

# Oracle database internals by Riyaj

**June 20, 2009**

## RAC, parallel query and udpsnoop

Filed under: [Oracle database internals](#), [Performance tuning](#), [RAC](#) — orainternals @ 10:37 pm

Tags: [battle of the nodes](#), [external table](#), [gv\\$px\\_session](#), [oracle](#), [oracle performance](#), [parallel query](#), [parallel instance group](#), [RAC interconnect traffic](#), [RAC performance myths](#), [UDP](#), [udpsnoop.d](#)

I presented about various performance myths in my 'battle of the nodes' presentation. One of the myth was that how spawning parallel query slaves across multiple RAC instances can cause major bottleneck in the interconnect. In fact, that myth was direct result of a lessons learnt presentation from a client engagement. Client was suffering from performance issues with enormous global cache waits running in to 30+ms average response time for global cache CR traffic and crippling application performance. Essentially, their data warehouse queries were performing hundreds of parallel queries concurrently with slaves spawning across three node RAC instances.

Of course, I had to hide the client details and simplified using a test case to explain the myth. Looks like either a)my test case is bad or b) some sort of bug I encountered in 9.2.0.5 version c) I made a mistake in my analysis somewhere. Most likely it is the last one 😊. [Greg Rahn](#) questioned that example and this topic deserves more research to understand this little bit further. At this point, I don't have 9.2.0.5 and database is in 10.2.0.4 and so we will test this in 10.2.0.4.

### **udpsnoop**

UDP is one of the protocol used for cache fusion traffic in RAC and it is the Oracle recommended protocol. In this article, UDP traffic size must be measured. Measuring Global cache traffic using AWR reports was not precise. So, I decided to use a dtrace tool kit tool:udpsnoop.d to measure the traffic between RAC nodes. There are two RAC nodes in this setup. You can read more about udpsnoop.d. That tool udpsnoop.d can be downloaded from [dtrace toolkit](#) . Output of this script is of the form:

```
PID      LADDR      LPORT      DR      RADDR      RPORT      SIZE
-----
```

```
15393      1.1.59.192      38395      ->      2.1.59.192      40449      8240
...
```

In the output above, PID 15393 sent an UDP packet of size 8240 from IP address 192.59.1.1 to 192.59.1.2 with local port as 38395 and remote port as 40449. As UDP traffic is flying between nodes, udpsnoop.d will print UDP traffic in to a file. So, I start collecting udpsnoop output before and end that collection immediately after our script is complete. Of course, we need to aggregate this data and play with it little bit, and so, will create an external table based upon this output file too.

```
- This is to read the file as an external table
drop table external_udpsnoop;
```

```
create table external_udpsnoop
(
  c_uid varchar2(10),
  pid varchar2(10),
  laddr varchar2(15),
  lport varchar2(15),
  dr  varchar2(10),
  raddr varchar2(15),
  rport varchar2(15),
  c_size  varchar2(10),
  cmd  varchar2(15)
)
organization external (
  type oracle_loader
  default directory UTL_FILE_DIR
  access parameters (
    records delimited by newline
    badfile APPS_DATA_FILE_DIR:'xtern_rpt.bad'
    logfile APPS_DATA_FILE_DIR:'xtern_rpt.log'
    discardfile APPS_DATA_FILE_DIR:'xtern_rpt.dsc'
    fields terminated by whitespace
    missing field values are null
  )
  location ('udpsnoop_ra_join_2_8th_iter.lst')
)
reject limit 1000
```

```
/
REM Reject limit is high since there are few packets with some junk outputs, might be due unstable fbt in
dtrace.
```

### Test case #1: Hash join – slaves from all instances

First, let's test for an hash join to show how UDP traffic is flowing between these ports. In this test case below, we use a big table and join a 10 Million rows to another 10 Million rows table. rownum is used so that script will complete in decent time, other wise, this ran for few hours before running in to errors. This selects few non-indexed columns so that SQL must do full table scan. SQL also has hint for 16 slaves from 2 instances.

Both instances will participate in this PQ operation as parallel\_instance\_group is set to ALL at session level.

```
alter session set parallel_instance_Group='ALL';
select /*+ parallel ( t1, 8,2)  parallel (t2, 8, 2) */
min (t1.CUSTOMER_TRX_LINE_ID +t2.CUSTOMER_TRX_LINE_ID ) , max ( t1.SET_OF_BOOKS_ID+t2.set_of_books_id ),
avg(t1.SET_OF_BOOKS_ID +t2.set_of_books_id),
      avg( t1.QUANTITY_ORDERED + t2.QUANTITY_ORDERED ), max(t1.ATTRIBUTE_CATEGORY ), max(t2.attribute1) ,
max(t1.attribute2)
from
  (select * from BIG_TABLE where rownum <=100000000)t1 ,
  (select * from BIG_TABLE where rownum <=100000000)t2
where t1.CUSTOMER_TRX_LINE_ID = t2.CUSTOMER_TRX_LINE_ID
;
```

### PQ in operation

We will also use yet another script to make sure that SQL is indeed getting 8 slaves in each instance. I don't remember, where I got this SQL to pull slaves information (may be Doug burns, Thanks!), but anyway, I modified that little bit for RAC.

```
select
  s.inst_id,
  decode(px.qcinst_id,NULL,s.username,
         ' - '||lower(substr(s.program,length(s.program)-4,4) ) ) "Username",
```

```

decode(px.qcinst_id,NULL, 'QC', '(Slave)') "QC/Slave" ,
to_char( px.server_set) "Slave Set",
to_char(s.sid) "SID",
decode(px.qcinst_id, NULL ,to_char(s.sid) ,px.qcsid) "QC SID",
px.req_degree "Requested DOP",
px.degree "Actual DOP", p.spid
from
gv$px_session px,
gv$session s, gv$process p
where
px.sid=s.sid (+) and
px.serial#=s.serial# and
px.inst_id = s.inst_id
and p.inst_id = s.inst_id
and p.addr=s.paddr
order by 5 , 1 desc
SQL> /

```

| INST_ID | Username | QC/slave | Slave Set | SID   | QC SID | Requested DOP | Actual DOP | SPID  |
|---------|----------|----------|-----------|-------|--------|---------------|------------|-------|
| 1       | SYS      | QC       |           | 10931 | 10931  |               |            | 7366  |
| 1       | - p000   | (Slave)  | 1         | 10925 | 10931  | 16            | 16         | 24762 |
| 1       | - p001   | (Slave)  | 1         | 10956 | 10931  | 16            | 16         | 24764 |
| 1       | - p002   | (Slave)  | 1         | 10955 | 10931  | 16            | 16         | 24766 |
| 1       | - p003   | (Slave)  | 1         | 10918 | 10931  | 16            | 16         | 24768 |
| 1       | - p004   | (Slave)  | 1         | 10941 | 10931  | 16            | 16         | 24778 |
| 1       | - p005   | (Slave)  | 1         | 10916 | 10931  | 16            | 16         | 24781 |
| 1       | - p006   | (Slave)  | 1         | 10945 | 10931  | 16            | 16         | 24787 |
| 1       | - p007   | (Slave)  | 1         | 10922 | 10931  | 16            | 16         | 24795 |
| 2       | - p000   | (Slave)  | 1         | 10943 | 10931  | 16            | 16         | 16920 |
| 2       | - p001   | (Slave)  | 1         | 10961 | 10931  | 16            | 16         | 16923 |
| 2       | - p002   | (Slave)  | 1         | 10920 | 10931  | 16            | 16         | 16970 |
| 2       | - p003   | (Slave)  | 1         | 10946 | 10931  | 16            | 16         | 16972 |
| 2       | - p004   | (Slave)  | 1         | 10935 | 10931  | 16            | 16         | 16974 |
| 2       | - p005   | (Slave)  | 1         | 10934 | 10931  | 16            | 16         | 16976 |
| 2       | - p006   | (Slave)  | 1         | 10899 | 10931  | 16            | 16         | 16988 |
| 2       | - p007   | (Slave)  | 1         | 10940 | 10931  | 16            | 16         | 16991 |
| 1       | SYS      | QC       |           | 10927 | 10927  |               |            | 9476  |
| 2       | - pz99   | (Slave)  | 1         | 10890 | 10927  | 2             | 2          | 17723 |
| 1       | - pz99   | (Slave)  | 1         | 10912 | 10927  | 2             | 2          | 25875 |

20 rows selected.

From the output above 8 slaves are from instance 1 and 8 are from instance 2 allocated, with query coordinator ( 7366) running instance 1. Above sample also captured my own session accessing gv\$ views ( Notice pz99, slaves for gv\$ access use different PQ slave naming conventions from 10.2 onwards).

## Results

We have established that slaves were allocated from multiple instances and udpsnoop is capturing UDP packet size between these instances. We also have external table mapping to that udpsnoop output file so as to query this data. Script completed in 1500 seconds. I mapped output of px slaves query above with UDP external table and here is the table I put together to show PQ slaves and their UDP size.

| INST | Username | QC/Slave | Slave Set | SID   | QC SID | Req. DOP | Actual DOP | SPID  | LADDR      | RADDR      | RPORT | SIZE      |
|------|----------|----------|-----------|-------|--------|----------|------------|-------|------------|------------|-------|-----------|
| 1    | SYS      | QC       |           | 10931 | 10931  |          |            | 7366  |            |            |       |           |
| 1    | - p000   | (Slave)  | 1         | 10925 | 10931  | 16       | 16         | 24762 |            |            |       |           |
| 1    | - p001   | (Slave)  | 1         | 10956 | 10931  | 16       | 16         | 24764 |            |            |       |           |
| 1    | - p002   | (Slave)  | 1         | 10955 | 10931  | 16       | 16         | 24766 |            |            |       |           |
| 1    | - p003   | (Slave)  | 1         | 10918 | 10931  | 16       | 16         | 24768 |            |            |       |           |
| 1    | - p004   | (Slave)  | 1         | 10941 | 10931  | 16       | 16         | 24778 |            |            |       |           |
| 1    | - p005   | (Slave)  | 1         | 10916 | 10931  | 16       | 16         | 24781 |            |            |       |           |
| 1    | - p006   | (Slave)  | 1         | 10945 | 10931  | 16       | 16         | 24787 |            |            |       |           |
| 1    | - p007   | (Slave)  | 1         | 10922 | 10931  | 16       | 16         | 24795 |            |            |       |           |
| 2    | - p000   | (Slave)  | 1         | 10943 | 10931  | 16       | 16         | 16920 | 2.1.59.192 | 2.1.59.192 | 62783 | 127068484 |
| 2    | - p001   | (Slave)  | 1         | 10961 | 10931  | 16       | 16         | 16923 | 2.1.59.192 | 2.1.59.192 | 62783 | 126904080 |
| 2    | - p002   | (Slave)  | 1         | 10920 | 10931  | 16       | 16         | 16970 | 2.1.59.192 | 2.1.59.192 | 62783 | 127767353 |
| 2    | - p003   | (Slave)  | 1         | 10946 | 10931  | 16       | 16         | 16972 | 2.1.59.192 | 2.1.59.192 | 62783 | 128154145 |
| 2    | - p004   | (Slave)  | 1         | 10935 | 10931  | 16       | 16         | 16974 | 2.1.59.192 | 2.1.59.192 | 62783 | 128096875 |
| 2    | - p005   | (Slave)  | 1         | 10934 | 10931  | 16       | 16         | 16976 | 2.1.59.192 | 2.1.59.192 | 62783 | 126057311 |
| 2    | - p006   | (Slave)  | 1         | 10899 | 10931  | 16       | 16         | 16988 | 2.1.59.192 | 2.1.59.192 | 62783 | 128228830 |
| 2    | - p007   | (Slave)  | 1         | 10940 | 10931  | 16       | 16         | 16991 | 2.1.59.192 | 2.1.59.192 | 62783 | 127471579 |

Few important points to make here:

1. In this case, all slaves running in node 2 were talking to one UDP port in node 1 (Port 62783). Isof shows that PID 7366 (Query co-ordinator) was listening on that UDP port. Point is that these slaves from node 2 were sending packets to the co-ordinator.
2. Interestingly, there is no UDP traffic from instance 1 to 2. I think, that looks due to the nature of aggregation in the SQL.

**Few minutes later..**

Interestingly, few minutes later, while I was watching UDP traffic, few other processes kicked in and started generating UDP traffic. Re-queried the database again to see what these processes are. Query has allocated 16 more slaves, 8 more running from node 1 and 8 more running in node 2 [processes p008-p015 below]. These slaves were talking to a different UDP port 62789 which was also listened by coordinator process 7366 in node 1.

| INST | Username | QC/Slave | Slave Set | SID   | QC SID | Req. DOP | Actual DOP | SPID  | LADDR      | RADDR      | RPORT | SIZE      |
|------|----------|----------|-----------|-------|--------|----------|------------|-------|------------|------------|-------|-----------|
| 1    | SYS      | QC       |           | 10931 | 10931  |          |            | 7366  |            |            |       |           |
| 1    | - p000   | (Slave)  | 1         | 10925 | 10931  | 16       | 16         | 24762 |            |            |       |           |
| 1    | - p001   | (Slave)  | 1         | 10956 | 10931  | 16       | 16         | 24764 |            |            |       |           |
| 1    | - p002   | (Slave)  | 1         | 10955 | 10931  | 16       | 16         | 24766 |            |            |       |           |
| 1    | - p003   | (Slave)  | 1         | 10918 | 10931  | 16       | 16         | 24768 |            |            |       |           |
| 1    | - p004   | (Slave)  | 1         | 10941 | 10931  | 16       | 16         | 24778 |            |            |       |           |
| 1    | - p005   | (Slave)  | 1         | 10916 | 10931  | 16       | 16         | 24781 |            |            |       |           |
| 1    | - p006   | (Slave)  | 1         | 10945 | 10931  | 16       | 16         | 24787 |            |            |       |           |
| 1    | - p007   | (Slave)  | 1         | 10922 | 10931  | 16       | 16         | 24795 |            |            |       |           |
| 1    | - p008   | (Slave)  | 1         | 10958 | 10931  | 16       | 16         | 24798 |            |            |       |           |
| 1    | - p009   | (Slave)  | 1         | 10938 | 10931  | 16       | 16         | 24818 |            |            |       |           |
| 1    | - p010   | (Slave)  | 1         | 10965 | 10931  | 16       | 16         | 24836 |            |            |       |           |
| 1    | - p011   | (Slave)  | 1         | 10953 | 10931  | 16       | 16         | 24838 |            |            |       |           |
| 1    | - p012   | (Slave)  | 1         | 10946 | 10931  | 16       | 16         | 24841 |            |            |       |           |
| 1    | - p013   | (Slave)  | 1         | 10929 | 10931  | 16       | 16         | 24843 |            |            |       |           |
| 1    | - p014   | (Slave)  | 1         | 10919 | 10931  | 16       | 16         | 24853 |            |            |       |           |
| 1    | - p015   | (Slave)  | 1         | 10942 | 10931  | 16       | 16         | 24855 |            |            |       |           |
| 2    | - p000   | (Slave)  | 1         | 10943 | 10931  | 16       | 16         | 16920 | 2.1.59.192 | 2.1.59.192 | 62783 | 127068484 |
| 2    | - p001   | (Slave)  | 1         | 10961 | 10931  | 16       | 16         | 16923 | 2.1.59.192 | 2.1.59.192 | 62783 | 126904080 |
| 2    | - p002   | (Slave)  | 1         | 10920 | 10931  | 16       | 16         | 16970 | 2.1.59.192 | 2.1.59.192 | 62783 | 127767353 |
| 2    | - p003   | (Slave)  | 1         | 10946 | 10931  | 16       | 16         | 16972 | 2.1.59.192 | 2.1.59.192 | 62783 | 128154145 |
| 2    | - p004   | (Slave)  | 1         | 10935 | 10931  | 16       | 16         | 16974 | 2.1.59.192 | 2.1.59.192 | 62783 | 128096875 |
| 2    | - p005   | (Slave)  | 1         | 10934 | 10931  | 16       | 16         | 16976 | 2.1.59.192 | 2.1.59.192 | 62783 | 126057311 |
| 2    | - p006   | (Slave)  | 1         | 10899 | 10931  | 16       | 16         | 16988 | 2.1.59.192 | 2.1.59.192 | 62783 | 128228830 |

|   |        |         |   |       |       |    |    |       |            |            |       |           |
|---|--------|---------|---|-------|-------|----|----|-------|------------|------------|-------|-----------|
| 2 | - p007 | (Slave) | 1 | 10940 | 10931 | 16 | 16 | 16991 | 2.1.59.192 | 2.1.59.192 | 62783 | 127471579 |
| 2 | - p008 | (Slave) | 1 | 10911 | 10931 | 16 | 16 | 16993 | 2.1.59.192 | 2.1.59.192 | 62989 | 182053370 |
| 2 | - p009 | (Slave) | 1 | 10949 | 10931 | 16 | 16 | 16995 | 2.1.59.192 | 2.1.59.192 | 62989 | 182490908 |
| 2 | - p010 | (Slave) | 1 | 10951 | 10931 | 16 | 16 | 17000 | 2.1.59.192 | 2.1.59.192 | 62989 | 181899025 |
| 2 | - p011 | (Slave) | 1 | 10890 | 10931 | 16 | 16 | 17007 | 2.1.59.192 | 2.1.59.192 | 62989 | 181858294 |
| 2 | - p012 | (Slave) | 1 | 10972 | 10931 | 16 | 16 | 17009 | 2.1.59.192 | 2.1.59.192 | 62989 | 182104499 |
| 2 | - p013 | (Slave) | 1 | 10950 | 10931 | 16 | 16 | 17011 | 2.1.59.192 | 2.1.59.192 | 62989 | 182334705 |
| 2 | - p014 | (Slave) | 1 | 10902 | 10931 | 16 | 16 | 17013 | 2.1.59.192 | 2.1.59.192 | 62989 | 181611641 |
| 2 | - p015 | (Slave) | 1 | 10955 | 10931 | 16 | 16 | 17023 | 2.1.59.192 | 2.1.59.192 | 62989 | 181816693 |

### In real life..

Summing this up, approximately, 2.4GB of UDP traffic was generated with one parallel query. Can you imagine what will happen if this inter-instance parallelism is allowed in data warehouse queries scanning many tables and partitions with many hash joins? Obviously, this has the effect of saturating Interconnect quickly and so performance will suffer. Our solution was to disallow parallel queries spawning multiple instances. All of them will be running within an instance boundary and effect of this change was immediately visible in the client environment. Back to our test, `parallel_execution_message_size` was set to 8192. Increasing this parameter will decrease elapsed time little bit, but we are worried about saturating interconnect traffic not just elapsed time of that query.

Further, I ran this query with `parallel_instance_group` set to one instance and then all instances, few times. Spawning across multiple instances, in fact, increases elapsed time too.

```
Parallel_instance_group :ALL
```

| call    | count | cpu     | elapsed | disk   | query | current | rows |
|---------|-------|---------|---------|--------|-------|---------|------|
| Parse   | 1     | 0.00    | 0.00    | 0      | 0     | 0       | 0    |
| Execute | 1     | 0.20    | 0.26    | 0      | 76    | 0       | 0    |
| Fetch   | 2     | 1481.76 | 1509.95 | 701158 | 76    | 0       | 1    |
| total   | 4     | 1481.96 | 1510.22 | 701158 | 152   | 0       | 1    |

```
parallel_instance_group :INST1
```

| call    | count | cpu     | elapsed | disk   | query | current | rows |
|---------|-------|---------|---------|--------|-------|---------|------|
| Parse   | 1     | 0.00    | 0.00    | 0      | 0     | 0       | 0    |
| Execute | 1     | 0.20    | 0.23    | 0      | 76    | 0       | 0    |
| Fetch   | 2     | 1321.05 | 1331.67 | 701344 | 76    | 0       | 1    |
| total   | 4     | 1321.25 | 1331.90 | 701344 | 152   | 0       | 1    |

### What about the original example ?

Of course, let's talk about that original example also. In this example, there was just one table and data was aggregated.

```
select /*+ parallel ( t1, 8,2) */
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
;
```

Measuring, UDP traffic, It is visible that for this huge table, Interconnect traffic is kept minimal. It looks like, there are some optimization techniques for this single table aggregation query minimizing cache fusion traffic to a minimal level, just 2152. This convinces that, just the SQL in that myth is a bad example, but that myth is still a myth. I should have used original SQL joining multiple tables with hash join for my presentation, but as a consultant, I have a responsibility to keep clients information confidential and protect. At the end of the day, they pay for my bread.

| INST_ID | Username | QC/Slave | Slave Set | SID   | QC SID | Requested DOP | Actual DOP | SPID  | Size |
|---------|----------|----------|-----------|-------|--------|---------------|------------|-------|------|
| 1       | SYS      | QC       |           | 10933 | 10933  |               |            | 3314  |      |
| 1       | - p000   | (Slave)  | 1         | 10958 | 10933  | 16            | 16         | 24762 |      |
| 1       | - p001   | (Slave)  | 1         | 10948 | 10933  | 16            | 16         | 24764 |      |
| 1       | - p002   | (Slave)  | 1         | 10953 | 10933  | 16            | 16         | 24766 |      |
| 1       | - p003   | (Slave)  | 1         | 10925 | 10933  | 16            | 16         | 24768 |      |
| 1       | - p004   | (Slave)  | 1         | 10916 | 10933  | 16            | 16         | 24778 |      |

|   |        |         |   |       |       |    |    |       |       |
|---|--------|---------|---|-------|-------|----|----|-------|-------|
| 1 | - p005 | (Slave) | 1 | 10938 | 10933 | 16 | 16 | 24781 |       |
| 1 | - p006 | (Slave) | 1 | 10951 | 10933 | 16 | 16 | 24787 |       |
| 1 | - p007 | (Slave) | 1 | 10946 | 10933 | 16 | 16 | 24795 |       |
| 2 | - p000 | (Slave) | 1 | 10949 | 10933 | 16 | 16 | 16920 | 2152  |
| 2 | - p001 | (Slave) | 1 | 10937 | 10933 | 16 | 16 | 16923 | 2152  |
| 2 | - p002 | (Slave) | 1 | 10946 | 10933 | 16 | 16 | 16970 | 2152  |
| 2 | - p003 | (Slave) | 1 | 10956 | 10933 | 16 | 16 | 16972 | 2152  |
| 2 | - p004 | (Slave) | 1 | 10902 | 10933 | 16 | 16 | 16974 | 2152  |
| 2 | - p005 | (Slave) | 1 | 10981 | 10933 | 16 | 16 | 16976 | 2152  |
| 2 | - p006 | (Slave) | 1 | 10899 | 10933 | 16 | 16 | 16988 | 2152  |
| 2 | - p007 | (Slave) | 1 | 10927 | 10933 | 16 | 16 | 16991 | 2152  |
| 1 | SYS    | QC      |   | 10945 | 10945 |    |    | 3527  |       |
| 1 | - pz99 | (Slave) | 1 | 10942 | 10945 | 2  | 2  | 25875 |       |
| 2 | - pz99 | (Slave) | 1 | 10962 | 10945 | 2  | 2  | 17723 | 72344 |

## Summary

In summary, having too many parallel query slaves spawning across multiple instances can cripple interconnect. There are some optimization techniques that seem to help in the case of single table aggregation and of course, that must be considered as an exception. I have modified the presentation little bit below, but will do a second and complete update on this presentation later: