INTRODUCTION: WHAT IS A RESOURCE MASTER?

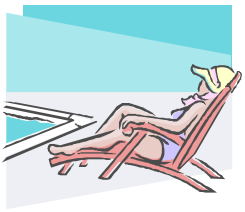
On a RAC cluster, only one node maintains extensive information about a particular resource (e.g., a data block) in use. We call that node the "resource master." As part of GCS (Global Cluster Services), the master maintains current information about the precise state of locking attributes for that block. Then, when other nodes seek access to that resource, the master plays traffic cop, and directs the node owning a block to gracefully relinquish control to a requesting node.

OBEY THE MASTER!

Normally, a group of contiguous blocks (typically 128) are assigned the same master, and this master doesn't change (except for special events, such as a node failure.) So, if you were to examine the master for a certain object in cache, you would normally see the same master for the adjacent blocks. We'll discuss how to find the master later in this paper, and see whether the theory fits the facts.

There is an interesting feature, however, called DRM, for "Dynamic Resource Remastering," in which the master *changes*, based on how often a node accesses the blocks in an object[1]. Of particular note is that DRM works at the *object* level. So if DRM is executed for an object, it will cause *all blocks* for the object to be assigned to a single master node. The idea is, have the node that accesses an object most, be the resource master for that object.

Before delving into the details, let's do some housekeeping preparation.



AN EASIER WAY TO ACCESS THE X$ TABLES

[1] This feature has proven troublesome, however, and is often disabled by the DBA.

I find it annoying to have to connect as *Sys*, which is typically required for accessing the X$ views. (Note that it is not possible to grant access directly to the X$ views.) Additionally, some companies have rules about connecting as *Sys*, so I prefer to avoid it when possible. To accomplish that, let's just create some views under *Sys*, and then grant access to these new views to the user of interest. [2] Here's what I did:

Connected as sys:

```
Create View V_X$Bh As Select * From X$Bh;
Create View V_X$Le As Select * From X$Le;
Create View V_X$Kjbl As Select * From X$Kjbl;

Grant Select On V_X$Bh To Chris;
Grant Select On V_X$Le To Chris;
Grant Select On V_X$Kjbl To Chris;
```

In my own schema:

```
Create Synonym Chris.X$Bh For Sys.V_X$Bh;
Create Synonym Chris.X$Le For Sys.V_X$Le;
Create Synonym Chris.X$Kjbl For Sys.V_X$Kjbl;
```

Once you have created these views and synonyms, you can use your normal DBA account, rather than connecting as *Sys*.

FINDING THE RESOURCE MASTER

For a given object, we can see how the blocks are distributed to which master. The following script looks at the buffer for the local node, and shows how the blocks are assigned to the various master nodes. Note that this can take a few minutes to run for a large table. Simply add the table name where indicated below.

Script *FIND_MASTER.SQL*

```
--FIRST FIND THE OBJECT IDS
With OBJ_IDS As (Select DATA_Object_Id OBJECT_ID From Dba_Objects
 Where Object_Name = 'TBD'),           --Customize
Addr As (Select /*+materialize */
```

[2] I believe Steve Adams originated this idea.

```
Le_Addr, class, state From X$Bh, OBJ_IDS
Where Object_Id = Obj),
--NOW GET THE RESOURCE NAME IN HEX
Hexnames As (Select
Rtrim(B.Kjblname, ' '||chr(0)) Hexname
From X$Le A, X$Kjbl B, Addr
Where A.Le_Kjbl=B.Kjbllockp
and class = 1 and state <> 3
And A.Le_Addr = Addr.Le_Addr)
-- NOW FIND THE NODE MASTERS
Select A.Master_Node Mast, Count(*)
From Gv$Dlm_Ress  A, Hexnames H
Where
Rtrim(A.Resource_Name, ' '||chr(0)) = H.Hexname
Group by A.Master_Node
```

In this script, I use query subfactoring—sometimes called the "with" syntax, to make it easier to follow. Nevertheless, the script is confusing, so let's review a few quirks.

SOME NOTES ON THE SCRIPT

(1) It is necessary to use employ Rtrim on *Resource_Name*, so that this field can be used in a join. However, it wasn't sufficient to just remove trailing blanks because the field also has a *null character* before the trailing blanks. So, Rtrim must look for both trailing spaces as well as the null character. These become input parameters to Rtrim, like this:

' ' || chr(0)

(2) Initially, I used *Object_Id* for identifying the table (or partition) in question, for which I was examining the cache. I discovered, however, that I would sometimes not find any blocks in the cache for the table I had just queried. It turns out that ***Data_Object_Id*** should be used instead. The difference is, ***Data_Object_Id*** changes as an object is significantly altered—e.g., moved to a different tablespace, or redefined in some way. The other field, Object_Id is assigned at creation time, and stays the same.

(3) The reason I use the hint, *Materialize*, is for performance reasons. I really only want the rows from X$Bh that match the condition, Class = 1 and State <> 3. I discovered, however, that adding that filter on the query of X$Bh completely wrecked performance. I can, however, retrieve those two columns without hurting performance. So, I just retrieve those extra fields, and then apply the filter condition later. The Materialize hint

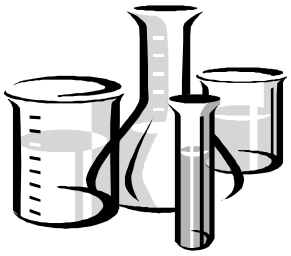
ensures that the optimizer doesn't try to co-mingle the query on X$Bh. (Maybe the *No_Merge* hint would work also.)

(4) The filter conditions slightly refine the block counts in this way:

Class = 1	Data blocks (4 = header block)
State <> 3	We don't want CR (Consistent Read), i.e., stale block.

(5) We use X$Bh, not V$Bh, because the needed fields are not in V$Bh. Also, querying V$Bh may prove costly, performance wise, since behind the scenes it must join views from each of the nodes in the cluster.

(6) The view *X$kjbr* also has a column, KjbrMaster that shows the master node. In the script above, I instead use Gv$Dlm_Ress, which finds the master no matter what node it is on. It appears that X$kjbr only includes the local master.



A TEST CASE

Let's see how the script works. We use a sample table, having the creative name, "Test_Table." It has a few millions rows, and several partitions. We will arbitrarily pick a node (I picked node 5), and see who the master is for the blocks in our test table.

Initially, since no one has queried this object on this node, it turns out there are no blocks from that table cached on our local node (node 5 in my test case). Remember—the script finds the master for the blocks that are currently cached on the local node. So, as shown here, there is initially nothing to see:

```
SQL> @find_master

no rows selected
```

Now, on node 5, we run a simple query to retrieve 100 rows from Test_Table.

```
Select * From Test_Table Where Rownum < 101;
```

Now we try Find_Master again:

```
SQL> @find_master
```

```
      MAST   COUNT(*)
---------- ----------
         1         16
```

For our sample query, we cached 16 blocks in node 5, and the master for all these blocks happens to be a single node--node 2. (Note that Oracle begins the node numbering with 0, not 1, so a result of 1 really means our node 2.)

A BIGGER TEST

So far, the test case has been pretty boring. Let's expand the test, and query a million rows for a particular partition of Test_Table. First we turn on Sql*Plus Autotrace, then run this query, which will touch 1 million rows (but of course, much fewer blocks):

```
Select Max(Big_Col)
From Test_Table Partition (TEST_TABLE_PART_210)
Where Rownum < 1000001;
```

Here's a copy of the screen. We see that Oracle touch about 26k blocks.

```
Execution Plan
----------------------------------------------------
Plan hash value: 2667524378

----------------------------------------------------
| Id  | Operation                | Name  | Rows  |
----------------------------------------------------
|   0 | SELECT STATEMENT         |       |     1 |
|   1 |  SORT AGGREGATE          |       |     1 |
|*  2 |   COUNT STOPKEY          |       |       |
|   3 |    PARTITION RANGE SINGLE|       |  7132K|
|   4 |     TABLE ACCESS FULL    | TEST  |  7132K|
----------------------------------------------------

Statistics
----------------------------------------------------
          0  recursive calls
          0  db block gets
      26176  consistent gets
```

Let's rerun our master script, and see how the master is distributed across these 26k blocks:

```
SQL> @find_master
```

```
    MAST   COUNT(*)
-------- ----------
       0       3320
       1       3181
       2       2407
       3       2526
       4       3185
       5       3933
       6       3938
       7       3702
```

We see that each node is the master for about 3k blocks. These results confirm a few things. First, that the resource-master is indeed assigned at the *block level*; secondly, that there are about 26,000 blocks from this table cached on our local node. Of course, since Oracle touched 26k blocks, this is exactly what we would expect.

CHECKING INDIVIDUAL BLOCKS

The above simple test revealed the *count* of blocks, as they are distributed amongst the nodes. But what about the individual blocks—that is, how is each block distributed to master nodes? Does Oracle assign the first block to node 1, the second block to node 2, and so on? If so, this would seem to be a very inefficient (but finely grained) way.

Let's see how Oracle actually assigns the master node. To do this, we'll have to slightly modify our *Find_Master* script, so that we see the actual block address, not just the count of the blocks.

First however, I created a 100,000-row table, then ran a simple query to cause a scan of all 100k rows. Running the *Find_Master* script (from above) I see this distribution:

```
      MAST   COUNT(*)
---------- ----------
         0       1376
         1        378
         2        126
         3        126
         4        487
         5        126
```

So far, so good. Now, let's now modify the *Find_Master* script to not just count the blocks, but actually display the hex resource name. The modification, which is made to the last part of Find_Master, is shown below. Let's call this script, ***Master_Details***:

```
        * * *
Select A.Master_Node Mast, H.Hexname
From Gv$Dlm_Ress  A, Hexnames H
Where Rtrim(A.Resource_Name, ' '||Chr(0)) = H.Hexname
And A.Master_Node = 0
Order By 1,2
```

I also found it convenient to just examine the results for one node. Hence, the additional filter to only include blocks assigned to Master_Node = 0 (i.e., node 1.) Of course, my selection of node 1 is complete arbitrary. Now, if we look at the block hex address for master 0, we see this:

```
SQL> @Master_Details

      MAST HEXNAME
---------- ------------------------
         0 [0x10180][0xb0000],[BL]
         0 [0x10181][0xb0000],[BL]
         0 [0x10182][0xb0000],[BL]
         0 [0x10183][0xb0000],[BL]
         0 [0x10184][0xb0000],[BL]
         0 [0x10185][0xb0000],[BL]
         0 [0x10186][0xb0000],[BL]
             * * *
```

As expected, the blocks are distributed to a master in contiguous chunks, not singly. This grouping is controlled by the special parameter *_lm_contiguous_res_count,* which defaults to 128 (see script, HIDDEN_PAR, below).=

SOME USEFUL SCRIPTS

Here are a few other scripts that I have found useful

LISTING THE SPECIAL PARAMETERS

The parameter *_lm_contiguous_res_count* is defined to be, "number of contiguous blocks that will hash to the same HV bucket." In case you want to see all the special, hidden parameters, here is a script you can use.

```
Set Linesize 145
Set Pagesize 9999
Set Verify   Off
Column Ksppinm   Format A42   Head 'Parameter Name'
Column Ksppstvl  Format A39   Head 'Value'
Column Ksppdesc  Format A60   Head 'Description'    Trunc
Select Ksppinm, Ksppstvl, Ksppdesc
FROM X$Ksppi X, X$Ksppcv Y
WHERE X.Indx = Y.Indx
AND TRANSLATE(ksppinm,'_','#') like '#%'
```

LISTING BLOCKS IN CACHE FOR AN OBJECT

If desired, you can see the actual blocks cached:

```
Select File#, Block#, Status From V$Bh Where Objd IN
(Select Data_Object_Id From Dba_Objects
Where Object_Name = 'TEST_TABLE')
Order By Block#;
```

REFERENCES

I found the following book very helpful, and credit the author with several scripts that form the basis of this paper: K.Gopalakrishnam, *Oracle Database 10g Real Application Clusters Handbook.* Especially Chapter 12, "A Closer Look at Cache Fusion."

Also, Julian Dyke has written some very detailed analysis on RAC internals. Especially useful is his presentation, "Inside RAC."

* * *

Chris Lawson is an Oracle *Ace* and performance specialist in the San Francisco Bay Area. He is the author of *The Art & Science of Oracle Performance Tuning*, as well as *Snappy Interviews: 100 Questions to Ask Oracle DBAs*. Chris can be reached at, RAC@OracleMagician.com