USING RAC PARALLEL INSTANCE GROUPS

WHY USE PARALLEL INSTANCE GROUPS?

Oracle's Parallel Query Option (PQO) provides a lot of flexibility, but also some pitfalls. It's very tempting for the novice DBA, when working with a long-running query, to quickly add a high degree of parallelism. Of course, parallelism should not be the first resort; instead, the query should first be optimized, and then, parallelism can be invoked where needed. In other words, parallelism should never be used as way to cover-over badly designed code.

With Oracle RAC we add more capability—and more hazards to PQO. Now, we can run parallel processes on a variety of nodes—regardless of where we are currently connected. In this paper we explore some ideas on using parallel instance groups. (All of our examples were run on 10g release 2.)

SETUP

Parallel instance groups rely on two init.ora parameters. They have very similar names, but very different purposes, confusing the issue. One parameter *defines* the node set-up, and the other parameter specifies how we *use* the set-up:

- The parameter *Instance_Groups* defines a particular node as belonging to a logical group of resources. A node may belong to more than one Instance_Group.

- The parameter *Parallel_Instance_Group* chooses the Instance_Group that should run the parallel processes. This parameter can be set at the instance level, as well as changed for an individual session.

Now let's take a look at some practical ways to use instance groups.

SOME IDEAS FOR USING INSTANCE GROUPS

There are a lot of possibilities on using this feature. Here are a few ways to consider managing the parallel processes.

- Allow certain critical queries to be routed to any server
- Restrict non-critical queries to certain servers

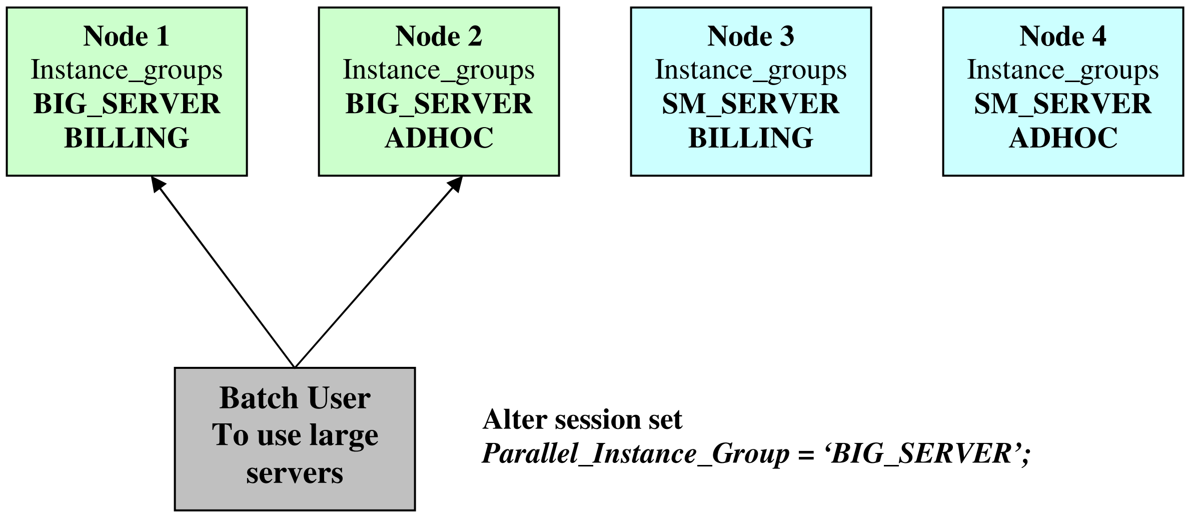
- Distribute processing based on geographical considerations
- Distribute processing based on expected runtime of query

In some cases, you will want to use *Alter Session Set Parallel_Instance_Group* as a way to distribute processing. Alternatively, you can set this parameter for the database as a whole so that parallel processing is automatically distributed according to your plan.

Example 1

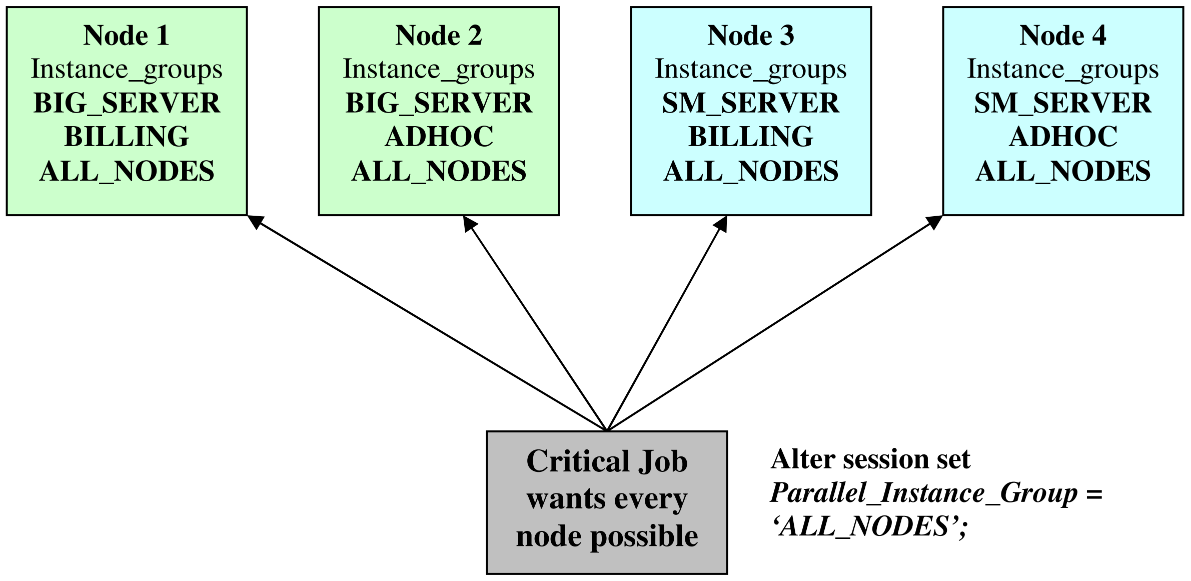
Assume we are running a four-node RAC cluster, with each node having different capability and function. We invent an Instance_Group called "BIG_SERVER" and assign nodes 1 and 2 to this group. Nodes 3 and 4 get assigned to the Instance_Group called "SM_SERVER."

Similarly, we want to use two of the nodes for Billing jobs, and the two remaining nodes for adhoc work. Here's how this would look:



Alter session set
Parallel_Instance_Group* = *'BIG_SERVER';

Now, if we need to run a large batch job, invoking parallelism, on the most powerful servers (i.e., nodes 1 and 2), we can use the *Alter Session* method to ensure the parallel processing is divided between Nodes 1 and 2—*no matter which node we are connected to at the time.*



Alter session set
Parallel_Instance_Group* = *'ALL_NODES';

We can even make it possible for a job to run parallelism on every node, by creating an Instance_Group (such as "ALL_NODES") that contains every node. Then, we alter our session and invoke the "ALL_NODE" instance groups.

FINDING PARALLEL STATISTICS

It's not too difficult o find the statistics for all the parallel processes for a given Sql. Let's try the following Sql, which uses moderate parallelism, and see how it works:

```
Select /*+Parallel(Ft 4) */
Max(Open_Dt) C1test From Test1
Where Rownum < 100000;
```

In the above Sql, we first find the Sql_id in the view V$Sql (I simply looked for the word "C1TEST.") Knowing the Sql_id, let's use that to find the overall statistics for the above Sql, across all RAC nodes[1]:

[1] Of course, this preliminary step can easily be combined into the main query.

```
Col SECS format 99999
Col EXECS format 9999
Col DISK format 99999
Col GETS format 99999

Select Inst_Id Instance, Sum(Px_Servers_Executions) Par_Execs,
Sum(Elapsed_Time/1000000) Secs, Sum(Executions) Execs,
Sum(Buffer_Gets) Gets, Sum(Disk_Reads) Disk
From Gv$Sql
Where Sql_Id = 'chn8g49nskfaa'
Group By Inst_Id;
```

```
  INSTANCE  PAR_EXECS SECS      EXECS   GETS       DISK
---------- ---------- ---- ---------- ------ ----------
         1          0    5          1   3578        681
         3          4    1          0   2661       2764
```

We see that Oracle distributed the child parallel processes to node 3. As expected, there were 4 executions on this node.

Note that the child processes were apparently executed zero times! Of course, this is not really true—we just aren't looking in the right place. An important column used in the above script is the field, *Px_Servers_Executions*. This is the place where we find the child executions. This is the official definition from the Oracle documentation for this field:

> *Total number of executions performed by Parallel execution servers. The value is 0 when the statement has never been executed in parallel.*

LISTING INSTANCE GROUPS

Here is a simple way to see how your RAC cluster is setup:

```
Col Name Format A33
Col Value Format A33
Select Inst_Id, Name, Value From Gv$Parameter
Where Upper(Name) Like '%INST%GROUP%'
Order By Value, Inst_Id;

INST_ID NAME                              VALUE
------- --------------------------------- -------------
      1 instance_groups                   WEST COAST
      1 parallel_instance_group           WEST COAST

      2 instance_groups                   WEST COAST
      2 parallel_instance_group           WEST COAST

      3 parallel_instance_group           EAST COAST
      3 instance_groups                   EAST COAST

      4 parallel_instance_group           EAST COAST
      4 instance_groups                   EAST COAST
```

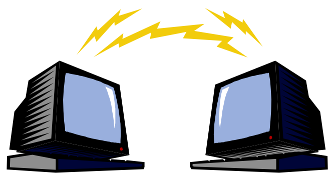
```
5 instance_groups                  REPORTS
5 parallel_instance_group          REPORTS

6 instance_groups                  RESERVIED
6 parallel_instance_group
```

With the above settings, parallel processing is mostly geographically distributed, with two servers designated for queries associated with east coast customers, and two for west coast customers. On the other hand, parallel queries originating on node 5 will launch the slaves only on node 5 (by default).

In the above list, node 6 is apparently special--queries originating on node 6 will be distributed to *any* of the other nodes, since each user on node 6 will, by default, not have a parallel_instance_group set. On the other hand, the nodes *other than node 6* won't be directed to node 6, because the other nodes don't specify the *Reserved* instance group.

Keep in mind that the setting for parallel_instance_group, when set at the database level, is simply the default setting for each session. This parameter is easily changeable at the session level.



SPLITTING PARALLEL PROCESSES ACROSS MULTIPLE NODES

It's conceivable that you may wish to have *multiple* nodes service one query. (Perhaps an extraordinary amount of CPU power is needed.) This feature is sometimes called, "cross-instance parallelism." It is controlled by the table-level parameter, *instances*. The default setting for *instances* is 1, which keeps all the child processes on a single node—either a different node, or the local node. In either case, the query coordinator stays on the local node.

When you increase the *Instances* parameter beyond 1, Oracle will consider distributing the parallel processes across multiple nodes. For example, the command below should cause Oracle to split the work between two nodes—with 4 child processes on *each* node:

```
ALTER TABLE NODETEST1 PARALLEL(DEGREE 4 INSTANCES 2)
```

We can also use the *parallel* hint to enable this feature, by adding an additional argument:

```
Select /*+PARALLEL(TAB, DEGREE, INSTANCES) */
```

In the hint above, the third field, *Instances*, serves the same function as the table-level parameter.

It seems that the *Instances* feature should be used with great care. Although using more cpu will likely prove beneficial, it obviously can't change the maximum disk i/o throughput capacity. It is also not clear if there are any downsides to coordinating child processes across different nodes. As with any new feature, test carefully before implementing in production.

WRAP-UP

There are quite a few possibilities for using parallel processing on RAC. When properly setup, the DBA can efficiently allocate resources to the proper node. The DBA should exercise great care and test the scenarios.

In concept, RAC instance groups are easy to understand, but the terms can be confusing. It's worth spending the time to sort-out the confusing terminology.

Chris Lawson is an Oracle Performance specialist and Oracle "Ace" who lives with his family in Dublin, California. He is the author of ***The Art & Science of Oracle Performance Tuning***, as well as ***Snappy Interviews: 100 Questions to Ask Oracle DBAs,*** both available on Amazon. Chris' web site is: www.Oraclemagician.Com. In his spare time, Chris enjoys golf, choral singing (bass), and throwing frisbees to Morgan, the resident border collie.