Load Balancing and Fail Over In Oracle Parallel Server

By Arup Nanda

Learn how to distribute client connection load among nodes of Oracle Parallel Server using no special tools but Net8.

Toolbox: Oracle Products Used: Oracle Parallel Server 8.1.7 or 9*i*, Net 8*i* or Oracle Net (9*i*). User Expertise Required: Intermediate DBA Experience, Multithreaded Server, Net8 Configuration.

One of the most challenging tasks for Oracle Parallel Server DBAs, or as it is now called, Real Application Clusters, is to effectively balance load across all the nodes of the cluster. Most of the time, the load balancing is accomplished in one of two ways:

- Using third party tools that redirect connections to individual nodes
- Assigning users to specific nodes

The first solution is viable but may be expensive to purchase and maintain. This option also adds an administrative overhead to the IT infrastructure.

The second option is not truly a load balancing solution. It is often possible that all of the users assigned to one node are connected whereas the other users assigned to another node are not, creating a severe imbalance.

The other problem is the need to have nodes serving as backups for each other, as a failover strategy. This article discusses how to set up the OPS environment using the standard Oracle tools, i.e. Net8, also called Oracle Net.

Background/Overview

Before starting to explore the topic further, it is necessary to understand the way Oracle handles connection requests from clients as shown in the example system in Figure 1.

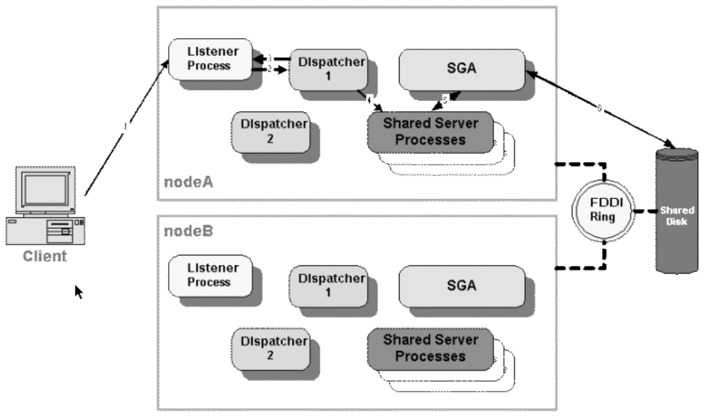

Figure 1: The Normal Process

This system assumes is a two-node parallel cluster on a FDDI ring. But it could also have been any number of nodes on an Ethernet-based backbone. The two nodes are named nodeA and nodeB. Each runs one listener and starts off two dispatchers of TCP/IP protocol. The process is as follows:

1. The client requests connection to the listener process on nodeA
2. After authentication, the listener redirects the connection to one of the dispatchers in the node available to the listeners. The listener has the data on the load status of the dispatchers so it assigns the request to the least loaded dispatcher.
3. The dispatcher updates the listener with the new load data.
4. The dispatcher puts it in a queue. It is picked up from the queue and assigned to one of the shared servers. If a shared server is not available, one is started.
5. The shared server picks up the data from SGA. This step is more complicated and a detailed discussion is beyond the scope of this article.
6. If not present in the SGA, the data is transferred from the disk and eventually to the client.

continued on page 38

The problem is the creation of two potential bottlenecks. If more than one session is started from the client, the listener and dispatchers of nodeA will become overloaded while those in nodeB will be unutilized. Since the clients are set up to access either nodeA or nodeB, it's possible that the load on one node will significantly outweigh that on the other if the clients accessing one outnumber those accessing the other. Therefore the listener and the dispatchers can become possible bottlenecks.

The description above is evidence for the need to create a load-balanced system. To solve this problem, two objectives can be met:

1. The client should be able to choose between the nodes at random and should not be tied to a particular node.
2. After the listener listens to the request, it should redirect the connection to a least-loaded dispatcher in either node, regardless of the node in which the listener is started.

Client Load Balancing

On the client side, the file tnsnames.ora determines how to translate the connect string to the host and service names. You can find this file in the sub-directory network/admin under the Oracle home directory. In some flavors of Unix (e.g. Solaris) it may be found in the /var/opt/oracle directory instead. If the TNS_ADMIN environment variable is set, it specifies the directory where the tnsnames.ora file can be found. The existing tnsnames.ora file probably looks like Listing 1:

```
Listing 1 : Existing TNSNAMES.ORA File on Client

MYINST1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeA)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = MYINST)
    )
  )

MYINST2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeB)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = MYINST)
    )
  )
```

Note that MYINST1 refers the instance on nodeA and MYINST2 refers to node2. Typically the client either connects using SQL Plus scott/tiger@MYINST1 or scott/tiger@MYINST2.

On the client side, this is not a true load balancing setup. You can make the necessary changes to instruct the client to connect to either node's listener at random and not have to specify the explicit node to connect. The fact that a listener is picked up at random will ensure that both listeners will be loaded fairly evenly. Make changes to file tnsnames.ora as shown in Listing 2:

```
Listing 2 : Changed TNSNAMES.ORA File on Client

MYINST =
    (DESCRIPTION =
        (LOAD_BALANCE = ON)
        (FAILOVER = ON)
        (ADDRESS_LIST =
            (FAIL_OVER=ON)
            (ADDRESS= (PROTOCOL=TCP) (HOST=nodeA) (PORT=1521))
            (ADDRESS= (PROTOCOL=TCP) (HOST=nodeA) (PORT=1526))
        )
        (ADDRESS_LIST =
            (FAIL_OVER=ON)
            (ADDRESS= (PROTOCOL=TCP) (HOST=nodeB) (PORT=1521))
            (ADDRESS= (PROTOCOL=TCP) (HOST=nodeB) (PORT=1526))
        )
        (CONNECT_DATA =
            (SERVICE_NAME = MYINST)
        )
    )
```

Note the way that the service MYINST has been defined with two nodes and two listeners per node. The CONNECT_DATA has been described only once whereas the ADDRESS_LIST has been described twice, once for each node's listener. The parameter LOAD_BALANCE=ON makes sure that the client chooses between the listeners at random. The parameter FAIL_OVER=ON instructs the client to try the next ADDRESS_LIST if the one chosen at random is down. The FAIL_OVER=ON inside the ADDRESS_LIST has been provided with an assumption that there are two listeners on each node listening on ports 1521 and 1526 respectively. If one listener fails, the other can be picked up for connection.

Cross-Registration of Listeners

The next step is to configure the listeners in each node. On nodeA, replace the file listener.ora as shown in Listing 3.

```
Listing 3 : Changed LISTENER.ORA File on nodeA

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = nodeA)(PORT = 1521))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
    )
  )

LISTENER1 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = nodeA)(PORT = 1526))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /oracle/product/8.1.7)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = MYINST1.WORLD)
      (ORACLE_HOME = /oracle/product/8.1.7)
      (SID_NAME = MYINST1)
    )
  )


SID_LIST_LISTENER1 =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = MYINST1.WORLD)
      (ORACLE_HOME = /oracle/product/8.1.7)
      (SID_NAME = MYINST1)
    )
  )

LOG_FILE_LISTENER = listener
LOG_FILE_LISTENER1 = listener1
```

Note the way in which LISTENER and LISTENER1 (which listen to ports 1521 and 1526 respectively) have been defined. Make the same change in nodeB, replacing nodeA with nodeB in the listener.ora file.

Next, it is necessary to register the instance on nodeA with the listeners on nodeB and vice versa. First, we need to describe the aliases for the listeners by making some changes in tnsnames.ora file on both the servers as shown in Listing 4.

```
Listing 4 : Changed TNSNAMES.ORA File on Server

MYINST1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeA)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeA)(PORT = 1526))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = MYINST)
    )
  )

MYINST2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeB)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = nodeB)(PORT = 1526))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = MYINST)
    )
  )
```

Then we will make the instance register itself to listeners on the other node. Make the change in init.ora file as in Listing 5.

```
Listing 5 : Changed INIT.ORA on nodeA and nodeB.

mts_dispatchers="(protocol=TCP)(listener=MYINST1)(dispatchers=4)(connections=5
00)"
mts_dispatchers="(protocol=TCP)(listener=MYINST2)(dispatchers=4)(connections=5
00)"
mts_multiple_listeners=TRUE
```

Note the way in which the listener parameter is described as MYINST1, meaning that the instance is registering itself with the listener as defined in the tnsnames.ora, which points it to the listener listening on nodeA. The line states that 4 dispatchers of TCP/IP protocol are to be started and be registered with the listener. The instance also registers itself with the listener on nodeB as indicated by the listener=MYINST2 option.

Make the same changes in init.ora of nodeB as well. Shut down and restart the instances and listeners on both nodes. Since you have two listeners on each node now, makes sure that you start the non-default listener by issuing the command lsnrctl START LISTENER1. Test it to make sure that the cross-registration works by issuing the command lsnrctl SERVICES from the Unix command prompt on nodeA. The output should resemble Listing 6.

```
Listing 6 : Output from LSNRCTL SERVICES Command

MYINST         has 5 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D003 <machine: nodeA, pid: 6872>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeA)(PORT=54860))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D002 <machine: nodeA, pid: 6870>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeA)(PORT=54859))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D001 <machine: nodeA, pid: 6868>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeA)(PORT=54858))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D000 <machine: nodeA, pid: 6866>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeA)(PORT=54857))
  MYINST         has 4 service handler(s)
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D003 <machine: nodeB, pid: 6969>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeB)(PORT=54941))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D002 <machine: nodeB, pid: 6967>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeB)(PORT=54940))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D001 <machine: nodeB, pid: 6965>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeB)(PORT=54939))
    DISPATCHER established:0 refused:0 current:0 max:500 state:ready
      D000 <machine: nodeB, pid: 6963>
      (ADDRESS=(PROTOCOL=tcp)(HOST=nodeB)(PORT=54938))
```

This listing states that there are 4 TCP/IP dispatchers with process ids 6866, 6868, 6870 and 6872 on nodeA and 4 TCP/IP dispatchers on nodeB with process ids 6963, 6965, 6967 and 6969.

If the command above does not display as expected, it probably means that the multi-threaded option is not enabled. Check the MTS options in init.ora file and try again. Check for the parameter AUTOMATIC_IPC in sqlnet.ora file. It should be set to FALSE.

If the HOST value in the output displays 0.0.0.0, it means that the hostname was not resolved. From the Unix prompt, enter the hostname command. If it gives an error or does not return anything, the names are not resolved. In that case, simply enter the IP address wherever the name appears in init.ora and tnsnames.ora.

Testing the Load Balanced Setup

Now that all of the parts are set up, the next step is to test it to make sure that it is working as intended. From the client, connect to the database by using SQL Plus scott/tiger@MYINST and issue a statement SELECT HOST_NAME FROM V$INSTANCE to check the machine connection. Disconnect and connect again and it might show

continued on page 40

nodeB this time. This means that it's connected to the listener on the node that comes up and may not be connected to the instance on that node.

To check it, log on as sys from another terminal connecting to nodeA and issue the following command: SELECT USERNAME FROM V$SESSION. You may not find SCOTT there. Try the same query in nodeB. Connect a few times from the client and check both the nodes in v$session view to see which instance the client finally connected to. It should be connecting at an even rate to each instance. Alternatively, you can query the gv$session and check the instance number and the username of the connections. However, to use gv$session, you must have parallel query servers enabled in both instances.

To see how it helps load balancing, see Figure 2, which is a variation of Figure 1 (the non-load-balanced approach).

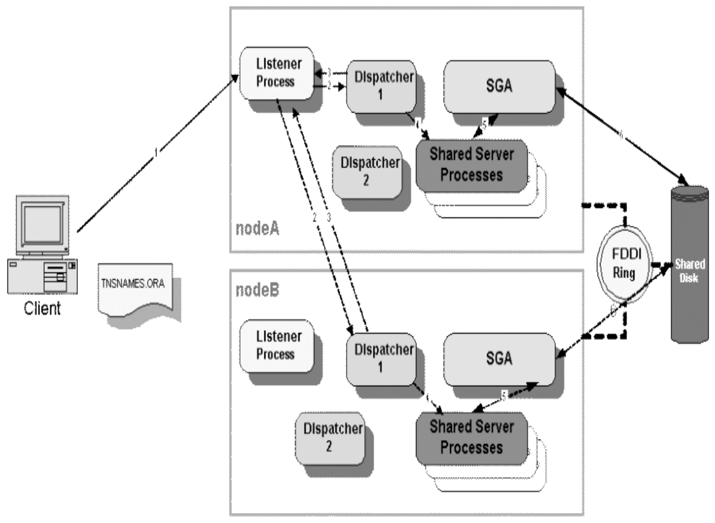

Fig. 2: The Load Balanced Process

Using the approach described above, the new path is indicated by broken lines. The revised steps are listed below. To simplify the diagram, the second listener LISTENER1 has not been shown in either node.

Client makes a connection request to alias MYINST, *not* MYINST1 or MYINST2 as was the case before. The Net8 client chose the nodeA at random. It could have chosen nodeB with the same probability.

The Listener on nodeA has the load data on dispatchers not just in nodeA but also in nodeB since both the instances has registered themselves with both the listeners. From that data, it identified that a dispatcher in nodeB is under-loaded and it redirected the connection to it. Technically, however, the listener does not redirect the connection since the dispatcher is in a separate machine. It will pass the address of the dispatcher to the client and the client then will connect to the dispatcher in that address.

The rest of the process is the same as above with the dispatcher putting in a queue to be picked up by a shared server and so on. The only difference is everything from the dispatcher now happens on nodeB whereas the client initially got connected to nodeA. In this setup, both the nodes will be evenly loaded.

Conclusions

In the example discussed here, a two-node cluster was used. But the same strategy can work with any number of nodes and any number of listeners on each node. You can even make the load balancing between listeners on one node.

The beauty of this approach goes beyond just OPS installations. You can use it to provide fail-over and load balancing to any number of databases that can be identical. For example you may have two databases serving as reporting servers containing identical data. The clients can connect to either one and have the servers evenly balanced for load

About the Author

Arup Nanda (Arup@InfoSpectrumSolutions.com) has been an Oracle DBA for more than nine years and is the founder of InfoSpectrum Solutions (www.InfoSpectrumSolutions.com), a New York area based consulting company providing highly specialized Oracle-only database solutions and services like Disaster Recovery Solutions, Replication Setup, Large Database Strategies and much more.