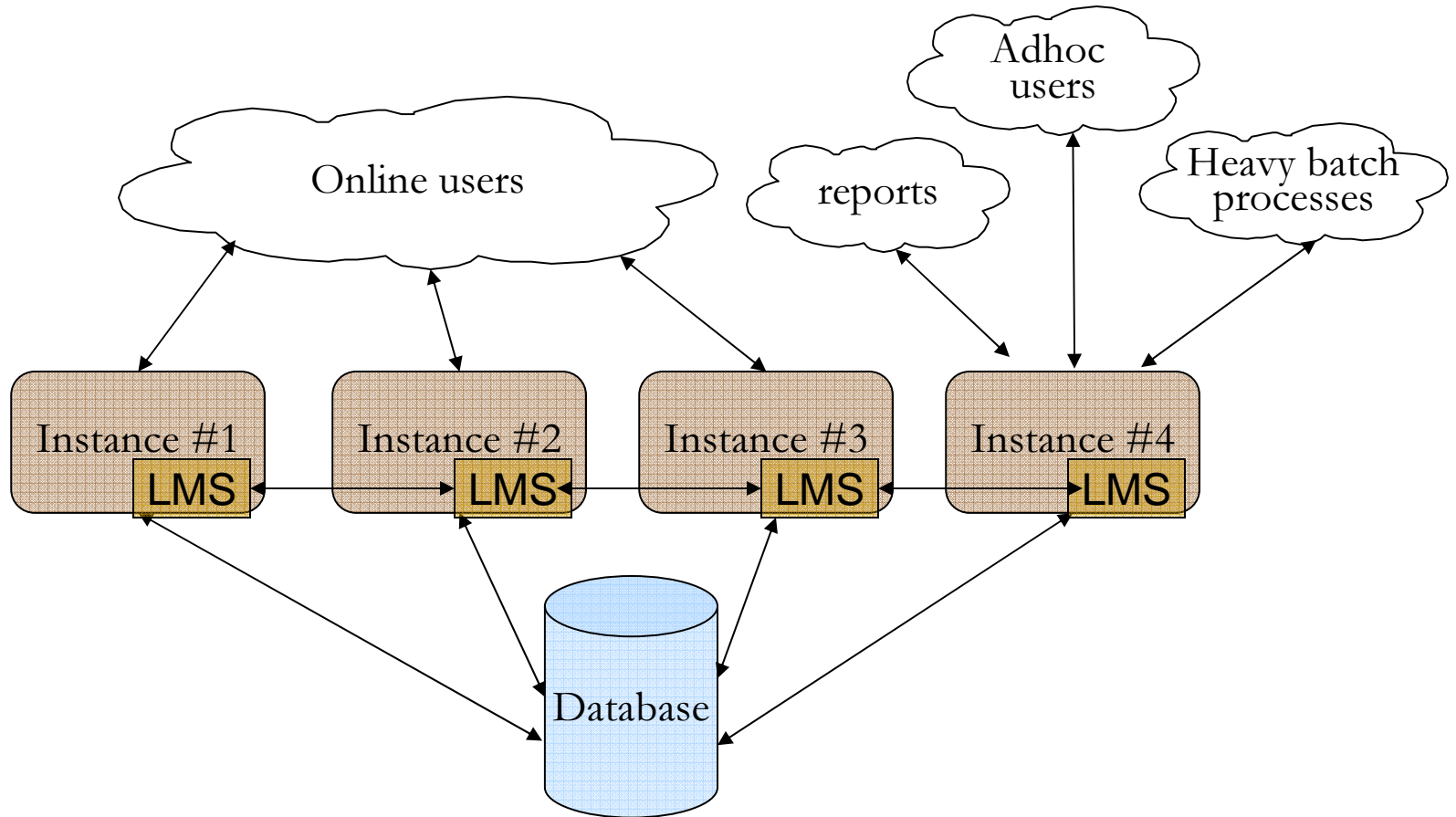


Reporting node

- Idea here is to put online “money-paying” users to all nodes and throw costly reports/adhoc SQL/batch in to one node.
- High CPU usage in the batch node shouldn't cause any issues to online users, right?
- Not Exactly!

LMS processes

- Blocks are transferred from remote cache if a suitable block is found in the remote cache avoiding costly disk reads.

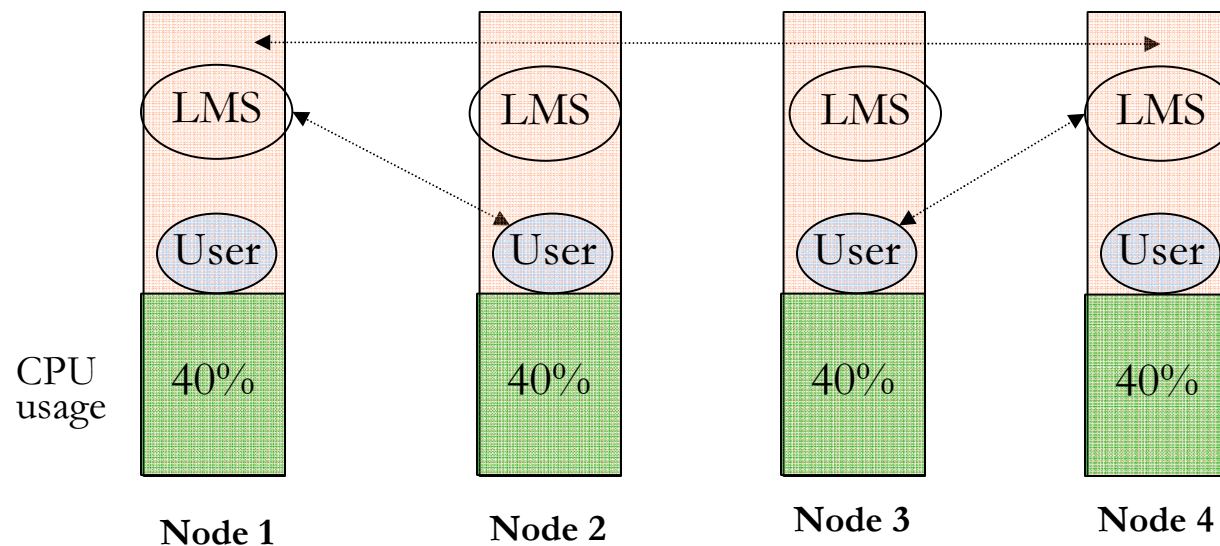


Global cache transfer

- Block transfer between instances are done by LMS processes.
- Until 10.2.0.1, LMS processes are running in normal CPU priority.
- If there is CPU starvation in any server, then all instances will be affected due to LMS latency.

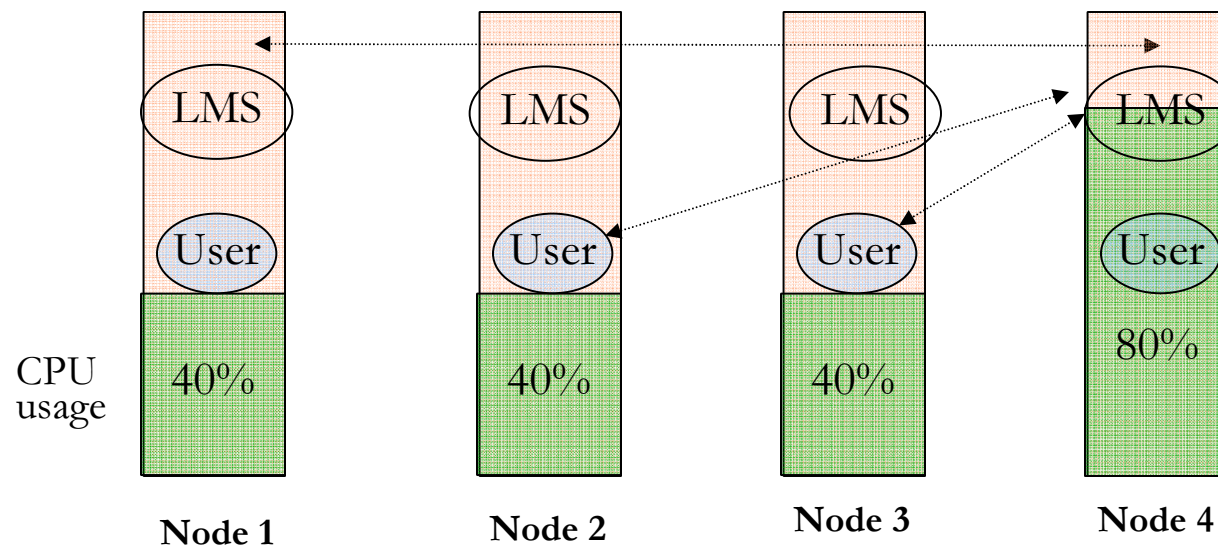
LMS processes – normal state

- During normal conditions, LMS processes are operating with no CPU latency.
- So, there is no Global cache latency either.



CPU latency

- If one node is suffering from CPU starvation then LMS process running in that node will suffer from CPU latency.
- This will result in Global cache latency in other nodes.



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.

Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

Node1 GC workload

Global Cache and Enqueue Services - workload Characteristics

Average GC CR block
Receive time is very high

~~~~~

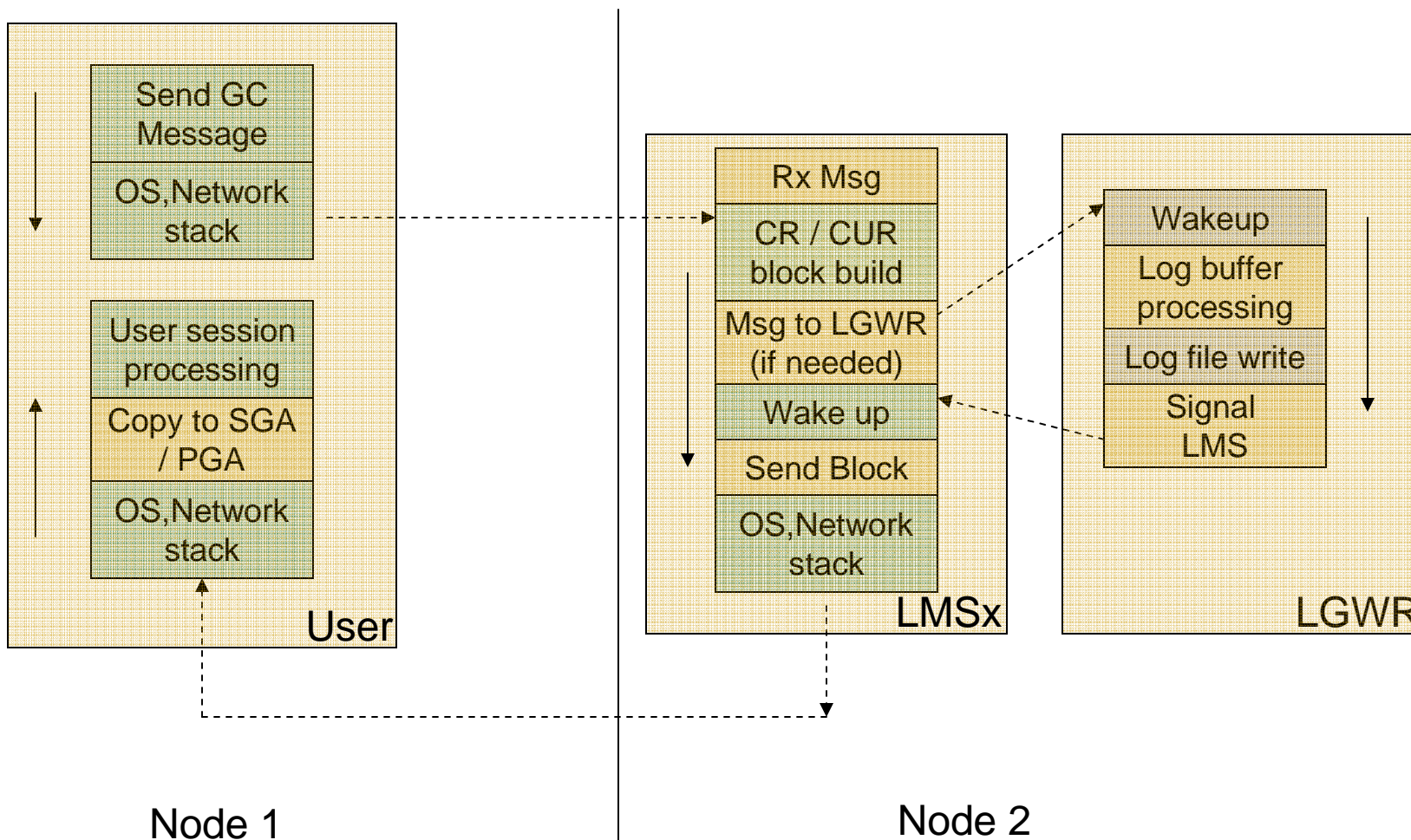
|                                                       |      |
|-------------------------------------------------------|------|
| Avg global enqueue get time (ms):                     | 8.9  |
| Avg global cache cr block receive time (ms):          | 63.3 |
| Avg global cache current block receive time (ms):     | 42.1 |
| Avg global cache cr block build time (ms):            | 0.3  |
| Avg global cache cr block send time (ms):             | 0.1  |
| Global cache log flushes for cr blocks served %:      | 4.5  |
| Avg global cache cr block flush time (ms):            | 51.5 |
| Avg global cache current block pin time (ms):         | 0.0  |
| Avg global cache current block send time (ms):        | 4.8  |
| Global cache log flushes for current blocks served %: | 0.1  |
| Avg global cache current block flush time (ms):       | 30.0 |

---

## Could this be interconnect issue?

- Common reaction to any Global cache performance issue: It is an interconnect hardware or network problem.
- It could be, but not necessarily.
- Unless interconnect is flooded, wire latency is a very small fraction of global cache latency.

# LMS Processing (over simplified)



---

# LMS and Log flush

- For CR blocks, if the buffer is NOT busy, LMS process can send block immediately.
- For CR blocks, if the buffer is busy, LMS process waits for Log flush to complete.
- For CURRENT blocks, LMS process waits for log flush to complete.

---

# GC CR latency

- GC CR latency  $\approx$ 
  - Time spent in sending message to LMS +
  - LMS processing +
  - LGWR latency ( if any) +
  - Wire latency

---

# GC CURRENT latency

- GC CUR latency  $\approx$

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

# Node2 GC workload

In this specific case, log flush was very slow due to an hardware issue

## Global Cache and Enqueue Services - workload Characteristics

```
~~~~~
```

|                                                       |        |
|-------------------------------------------------------|--------|
| Avg global enqueue get time (ms):                     | 0.3    |
| Avg global cache cr block receive time (ms):          | 10.4   |
| Avg global cache current block receive time (ms):     | 3.2    |
| Avg global cache cr block build time (ms):            | 0.1    |
| Avg global cache cr block send time (ms):             | 0.0    |
| Global cache log flushes for cr blocks served %:      | 5.0    |
| Avg global cache cr block flush time (ms):            | 4380.0 |
| Avg global cache current block pin time (ms):         | 0.0    |
| Avg global cache current block send time (ms):        | 0.1    |
| Global cache log flushes for current blocks served %: | 0.1    |
| Avg global cache current block flush time (ms):       | 0.0    |

---

## 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='*';
```

---

# Pitfalls of RT mode

- Of course, there are few!
- 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.
- Hyper-active LGWR can lead to latch contention issues.

---

## 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 platforms.

---

# 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 - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

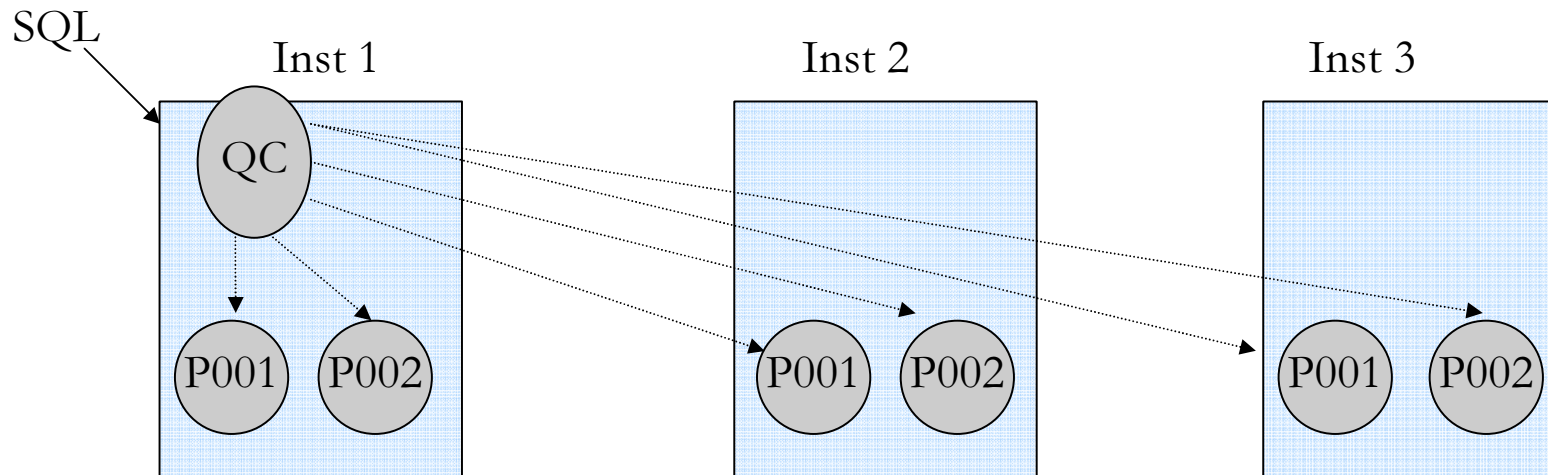
---

# Parallelism

- Few parameters controls this behaviour:
  - `parallel_min_servers`
  - `parallel_max_servers`
- Two more parameters, RAC specific (In 11g, use Services instead):
  - `instance_group`
  - `parallel_instance_group`
- In a multi-instance RAC cluster, we can control parallelism to specific instances.

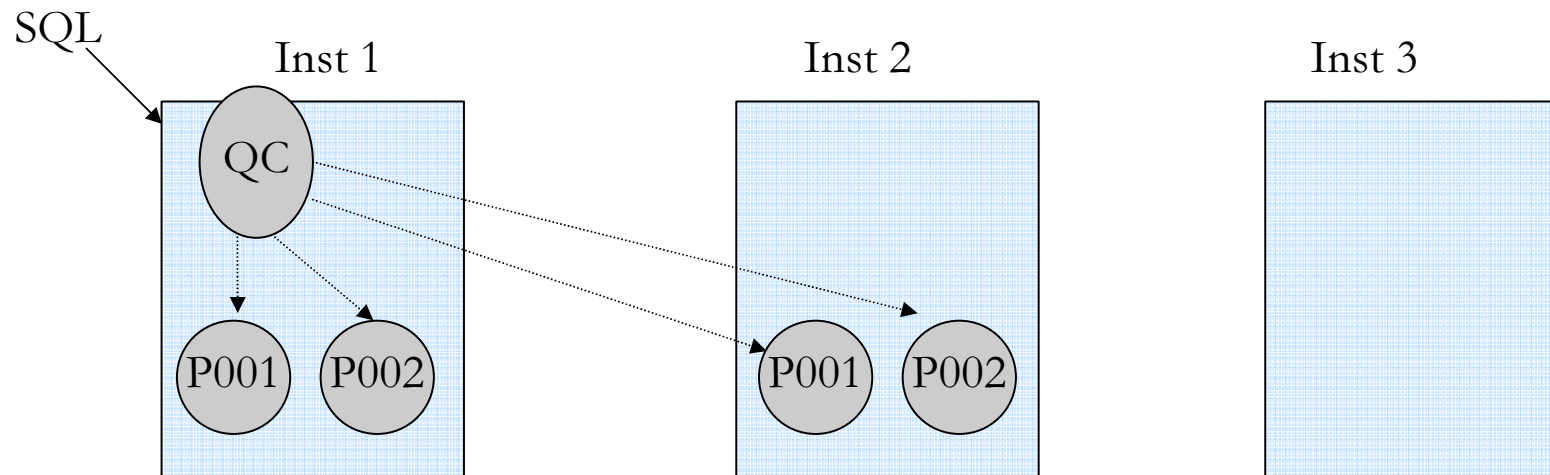
# Parallelism

- Let's say that there are three instances: inst1, inst2, inst3.
- To span slaves across all instances
  - `inst1.instance_groups='inst1','all'`
  - `inst2.instance_groups='inst2','all'`
  - `inst3.instance_groups='inst3','all'`
  - `inst1.parallel_instance_group='all'`



# Parallelism

- To span slaves across all instances inst1 and inst2 alone, parameters will be:
  - `inst1.instance_groups='inst1','all','inst12'`
  - `inst2.instance_groups='inst2','all','inst12'`
  - `inst3.instance_groups='inst3','all'`
  - `inst1.parallel_instance_group='inst12'`



---

## So, how do we measure?

- Parallel query traffic is not factored in Statspack or AWR report until Version 11g.
- There is no easy way to measure PQ traffic using database utilities.
- I used **dtrace based Solaris tool udpsnoop.d** to measure UDP traffic between various nodes for this presentation.
- Packet sizes were aggregated to calculate interconnect traffic
- This method is very accurate and reliable.

# SQL 1

- This SQL is performing a simple aggregation with parallel servers from two nodes.

```
alter session set parallel_instance_group='inst12';
```

```
select /*+ parallel (t1, 8) */
 min (t1.CUSTOMER_TRX_LINE_ID +t1.CUSTOMER_TRX_LINE_ID) ,
 avg(t1.SET_OF_BOOKS_ID+t1.set_of_books_id),
 max (t1.SET_OF_BOOKS_ID+t1.set_of_books_id),
 avg(t1.QUANTITY_ORDERED + t1.QUANTITY_ORDERED),
 max(t1.ATTRIBUTE_CATEGORY), max(t1.attribute1),
 max(t1.attribute2)
From BIG_TABLE t1;
```

This is a single table aggregation.

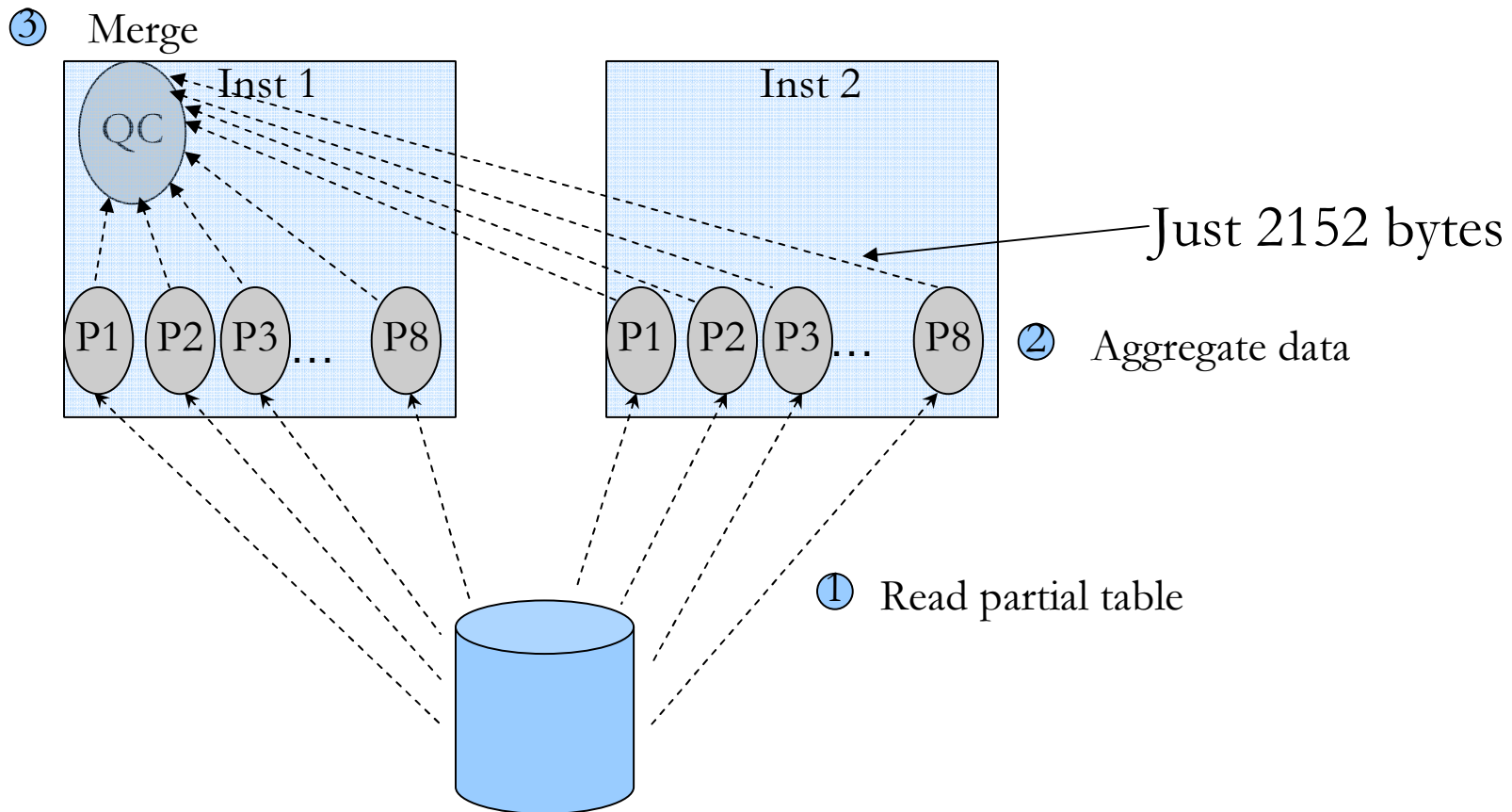
# PQ allocation

Eight slaves allocated from node 1 and node 2 each.

| INST | PROC   | slave   | Set | slave | QC    | RDOP | DOP | SIZE |
|------|--------|---------|-----|-------|-------|------|-----|------|
| 1    | SYS    | QC      |     | 10933 | 10933 |      |     | 3314 |
| 1    | - p000 | (slave) | 1   | 10958 | 10933 | 16   | 16  |      |
| 1    | - p001 | (slave) | 1   | 10948 | 10933 | 16   | 16  |      |
| 1    | - p002 | (slave) | 1   | 10953 | 10933 | 16   | 16  |      |
| 1    | - p003 | (slave) | 1   | 10925 | 10933 | 16   | 16  |      |
| 1    | - p004 | (slave) | 1   | 10916 | 10933 | 16   | 16  |      |
| 1    | - p005 | (slave) | 1   | 10938 | 10933 | 16   | 16  |      |
| 1    | - p006 | (slave) | 1   | 10951 | 10933 | 16   | 16  |      |
| 1    | - p007 | (slave) | 1   | 10946 | 10933 | 16   | 16  |      |
| 2    | - p000 | (slave) | 1   | 10949 | 10933 | 16   | 16  | 2152 |
| 2    | - p001 | (slave) | 1   | 10937 | 10933 | 16   | 16  | 2152 |
| 2    | - p002 | (slave) | 1   | 10946 | 10933 | 16   | 16  | 2152 |
| 2    | - p003 | (slave) | 1   | 10956 | 10933 | 16   | 16  | 2152 |
| 2    | - p004 | (slave) | 1   | 10902 | 10933 | 16   | 16  | 2152 |
| 2    | - p005 | (slave) | 1   | 10981 | 10933 | 16   | 16  | 2152 |
| 2    | - p006 | (slave) | 1   | 10899 | 10933 | 16   | 16  | 2152 |
| 2    | - p007 | (slave) | 1   | 10927 | 10933 | 16   | 16  | 2152 |

# PQ Optimization

- PQ code is optimized to reduce GC traffic.
- For a single table, GC traffic is minimal.



---

# SQL 2

- This SQL is an hash join between two big tables and then aggregation.

```
alter session set parallel_instance_group='inst12';
```

```
select /*+ parallel (t1, 16) parallel (t2, 16) */
 t1.CUSTOMER_TRX_LINE_ID , t1.SET_OF_BOOKS_ID,
 max(t1.ATTRIBUTE_CATEGORY), s
um(t2.QUANTITY_ORDERED), sum(t1.QUANTITY_ORDERED)
from
 backup.big_table1_part t1,
 backup.big_table1_part t2
where t1.CUSTOMER_TRX_LINE_ID = t2.CUSTOMER_TRX_LINE_ID
group by
 t1.CUSTOMER_TRX_LINE_ID , t1.SET_OF_BOOKS_ID
;
```

# PQ allocation -2

Two slave sets allocated per node.

|   |        |         |   |      |      |    |    |
|---|--------|---------|---|------|------|----|----|
| 1 | SYS    | QC      |   | 9868 | 9868 |    |    |
| 1 | - p007 | (Slave) | 1 | 9890 | 9868 | 16 | 16 |
| 1 | - p006 | (Slave) | 1 | 9949 | 9868 | 16 | 16 |
| 1 | - p005 | (Slave) | 1 | 9871 | 9868 | 16 | 16 |
| 1 | - p004 | (Slave) | 1 | 9878 | 9868 | 16 | 16 |
| 1 | - p003 | (Slave) | 1 | 9898 | 9868 | 16 | 16 |
| 1 | - p002 | (Slave) | 1 | 9969 | 9868 | 16 | 16 |
| 1 | - p001 | (Slave) | 1 | 9897 | 9868 | 16 | 16 |
| 1 | - p000 | (Slave) | 1 | 9924 | 9868 | 16 | 16 |
| 1 | - p015 | (Slave) | 2 | 9867 | 9868 | 16 | 16 |
| 1 | - p014 | (Slave) | 2 | 9908 | 9868 | 16 | 16 |
| 1 | - p013 | (Slave) | 2 | 9917 | 9868 | 16 | 16 |
| 1 | - p012 | (Slave) | 2 | 9938 | 9868 | 16 | 16 |
| 1 | - p011 | (Slave) | 2 | 9877 | 9868 | 16 | 16 |
| 1 | - p010 | (Slave) | 2 | 9895 | 9868 | 16 | 16 |
| 1 | - p009 | (Slave) | 2 | 9942 | 9868 | 16 | 16 |
| 1 | - p008 | (Slave) | 2 | 9912 | 9868 | 16 | 16 |

Instance 1

|   |        |         |   |      |      |    |    |
|---|--------|---------|---|------|------|----|----|
| 2 | - p023 | (Slave) | 1 | 9904 | 9868 | 16 | 16 |
| 2 | - p022 | (Slave) | 1 | 9941 | 9868 | 16 | 16 |
| 2 | - p021 | (Slave) | 1 | 9928 | 9868 | 16 | 16 |
| 2 | - p020 | (Slave) | 1 | 9870 | 9868 | 16 | 16 |
| 2 | - p019 | (Slave) | 1 | 9880 | 9868 | 16 | 16 |
| 2 | - p018 | (Slave) | 1 | 9934 | 9868 | 16 | 16 |
| 2 | - p017 | (Slave) | 1 | 9910 | 9868 | 16 | 16 |
| 2 | - p016 | (Slave) | 1 | 9913 | 9868 | 16 | 16 |
| 2 | - p031 | (Slave) | 2 | 9902 | 9868 | 16 | 16 |
| 2 | - p030 | (Slave) | 2 | 9883 | 9868 | 16 | 16 |
| 2 | - p029 | (Slave) | 2 | 9882 | 9868 | 16 | 16 |
| 2 | - p028 | (Slave) | 2 | 9920 | 9868 | 16 | 16 |
| 2 | - p027 | (Slave) | 2 | 9916 | 9868 | 16 | 16 |
| 2 | - p026 | (Slave) | 2 | 9903 | 9868 | 16 | 16 |
| 2 | - p025 | (Slave) | 2 | 9893 | 9868 | 16 | 16 |
| 2 | - p024 | (Slave) | 2 | 9897 | 9868 | 16 | 16 |

Instance 2

# PQ size

Two slave sets allocated per node.

|   |        |         |   |      |      |    |    |           |            |
|---|--------|---------|---|------|------|----|----|-----------|------------|
| 1 | SYS    | QC      |   | 9868 | 9868 |    |    |           |            |
| 1 | - p007 | (Slave) | 1 | 9890 | 9868 | 16 | 16 |           |            |
| 1 | - p006 | (Slave) | 1 | 9949 | 9868 | 16 | 16 |           |            |
| 1 | - p005 | (Slave) | 1 | 9871 | 9868 | 16 | 16 |           |            |
| 1 | - p004 | (Slave) | 1 | 9878 | 9868 | 16 | 16 |           |            |
| 1 | - p003 | (Slave) | 1 | 9898 | 9868 | 16 | 16 |           |            |
| 1 | - p002 | (Slave) | 1 | 9969 | 9868 | 16 | 16 |           |            |
| 1 | - p001 | (Slave) | 1 | 9897 | 9868 | 16 | 16 |           |            |
| 1 | - p000 | (Slave) | 1 | 9924 | 9868 | 16 | 16 |           |            |
| 1 | - p015 | (Slave) | 2 | 9867 | 9868 | 16 | 16 | 128020044 |            |
| 1 | - p014 | (Slave) | 2 | 9908 | 9868 | 16 | 16 | 132011920 |            |
| 1 | - p013 | (Slave) | 2 | 9917 | 9868 | 16 | 16 | 127770024 |            |
| 1 | - p012 | (Slave) | 2 | 9938 | 9868 | 16 | 16 | 128154240 |            |
| 1 | - p011 | (Slave) | 2 | 9877 | 9868 | 16 | 16 | 183490248 |            |
| 1 | - p010 | (Slave) | 2 | 9895 | 9868 | 16 | 16 | 181900075 |            |
| 1 | - p009 | (Slave) | 2 | 9942 | 9868 | 16 | 16 | 181828128 |            |
| 1 | - p008 | (Slave) | 2 | 9912 | 9868 | 16 | 16 | 124428800 | Instance 1 |
| 2 | - p023 | (Slave) | 1 | 9904 | 9868 | 16 | 16 |           |            |
| 2 | - p022 | (Slave) | 1 | 9941 | 9868 | 16 | 16 |           |            |
| 2 | - p021 | (Slave) | 1 | 9928 | 9868 | 16 | 16 |           |            |
| 2 | - p020 | (Slave) | 1 | 9870 | 9868 | 16 | 16 |           |            |
| 2 | - p019 | (Slave) | 1 | 9880 | 9868 | 16 | 16 |           |            |
| 2 | - p018 | (Slave) | 1 | 9934 | 9868 | 16 | 16 |           |            |
| 2 | - p017 | (Slave) | 1 | 9910 | 9868 | 16 | 16 |           |            |
| 2 | - p016 | (Slave) | 1 | 9913 | 9868 | 16 | 16 |           |            |
| 2 | - p031 | (Slave) | 2 | 9902 | 9868 | 16 | 16 | 127068484 |            |
| 2 | - p030 | (Slave) | 2 | 9883 | 9868 | 16 | 16 | 126904080 |            |
| 2 | - p029 | (Slave) | 2 | 9882 | 9868 | 16 | 16 | 127767353 |            |
| 2 | - p028 | (Slave) | 2 | 9920 | 9868 | 16 | 16 | 128154145 |            |
| 2 | - p027 | (Slave) | 2 | 9916 | 9868 | 16 | 16 | 128096875 |            |
| 2 | - p026 | (Slave) | 2 | 9903 | 9868 | 16 | 16 | 182490908 |            |
| 2 | - p025 | (Slave) | 2 | 9893 | 9868 | 16 | 16 | 181899025 |            |
| 2 | - p024 | (Slave) | 2 | 9897 | 9868 | 16 | 16 | 181611641 | Instance 2 |

# PQ explain plan

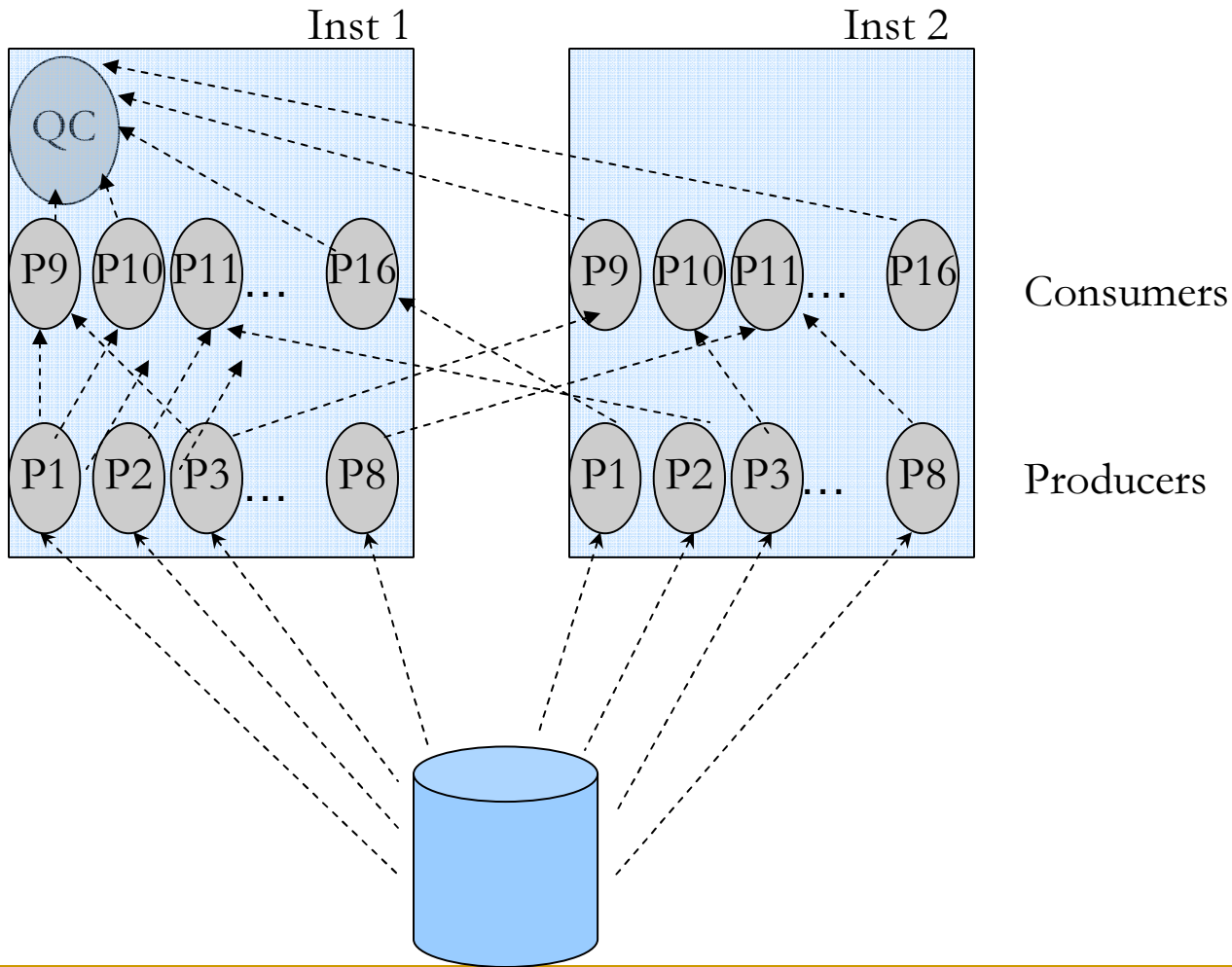
| Id  | Operation              | Name            | E-Rows | OMem  | lMem  | Used-Mem |
|-----|------------------------|-----------------|--------|-------|-------|----------|
| 1   | PX COORDINATOR         |                 |        |       |       |          |
| 2   | PX SEND QC (RANDOM)    | :TQ10001        | 85M    |       |       |          |
| 3   | HASH GROUP BY          |                 | 85M    | 1096K | 1096K |          |
| 4   | PX RECEIVE             |                 | 85M    |       |       |          |
| 5   | PX SEND HASH           | :TQ10000        | 85M    |       |       |          |
| 6   | HASH GROUP BY          |                 | 85M    | 1180K | 1180K |          |
| 7   | PX PARTITION RANGE ALL |                 | 85M    |       |       |          |
| * 8 | HASH JOIN              |                 | 85M    | 2047M | 37M   |          |
| 9   | TABLE ACCESS FULL      | BIG_TABLE1_PART | 85M    |       |       |          |
| 10  | TABLE ACCESS FULL      | BIG_TABLE1_PART | 85M    |       |       |          |

Predicate Information (identified by operation id):

8 - access("T1"."CUSTOMER\_TRX\_LINE\_ID"="T2"."CUSTOMER\_TRX\_LINE\_ID")

# PQ Optimization 2

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



---

# SQL 3

- Decreasing parallelism is keeping all slaves in the same instance.

```
alter session set parallel_instance_group='inst12';
```

```
select /*+ parallel (t1, 8) parallel (t2, 8) */
 t1.CUSTOMER_TRX_LINE_ID , t1.SET_OF_BOOKS_ID,
 max(t1.ATTRIBUTE_CATEGORY), s
um(t2.QUANTITY_ORDERED), sum(t1.QUANTITY_ORDERED)
from
 backup.big_table1_part t1,
 backup.big_table1_part t2
where t1.CUSTOMER_TRX_LINE_ID = t2.CUSTOMER_TRX_LINE_ID
group by
 t1.CUSTOMER_TRX_LINE_ID , t1.SET_OF_BOOKS_ID
;
```

# PQ allocation -3

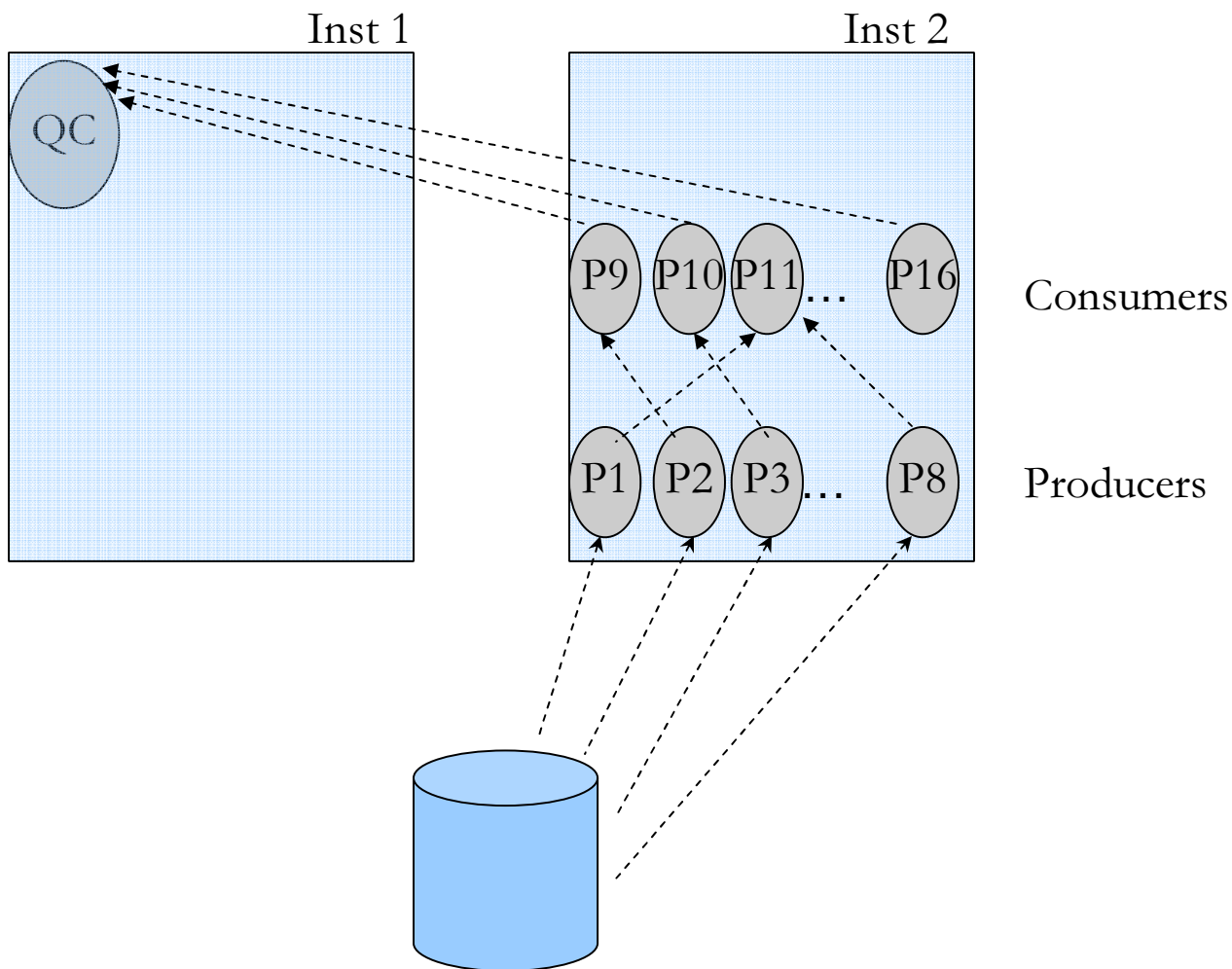
All slaves allocated in the same node reducing Interconnect traffic.

| INST_ID | Username | QC/Slave | Slave Set | SID  | QC SID | Requested DOP | Actual DOP |
|---------|----------|----------|-----------|------|--------|---------------|------------|
| 1       | SYS      | QC       |           | 9897 | 9897   |               |            |
| 2       | - p002   | (Slave)  | 1         | 9870 | 9897   | 8             | 8          |
| 2       | - p000   | (Slave)  | 1         | 9877 | 9897   | 8             | 8          |
| 2       | - p003   | (Slave)  | 1         | 9936 | 9897   | 8             | 8          |
| 2       | - p004   | (Slave)  | 1         | 9927 | 9897   | 8             | 8          |
| 2       | - p005   | (Slave)  | 1         | 9926 | 9897   | 8             | 8          |
| 2       | - p001   | (Slave)  | 1         | 9913 | 9897   | 8             | 8          |
| 2       | - p006   | (Slave)  | 1         | 9909 | 9897   | 8             | 8          |
| 2       | - p007   | (Slave)  | 1         | 9893 | 9897   | 8             | 8          |
| 2       | - p013   | (Slave)  | 2         | 9949 | 9897   | 8             | 8          |
| 2       | - p014   | (Slave)  | 2         | 9883 | 9897   | 8             | 8          |
| 2       | - p011   | (Slave)  | 2         | 9904 | 9897   | 8             | 8          |
| 2       | - p008   | (Slave)  | 2         | 9916 | 9897   | 8             | 8          |
| 2       | - p012   | (Slave)  | 2         | 9920 | 9897   | 8             | 8          |
| 2       | - p009   | (Slave)  | 2         | 9922 | 9897   | 8             | 8          |
| 2       | - p015   | (Slave)  | 2         | 9928 | 9897   | 8             | 8          |
| 2       | - p010   | (Slave)  | 2         | 9882 | 9897   | 8             | 8          |

Instance 2

# PQ Optimization 3

Still, consumers to Query Co-ordinator PQ traffic  
Uses Interconnect.



---

# Analysis

- It's hard to see the issues with AWR report since the PQ interconnect traffic is not correctly printed.
- Oracle Version 10g and 11g has reduced interconnect traffic with innovative optimizer plans. PQ server allocation was optimized to minimize interconnect traffic.
- For example, with few parallel slaves all servers were allocated in the same node reducing GC traffic.
- But, QC and servers were not running in the same node in test case 3.

---

# 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 may 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 - Myths

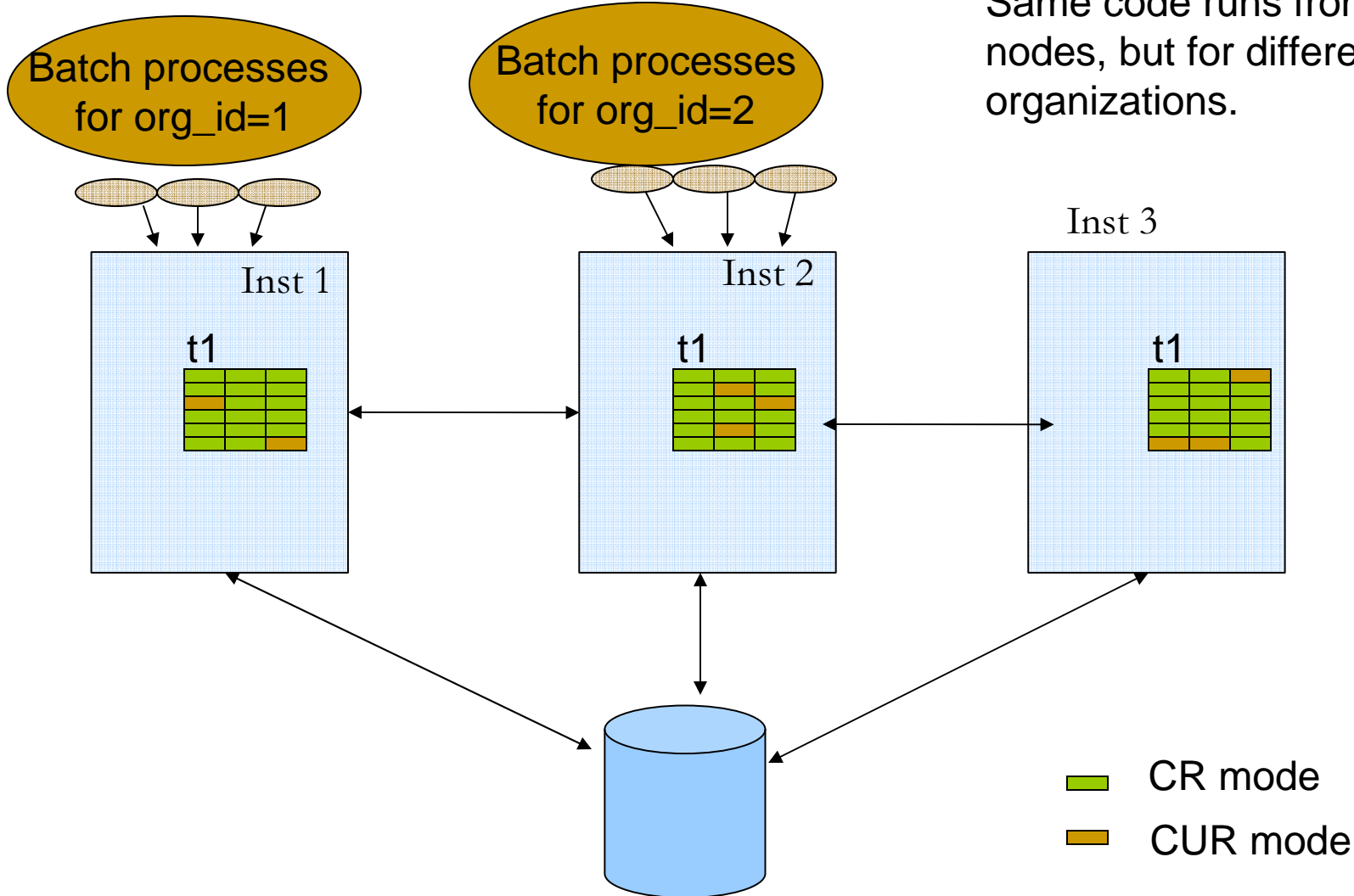
- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

---

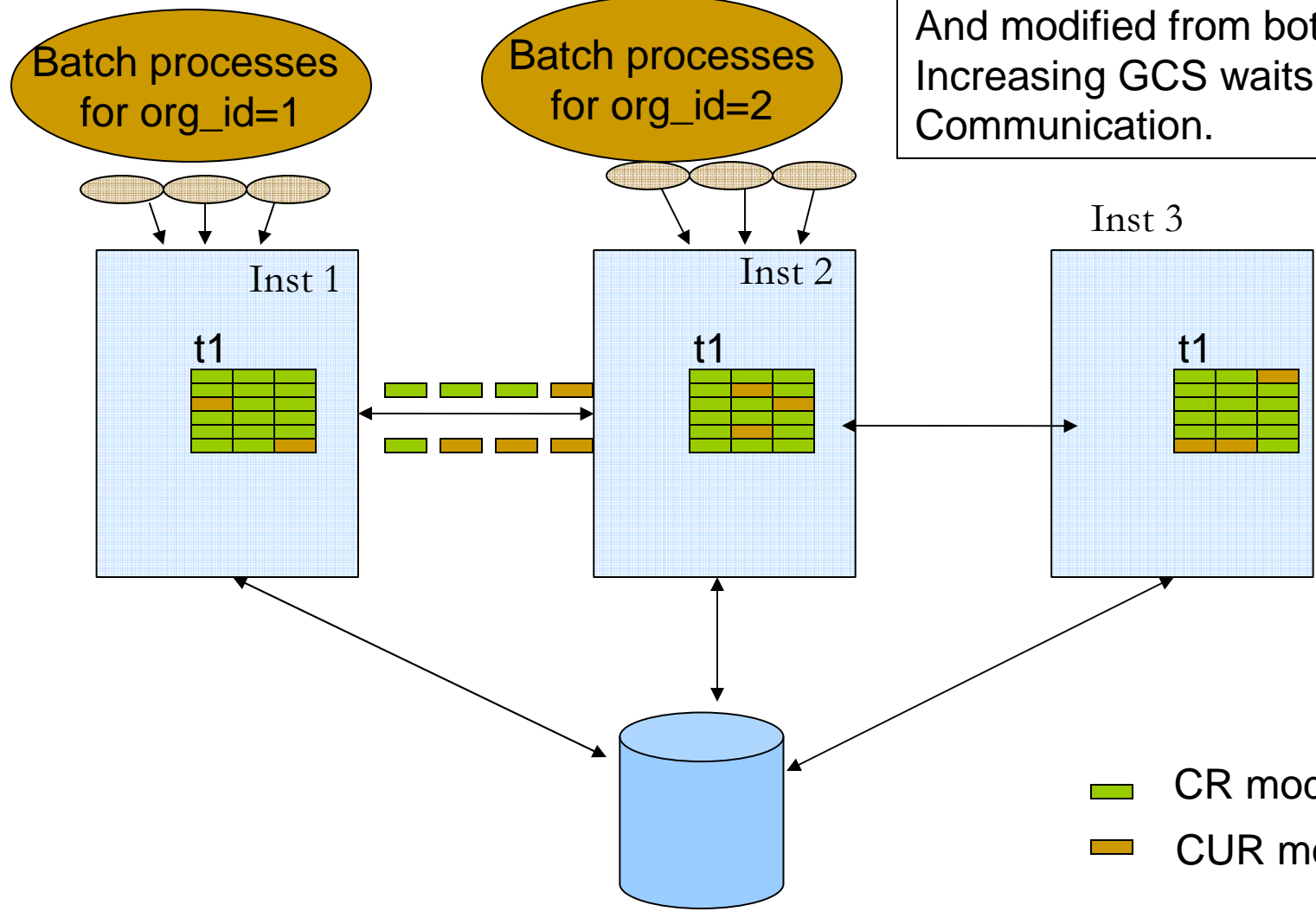
# Concurrent processing?

- A batch process was designed to accept `organization_id` as an argument, and spawns multiple threads for each organization.
- This is multi-node RAC cluster and so
  - Batch process was started in node 1 for `org_id=1`.
  - Same batch process was started in node 2 for `org_id=2`.
- But batch code accessing one set of tables and they are not partitioned.

# Concurrent processing?



# Concurrent processing?



But, same blocks are accessed  
And modified from both nodes,  
Increasing GCS waits and GCS  
Communication.

---

## Are GCS waits cheap?

- Typical Global cache waits are 1-3ms for a decent hardware setup.
- Typical Single block or multi-block reads are 2-4ms for decent hardware.
- But typical access to the buffer in local SGA is just nano seconds (10-100ns).
- Higher global cache waits still introduces latency comparable to I/O based latencies.

# What happened?

- Enormous increase in Global cache waits!

(5 minutes statspack)

Top 5 Timed Events

~~~~~

| Event                   | Waits   | Time (s) | % Total<br>Ela Time |
|-------------------------|---------|----------|---------------------|
| -----                   | -----   | -----    | -----               |
| CPU time                |         | 3,901    | 46.17               |
| PL/SQL lock timer       | 80      | 2,316    | 27.42               |
| global cache cr request | 123,725 | 670      | 7.93                |
| global cache null to x  | 13,551  | 446      | 5.28                |
| buffer busy global CR   | 12,383  | 215      | 2.54                |

- If this has been a non-RAC, access to buffers would be in terms of nano seconds instead of milli-seconds.
- Point is that since these batch processes are accessing same physical blocks, they need to be grouped at physical level and scheduled to run from one node.
- Proper partitioning will help, but global indices needs special attention.

# Tkprof output

- Trace file analysis also shows excessive global cache waits.

Elapsed times include waiting on following events:

| Event waited on                     | Times<br>waited | Max. Wait | Total waited |
|-------------------------------------|-----------------|-----------|--------------|
| log file sync                       | 13              | 0.08      | 0.28         |
| latch free                          | 2185            | 0.17      | 32.76        |
| global cache cr request             | 21719           | 0.31      | 71.18        |
| db file sequential read             | 50              | 0.04      | 0.33         |
| global cache s to x                 | 791             | 0.06      | 5.31         |
| row cache lock                      | 113             | 0.02      | 0.41         |
| global cache null to x              | 4881            | 0.30      | 97.41        |
| buffer busy global CR               | 3306            | 0.23      | 15.26        |
| global cache open x                 | 903             | 0.06      | 5.83         |
| buffer busy waits                   | 1462            | 0.98      | 17.90        |
| global cache busy                   | 644             | 0.98      | 10.36        |
| buffer deadlock                     | 224             | 0.02      | 0.07         |
| global cache null to s              | 89              | 0.15      | 0.77         |
| buffer busy global cache            | 1066            | 0.31      | 27.62        |
| enqueue                             | 302             | 0.17      | 2.61         |
| KJC: wait for msg sends to complete | 102             | 0.00      | 0.00         |
| global cache open s                 | 62              | 0.01      | 0.13         |
| cr request retry                    | 4               | 0.00      | 0.00         |

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

# Sequence operation in RAC

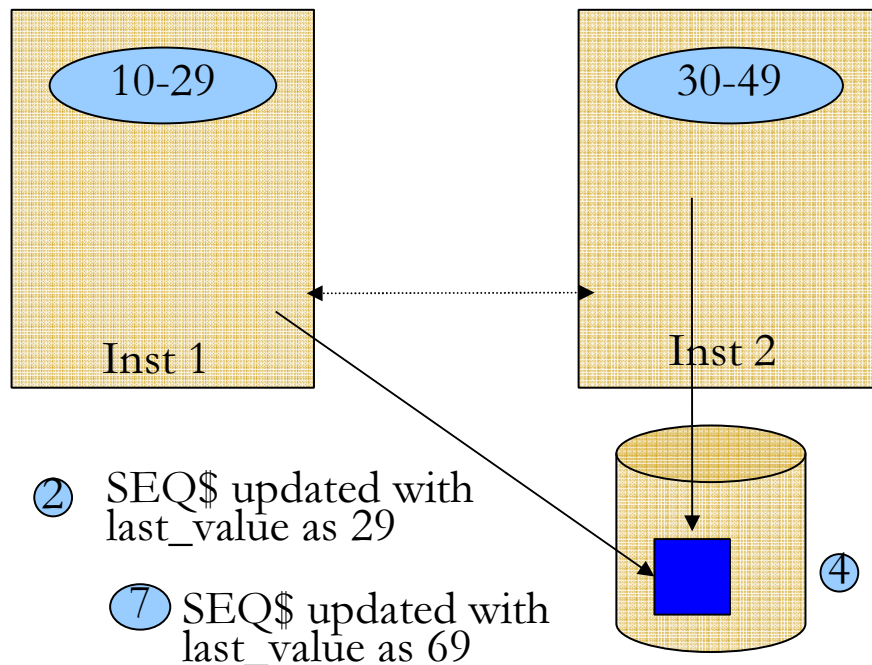
emp\_seq  
cache 20  
start with 10

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

⑤ Subsequent accesses returns 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

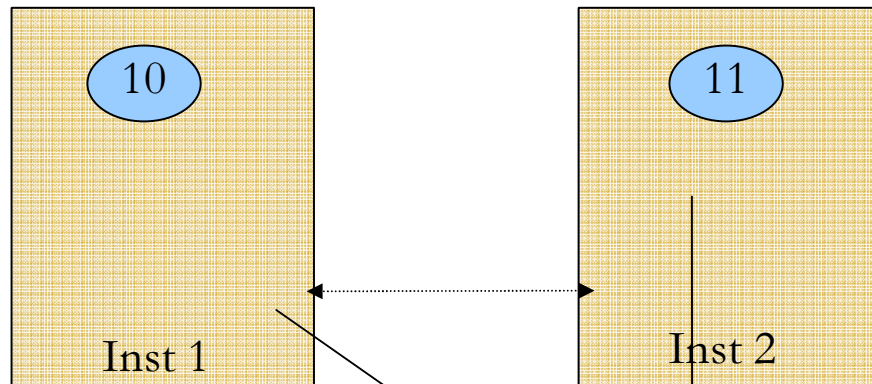
emp\_seq  
nocache  
start with 10

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

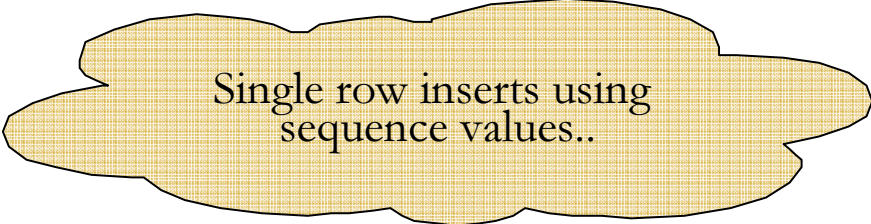
⑦ SEQ\$ updated with  
last\_value as 12

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 – Test case

```
set timing on
alter session set events '10046 trace name context forever, level 8';
declare
 t_v1 varchar2(512);
 t_n1 number :=0;
begin
 for loop_cnt in 1 .. 10000
 loop
 -- Random access
 -- Also making undo blocks to be pinged..
 insert into t1
 select t1_seq.nextval, lpad(loop_cnt, 500, 'x') from dual;
 if mod(loop_cnt, 1000) =0 then
 commit;
 end if;
 end loop;
end;
/
```



Single row inserts using  
sequence values..

# Code executions – one node

```
INSERT INTO T1 SELECT T1_SEQ.NEXTVAL, LPAD(:B1 , 500, 'x') FROM DUAL
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 5.28 | 7.66    | 1    | 794   | 25670   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 10001 | 5.29 | 7.66    | 1    | 794   | 25670   | 10000 |

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,
cache=:7,highwater=:8,audit$=:9,flags=:10 where obj#=:1
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 10000 | 0.32 | 0.30    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 2.74 | 3.04    | 0    | 10000 | 20287   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 20000 | 3.06 | 3.34    | 0    | 10000 | 20287   | 10000 |

# Code executions – two nodes

```
INSERT INTO T1 SELECT T1_SEQ.NEXTVAL, LPAD(:B1 , 500, 'x') FROM DUAL
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 8.02 | 81.23   | 0    | 1584  | 27191   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 10001 | 8.02 | 81.23   | 0    | 1584  | 27191   | 10000 |

Excessive row cache lock  
waits

Elapsed times include waiting on following events:

| Event waited on        | Times<br>Waited | Max. wait | Total waited |
|------------------------|-----------------|-----------|--------------|
| row cache lock         | 5413            | 2.93      | 62.86        |
| gc current block 2-way | 63              | 0.16      | 0.41         |
| gc cr block 2-way      | 46              | 0.00      | 0.06         |

# Code executions – two nodes

5000 blocks transferred  
between nodes..

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,
 cache=:7,highwater=:8,audit$=:9,flags=:10
```

```
where obj#=:1
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 10000 | 0.35 | 0.30    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 4.08 | 11.18   | 0    | 10000 | 20290   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 20000 | 4.44 | 11.49   | 0    | 10000 | 20290   | 10000 |

| Event waited on            | Times<br>waited | Max. wait | Total waited |
|----------------------------|-----------------|-----------|--------------|
| gc current block 2-way     | 5166            | 0.01      | 5.39         |
| log file switch completion | 3               | 0.16      | 0.22         |
| gc current grant busy      | 1               | 0.00      | 0.00         |

# Cache and Order

- If ordering is a must, then use ORDER and cache attributes for sequences:

```
create sequence test_seq order cache 1000;
```

- With 'cache' and 'order', RAC is using GES layer to synchronize the sequence values (at least from 10g onwards).

Elapsed times include waiting on following events:

| Event waited on        | Times<br>waited | Max. Wait | Total waited |
|------------------------|-----------------|-----------|--------------|
| -----                  | -----           | -----     | -----        |
| row cache lock         | 24              | 0.00      | 0.02         |
| DFS lock handle        | 3430            | 0.03      | 2.94         |
| gc current block 2-way | 13              | 0.21      | 0.24         |
| gc cr block 2-way      | 4               | 0.00      | 0.00         |

---

## Sequence- summary

- Nocache sequences increases 'row cache lock' waits. Use cache and Order attributes (tested in 10g).
- Nocache increases interconnect traffic.
- Nocache increases elapsed time.
- If no gaps are needed, control sequence access from just one node or use non-sequence based techniques.

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC. (Skipped due to time constraints)
- ~~Bitmap index performance is worse in RAC. (Skipped)~~

---

## Small tables

- Even small tables must be indexed.
- Excessive full table scans on smaller tables will increase CPU usage.
- This guideline applies to RAC environments too.
- I think, this myth arises due to misunderstanding of the problem.

---

# Small tables

```
set timing on
drop table t_small2;
create table t_small2 (n1 number, v1 varchar2(10)) tablespace users
;
insert into t_small2 select n1, lpad(n1,10,'x')
from (select level n1 from dual connect by level <=10001);
commit;
```

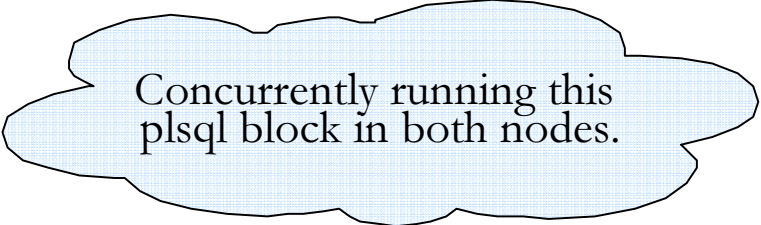
```
select segment_name, sum(bytes)/1024 from dba_segments where segment_name='T_SMALL2'
and owner='CBQT' group by segment_name
```

```
SQL> /
```

| SEGMENT_NAME | SUM(BYTES)/1024 |
|--------------|-----------------|
| T_SMALL2     | 256             |

# Test case

```
alter session set events '10046 trace name context forever , level 8';
set serveroutput on size 100000
declare
 v_n1 number;
 v_v1 varchar2(512);
 b_n1 number;
begin
 for i in 1 .. 100000 loop
 b_n1 := trunc(dbms_random.value (1,10000));
 select n1, v1 into v_n1, v_v1 from t_small2 where n1 =b_n1;
 end loop;
exception
 when no_data_found then
 dbms_output.put_line (b_n1);
end;
/
```



Concurrently running this  
plsql block in both nodes.

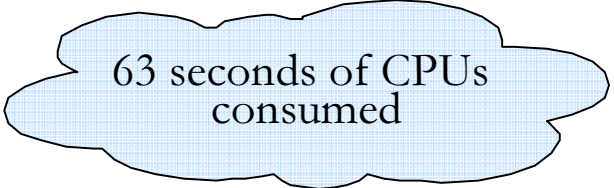
# Results from RAC nodes.

```
SELECT N1, V1
```

```
FROM
```

```
 T_SMALL2 WHERE N1 =:B1
```

| call    | count  | cpu   | elapsed | disk | query   | current | rows   |
|---------|--------|-------|---------|------|---------|---------|--------|
| Parse   | 1      | 0.00  | 0.00    | 0    | 0       | 0       | 0      |
| Execute | 100000 | 2.81  | 3.08    | 0    | 1       | 0       | 0      |
| Fetch   | 100000 | 62.72 | 63.71   | 0    | 3100000 | 0       | 100000 |
| total   | 200001 | 65.54 | 66.79   | 0    | 3100001 | 0       | 100000 |



63 seconds of CPUs  
consumed

```
Rows Row Source Operation
```

```

100000 TABLE ACCESS FULL T_SMALL2 (cr=3100000 pr=0 pw=0 time=63391728 us)⌈
```

# Results with an index

REM adding an index and repeating test  
create index t\_small2\_n1 on t\_small2(n1);

CPU usage dropped from  
63 seconds to 3.5 seconds.

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.00    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 1.64 | 1.61    | 0    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.79 | 1.78    | 23   | 300209 | 0       | 100000 |
| total   | 200001 | 3.43 | 3.40    | 23   | 300211 | 0       | 100000 |

Rows Row Source Operation

```

100000 TABLE ACCESS BY INDEX ROWID T_SMALL2 (cr=300209 pr=23 pw=0 time=1896719 us)⌈
100000 INDEX RANGE SCAN T_SMALL2_N1 (cr=200209 pr=23 pw=0 time=1109464 us)(object id
53783)⌈
```

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

---

# Bitmap index

- Bitmap indices are optimal for low cardinality columns.
- Bitmap indices are not suitable for table with massive DML changes.
- Bitmap index performance does not worsen because of RAC for select queries.
- Of course, having bitmap indices on columns with enormous DML changes is not optimal even in single instance databases.

---

# Test case - Select

```
Create bitmap index t_large2_n4 on t_large2(n4);
```

```
alter session set events '10046 trace name context forever , level 8';
```

```
set serveroutput on size 100000
```

```
declare
```

```
 v_n1 number;
```

```
 v_v1 varchar2(512);
```

```
 b_n1 number;
```

```
begin
```

```
 for i in 1 .. 100000 loop
```

```
 b_n1 := trunc(dbms_random.value (1,10000));
```

```
 select count(*) into v_n1 from t_large2 where n4 =b_n1;
```

```
 end loop;
```

```
exception
```

```
 when no_data_found then
```

```
 dbms_output.put_line (b_n1);
```

```
end;
```

```
/
```

# Result – Single thread

```
SELECT COUNT(*) FROM T_LARGE2 WHERE N4 =:B1
```

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.00    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 2.87 | 2.93    | 2    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.86 | 2.03    | 78   | 200746 | 0       | 100000 |
| total   | 200001 | 4.73 | 4.97    | 80   | 200748 | 0       | 100000 |

```
Rows Row Source Operation
```

```

100000 SORT AGGREGATE (cr=200746 pr=78 pw=0 time=2854389 us)†
100000 BITMAP CONVERSION COUNT (cr=200746 pr=78 pw=0 time=1766444 us)†
```

# Result – From two nodes

```
SELECT COUNT(*)
FROM
 T_LARGE2 WHERE N4 =:B1
```

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.01    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 2.82 | 2.95    | 0    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.90 | 1.94    | 3    | 200753 | 0       | 100000 |
| total   | 200001 | 4.73 | 4.90    | 3    | 200755 | 0       | 100000 |

Misses in library cache during parse: 1

---

# References

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

[www.ixora.com.au](http://www.ixora.com.au)

- Jonathan Lewis' website

[www.jlcomp.daemon.co.uk](http://www.jlcomp.daemon.co.uk)

- Julian Dyke's website

[www.julian-dyke.com](http://www.julian-dyke.com)

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

- Tom Kyte's website

Asktom.oracle.com