

## RAC Tips & Best Practices

By Murali Vallath

### Tip: Cluster Interconnect

This is a very important component of the clustered configuration. Oracle depends on the cluster interconnect for movement of data between the instances. Chapter 2 provides detailed explanation on how global data movement occurs.

Testing the cluster interconnect should start with a test of the hardware configuration. This basic test should ensure that the database is using the correct IP addresses or NIC's for the interconnect. The following query provides a list of IP addresses registered with Oracle.

```
COL PICKED_KSXPIA FORMAT A15  
COL INDX FORMAT 99999  
SELECT * FROM X$KSPXPIA;
```

ADDR	INDX	INST_ID	PUB_KSXPIA	PICKED_KSXPIA	NAME_KSXPIA	IP_KSXPIA
3FE47C74	0	1	N	OCR	bond1	10.168.2.130
3FE47C74	1	1	Y	OCR	bond0	192.168.2.30

In the output, bond0 is the public interface (identified by the value “Y” in column PUB\_KSXPIA) and bond1 is the private interface (identified by the value “N” in column PUB\_KSXPIA). If the correct IP addresses are not visible, it is an indication of incorrect installation and configuration of the RAC environment.

Column PICKED\_KSXPIA indicates the type of Clusterware implemented on the RAC cluster, where the interconnect configuration is stored and the cluster communication method that RAC will use. The valid values in this column are:

- OCR = Oracle Clusterware is configured.
- OSD = Operating System dependent, meaning a third party cluster manager is configured, and Oracle Clusterware is only a bridge between Oracle RDBMS and the third party cluster manager.
- CI = Interconnect is defined using the CLUSTER\_INTERCONNECT parameter in the instance.

Cluster interconnect can also be verified using the ORADEBUG utility (discussed later) and verifying the trace file for the appropriate IP address.

**Note:** In the output above, NIC pairing/bonding is used for both public (bond0) and private (bond1) networks.

**CLUSTER\_INTERCONNECTS**

This parameter provides Oracle with information on the availability of additional cluster interconnects that could be used for cache fusion activity. The parameter overrides the default interconnect settings at the operating system level with a preferred cluster traffic network. While this parameter does provide certain advantages over systems where high interconnect latency is noticed by helping reduce such latency, configuring this parameter could affect the interconnect high-availability feature. In other words, an interconnect failure that is normally unnoticeable would instead cause an Oracle cluster failure as Oracle still attempts to access the network interface.

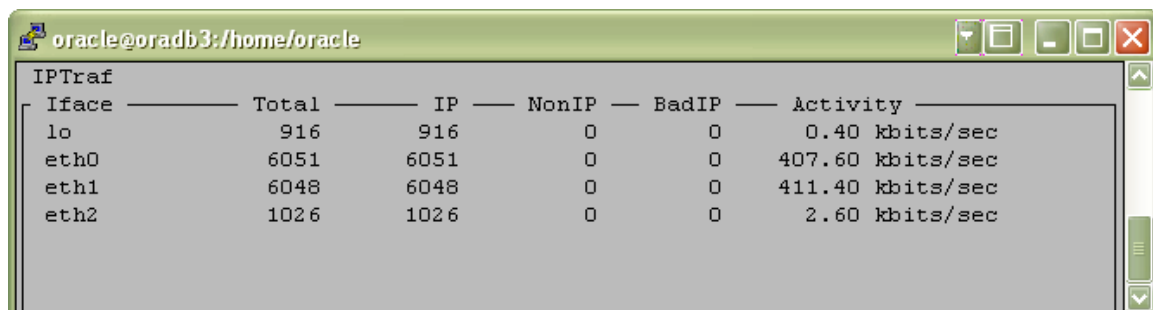
**Best Practice:** NIC pairing /bonding should be a preferred method over using the `CLUSTER_INTERCONNECTS` parameters to provide load balancing and failover of the interconnects.

### Interconnect Transfer Rate

The next important verification would be to determine the transfer rate versus the actual implemented packet size to ensure the installation has been carried out per specification.

The speed of the cluster interconnect solely depends on the hardware vendor and the layered operating system. Oracle depends on the operating system and the hardware for sending packets of information across the cluster interconnect. For example, one type of cluster interconnect supported in Sun 4800s is the UDP protocol. However, Solaris in this specific version has an O/S limitation of a 64 KB packet size for data transfer. To transfer 256 KB worth of data across this interconnect protocol would take over four round trips. Comparing this to another operating system for example Linux, the maximum support packet size is 256K. On a high-transaction system where there is a large amount of interconnect traffic, because of user activity on the various instances participating in the clustered configuration, limitations on the packet size could cause serious performance issues.

Tools such as `iptraf` on Linux environments (figure below), `glance` on HP-UX environments or utilities such as `netstat` should help monitor network traffic and transfer rates between instance and client configurations.



The screenshot shows a terminal window titled 'oracle@oradb3:/home/oracle' running the IPtraf utility. The utility displays a table of network traffic statistics for four interfaces: lo, eth0, eth1, and eth2. The table has columns for 'Iface', 'Total', 'IP', 'NonIP', 'BadIP', and 'Activity'. The 'Activity' column shows traffic rates in kbits/sec.

Iface	Total	IP	NonIP	BadIP	Activity
lo	916	916	0	0	0.40 kbits/sec
eth0	6051	6051	0	0	407.60 kbits/sec
eth1	6048	6048	0	0	411.40 kbits/sec
eth2	1026	1026	0	0	2.60 kbits/sec

*IPtraf General network traffic*

IPtraf also helps to looking into a specific network and monitor its performance in detail by the type of protocol used for network traffic. For example, in figure 9.5 the network traffic by protocol (TCP and UDP) giving outgoing and incoming rates is displayed.

```

oracle@pradb3:/home/oracle
IPTraf
Statistics for eth0
-----
              Total      Total      Incoming      Incoming      Outgoing      Outgoing
              Packets    Bytes     Packets      Bytes     Packets      Bytes
Total:         1426     501164         0         0         1426     501164
IP:            1426     481200         0         0         1426     481200
TCP:           477     193864         0         0          477     193864
UDP:           943     283880         0         0          943     283880
ICMP:           6         3456         0         0           6         3456
Other IP:       0           0         0         0           0           0
Non-IP:         0           0         0         0           0           0

Total rates:           67.0 kbits/sec      Broadcast packets:      0
                   26.4 packets/sec      Broadcast bytes:       0

Incoming rates:        0.0 kbits/sec
                   0.0 packets/sec

Outgoing rates:        67.0 kbits/sec
                   26.4 packets/sec

IP checksum errors:      0

Elapsed time:  0:01
X-exit

```

*IPTraf Statistics for eth0*

After the initial hardware and operating-system-level tests to confirm the packet size across the interconnect, subsequent tests could be done from the Oracle database to ensure that there is not any significant added latency from using cache-to-cache data transfer or the cache fusion technology. The query below provides the average time to receive a consistent read (CR) block on the system.

```

set numwidth 20
column "AVG CR BLOCK RECEIVE TIME (ms)" format 9999999.9
select
    b1.inst_id,
    b2.value "GCS CR BLOCKS RECEIVED",
    b1.value "GCS CR BLOCK RECEIVE TIME",
    ((b1.value / b2.value) * 10) "AVG CR BLOCK 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 ;

INST_ID GCS CR BLOCKS RECEIVED GCS CR BLOCK RECEIVE TIME AVG CR BLOCK RECEIVE TIME (ms)
-----
1          2758                112394                443.78
2          1346                1457                  10.8

```

**Note:** The data in the GV\$SYSSTAT view is a cumulative figure since the last time the Oracle instance was bounced. This does not reflect the true performance of the

interconnect or give a true picture of the latency in transferring data. To get a more realistic picture of the performance, it would be good to bounce all the Oracle instances and test again.

In the output above, it can be noticed that the `AVG CR BLOCK RECEIVE TIME` is 443.78 ms; this is significantly high when the expected average latency as recommended by Oracle should not exceed 15 ms. A high value is possible if the CPU has limited idle time and the system typically processes long-running queries. However, it is possible to have an average latency of less than 1 ms with user-mode IPC. Latency can also be influenced by a high value of the `DB_MULTI_BLOCK_READ_COUNT` parameter. This is because this parameter determines the size of the block that each instance would request from the other during read transfers and a requesting process can issue more than one request for a block depending on the setting of this parameter and may have to wait longer. This kind of high latency requires further investigation of the cluster interconnect configuration and tests should be performed at the operating system level to ensure this is not something from Oracle or the parameter.

**Note:** Sizing of the `DB_MULTI_BLOCK_READ_COUNT` parameter should be based on the interconnect latency and the packet sizes as defined by the hardware vendor, and after considering the operating system limitations (e.g., the Sun UDP max setting is only 64 KB).

### **Sequences and Index Contention**

Indexes, with key values generated using sequences, tend to be subject to leaf block contention when the insert rate is high. This is because the index leaf block holding the highest key value is changed for every row inserted, as the values are monotonically ascending. In a RAC environment, this may lead to a high rate of current and CR blocks being transferred between nodes.

- Increase sequence cache size, the difference between sequence values generated by different instances increases successive index block splits and tends to create instance affinity to index leaf blocks.
- Increase sequence cache also improves instance affinity to index keys deriving their values from sequences. This technique may result in significant performance gains for multi instance `INSERT` intensive applications.
- Implement database partitioning option to physically distribute the data.
- Use locally managed tablespaces (LMT) over dictionary-managed tablespaces.
- Use automatic segment space management (ASSM). ASSM can provide instance affinity to table blocks. ASSM is the default in Oracle Database 10g Release 2

## Identifying Blockers Across Instances

When two users request access to same row of data for update purposes, all users except the first will remain in wait mode until the first user has committed or rolled back the change. This is true irrespective of whether it is a single instance configuration or a clustered RAC configuration, with an additional possibility. Apart from users on the same instance, users from other instances can also request the same row at the same time. The following query helps identify blocking session in a RAC environment:

```
SELECT DECODE(G.INST_ID,1,'SSKY1',2,'SSKY2') INSTANCE,
       S.SID,
       G.TYPE,
       S.USERNAME,
       S.SERIAL#,
       S.PROCESS,
       DECODE(LMODE,0,'None',1,'Null',2,'Row-S',3,'Row-X',4,'Share',5,'S/ROW',
6,'Exclusive') LMODE,
       DECODE(REQUEST,0,'None',1,'Null',2,'Row-S',3,'Row-X',4,'Share',
5,'S/ROW',6,'Exclusive')REQUEST,
       DECODE(REQUEST,0,'BLOCKER','WAITER') STATE
FROM   GV$GLOBAL_BLOCKED_LOCKS G,
       GV$SESSION S
WHERE  G.SID = S.SID
AND    G.INST_ID = S.INST_ID
ORDER BY STATE
```

INSTANCE	<b>SID</b>	TY	USERN	SERIAL#	PROCESS	LMODE	REQUEST	STATE
SSKY2	<b>132</b>	TX	OE	519	2576:820	<b>Exclusive</b>	None	<b>BLOCKER</b>
SSKY2	<b>148</b>	TX	OE	78	980:3388	None	Exclusive	<b>WAITER</b>
SSKY1	<b>139</b>	TX	OE	114	3192:2972	None	Exclusive	<b>WAITER</b>

In the above output, session with SID# 132 on instance SSKY2 is the BLOCKER because this session accessed this row first in an exclusive mode and has not either committed or rolled back the transaction. Subsequently session with SID# 148 also on instance SSKY2 requested for this same row for update, followed by SID # 139 from instance SSKY1. Both these SID's will remain as WAITER(s) until such time the row is available to the session to complete its update operation.

Blockers can also be determined from EM by selecting “Blocking Sessions” option from the database performance page.