BOOT CAMP: RAC PERFORMANCE TUNING

Introduction

This paper is to explore concepts and techniques behind performance tuning in RAC (Real Application Clusters) environments. This paper is primarily designed to introduce methods, concepts and tools for performance analysis in RAC. Some familiarity with RAC technology is needed as a prerequisite for this presentation.

1. Global cache performance

In RAC, multiple instances can be mounted in a database. In a single instance mode, there is one-to-one relationship between an instance and a database. In RAC, there is a many-to-one relationship between instance(s) and a database. This means that various resources are co-ordinated, blocks transferred between various instances.

RAC traffic can be divided in to three major groups: (Not an exhaustive list)

- 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
  - etc..

Consistent Read (CR) blocks

For consistent read blocks, Blocks are requested by other nodes for mostly read operations. User processes running in local node requests LMS background processes running in the remote nodes for a block or set of blocks. These user requests include version of the block.

A process finds that local instance does not own a PR lock on that block range. One of three things can happen for a CR request:

1. A user process requests master instance. No instance currently owns the block. So, Master node grants a PR lock (Protected Read) request to the requesting instance. User process reads the block from the disk. (or)
2. Master node and owner nodes are one and the same. That block is in the Master instance. Master instance sends the block to instance 1 and grants a PR lock. (or)
3. Master node and owner nodes are not the same. Master node sends a message to owner node to transfer the block and sends a grant request to the requestor.

In Figure 1-1, User process identifies that there are no suitable block image on the local instance. So, master node of the block must be queried to co-ordinate global resources. Blocks are globally co-ordinated by GCS (Global Cache Services) layer.

1. User process in instance 1 requests master for a PR lock on the block.
2. Assuming no current owner, master node grants the lock to inst 1.

3. User process in instance 1 reads the block from the disk and holds PR.

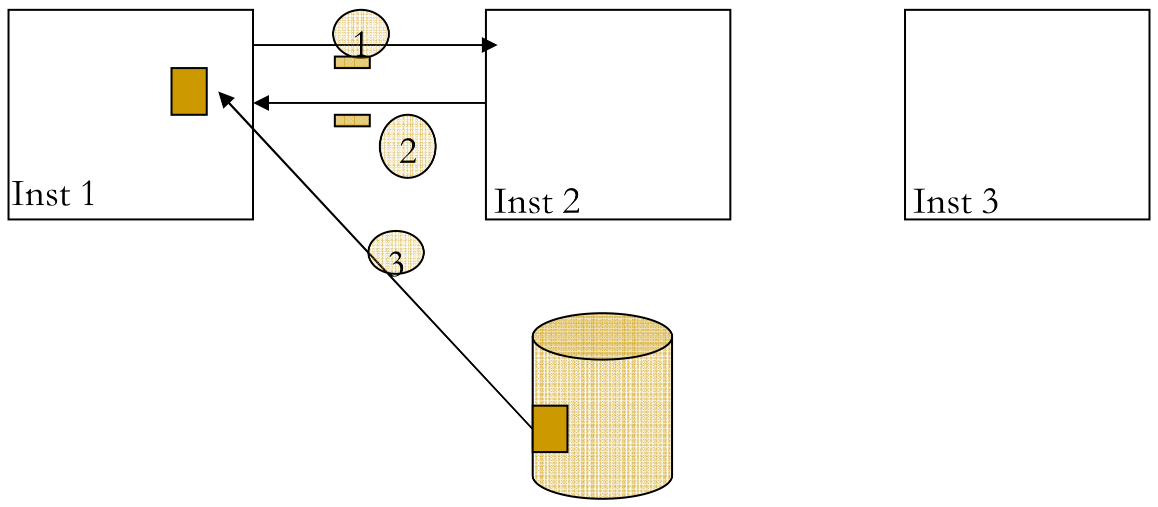

Figure 1-1: CR disk read

In Figure 1-2, block is in the master instance cache, meaning both master instance and owner instance one and the same. So, basic messaging between the instances is:

1. User process in instance 1 requests master for the block in PR mode.
2. Current owner (2) holds the block in Protected Read (PR) mode and so master grants a PR to instance 1.
3. In this case master instance and owner instance are the same. Owner instance LMS process serves the block to the user process in instance1.

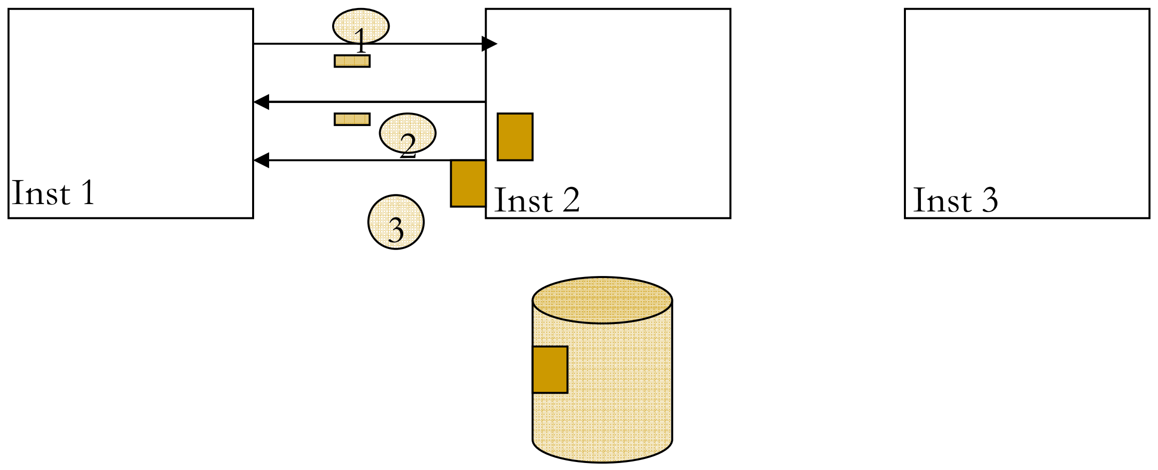

Figure 1-2: CR immediate 2-way transfer

Suppose, if the master and the current owners of the blocks are different, then an additional message occurs. Figure 1-3 represents this transfer.

1. User process in instance 1 requests master for the block.
2. Current owner holds Protected Read (PR) and so master grants a PR lock to instance 1.
3. Master sends the owner to send the block to instance1.

4. Owner instance LMS process serves the block to the user process in instance1

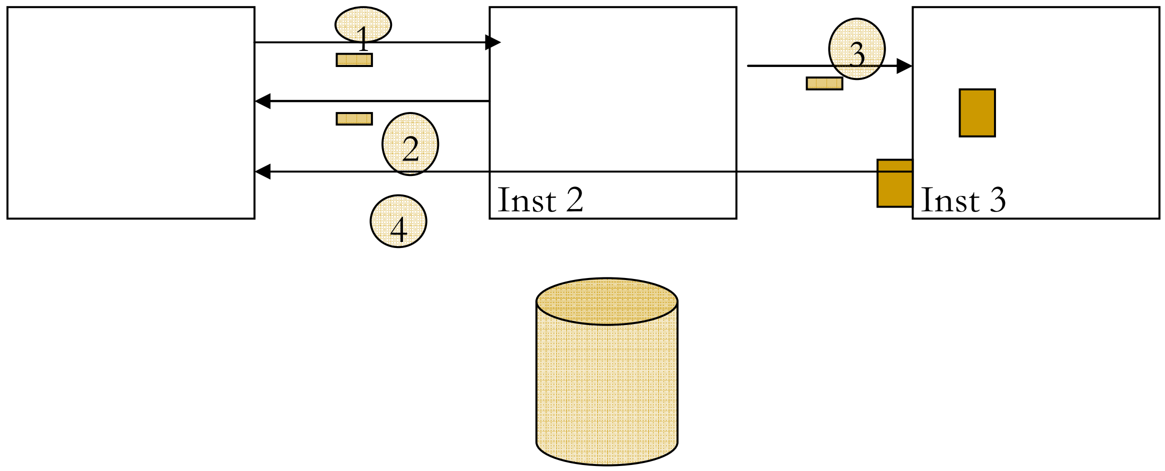

Figure 1-3: CR immediate 3-way transfer

<u>CR complications</u>

In reality, these transfers are not so simple. Many different complexities arise and detailed discussion is beyond the scope of this paper. Some complications are:

What if the block has uncommitted transaction?
What if the block SCN is ahead the query environment SCN?
What if the block is undergoing changes and the buffer is pinned by another process?
What if the block is transaction table, undo block?
What if the block is index branch block that was just split?

<u>CURrent mode (CUR) blocks</u>

Blocks requested by other nodes to modify the blocks. LMS will send the current mode block to the requesting process. LMS process will convert the block in the local instance to a Past Image (PI). For both CR and CUR transfers, at the most, only three instances participate in the global cache transfer.

For CUR mode blocks, LGWR must do a log flush before sending the block to the remote instance. In the case of CUR mode transfer[1] sending the block to the remote instance also has same transactional issues associated with commit mechanism. So, before sending a block to the remote instance, LMS process will request a log flush. LGWR must complete a log flush before LMS process can send the block to the remote instance.

Figure 1-4 shows the communication mechanism between various layers involved in a CUR mode block transfer. We can see that various technical stacks and processes are involved in transferring blocks from one instance to another instance cache.

[1] Log flush need to happen for CR mode transfer if the block is considered "busy", meaning that LMS process applied undo records to create a consistent version of the block. In this case, LMS process must request LGWR process for a log flush. In RAC, even select statement can suffer from LGWR performance issues, but this log flush event is essential to maintain data integrity.

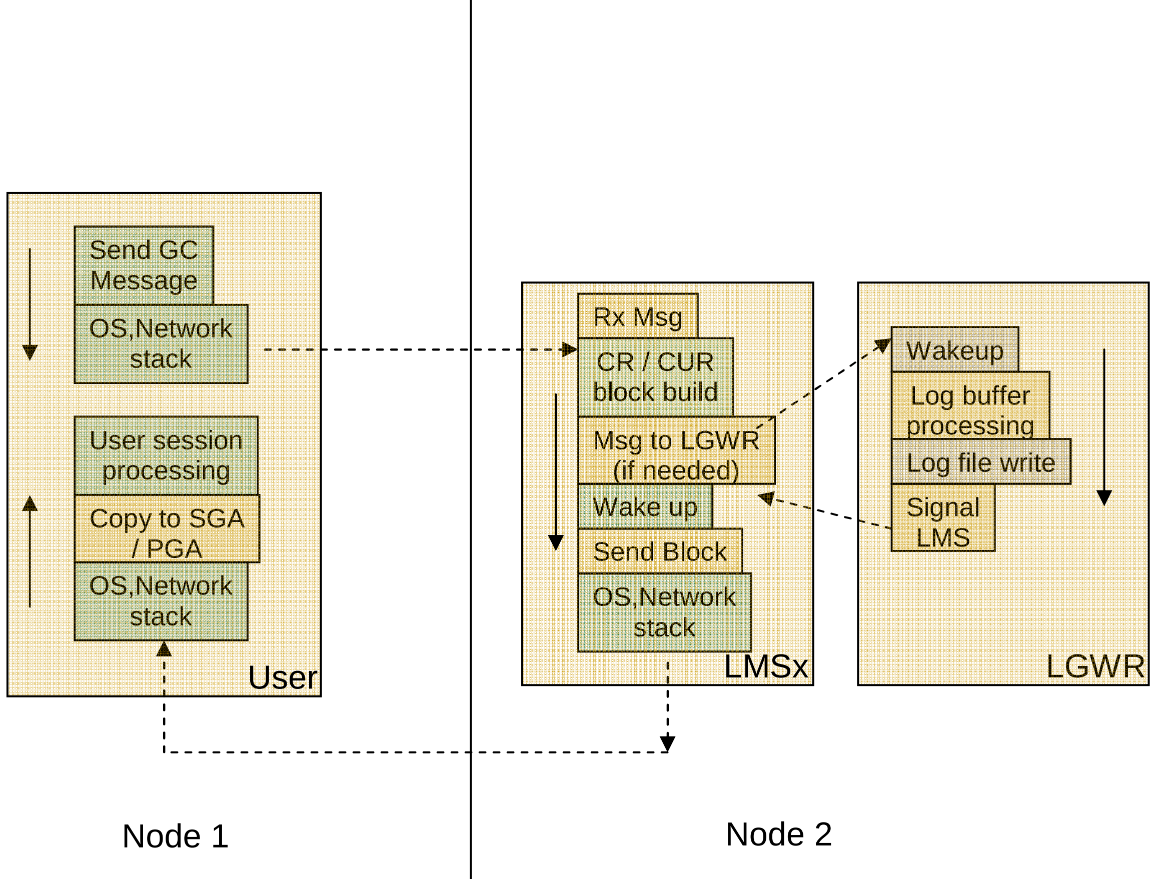

Figure 1-4: A typical CUR transfer with directional communication

<u>GC CR latency</u>

Generally, GC CR latency can be broken down to the following components. Following formula depicts an approximate representation of GC CR latency. Any bottlenecks in the stack can cause spikes in the Global Cache latency.

```
GC CR latency ~=
                Time spent in sending message to LMS +
                LMS processing (building blocks etc) +
                LGWR latency ( if any) +
                LMS send time +
                Wire latency
Listing 1-1 GC CR latency
```

Table 1-1 shows various global cache timing statistics that can be used to understand global cache performance issues. Average global cache response time was 6.2ms and much of that can be attributed to gc cr block flush time, which is due to LGWR I/O performance issues.

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

Table 1-1: GC CR breakup

<u>GC CUR latency</u>

GC CUR mode transfer can be broken down using following approximate formula:

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

Listing 1-2: GC CUR latency

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

<u>A word of caution</u>: Don’t derive averages using gv$ views. It can be misleading. Data from gv$views are aggregated and can be misleading. For example, listing 1-3 is querying gv$ views to calculate average GC receive time. But, this 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
```

Listing 1-3: Average CR receive time (misleading)

Use custom scripts to understand the global cache performance issues. You can use my script to print global cache performance data for the past minute. Download from scripts archive: Please refer to the presentation for the complete script output as the listing 1-4 is not showing all column output of that script. Script is also attached in appendix #1, but formatting is not good in the appendix, so please download from the script archive.

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

gc_traffic_print.sql script showed CR average response time of 13.8 ms in the last minute. Indeed, Instance 1 was suffering from CR response time of 13.82ms.

```
---------|--------------|---------|...
Inst     | CR blocks Rx | CR time |
---------|--------------|---------|
1        |         40999|    13.82|
2        |         12471|     5.85|
3        |         28795|     4.11|
4        |         33105|     4.54|
---------|--------------|---------|...
```

Listing 1-4: CR average response time for the last minute

During the same time frame, script output from the gv$ views shows that the average response time is 6.69ms, which is misleading. These instances have been running for only a short period of time, and if these have been running for longer time, then this could be far misleading.

```
 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
```

Listing 1-5: CR average response time from gv$ views

It is also important to review performance data from all nodes. Of course, it is easy to create AWR report from all nodes awrrpt_all_gen.sql. This script is also available from orainternals script archive or my blog entry.

Listing 1-6 shows the CR break response time. Much of the time is spent CR flush time, which is due LGWR performance.

```
---------|-----------|---------|-----------|----------|...
Inst     | CR blk Tx | CR bld  | CR fls tm | CR snd tm|
---------|-----------|---------|-----------|----------|
2        |      67061|      .08|        .88|       .23|
3        |      38207|      .17|       2.19|       .26|
4        |      72820|      .06|       1.76|        .2|
5        |      84355|      .09|       2.42|       .23|...
```

Listing 1-6: Breakdown of CR response time

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 in v$session_wait, but that 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.

As we saw in the prior section, there are 2-way and 3-way block transfer events.
GC CR block 2-way
GC CR block 3-way
GC CUR block 2-way
GC CUR block 3-way
Even if there are many instances, only three instances participate in a block transfer. But, flush messages can be sent to all instances in few cases.

gc cr grants:
Wait events 'gc cr grant 2-way' and 'gc current grant 2-way' indicates
Block is not in any cache
Permission granted to read from the disk.

```
WAIT #6: nam='gc cr grant 2-way' ela= 567 p1=295 p2=770871 p3=1
obj#=5153800 tim=817052932927

WAIT #6: nam='db file sequential read' ela= 11003 file#=295
block#=770871 blocks=1 obj#=5153800 tim=817052943998
```

Congestion:
Congestion indicates that LMS processes were not able to service fast enough: Focus on LMS processes and usual culprits are load, SQL performance or longer CPU queue etc.

gc cr grant congested, gc current grant congested
gc cr block congested, gc current block congested

2. RAC background process tuning

Typically, Global cache CR waits such as 'gc cr grant 2 way' (10g), 'global cache cr request' latency increases due to global communication or cache fusion latencies. This global cache performance is blamed on interconnect network hardware or path. But, interconnect is probably performing fine and global cache latency is caused by LMS process latencies.

1. More LMS processes? Typical response to this issue, if LMS processes have been identified as a culprit, to increase number of LMS processes. This change has negative effect on performance. If a RAC node is suffering from resource starvation, then adding more processes typically results in more performance degradation.

   Modern CPUs have cache affinity designed in them and so, processes tend to be executed in the same CPU sets to improve TLB efficiencies. By adding more LMS processes, TLB misses and cache thrashing increases. This can be evidenced by carefully monitoring xcalls/migrates/TLB misses in mpstat and trapstat outputs. In summary, few busy LMS processes are better then many quasi-busy LMS processes.

   Same number of LMS processes as # of interconnects or number of remote nodes is a good starting point. For example, in a four node cluster, three LMS processes per node is usually sufficient.

2. LMS process priority? In release 9i, increasing LMS process priority to RT or FX (with larger CPU quanta scheduling schemes) helps. Oracle Corporation has already identified this potential issue and in release 10g, this issue is taken care of, and LMS processes are running with RT priority. This alleviates many issues with global cache transfer.

   Two parameters control this behaviour[2]:
   a. _high_priority_processes: This controls processes that will have higher priority and v$parameter specifies this as "High priority process name mask" with default value of LMS*.
   b. _os_sched_high_priority: This controls whether OS scheduling high priority is enabled or not and defaults to 1.

3. Reducing LMS process to a very minimal level also has a side effect, so don't reduce to 1. From Version 10g onwards, instance reconfiguration is done in parallel and LMS process are

[2] In some cases, I have noticed that running these processes in RT can have negative effect too. Especially, if the node size is very small and overloaded, then kernel and other critical processes doesn't get enough CPU. This can result in node evictions due to timeout failures.

involved in instance reconfiguration. If you reduce parallelism to a small number then instance reconfiguration performance can suffer. This affects object remastering too as the object remastering involves instance reconfiguration.

In lieu of discussion about LMS processes, it is also prudent to consider following:

1. LGWR processes should also run with higher priority, in addition to LMS processes (only applicable to 9i and above).
2. Better write throughput for redo log files is quite essential. Higher interconnect traffic eventually lead higher or hyperactive LGWR. Consider using direct I/O and asynchronous I/O for redo log files. Asynchronous I/O is quite important if there are multiple log group members as LGWR has to wait for I/O to each member complete sequentially, before releasing commit markers.
3. In solaris platform, priocntl can be used to increase priority of LGWR and LMS processes.
   ```
   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}`
   ```
4. Binding: Another option is bind LMS and LGWR processes to specific processors or processor sets. This should reduce TLB thrashing and improve efficiency of LMS/LGWR processes.
5. It is worthwhile to fence interrupts to one or two processor sets and use remaining processor sets for LMS/LGWR processes. Refer psradm for further reading in Solaris environments. But, of course, processor binding is only applicable to servers with high # of CPUs and has repercussions on availability.

3. Interconnect issues, lost packets and network layer

Statistics 'gc blocks lost' is useful in debugging performance issues due to interconnect problems. Consistent high amount of 'gc blocks lost' is an indication of problem with underlying network infrastructure. (Hardware, firmware etc). It is important to understand which tech stack is causing the issue and usually, it is an inter-disciplinary exercise. Ideal value is near zero. But, only worry about this, if there are consistently higher values.

Higher amount of lost packets can lead to timeouts in GC traffic wait events. Many processes will be waiting for place-holder events such as 'gc cr request' and 'gc current request'. Use the total_timeouts column in v$system_event to see if the timeouts are increasing.

Figure 3-1 shows layers involved in the communication for UDP protocol. Detailed discussion about various layers in the communication is out of the scope. We will briefly discuss the UDP communication issues here. UDP is a "send-and-forget" type protocol. There is no acknowledgement like TCP protocol. 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.

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 CPU scheduling latencies and lost packets.

If there is not sufficient amount of UDP receive/transmit buffers, then it can lead to packets being dropped. It is generally a good practice to increase udp_xmit_hiwat, udp_recv_hiwat to 256KB. Increase udp_max_buf to 8M.

If your network supports, then use jumbo frames. Typically, MTU size is set to 1500 bytes. A 8K block must be split in to 6 packets for one DB block transfer. Disassembly and Assembly is a costly process and can increase CPU usage. Use Jumbo frames with a MTU setting of ~9000 bytes. Of course, every component in the path must support jumbo frame, other wise, packets can be corrupted and dropped.

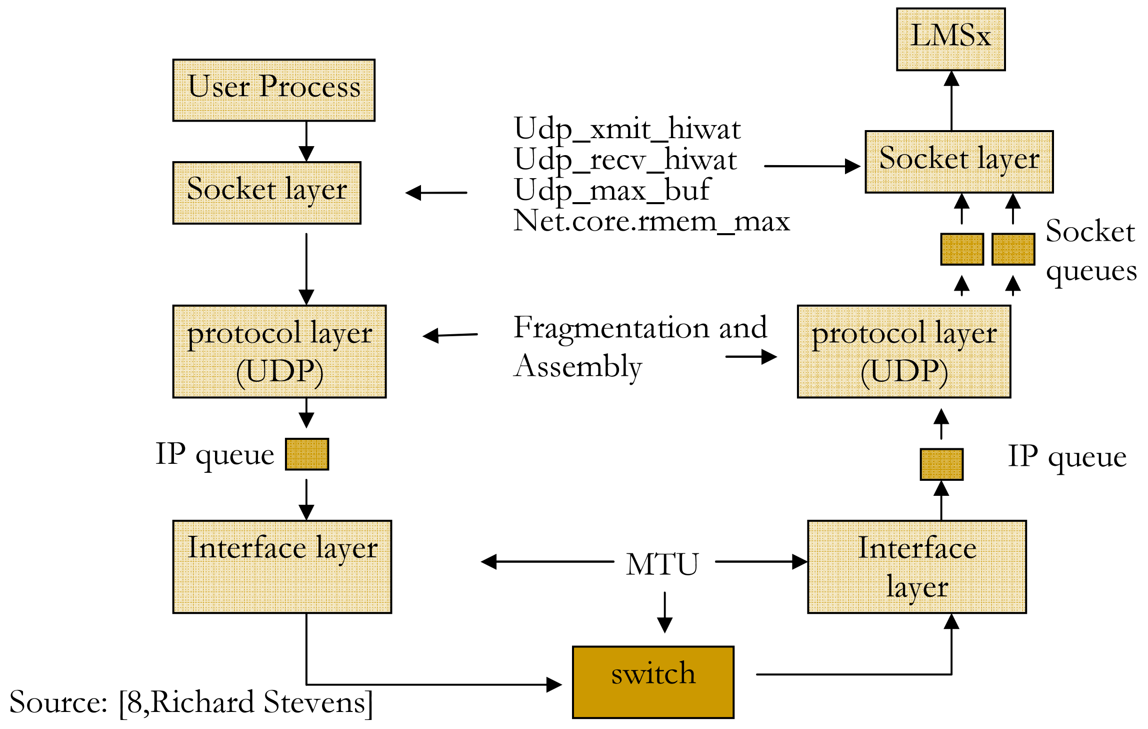

Figure 3-1: UDP tech stack

4. Effective use of parallel query

In Real Application Cluster databases, a query can be executed in parallel allocating slaves from multiple nodes. For example, in a 6 node RAC cluster, a parallel query can allocate 4 slaves from each node with a combined parallelism of 24. Allocating slaves from multiple nodes to execute SQL statements *can* cause performance issues. Let me be very clear here, I am not suggesting that inter-node parallelism never should be used, only that all ramifications carefully considered.

Intra instance parallelism is controlled by parameters such as parallel_min_servers, parallel_max_servers and other parallel* parameters. Inter-node parallel execution is controlled by RAC specific parameters. Specifically, instance_group and parallel_instance_group parameters determine the allocation of parallel servers in the local and remote nodes (until version 10g). From Oracle Database version 11g onwards, you need to use database services to implement inter-node parallel execution.

Parameter instance_group dictates the group membership of an instance. For example, we could specify that instance group OLTP is made up of instances 1,2 and 5. Parameter parallel_instance_group dictates the parallel slave instance group. It is easier to explain these two parameters with few examples:

Say, there are three nodes in this RAC cluster inst1, inst2 and inst3.

Scenario #1:
In this first scenario, an instance_group 'all' encompassing all three instances is setup and the parameter parallel_instance_group is set to 'all'. So, parallel slaves for a parallel SQL statements initiated from any instance *can* be allocated from all three nodes. In the figure 4-1 show how parallel slaves can be executed from all three nodes with global co-ordinator process is executing in node 1. Further, these slaves are not randomly allocated and there is much intelligence and usually least loaded nodes are selected, execution plan of the SQL statements, type of SQL statement etc.

Parameters:

```
inst1.instance_groups='inst1','all'
inst2.instance_groups='inst2','all'
inst3.instance_groups='inst3','all'
inst1.parallel_instance_group='all'
```

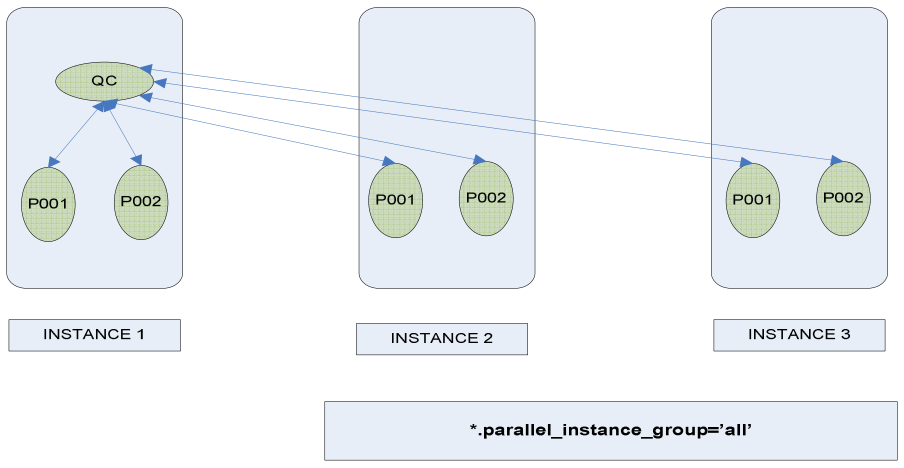

Figure 4-1 Parallel query slaves allocated in all three nodes

Scenario #2:
In this scenario, both inst1 and inst2 are members of instance_group inst12. But, inst3 is not a member of this instance_group. Figure 4-2 depicts a scenario where only two nodes are participating to execute this parallel SQL statement.

```
inst1.instance_groups='inst12','all'
inst2.instance_groups='inst12','all'
inst3.instance_groups='inst3','all'
inst1.parallel_instance_group='inst12'
```

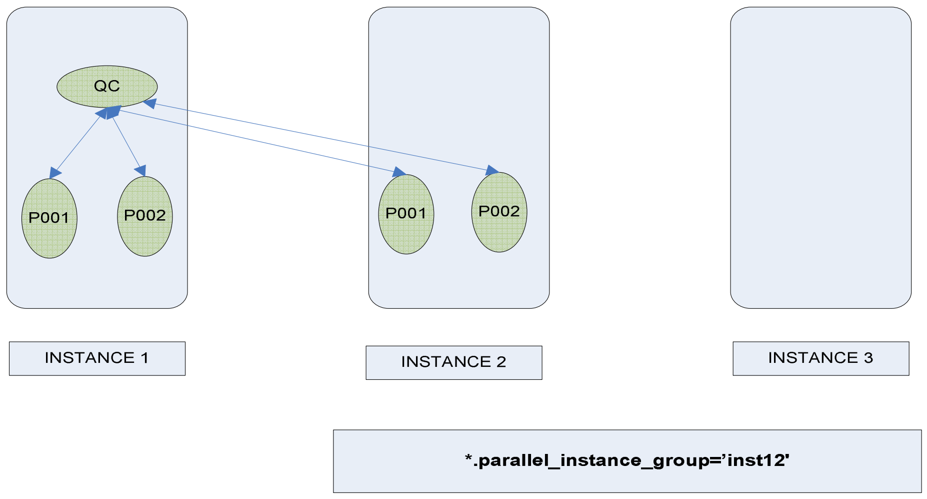

Figure 4-2 Parallel query slaves allocated in two nodes

Parallel Query and RAC optimization

We also need to discuss recent optimization changes, especially, in 10g and 11g with PQ slaves allocation, communication and RAC. We will consider two SQL statements to establish these optimization features:

SQL 1

Consider the SQL statement in Listing 1.3 performing a simple aggregation of a big table with 8 parallel slaves. Parallel_instance_group was set to inst12 so as to allocate parallel slaves from instance 1 and instance 2. Due to aggregation step, there can be twice as many as slaves allocated and so 16 slaves are allocated from two instances for this SQL statement.

```
select /*+ parallel ( t1, 8) */
      min (t1.CUSTOMER_TRX_LINE_ID +t1.CUSTOMER_TRX_LINE_ID ) ,
      max ( t1.SET_OF_BOOKS_ID+t1.set_of_books_id ),
      avg(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;
```

Listing 4-1 Parallel SQL with simple aggregation

Listing 1.4 shows the allocation of slaves between two instances. Eight slaves are allocated in instance 1 and 8 slaves are allocated in instance 2, Query coordinator (QC) process is running from instance 1. Notice that amount of communication between query co-ordinator process and the slaves running in instance 2 under the size column. Size column[3] is printing the amount of interconnect traffic between QC process running in instance 1 and slaves running in instance 2. It looks like parallel slave processes are aggregating data and sending only aggregated results to the Query Co-ordinator process. Query co-ordinator must perform second level aggregation to derive final results. In a

[3] Size column is calculated from a Solaris tool. Until 11g, PQ traffic is not accounted in cluster traffic in the database. These measurements were taken in Oracle Database version 10g. To measure UDP traffic over the interconnect, a Solaris tool udpsnoop.d was used. This tool is part of dtrace tool kit.

nutshell, for these types of SQL statements with simple aggregation, interconnect traffic is kept to a minimum[4].

```
   INST Username     QC/   Slave SID    QC     Req. Act. Size
                     Slave   Set                 DOP  DOP
------- ------------ ----- ----- ----- ------- ---- ---- -----
     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
```

Listing 4-2 Parallel slave allocation and message size

SQL 2

Consider the SQL statement in Listing 1.5. This SQL statement is performing slightly complex aggregation. Two tables are joined in this example using HASH JOIN technique and then aggregated to derive final result set.

```
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
;
```

Listing 4-3 Parallel SQL for a slightly complex execution

Listing 1.6 shows the slave allocation for the SQL statement in listing 4-4. You can see that 16 slaves were allocated in instance 1 and 16 slaves were allocated in instance 2. Also notice the amount of interconnect traffic between the PQ slaves in instance 2 and the QC process. Approximately, total interconnect traffic for this SQL statement in one direction was 2.4GB.

```
1 SYS QC  9868   9868
1  - p007      (Slave)    1 9890   9868     1616
1  - p006      (Slave)    1 9949   9868     1616
1  - p005      (Slave)    1 9871   9868     1616
1  - p004      (Slave)    1 9878   9868     1616
1  - p003      (Slave)    1 9898   9868     1616
```

[4] Size column is empty for local slaves since udpsnoop.d is measuring traffic over interconnect and local slaves are not using the interconnect for communication with QC process.

```
1   - p002      (Slave)    1 9969    9868      1616
1   - p001      (Slave)    1 9897    9868      1616
1   - p000      (Slave)    1 9924    9868      1616

1   - p015      (Slave)    2 9867    9868      1616  128020044
1   - p014      (Slave)    2 9908    9868      1616  132011920
1   - p013      (Slave)    2 9917    9868      1616  127770024
1   - p012      (Slave)    2 9938    9868      1616  128154240
1   - p011      (Slave)    2 9877    9868      1616  183490248
1   - p010      (Slave)    2 9895    9868      1616  181900075
1   - p009      (Slave)    2 9942    9868      1616  181828128
1   - p008      (Slave)    2 9912    9868      1616  124428800

2   - p023      (Slave)    1 9904    9868      1616
2   - p022      (Slave)    1 9941    9868      1616
2   - p021      (Slave)    1 9928    9868      1616
2   - p020      (Slave)    1 9870    9868      1616
2   - p019      (Slave)    1 9880    9868      1616
2   - p018      (Slave)    1 9934    9868      1616
2   - p017      (Slave)    1 9910    9868      1616
2   - p016      (Slave)    1 9913    9868      1616

2   - p031      (Slave)    2 9902    9868      1616  127068484
2   - p030      (Slave)    2 9883    9868      1616  126904080
2   - p029      (Slave)    2 9882    9868      1616  127767353
2   - p028      (Slave)    2 9920    9868      1616  128154145
2   - p027      (Slave)    2 9916    9868      1616  128096875
2   - p026      (Slave)    2 9903    9868      1616  182490908
2   - p025      (Slave)    2 9893    9868      1616  181899025
2   - p024      (Slave)    2 9897    9868      1616  181611641
```

Listing 4-4 Parallel SQL for a slightly complex execution

This convincingly prove that amount of interconnect communication increases if parallel slaves are allocated from multiple instances to execute a parallel query. Essentially, if the application needs to use parallelism across many instances, then interconnect hardware must be properly designed and configured.
A white paper supplied by Oracle Corporation also has recommendations along the same line. Listing 4-5 shows few lines from the white paper available free from the URL: http://www.oracle.com/technology/products/bi/db/11g/pdf/twp_bidw_parallel_execution_11gr1.pdf

> "If you use a relatively weak interconnect, relative to the I/O bandwidth from the server to the storage configuration, then you may be better of restricting parallel execution to a single node or to a limited number of nodes; inter-node parallel execution will not scale with an undersized interconnect. As a general rule of thumb, your interconnect must provide the total I/O throughput of all nodes in a cluster (since all nodes can distribute data at the same point in time with the speed data is read from disk); so, if you have a four node cluster, each node being able to read 1GB/sec from the I/O subsystem, the interconnect must be able to support 4 x 1GB/sec = 4GB/sec to scale linearly for operations involving inter-node parallel execution. It is not recommended to use inter-node parallel execution unless your interconnect satisfies this requirement (or comes very close)."

Listing 4-5 Few lines from Oracle White paper.

In summary, if you are planning to allow parallelism to spawn multiple nodes, verify that interconnect hardware can support it. Especially, if the SQL statements are executed concurrently by many processes then interconnect hardware can quickly become a bottleneck. In fact, this is a customer performance issue we resolved, in which, parallelism on many queries were turned on causing enormous performance issues. Since the Global cache statistics for Parallel queries are not captured in the statspack they were not easily visible.

5. Dynamic remastering

Before reading the block, a user process must request master node of the block to access that block. Statistics 'gc remote grants' keeps track number of remote requests. If the local node is the master of that block, then GC affinity locks acquired, which is more efficient than remote grants.

Typically, a batch process will access many blocks aggressively. Performance can be improved if those blocks are mastered in the local node avoiding costly remote grants. That is exactly what 'dynamic remastering' feature is designed to achieve. If an object is detected to be accessed excessively from one node, then all blocks belonging to that object is remastered locally, avoiding remote grants.

This is especially useful, if there is an application level node affinity. For example, all FA users go to node 1, supply chain users to node 2 etc. In these cases, it is beneficial to master all FA related objects to node 1 and all supply chain related objects to node 2 etc.

In 10gR1, dynamic remastering is at file level, not so useful. All blocks belonging to a file are remastered to a node. In 10gR2 onwards, dynamic remastering is at object level. All blocks of an object is remastered to a node. Three background processes work together to implement dynamic remastering functionality

High level overview in 10gR2

Following gives a brief summary of the activity.

1. LCK0 process maintains object level statistics and determines if remastering must be triggered.
2. 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
   ```
3. LMON performs reconfiguration and uses LMS processes to do actual work.
   ```
   *** 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

Three parameters affects behaviour of dynamic remastering.

```
_gc_affinity_limit
 _gc_affinity_time
 _gc_affinity_minimum
```

Parameter	Value	Comment
_gc_affinity_limit	50	Not documented well, but, it is number of times a node should access an object more than other nodes.
_gc_affinity_time	10	Frequency in seconds to check if remastering to be triggered or not.
_gc_affinity_minimum	600	determines number of DRM requests to enqueue

Default for these parameters may be too low in a busy high-end instance or if you don't have application level node affinity. 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 three parameters are completely removed. Three new parameters are introduced:
 _gc_affinity_locking
 _gc_affinity_locks
 _gc_affinity_ratio

Sorry, I have not tested these parameters thoroughly yet.

```
Top 5 Timed Events                                         Avg %Total
~~~~~~~~~~~~~~~~~~                                        wait   Call
Event                                 Waits    Time (s)   (ms)   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
```

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
...
```

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
```

<u>Manual remastering</u>

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)
```

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
```

About the author

Riyaj Shamsudeen has 15+ years of experience in Oracle and 14+ years as an Oracle DBA/Oracle Applications DBA. He is the principal DBA behind ora!nternals (http://www.orainternals.com - performance/recovery/EBS11i consulting company). He specializes in RAC, performance tuning and database internals and frequently blogs about them in http://orainternals.wordpress.com . He has authored many articles such as internals of locks, internals of hot backups, redo internals etc. He also teaches in community colleges in Dallas such as North lake college and El Centro College. He is a proud member of OakTable network.

References

1. Oracle support site. Metalink.oracle.com. numerous documents
2. Innovate faster with Oracle database 11g- An Oracle white paper
3. My blog: http://orainternals.wordpress.com
4. Oracle database 11g: An Oracle white paper:
5. Internal's guru Steve Adam's: http:// www.ixora.com.au
6. Jonathan Lewis: http://www.jlcomp.daemon.co.uk
7. Julian Dyke: http://www.julian-dyke.com
8. Costing PL/SQL functions: Joze Senegachnick – HOTSOS 2006
9. Pythian blog: Query result cache: by Alex fatkulin:
http://www.pythian.com/authors/fatkulin
10. Metalink note : 453567.1 on result cache
11. Oracle database 11g: secure files : An Oracle white paper.
12. Securefile performance:
http://www.oracle.com/technology/products/database/securefiles/pdf/securefilesperformancepaper.pdf
13. shmdt calls.
http://www.opengroup.org/onlinepubs/007908799/xsh/shmdt.html

<u>Appendix #1</u>

<u>Script: </u>gc_traffic_processing.sql

```
set serveroutput on size 100000
REM --------------------------------------------------------------------------------------------------
REM Author: Riyaj Shamsudeen @OraInternals, LLC
REM         www.orainternals.com
REM
REM Functionality: This script is to print GC processing timing for the past N seconds or
so
REM **************
REM
REM Source  : gv$sysstat
REM
REM Note : 1. Keep window 160 columns for better visibility.
REM
REM Exectution type: Execute from sqlplus or any other tool.  Modify sleep as needed.
Default is 60 seconds
REM
REM Parameters:
REM No implied or explicit warranty
REM
REM Please send me an email to rshamsud@orainternals.com for any question..
REM  NOTE   1. Querying gv$ tables when there is a GC performance issue is not
exactly nice. So, don't run this too often.
REM         2. Until 11g, gv statistics did not include PQ traffic.
REM         3. Of course, this does not tell any thing about root cause :-)
REM @copyright : OraInternals, LLC. www.orainternals.com
REM Version  Change
REM ---------- --------------------
REM 1.10       Corrected zero divisor issue
REM --------------------------------------------------------------------------------------------------
PROMPT
PROMPT
PROMPT  gc_traffic_processing.sql v1.10 by Riyaj Shamsudeen @orainternals.com
PROMPT
PROMPT  ...Prints various timing related information for the past N seconds
PROMPT  ...Default collection period is 60 seconds.... Please wait for at least 60
seconds...
PROMPT
PROMPT Column name key:
PROMPT        CR blk TX     : GC CR blocks served
PROMPT   CR bld tm: Average GC CR build time/CR block served
PROMPT   CR fls tm : Average GC CR flush time/CR block served
PROMPT   CR snd tm           : Average GC CR send time/CR block served
```

```
PROMPT
PROMPT   CUR blk TX        : GC CUR blocks served
PROMPT   CUR pin tm        : Average GC Current pin time /CUR block served
PROMPT   CUR fls tm        : Average GC Current flush time/CUR block served
PROMPT   CUR snd tm        : Average GC current send time/CUR block served
undef sleep
set lines 170 pages 100
set verify off
declare
        type number_table   is table of number       index by binary_integer;
        b_inst_id  number_table ;
        b_cr_blks_serv  number_table ;
        b_cur_blks_serv  number_table ;
        b_cr_bld_tm  number_table ;
        b_cr_flsh_tm    number_table ;
        b_cr_snd_tm    number_table ;
        b_cur_bld_tm  number_table ;
        b_cur_pin_tm    number_table ;
        b_cur_flsh_tm    number_table ;
        b_cur_snd_tm    number_table ;
        b_cr_tot_snd_tm    number_table ;
        b_cur_tot_snd_tm    number_table ;

        e_inst_id  number_table ;
        e_cr_blks_serv  number_table ;
        e_cur_blks_serv  number_table ;
        e_cr_bld_tm  number_table ;
        e_cr_flsh_tm    number_table ;
        e_cr_snd_tm    number_table ;
        e_cur_bld_tm  number_table ;
        e_cur_pin_tm    number_table ;
        e_cur_flsh_tm    number_table ;
        e_cur_snd_tm    number_table ;
        e_cr_tot_snd_tm    number_table ;
        e_cur_tot_snd_tm    number_table ;
        v_tot_instances number;

        v_cr_blks_serv  varchar2(256);
      v_cur_blks_serv  varchar2(256);
      v_cr_blks_bld    varchar2(256);
      v_cr_blks_flsh   varchar2(256);
      v_cr_blks_sndt    varchar2(256);
      v_cur_blks_pin    varchar2(256);
      v_cur_blks_flsh   varchar2(256);
      v_cur_blks_sndt    varchar2(256);
```

```
        v_ver number;
        l_sleep number:=60;
        l_cr_blks_served number :=0;
        l_cur_blks_served number :=0;
begin
         select count(*) into v_tot_instances from gv$instance;
         select to_number(substr(banner, instr(banner, 'Release ')+8,2)) ver into v_ver
from v$version where rownum=1;
         if (v_ver  <=9) then
           v_cr_blks_serv :='global cache cr blocks served';
           v_cur_blks_serv := 'global cache current blocks served';
           v_cr_blks_bld := 'global cache cr block build time';
           v_cr_blks_flsh := 'global cache cr block flush time';
           v_cr_blks_sndt := 'global cache cr block send time';
           v_cur_blks_pin := 'global cache current block pin time';
           v_cur_blks_flsh := 'global cache current block flush time';
           v_cur_blks_sndt := 'global cache current block send time';
      else
           v_cr_blks_serv :='gc cr blocks served';
           v_cur_blks_serv := 'gc current blocks served';
           v_cr_blks_bld := 'gc cr block build time';
           v_cr_blks_flsh := 'gc cr block flush time';
           v_cr_blks_sndt := 'gc cr block send time';
           v_cur_blks_pin := 'gc current block pin time';
           v_cur_blks_flsh := 'gc current block flush time';
           v_cur_blks_sndt := 'gc current block send time';
      end if;

      select
           evt_cr_serv.inst_id,
                evt_cr_serv.value cr_blks_serv,
                evt_cur_serv.value cur_blks_serv,
                evt_cr_bld.value cr_bld_tm,
                evt_cr_flsh.value cr_flsh_tm,
                evt_cr_snd.value cr_snd_tm,
                evt_cur_pin.value cur_pin_tm,
                evt_cur_flsh.value cur_flsh_tm,
                evt_cur_snd.value cur_snd_tm,
                evt_cr_bld.value + evt_cr_flsh.value + evt_cr_snd.value cr_tot_snd_tm,
                evt_cur_pin.value + evt_cur_flsh.value + evt_cur_snd.value
cur_tot_snd_tm
           bulk collect into

        b_inst_id,b_cr_blks_serv,b_cur_blks_serv,b_cr_bld_tm,b_cr_flsh_tm,b_cr_snd_t
m,b_cur_pin_tm,b_cur_flsh_tm,b_cur_snd_tm,b_cr_tot_snd_tm,b_cur_tot_snd_tm
        from
```

```
                gv$sysstat evt_cr_serv,
                gv$sysstat evt_cur_serv,
                gv$sysstat evt_cr_bld,
                gv$sysstat evt_cr_flsh,
                gv$sysstat evt_cr_snd,
                gv$sysstat evt_cur_pin,
                gv$sysstat evt_cur_snd,
                gv$sysstat evt_cur_flsh
          where
                  evt_cr_serv.name =v_cr_blks_serv
                and evt_cur_serv.name =v_cur_blks_serv
                and evt_cr_bld.name =v_cr_blks_bld
                and evt_cr_flsh.name =v_cr_blks_flsh
                and evt_cr_snd.name =v_cr_blks_sndt
                and evt_cur_pin.name =v_cur_blks_pin
                and evt_cur_flsh.name =v_cur_blks_flsh
                and evt_cur_snd.name =v_cur_blks_sndt
           and evt_cr_serv.inst_id=evt_cur_serv.inst_id
           and evt_cr_serv.inst_id=evt_cr_bld.inst_id
           and evt_cr_serv.inst_id=evt_cr_flsh.inst_id
           and evt_cr_serv.inst_id=evt_cr_snd.inst_id
           and evt_cr_serv.inst_id=evt_cur_pin.inst_id
           and evt_cr_serv.inst_id=evt_cur_snd.inst_id
           and evt_cr_serv.inst_id=evt_cur_flsh.inst_id
                order by inst_id
                ;
       select upper(nvl('&sleep',60)) into l_sleep from dual;
          dbms_lock.sleep(l_sleep);
       select
           evt_cr_serv.inst_id,
                evt_cr_serv.value cr_blks_serv,
                evt_cur_serv.value cur_blks_serv,
                evt_cr_bld.value cr_bld_tm,
                evt_cr_flsh.value cr_flsh_tm,
                evt_cr_snd.value cr_snd_tm,
                evt_cur_pin.value cur_pin_tm,
                evt_cur_flsh.value cur_flsh_tm,
                evt_cur_snd.value cur_snd_tm,
                evt_cr_bld.value + evt_cr_flsh.value + evt_cr_snd.value cr_tot_snd_tm,
                evt_cur_pin.value + evt_cur_flsh.value + evt_cur_snd.value
cur_tot_snd_tm
           bulk collect into

        e_inst_id,e_cr_blks_serv,e_cur_blks_serv,e_cr_bld_tm,e_cr_flsh_tm,e_cr_snd_t
m,e_cur_pin_tm,e_cur_flsh_tm,e_cur_snd_tm,e_cr_tot_snd_tm,e_cur_tot_snd_tm
        from
```

```
                gv$sysstat evt_cr_serv,
                gv$sysstat evt_cur_serv,
                gv$sysstat evt_cr_bld,
                gv$sysstat evt_cr_flsh,
                gv$sysstat evt_cr_snd,
                gv$sysstat evt_cur_pin,
                gv$sysstat evt_cur_snd,
                gv$sysstat evt_cur_flsh
         where
                   evt_cr_serv.name =v_cr_blks_serv
                and evt_cur_serv.name =v_cur_blks_serv
                and evt_cr_bld.name =v_cr_blks_bld
                and evt_cr_flsh.name =v_cr_blks_flsh
                and evt_cr_snd.name =v_cr_blks_sndt
                and evt_cur_pin.name =v_cur_blks_pin
                and evt_cur_flsh.name =v_cur_blks_flsh
                and evt_cur_snd.name =v_cur_blks_sndt
          and evt_cr_serv.inst_id=evt_cur_serv.inst_id
          and evt_cr_serv.inst_id=evt_cr_bld.inst_id
          and evt_cr_serv.inst_id=evt_cr_flsh.inst_id
          and evt_cr_serv.inst_id=evt_cr_snd.inst_id
          and evt_cr_serv.inst_id=evt_cur_pin.inst_id
          and evt_cr_serv.inst_id=evt_cur_snd.inst_id
          and evt_cr_serv.inst_id=evt_cur_flsh.inst_id
                order by inst_id
                ;
         dbms_output.put_line ( '---------|-----------|---------|-----------|----------|------------|--
----------|------------|----------|');
         dbms_output.put_line ( 'Inst     | CR blk Tx |CR bld tm| CR fls tm | CR snd tm|
CUR blk TX | CUR pin tm | CUR fls tm |CUR snd tm|');
         dbms_output.put_line ( '---------|-----------|---------|-----------|----------|------------|--
----------|------------|----------|');
         for i in  1 ..  v_tot_instances
                loop
                        l_cr_blks_served := e_cr_blks_serv (i) - b_cr_blks_serv (i);
                        l_cur_blks_served := e_cur_blks_serv (i) - b_cur_blks_serv (i);
                        dbms_output.put_line ( rpad( e_inst_id (i), 9)
        || '|' ||
                                lpad(to_char(e_cr_blks_serv (i) - b_cr_blks_serv(i)),11)
        || '|' ||
                                (case when l_cr_blks_served > 0  then
                                        lpad(to_char(trunc(10*( e_cr_bld_tm(i) -
b_cr_bld_tm(i) )/l_cr_blks_served, 2)),9)
                                 else
                                        lpad ('0',9)
                                        end)
          ||'|'||
                                       (case when l_cr_blks_served > 0  then
                                                 lpad(to_char(trunc(10*( e_cr_flsh_tm(i) -
b_cr_flsh_tm(i) )/l_cr_blks_served, 2)),11)
                                        else
                                                 lpad ('0',11)
                                        end)
          ||'|'||
                                       (case when l_cr_blks_served > 0  then
                                                 lpad(to_char(trunc(10*( e_cr_snd_tm(i) -
b_cr_snd_tm(i) )/l_cr_blks_served, 2)),10)
                                        else
                                                 lpad ('0',10)
                                        end)
          ||'|'||
                                       lpad(to_char( e_cur_blks_serv (i) - b_cur_blks_serv (i) ),
12)       || '|' ||
                                       (case when l_cur_blks_served > 0  then
                                                 lpad(to_char(trunc(10*( e_cur_pin_tm(i) -
b_cur_pin_tm(i) )/l_cur_blks_served, 2)),11)
                                        else
                                                 lpad ('0',11)
                                        end)
          ||'|'||
                                       (case when l_cur_blks_served > 0  then
                                                 lpad(to_char(trunc(10*( e_cur_flsh_tm(i) -
b_cur_flsh_tm(i) )/l_cur_blks_served, 2)),12)
                                        else
                                                 lpad ('0',12)
                                        end)
          ||'|'||
                                       (case when l_cur_blks_served > 0  then
                                                 lpad(to_char(trunc(10*( e_cur_snd_tm(i) -
b_cur_snd_tm(i) )/l_cur_blks_served, 2)),11)
                                        else
                                                 lpad ('0',11)
                                        end)
          ||'|'
                              );
                    end loop;
           dbms_output.put_line ( '-----------------------------------------------------------------
------------------------------------');
end;
/
set verify on
```

<u>Script:</u> gc_traffic_print.sql

```
set serveroutput on size 100000
set lines 120 pages 100
REM ------------------------------------------------------------------------------------------------
REM Author: Riyaj Shamsudeen @OraInternals, LLC
REM         www.orainternals.com
REM
REM Functionality: This script is to print GC timing for the past few minutes.
REM **************
REM
REM Source  : GV$ views
REM
REM Note : 1. Keep window 160 columns for better visibility.
REM
REM Exectution type: Execute from sqlplus or any other tool.  Modify sleep as needed.
Default is 60 seconds
REM
REM Parameters: Modify the script to use correct parameters. Search for PARAMS
below.
REM No implied or explicit warranty
REM
REM Please send me an email to rshamsud@orainternals.com for any question..
REM  NOTE   1. Querying gv$ tables when there is a GC performance issue is not
exactly nice. So, don't run this too often.
REM         2. Until 11g, gv statistics did not include PQ traffic.
REM         3. Of course, this does not tell any thing about root cause :-)
REM @ copyright :  www.orainternals.com
REM Version             Change
REM ----------          ----------------------------------------------------------
REM 1.21                Modified to handle zero divisor condition.
REM ------------------------------------------------------------------------------------------------
PROMPT
PROMPT
PROMPT  gc_traffic_print.sql v1.21 by Riyaj Shamsudeen @orainternals.com
PROMPT
PROMPT  ...Calculating GC Rx and Tx timing and blocks..
PROMPT  ...Default collection period is 60 seconds.... Please wait for at least 60
seconds...
PROMPT
PROMPT Column key:
PROMPT =========
PROMPT   CR block RX    : GC CR blocks received
PROMPT   CR time        : Average GC CR receive time
PROMPT   CUR blocks RX  : GC CUR blocks received
PROMPT   CUR time       : Average GC CuR receive time
PROMPT   CR blocks TX   : GC CR blocks transmitted
PROMPT   CUR blocks TX  : GC CUR blocks transmitted
PROMPT   tot blocks     : Sum of transmitted + received for both CR and CUR traffic
PROMPT
set lines 160 pages 100
set verify off
undef sleep
declare
        type number_table   is table of number       index by binary_integer;
        b_inst_id  number_table ;
        b_cr_blks_serv  number_table ;
        b_cr_blks_recv  number_table ;
        b_cr_tm_recv    number_table ;

        b_cur_blks_serv  number_table ;
        b_cur_blks_recv  number_table ;
        b_cur_tm_recv    number_table ;
        b_tot_blocks     number_table ;
        e_inst_id  number_table ;
        e_cr_blks_serv  number_table ;
        e_cr_blks_recv  number_table ;
        e_cr_tm_recv    number_table ;

        e_cur_blks_serv  number_table ;
        e_cur_blks_recv  number_table ;
        e_cur_tm_recv    number_table ;
        e_tot_blocks     number_table ;
        v_ver number;

        v_tot_instances number;

        v_cr_blks_serv  varchar2(256);
        v_cr_blks_recv    varchar2(256);
        v_cur_blks_serv  varchar2(256);
        v_cur_blks_recv   varchar2(256);
        v_cr_blks_rcv_time varchar2(256);
        v_cur_blks_rcv_time varchar2(256);
        l_sleep number;

begin
         select count(*) into v_tot_instances from gv$instance;
         select to_number(substr(banner, instr(banner, 'Release ')+8,2)) ver into v_ver
from v$version where rownum=1;
         if (v_ver  <=9) then
                 v_cr_blks_serv :='global cache cr blocks served';
                 v_cr_blks_recv := 'global cache cr blocks received';
```

```
                v_cur_blks_serv := 'global cache current blocks served';
                v_cur_blks_recv := 'global cache current blocks served';
                v_cr_blks_rcv_time :='global cache cr block receive time';
                v_cur_blks_rcv_time := 'global cache current block receive time';
        else
                v_cr_blks_serv :='gc cr blocks served';
                v_cr_blks_recv := 'gc cr blocks received';
                v_cur_blks_serv := 'gc current blocks served';
                v_cur_blks_recv := 'gc current blocks received';
                v_cr_blks_rcv_time :='gc cr block receive time';
                v_cur_blks_rcv_time := 'gc current block receive time';
        end if;

      select
          evt_cr_recv.inst_id,
                evt_cr_serv.value cr_blks_serv,
                evt_cr_recv.value cr_blks_recv, evt_cr_tm.value cr_tm_recv   ,
                evt_cur_serv.value cur_blks_serv,
                evt_cur_recv.value cur_blks_recv, evt_cur_tm.value cur_tm_recv,
                evt_cr_serv.value + evt_cr_recv.value + evt_cur_serv.value +
evt_cur_recv.value tot_blocks
          bulk collect into
                b_inst_id, b_cr_blks_serv ,b_cr_blks_recv ,b_cr_tm_recv ,
                b_cur_blks_serv, b_cur_blks_recv, b_cur_tm_recv , b_tot_blocks
       from
                gv$sysstat evt_cr_tm,
                gv$sysstat evt_cr_recv,
                gv$sysstat evt_cur_tm,
                gv$sysstat evt_cur_recv,
                gv$sysstat evt_cr_serv,
                gv$sysstat evt_cur_serv
         where
                   evt_cr_recv.name = v_cr_blks_recv
                and evt_cr_serv.name = v_cr_blks_serv
                and evt_cur_recv.name =v_cur_blks_recv
                and evt_cur_serv.name =v_cur_blks_serv
                and evt_cr_tm.name =v_cr_blks_rcv_time
                and evt_cur_tm.name =v_cur_blks_rcv_time
          and evt_cr_tm.inst_id=evt_cr_recv.inst_id
          and evt_cr_tm.inst_id=evt_cur_tm.inst_id
          and evt_cr_tm.inst_id=evt_cur_recv.inst_id
          and evt_cr_tm.inst_id=evt_cr_serv.inst_id
          and evt_cr_tm.inst_id=evt_cur_serv.inst_id
                order by inst_id
                ;
        select nvl ('&sleep',60) into l_sleep from dual;
```

```
            dbms_lock.sleep(l_sleep);
         select
              evt_cr_recv.inst_id,
                     evt_cr_serv.value cr_blks_serv,
                     evt_cr_recv.value cr_blks_recv, evt_cr_tm.value cr_tm_recv   ,
                     evt_cur_serv.value cur_blks_serv,
                     evt_cur_recv.value cur_blks_recv, evt_cur_tm.value cur_tm_recv,
                     evt_cr_serv.value + evt_cr_recv.value + evt_cur_serv.value +
evt_cur_recv.value tot_blocks
              bulk collect into
                     e_inst_id, e_cr_blks_serv ,e_cr_blks_recv ,e_cr_tm_recv ,
                     e_cur_blks_serv, e_cur_blks_recv, e_cur_tm_recv , e_tot_blocks
           from
                     gv$sysstat evt_cr_tm,
                     gv$sysstat evt_cr_recv,
                     gv$sysstat evt_cur_tm,
                     gv$sysstat evt_cur_recv,
                     gv$sysstat evt_cr_serv,
                     gv$sysstat evt_cur_serv
             where
                        evt_cr_recv.name = v_cr_blks_recv
                     and evt_cr_serv.name = v_cr_blks_serv
                     and evt_cur_recv.name =v_cur_blks_recv
                     and evt_cur_serv.name =v_cur_blks_serv
                     and evt_cr_tm.name =v_cr_blks_rcv_time
                     and evt_cur_tm.name =v_cur_blks_rcv_time
              and evt_cr_tm.inst_id=evt_cr_recv.inst_id
              and evt_cr_tm.inst_id=evt_cur_tm.inst_id
              and evt_cr_tm.inst_id=evt_cur_recv.inst_id
              and evt_cr_tm.inst_id=evt_cr_serv.inst_id
              and evt_cr_tm.inst_id=evt_cur_serv.inst_id
                     order by inst_id
                     ;
            dbms_output.put_line ( '---------|--------------|---------|----------------|----------|-------
--------|---------------|-------------|');
            dbms_output.put_line ( 'Inst     | CR blocks Rx | CR time |  CUR blocks Rx |
CUR time |  CR blocks Tx | CUR blocks Tx |Tot blocks   |');
            dbms_output.put_line ( '---------|--------------|---------|----------------|----------|-------
--------|---------------|-------------|');
            for i in  1 ..  v_tot_instances
                     loop
                              dbms_output.put_line ( rpad( e_inst_id (i), 9)
           || '|' ||
                                      lpad(to_char(e_cr_blks_recv (i) - b_cr_blks_recv(i)),14)
           || '|' ||
                                    ( case  when (e_cr_blks_recv (i) - b_cr_blks_recv (i)) >0
then
                                      lpad(to_char(trunc(10*( e_cr_tm_recv(i) -
b_cr_tm_recv(i) )/(e_cr_blks_recv (i) - b_cr_blks_recv (i)), 2)),9)
                                     else
                                      lpad('0',9)
                                     end )
         || '|' ||
                                    lpad(to_char( e_cur_blks_recv (i) - b_cur_blks_recv (i) ),
16)              || '|' ||
                                    ( case  when (e_cur_blks_recv (i) - b_cur_blks_recv (i)) >0
then
                                      lpad(to_char(trunc(10*( e_cur_tm_recv(i) -
b_cur_tm_recv(i) )/(e_cur_blks_recv (i) - b_cur_blks_recv (i)), 2)),10)
                                     else
                                      lpad('0',10)
                                     end )
         || '|' ||
                                    lpad(to_char(e_cr_blks_serv (i) - b_cr_blks_serv (i) ),15)
         || '|' ||
                                    lpad(to_char( e_cur_blks_serv (i) - b_cur_blks_serv (i) ),
15)              || '|' ||
                                    lpad(to_char( e_tot_blocks (i) - b_tot_blocks (i) ) ,13)
         || '|'
                           );
                  end loop;
          dbms_output.put_line ( '---------|--------------|---------|----------------|----------|-------
--------|---------------|-------------|');
end;
/
set verify on
```