CONFIO is now part of solarwinds

Oracle RAC Tuning Tips

There is More to Know

By Kathy Gibbs, Product Manager

Confio Software
4772 Walnut Street, Suite 100
Boulder, CO 80301

www.confio.com

Introduction

When I first started working with RAC 8 years ago, I was told there was no difference between performances tuning on a RAC database versus a single instance. Even today Oracle maintains that an application performing well on a single instance will perform well on RAC. This is not entirely accurate. In fact, it is well known among database administrators (DBAs) working with RAC systems, that an application that isn't 'RAC Aware' will run slower in a RAC environment. In this paper I will explore the RAC-only waits as well as some tuning suggestions for code to help your RAC environment run more smoothly. First though a bit of an overview, what is RAC?

What is Oracle RAC?

RAC, or real application clusters, has been around since Oracle 9i and before that as parallel server although we will just concentrate on RAC in this paper. In order to have RAC you need at least two instances preferably on separate domains with shared storage so they can run the same database. The recommendation for RAC is to actually have a minimum of three instances. This ensures you will still have a two instance RAC even if one instance goes down. The instances communicate over a private network referred to as the interconnect. The interconnect uses cache fusion to be able to transfer blocks of data between the instances. The response time for cache fusion transfers is determined by the messaging and processing times reported from the physical interconnect components, the IPC protocol and the GCS protocol.

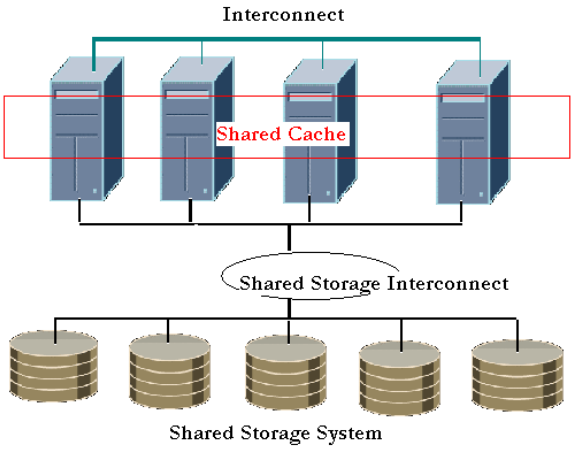
The other item we need to address for this article is a service. [1] What this means is that you can divide up your load or isolate work across the different instances while still hitting the same database. A Service, and just RAC in general, allows you to run on smaller machines and add in new machines when growth is needed potentially saving cost for licensing and hardware. It can do this by a divide and conquer method. If your data can be divided, i.e., reporting on one instance isolated from loading, you can also get more out of your hardware instead of having to carry a certain free area, i.e., 30% CPU or 8 GB of extra memory, for nightly or quarterly loads.

The other two things to mention about RAC are TAF and SCAN. TAF or Transparent Application Failover allows for clients to reconnect to a surviving instance in the event of an instance failure. When using TAF, reconnect happens automatically from the OCI (Oracle Call Interface). Any uncommitted transactions during a failure will be rolled back and session properties will be lost. In most cases, when using TAF, Select statements are automatically re-executed behind the scenes so if you are running a SQL select statement, you shouldn't lose connectivity. The SCAN or Single Client Access Name was introduced in 11g Release 2. It provides for a single domain name via DNS and allows users to connect to a RAC cluster as if was a single IP address. This makes it easier to setup applications using Java Thin drivers, especially ones that are coded into a GUI page, to connect to a RAC database. Before SCAN,

[1] Oracle defines a service as 'Entities that you can define in Oracle RAC databases that enable you to group database workloads and route work to the optimal instances that are assigned to offer the service.' 'http://download.oracle.com/docs/cd/B28359_01/rac.111/b28254/admcon.htm#RACAD7151 '

there are many instances where you couldn't use RAC in an automatic failover fashion because the Java Thin driver had to connect to one instance and one host. Using SCAN fixes this issue in theory above.

Below is a rough picture from Oracle's documentation of a 4 node RAC Cluster.



RAC Wait Events

When troubleshooting performance issues, the first thing I look at after checking the usual suspects; e.g., system, network, etc., is wait events. In a single instance database, if user response times increase and/or a high proportion of the time in the database, then a cause should be determined. In a RAC environment, we have specific wait events that are referred to as GC wait events or Global Cache wait events. Oracle wait times are attributed to an event which reflects the exact outcome of a request. For example, when a session on an instance is looking for a block in the global cache, it does not know whether it will receive the data cached by another instance or whether it will receive a message to read from disk. The events for the global cache related waits convey precise information on global cache blocks or messages. If you want one wait to look at for cluster health, the wait information in a broader category is 'Cluster Wait Class'. This has summarized wait information for the cluster. I find in 99.9% of situations, I need to dive in deeper.

The most important wait events for Oracle RAC are organized in four main categories.

Block –oriented : Indicates that a block was received as either the result of a 2-way or a 3-way message; that the block was sent from either the resource master requiring 1 message and 1 transfer; or was forwarded to a third node from which it was sent requiring 2 messages and 1 block transfer. These waits also indicate that the remotely cached blocks were shipped without having been busy, pinned, or requiring a log flush.

- gc current block 2-way
- gc current block 3-way
- gc cr block 2-way
- gc cr block 3-way

Message-oriented: Indicates that no block was received from being cached in any instance. Instead a global grant was given enabling the instance to read the block from disk. If this time is long, it may be that the frequently used SQL causes a lot of disk I/O (for the cr grant) or that the workload inserts a lot of data and needs to format new blocks (for the current grant)

- gc current grant 2-way
- gc cr grant 2-way

Contention-oriented: The gc current block busy and gc cr block busy events indicate that the remote instance received the block after a remote instance processing delay, in many cases due to a log flush. High concurrency is evidenced by the gc buffer busy event which indicates that the block was pinned or held up by a session on a remote instance. It can also indicate that a session on the same instance has already requested the block.

- gc current block busy
- gc cr block busy
- gc buffer busy

Load-oriented: These are usually the most frequent in the absence of block contention and indicate that a delay has occurred in the GCS. The available wait time and the total wait time should be considered when being alerted to performance issues when these events are high. Usually either interconnect, load issues, or SQL execution against a large working set is the root cause. For these events, the wait time includes the entire round trip from the time a session starts to wait until the block arrives.

- gc current block congested
- gc cr block congested:

RAC performance tuning

The methods to find and analyze the longest running waits for a RAC instance are similar to what you do now for a single instance, but there are a number of RAC-specific steps and scripts.

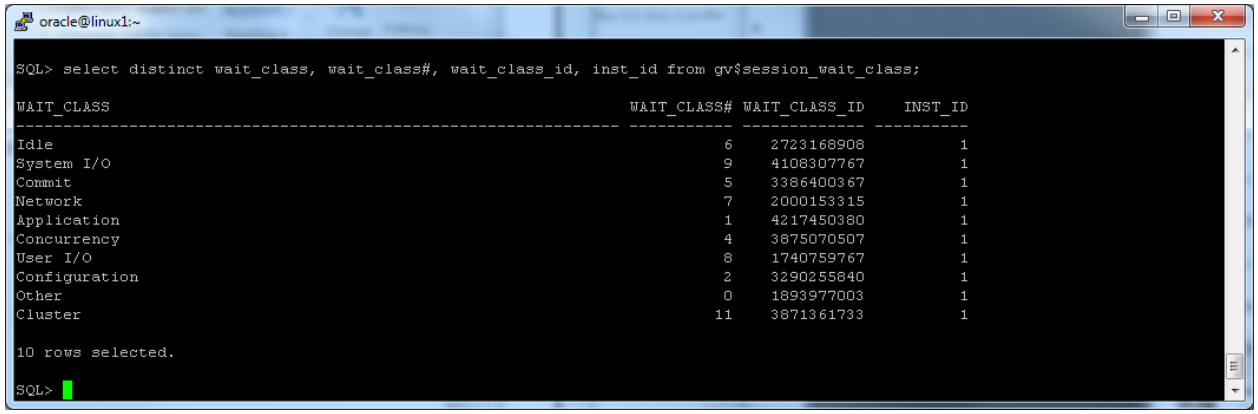
There is a script that you can download from Oracle Support (formerly Metalink); NOTE 135714.1 called racdiag.sql. This script is updated often so I recommend verifying you have the latest version before running the script. This script will collect items such as waiting sessions, GES lock information, system stats, and many other RAC-related items. This script is a great tool for establishing performance benchmarks and revealing performance problems. It is best to run this script in development first and then at a designated time in the production environment, because sometimes the script can further impact performance – especially where cluster wait problems already exist.

Another reliable source for troubleshooting performance problems is the v$views. In a RAC environment, there are additional views to look at that are gv$views (i.e. GV$SESSION_WAIT). The only difference between gv$views and the views you use with single instance database is they have

information about all instances noted by the 'INST_ID' column. The gv$session_wait is a great view to start with as it will allow you to limit the waits you are looking for:

```
Description of GV$SESSION_WAIT
Name Null? Type
----------------------------------------- -------- -------------
INST_ID NUMBER
SID NUMBER
SEQ# NUMBER
EVENT VARCHAR2(64)
P1TEXT VARCHAR2(64)
P1 NUMBER
P1RAW RAW(4)
P2TEXT VARCHAR2(64)
P2 NUMBER
P2RAW RAW(4)
P3TEXT VARCHAR2(64)
P3 NUMBER
P3RAW RAW(4)
WAIT_CLASS# NUMBER
WAIT_CLASS VARCHAR2(64)
WAIT_TIME NUMBER
SECONDS_IN_WAIT NUMBER
STATE VARCHAR2(19)
```

The screen print below provides an example on one way to query this view



Another helpful view is the DBA_HIST_ACTIVE_SESS_HISTORY view combined with DBA_HIST_WAITSTAT. This keeps instance information by active session history by snaphot range and can assist you to identify specific SQL statements that may be experiencing extended wait times.

Here is a sample of how to find SQL, and even objects and hot blocks, for SQL specific to high RAC waits by combining the wait class above with the DBA_HIST_ACTIVE_SESS_HISTORY:

```
select event_id, event, count(*) cnt from dba_hist_active_sess_history
where snap_id between 12831 and 12838 and wait_class_id=3871361733
group by event_id, event
order by 3;

  EVENT_ID EVENT                                          CNT
---------- ---------------------------------------- ----------
3905407295 gc current request                              4
3785617759 gc current block congested                     10
```

```
2705335821 gc cr block congested                          11
 512320954 gc cr request                                  13
3794703642 gc cr grant congested                          18
3897775868 gc current multi block request                 18
1742950045 gc current retry                               23
1445598276 gc cr disk read                               113
1457266432 gc current split                              188
2685450749 gc current grant 2-way                        209
 957917679 gc current block lost                         523
 737661873 gc cr block 2-way                             688
2277737081 gc current grant busy                         800
3570184881 gc current block 3-way                       1265
3151901526 gc cr block lost                             1734
 111015833 gc current block 2-way                       1801
3046984244 gc cr block 3-way                            1988
 661121159 gc cr multi block request                    2066
3201690383 gc cr grant 2-way                            3458
1520064534 gc cr block busy                             3522
2701629120 gc current block busy                       16845
1478861578 gc buffer busy                              43668
```

You can then drill in further to find out what was causing the two longest waits

```
select sql_id, count(*) cnt from dba_hist_active_sess_history
where snap_id between 16905 and 16928
and event_id in (2701629120, 1478861578)
group by sql_id
having count(*)>500
order by 2;

SQL_ID               CNT
------------- ----------
6kk6ydpp3u8xw        500
2hvs3mpab5j0w        998
292jxfuggtsqh       1101
3mcxaqffnzgfw       1190
a36pf34c87x7s       1221
4vs8wgvpfm87w       1257
22ggtj4z9ak3a       1298
gsqhbt5a6d4uv       1312
cyt90uk11a22c       2024
39dtqqpr7ygcw       3271
5p5gz205n93k7      32396
```

Clearly the sql id 5p5gz205n93k7 is the issue.

```
select sql_text from dba_hist_sqltext where sql_id='5p5gz205n93k7';

SQL_TEXT
--------------------------------------------------------------------------------
----

Insert into treatable(vendorid, amt, business_unit, order_id, desc)
```

You can isolate further from here to see what objects are affected to see if there is a hot block. This query provides you a list of the objects that are involved in the sql query that was isolated above.

```
select count(distinct(current_obj#)) from dba_hist_active_sess_history

where snap_id between 16905 and 16928

and event_id=1478861578 and sql_id='5p5gz205n93k7';

  COUNT(DISTINCT(CURRENT_OBJ#))
  -----------------------------
                            14
```

The query below is going to isolate objects even further to see if there is one particular object that has more of the system work going against it. In this case you can see that is object 78553.

```
select current_obj#, count(*) cnt from dba_hist_active_sess_history

where snap_id between 16905 and 16928

and event_id=1478861578 and sql_id='5p5gz205n93k7'

group by current_obj#

order by 2;

      CURRENT_OBJ#        CNT
      ------------ ----------
           88445        1039
           78624        1154
           78553       24521



select * from all_objects

where object_id in (78553,78624,88445)
```

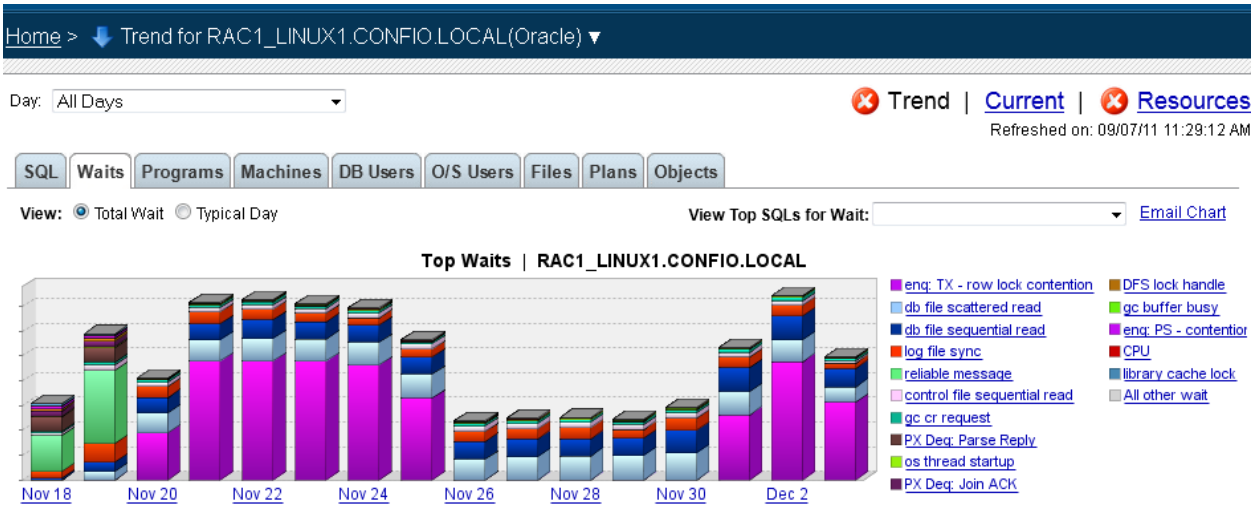
OBJECT_ID	OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_TYPE
78553	SCOTT	TREATABLE	P_2011_09	TABLE PARTITION
88445	SCOTT	TTLG_X_VENID	P_2011_09	INDEX PARTITION
78624	SCOTT	TTLG_X_ DATE	P_2011_09	INDEX PARTITION

The query below is then going to let us know if we have a hot block which would be bad and could indicate storage issues. In this case, the file 1830 has account of 328 which is something to start tracking. With this count it doesn't absolutely indicate a hot block, but it is something to keep an eye on.

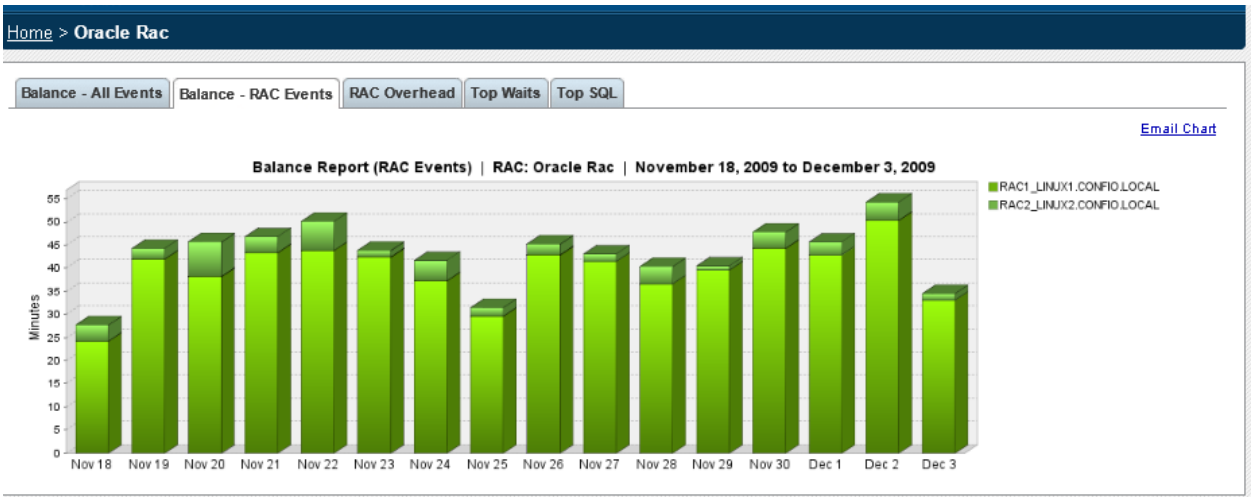
```
select current_file#, current_block#, count(*) cnt

from dba_hist_active_sess_history

where snap_id between 16905 and 16928

and event_id=1478861578 and sql_id='5p5gz205n93k7'

and current_obj# in (78553)

group by current_file#, current_block#

having count(*) > 50

order by 3;
```

CURRENT_FILE#	CURRENT_BLOCK#	CNT
1830	63013	51
1124	62456	55
1487	67910	56
1830	68742	101
1830	64129	176

Another way to isolate wait information is to employ 3rd-party tools help identify long wait events as well as problem SQL that can be the cause of the long waits. The screen capture below from SolarWinds Database Performance Analyzer (DPA) shows normal database waits along with some gc waits in a RAC database. This can be extremely beneficial to not only isolate individual queries, but to monitoring trending. These tools also use graphical representation of the longest running waits and the underlying causes making it easier for Oracle DBAs to communicate complex performance issues more effectively with IT managers and development teams.



In the screen above, I can see there is a lot of locking which could be cluster or RAC related, but it would require further investigation. The gc specific waits are not on the top 5 wait events so you would get a sense the cluster is not the issue here.



In this screen print, it shows the RAC specific events at a high level. This allows you to be able to focus on the days that had the highest amount of RAC events.

Cluster tuning

Though I won't spend a lot of time focusing on it, there are a couple of items related strictly to the CRS or GRID that you should be aware of that will help with interconnect latency and evictions. If you are on 10g you need to set the diagwait parameter to 13. This will increase logging in the crsd.log which will help you to troubleshoot any RAC issues. The other CRS parameter that should be increased is the CSS miscount parameter. The CSS miscount parameter points to interconnect latency. You want to be careful increasing this because there are cases if there is a miscount that is high enough you definitely want the cluster to fail. Just be aware that this depends on your system, it is common at least in all the environments I have worked at as well as consultants and experts I have talked to that this increased to 300 seconds from 30 or 60 seconds.

Patching

Patching can be such a dirty word to an Oracle DBA. I don't know a single DBA who would blindly apply a patch to any database. And most of them will only apply a patch when given no other alternative. But there is one really good reason to consider patching: RAC.

As we saw in the previous section, there are a lot of cluster wait events and each of these can have "undocumented features." For an example, 10.2.0.4 is still a very stable platform for the database. DBAs have enough excitement throughout their day without introducing risky patches into their environment. If it ain't broke, why fix it, right? What if I told you that sticking with a patch just because it works isn't always a good idea? Oracle is always improving RAC performance so they are constantly churning out patches with new fixes. In this example, the 10.2.0.5 patch has specific fixes for some

interconnect latencies as well as actual cluster node evictions that are the ultimate bad performance issue. At one company I worked at, when we applied the 10.2.0.5 patch, our evictions went from happening every week to about once a quarter. The CRS latency average cr block receive time when from 15 ms to less than 5 ms. At another company, we applied patch number 9352164 or PSU 10.2.0.4.4 and saw latency decrease to less than 5% on a very heavy OLTP database. Of course, it is important to test the application of these patches on a RAC system before going to PROD as there is different behavior for RAC patches.

SQL tuning

SQL tuning is important in a RAC environment and is often the culprit in performance tuning problems. How often have you heard that 80% of performance problems are SQL related? In a RAC environment there are additional factors to be considered. Some documentation suggests that you should run with a service or instance group for bulk loads when using a RAC database. If you are connecting to a service it not only assists in failover but also assists in knowing what hosts you can go to in a least loaded host scenario. If the application server is pointing to particular instance instead of using a server this will take failover out of the equation. It will also take some performance gains you could have been expecting out of the mix. For example, if you have a three node cluster and all the work is coming from appserver1 and it is directly attached to dbinst1, dbinst2 and dbinst3 are going to be sitting pretty idle.

The other role SQL tuning has to play is if developers design more RAC friendly code. For example, if the plan is to query a large table bringing back large amounts of data partitioning and parallelism, it should be tested to determine if some of the load can occur independently across the instances.

Is RAC part of the problem?

Do we need RAC? I think this is a valid question and one that must be asked. Too often, having RAC is seen as a 'status' symbol inside corporations. "I have to have RAC to be a tier 1 application so therefore I need RAC." I have heard statement countless times throughout my career. There are flaws in this statement, such as customers should never choose a solution just to be relevant in the company. If they truly need to be a tier 1 application that is one thing, but to state they want to have a RAC system so they feel important is a waste of money and resources. That decision should be made in the development phase as well. Suffice to say, there are databases and software out there that will a) not utilize RAC at all, b) may say they utilize RAC but it is really just one app server pointing to one instance which though beneficial if you have no other means of failover, really isn't what RAC is good at and, c) may just run slower on a RAC system due to high block transfer. People have asked me if there is a blanket statement on how to determine if an application will perform poorly on RAC. The truth is I have seen Data Warehouses perform very well on RAC and I have seen OLTP systems perform poorly. It truly should be a case by case study.

Conclusion

In conclusion, Oracle RAC can be very beneficial part of your MAA (Maximum Availability Architecture). If setup correctly with software that utilizes RAC options it will provide near seamless failover in the case

of instance crash and even hardware issues. You also can get RAC to run efficiently and close to, if not faster, that a single instance, you just need to be aware that tuning is not complete if you are only using single instance tuning techniques.

Documentation

http://download.oracle.com/docs/pdf/A95979_02.pdf
http://download.oracle.com/docs/cd/B19306_01/rac.102/b28759/intro.htm#CEGEIBBI
http://download.oracle.com/docs/cd/B28359_01/rac.111/b28254/monitor.htm#CFAHDADB
http://www.oracle.com/technetwork/database/clustering/overview/wp-oracleibm-2009-130764.pdf

About Confio Software

Confio Software, now a part of the SolarWinds family, builds award-winning database performance analysis tools for DBAs and developers. SolarWinds Database Performance Analyzer (formerly Confio Ignite) improves the productivity and efficiency of IT organizations. By resolving problems faster, speeding development cycles, and squeezing more performance out of expensive database systems, Database Performance Analyzer makes DBA and development teams more productive and valuable to the organization. Customers worldwide use our products to improve database performance on Oracle, SQL Server, Sybase and DB2 on physical and virtual machines.

For more information, please visit: http://www.confio.com/performance/oracle/rac/