CHAPTER 14

RAC Troubleshooting

In this chapter we look into the details of debugging Oracle Real Application Clusters, from simple startup problems to complex system hang or crash problems. As a single piece of software, Oracle RDBMS is the one of the most complex commercial products in the world. But with the help of a solid and extensive diagnostics framework, you can usually diagnose even complex problems simply by viewing and interpreting Oracle's detailed trace files.

Each instance in a cluster has its own alert logs, which will be the first and foremost thing to examine whenever a problem is reported. Alert logs show detailed information about the basic settings of the database, including the non-default parameters used. Alert logs also contain information about startup and shutdown, and details of node(s) joining and leaving with timestamps. The alert log is specific to each instance and the location of the log is specified by the initialization parameter `background_dump_dest`, which also defines the location of the background process trace files. The trace files of the other background process such as LMON or LMD traces are also written in the location specified by this parameter. If the shared server is configured, trace files of the shared servers are also written in the directory.

Log Directory Structure in Cluster Ready Services

To diagnose any problem, the first thing examined by Oracle Support are the installation log files. Anyone who knows anything about database administration knows the importance of the dump directories (bdump, udump, and cdump). Similarly, each component in the CRS stack has its respective directories created under the CRS home:

- **$ORA_CRS_HOME/crs/log** Contains trace files for the CRS resources.
- **$ORA_CRS_HOME/crs/init** Contains trace files of the CRS daemon during startup. Good place to start with any CRS login problems.
- **$ORA_CRS_HOME/css/log** The Cluster Synchronization (CSS) logs indicate all actions such as reconfigurations, missed check-ins, connects, and disconnects from the client CSS listener. In some cases, the logger logs messages with the category of auth.crit for the reboots done by Oracle. This could be used for checking the exact time when the reboot occurred.
- **$ORA_CRS_HOME/css/init** Contains core dumps from the Oracle Cluster Synchronization Service daemon (OCSSd) and the process ID (PID) for the CSS daemon whose death is treated as fatal. If abnormal restarts for CSS exist, the core files will have the format of `core.<pid>`.
- **$ORA_CRS_HOME/evm/log** Log files for the Event Volume Manager (EVM) and evmlogger daemons. Not used as often for debugging as the CRS and CSS directories.
- **$ORA_CRS_HOME/evm/init** PID and lock files for EVM. Core files for EVM should also be written here.
- **$ORA_CRS_HOME/srvm/log** Log files for Oracle Cluster Registry (OCR), which contains the details at the Oracle cluster level.
- **$ORA_CRS_HOME//log** Log files for Oracle Clusterware (known as the cluster alert log), which contains diagnostic messages at the Oracle cluster level. This is available from Oracle database 10g R2.

Log Directory Structure in the Oracle RDBMS

In a 10g RAC installation, a node consists of a CRS home and an Oracle RDBMS home. The RDBMS home is where you install the Oracle software with the RAC option. You create databases from this home. The log files are related to each of the database instances running out of this Oracle home. The most important log file that every database generates is the alert.log file that is created in the directory specified by the init.ora parameter `background_dump_dest`. For most sites, this directory is kept under $ORACLE_BASE/admin.

Suppose, for example, that you have a database named *test*. All its background process trace files, along with the alert.log, would be available in the $ORACLE_BASE/admin/test/bdump directory; that's the norm. Its not a hard and fast rule that you use this directory structure, but it does help to follow conventions.

These directories are also important for the RDBMS home:

- **$ORACLE_BASE/admin/udump** Contains any trace file generated by a user process.
- **$ORACLE_BASE/admin/cdump** Contains core files that are generated due to a core dump in a user process.

Let's take a closer look at the startup of an instance in a two-node RAC cluster running version 10.1.0.4. We use SQL *Plus to issue startup on node1 and then on node2. The sequence of events follows with additional explanations added as and when required:

```
SQL> startup nomount;
Cluster communication is configured to use the following interface(s) for this instance
  192.168.0.1
Mon Aug 29 07:25:09 2005
cluster interconnect IPC version:Oracle UDP/IP
IPC Vendor 1 proto 2 Version 1.0
PMON started with pid=2, OS id=31242
```

The correct interconnect must be used for the Cache Fusion traffic. Some may choose the public network for the interconnect traffic, but doing so will bring the database to its knees. To identify the network used for Cache Fusion or private traffic, you can do any of the following:

```
$ oifcfg getif


SQL> select INST_ID,PUB_KSXPIA,PICKED_KSXPIA,NAME_KSXPIA,IP_KSXPIA
from x$ksxpia;

   INST_ID P PICK NAME_KSXPIA     IP_KSXPIA
---------- - ---- --------------- ----------------
         1   OCR  prod1           192.168.0.1
```

Depending on the source of the information, the `PICKED_KSXPIA` is populated with the values `OSD`, `OCR`, and `CI`. If the `cluster_interconnects` parameter is set in the SPFILE, the query will return the following output:

```
SQL> select INST_ID,PUB_KSXPIA,PICKED_KSXPIA,NAME_KSXPIA,IP_KSXPIA
from x$ksxpia;
   INST_ID P PICK NAME_KSXPIA     IP_KSXPIA
---------- - ---- --------------- ----------------
         1   CI   prod1           192.168.0.1
```

Another option is the conventional method of finding the interconnect information from an Interprocess Communications (IPC) dump. This was the only method available in Oracle versions prior to 10g, but this method is also possible in Oracle 10g. Starting from 10g R2, X$KSXPIA is exposed as `GV$CONFIGURED_INTERCONNECTS`.

Log in to the database as user *SYS*:

```
SQL> oradebug setmypid
SQL> oradebug ipc
```

This will dump a trace file to `user_dump_dest`. The output will look something like this:

```
SSKGXPT 0x1a2932c flags SSKGXPT_READPENDING info for network 0
socket no 10 IP 192.168.0.1 UDP 43749
sflags SSKGXPT_WRITESSKGXPT_UP info for network 1
socket no 0 IP 0.0.0.0 UDP 0...
```

You can see that we are using IP 192.168.0.1 with a User Datagram Protocol (UDP).

To change the network that RAC uses, you can change the order of the network(s) in the operating system–dependent network configurations, such as /etc/hosts or by using the `cluster_interconnects` parameter.

NOTE
A little caution is required while using the `cluster_interconnects` parameter. With Oracle 10g, interconnect information is stored in OCR, so you don't need to specify the interconnect using the `cluster_interconnects` parameter. Although this parameter supports load balancing among the specified interfaces, it does not provide failover capabilities, and in case one of several interfaces goes down, Oracle will treat it as a complete failure and start evicting instances. You can still use `cluster_interconnects` with one IP address—for example, to work around a bug in the network selection or to use a specific NIC for a particular instance for test purposes.

The moment the first instance is started, its alert log gets populated with important information. The first thing to notice in the preceding example after the list of non-default parameters is the IP address used for the interconnect. On the first instance, mentioning the `cluster_interconnects` parameter is necessary so that the correct interface is picked up—due to a bug that doesn't pick up the correct interface for the interconnect. Also shown is the protocol used—in our case, UDP. The value shown would depend on your platform-specific usage—RSM could be used on Sun, HyperFabric on HP, and so on. Ensure that you see the correct protocol that you have configured for the interconnect usage. The supported interconnect protocols with respect to various operating systems are listed in Chapter 3.

```
Mon Aug 29 07:25:11 2005
lmon registered with NM - instance id 1 (internal mem no 0)
Mon Aug 29 07:25:11 2005
Reconfiguration started (old inc 0, new inc 1)
```

```
List of nodes:
 0
 Global Resource Directory frozen
 Update rdomain variables
 Communication channels reestablished
 Master broadcasted resource hash value bitmaps
 Non-local Process blocks cleaned out
Mon Aug 29 07:25:11 2005
 LMS 1: 0 GCS shadows cancelled, 0 closed
Mon Aug 29 07:25:11 2005
 LMS 0: 0 GCS shadows cancelled, 0 closed
 Set master node info
 Submitted all remote-enqueue requests
 Dwn-cvts replayed, VALBLKs dubious
 All grantable enqueues granted
 Post SMON to start 1st pass IR
Mon Aug 29 07:25:11 2005
 LMS 0: 0 GCS shadows traversed, 0 replayed
Mon Aug 29 07:25:11 2005
 LMS 1: 0 GCS shadows traversed, 0 replayed
Mon Aug 29 07:25:11 2005
 Submitted all GCS remote-cache requests
 Post SMON to start 1st pass IR
 Fix write in gcs resources
Reconfiguration complete
LCK0 started with pid=17, OS id=31272
```

When an instance starts up, it's the Lock Monitor's (LMON) job to register with the Node Monitor (NM). That's what we see in the alert.log with the instance ID that is getting registered. When any node joins or leaves a cluster, the global resource directory undergoes a reconfiguration event. We see the start of the reconfiguration event along with the old and new incarnation. Next, we see the number of nodes that have joined the cluster. As this was the first node to be started up, in the list of nodes we see only one node listed, and the number starts with 0. A reconfiguration event is a seven-step procedure and upon completion the "reconfiguration complete" message is logged into the alert.log.

The messages logged in the alert.log are summaries of the reconfiguration event. The LMON trace file would have more information about the reconfiguration. Following are the contents of the LMON trace file:

```
*** 2005-08-29 07:25:11.235
kjxgmrcfg: Reconfiguration started, reason 1
kjxgmcs: Setting state to 0 0.
```

Here, you can see the reason for the reconfiguration event. The most common reasons would be 1, 2, or 3. Reason 1 means that the NM initiated the reconfiguration event, as typically seen when a node joins or leaves a cluster. A reconfiguration event is initiated with reason 2 when an instance death is detected. How is an instance death detected? Every instance updates the control file with a heartbeat through its Checkpoint (CKPT) process. If heartbeat information is not present for *x* amount of time, the instance is considered to be dead and the Instance Membership Recovery (IMR) process initiates reconfiguration. This type of reconfiguration is commonly seen when significant time changes occur across nodes, the node is starved for CPU or I/O times, or some problems occur with the shared storage.

A reason 3 reconfiguration event is due to a communication failure. Communication channels are established between the Oracle processes across the nodes. This communication occurs over the interconnect. Every message sender expects an acknowledgment message from the receiver. If a message is not received for a timeout period, then a "communication failure" is assumed. This is more relevant for UDP, as Reliable Shared Memory (RSM), Reliable DataGram protocol (RDG), and Hyper Messaging Protocol (HMP) do not need it, since the acknowledgment mechanisms are built into the cluster communication and protocol itself.

When the block is sent from one instance to another using wire, especially when unreliable protocols such as UDP are used, it is best to get an acknowledgment message from the receiver. The acknowledgment is a simple side channel message that is normally required for most of the UNIX systems where UDP is used as the default IPC protocol. When user-mode IPC protocols such as RDG (on HP Tru64 UNIX TruCluster) or HP HMP are used, the additional messaging can be disabled by setting `_reliable_block_sends=TRUE`. For Windows-based systems, it is always recommended to leave the default value as is.

```
Database mounted in Shared Mode (CLUSTER_DATABASE=TRUE).
Completed: alter database mount
```

Since this is an RAC database, every instance mounts the database in shared mode. Sometimes you want to mount the database in exclusive mode, as in completing the actions of a patch set application. Checking the alert log is one way to confirm that:

```
Mon Aug 29 15:36:53 2005
alter database open
This instance was first to open
Picked Lamport scheme to generate SCNs
```

You may see the message depending on the version of the RAC and setting of the parameter `max_commit_propagation_delay`. In version 9i, every commit System Commit Numbers (SCN) is broadcasted to all the nodes, and the log writer is held up until all the interested redos are written to the disk. Starting with 10g, the wait is greatly reduced as the broadcast and commit are asynchronous. This means the system waits until it is sure that all nodes have seen the commit SCN. Any message with an SCN greater than commit SCN is deemed sufficient.

Before doing a broadcast, the process checks whether it has already received a higher SCN from that instance. It used the same SCN to determine whether a foreground or an LMS has to be posted. With 10g, this is decoupled: an SCN to release foregrounds and an SCN needed for shipping buffers. The init.ora parameter `_lgwr_async_broadcasts = true` can be used to change the broadcast method.

The Lamport Algorithm

The Lamport algorithm is fast and scalable in RAC as it generates the SCNs in parallel and the SCNs are assigned to the transactions on a first come, first serve basis. The Distributed Lock Manager (DLM) controls and monitors the lock management and conflict resolution.

Another algorithm that is simpler than Dr. Lamport's bakery algorithm is based on the broadcasting operation after a commit operation, and this is the default SCN generation algorithm in single instance Oracle implementations. In this method, the SCN is broadcasted to the participating nodes immediately after a commit operation. This ensures that read consistency is at the highest level so that the participating nodes know the current state of the transaction irrespective of the workload. This method puts additional load on the systems, as it has to broadcast the SCN for every commit; however, the other nodes can see the committed SCN immediately.

The initialization parameter `max_commit_propagation_delay` limits the maximum delay allowed for SCN propagation to the participating nodes and controls the SCN scheme used in the OPS/RAC environment. This parameter defaults to 7 seconds (700 centiseconds) in most of the platforms except for the Compaq Tru64 UNIX. When we set the `max_commit_propagation_delay` initialization parameter to any value less than 100, the broadcast on commit algorithm is used. The alert log is updated immediately after startup with the method used for SCN generation.

Let's start the second instance and check the reconfiguration event entries in the alert.log:

```
SQL> startup nomount;
```

In the alert log of the second instance we see this:

```
Mon Aug 29 17:40:35 2005
lmon registered with NM - instance id 2 (internal mem no 1)
Mon Aug 29 17:40:37 2005
Reconfiguration started (old inc 0, new inc 2)
List of nodes:
 0 1
 Global Resource Directory frozen
 Update rdomain variables
```

Now two nodes in the clusters have internal IDs of 0 and 1. The rest of the messages are pretty much the same as in the first instance's alert log file.

RAC ON and OFF

In some cases, you may want to disable the RAC options for testing purposes—perhaps to run a benchmark or convert the RAC binaries to single instance binaries. In such a case, you can use the following procedure to convert the RAC installation to non-RAC. Disabling and enabling RAC options are available only for UNIX platforms. Windows installations do not support relinking binaries with RAC ON and OFF.

Use the following steps to disable RAC (known as RAC OFF):

1. Log in as the Oracle software owner (which is typically the UNIX account *oracle*) in all nodes.
2. Shut down all the instances from all the nodes using a NORMAL or IMMEDIATE option.
3. Change the working directory to $ORACLE_HOME/lib:

   ```
   cd $ORACLE_HOME/lib
   ```

4. Run the following `make` command to relink the Oracle binaries without the RAC option:

   ```
   make -f ins_rdbms.mk rac_off
   ```

   This normally runs for few minutes and should not pose any errors.
5. Now relink the Oracle binaries:

   ```
   make -f ins_rdbms.mk ioracle
   ```

Now the Oracle binaries are relinked with the RAC OFF option. You may have to edit the init.ora or SPFILE parameters accordingly. If errors occur in step 4, you may need to contact Oracle Support and log a service request with the trace and log files.

Use the following steps to enable RAC (known as RAC ON):

1. Log in as the Oracle software owner (typically the UNIX account *oracle*) in all nodes.
2. Shut down all the instances from all the nodes using a NORMAL or IMMEDIATE option.
3. Change the working directory to $ORACLE_HOME/lib:

   ```
   cd $ORACLE_HOME/lib
   ```

4. Run the following `make` command to relink the Oracle binaries without the RAC option:

   ```
   make -f ins_rdbms.mk rac_on
   ```

   This normally runs for a few minutes and should not pose any errors.
5. Now relink the Oracle binaries:

   ```
   make -f ins_rdbms.mk ioracle
   ```

Now the Oracle binaries are relinked with the RAC ON option. You may need to edit the init.ora or SPFILE parameters accordingly. If any errors occur in step 4, you may need to contact Oracle Support and log a service request with the trace and log files.

Database Performance Issues

RAC databases have more than one instance using the same set of resources, and a resource may be requested by more than one instance. Resource sharing is well managed by Global Cache Services (GCS) and Global Enqueue Services (GES). However, in some cases, the resource management operations could run into a deadlock situation and the entire database may hang because of serialization issues. Sometimes, software bugs also cause database-hang issues, and these situations almost always require the intervention of Oracle Support in the form of a service request.

Database hang issues can be placed in the following categories:

- Hung database
- Hung session(s)
- Overall instance/database performance
- Query performance

We will examine only the hung database, as that is more critical and complex than the others and also related to our point of interest. Detailed texts are available for analyzing the database and query performance.

Hung Database

Oracle Support defines a "true" database hang as "an internal deadlock or a cyclical dependency between two or more processes." When dealing with DML locks (that is, enqueue type TM), Oracle is able to detect this dependency and roll back one of the processes to break the cyclical condition. On the other hand, when this situation occurs with internal kernel-level resources (such as latches or pins), Oracle is usually unable to automatically detect and resolve the deadlock.

If you encounter a database hang situation, you need to take system state dumps so that Oracle Support can begin to diagnose the root cause of the problem. Whenever you take such dumps for a hang, it is important to take at least three of them a few minutes apart, on all instances of your database. That way, evidence shows whether a resource is still being held from one time to the next.

The maxdump file size should be set to unlimited, as this will generate bigger and larger trace files, depending on the size of the System Global Area (SGA), the number of sessions logged in, and the workload on the system. The SYSTEMSTATE dump contains a separate section with information for each process. Normally, you need to take two or three dumps in regular intervals. Whenever you make SYSTEMSTATE dumps repetitively, make sure you reconnect every time so that you get a new process ID and also the new trace files. Expect HUGE trace files!

Starting with Oracle Database 10g, a SYSTEMSTATE dump includes session wait history information. If you are using Oracle 10g or later, you don't need to take multiple system state dumps. The SYSTEMSTATE dump can be taken by any of the following methods.

From SQL *Plus:

```
alter session set max_dump_file_size = unlimited;
alter session set events 'immediate trace name systemstate level 10';
```

Using oradebug:

```
REM The select below is to avoid problems on pre 8.0.6 databases
select * from dual;
oradebug setmypid
oradebug unlimit
oradebug dump systemstate 10
```

When the entire database is hung and you cannot connect to SQL *Plus, you can try invoking SQL *Plus with the `prelim` option if you're using Oracle 10g or later. This attaches the process to the Oracle instance and no SQL commands are run. No login triggers or no pre-processing is done, and no SQL queries are allowed to run. See the change in banner from normal sqlplus and prelim sqlplus.

```
$sqlplus -prelim
SQL*Plus: Release 10.2.0.1.0 - Production on Wed Nov 9 11:42:23 2005
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
Enter user-name: / as sysdba
SQL>
```

Alternatively, oradebug allows you to dump the global system state by connecting to one node. The following shows the global SYSTEMSTATE dump from oradebug:

```
oradebug –g all dump systemstate 10
```

The –g option is used for RAC databases only. This will dump system states for all the instances. The SYSTEMSTATE dump/trace file can be found in the user_dump_dest directory on the instance where the dump was generated.

Hanganalyze Utility

A severe performance problem cam be mistaken for a hang. This usually happens when contention is so bad that it *seems like* the database is completely hung. Usually, a SYSTEMSTATE dump is used to analyze these situations. However, if the instance is large with more than a few gigabytes of SGA and with a heavy workload, a SYSTEMSTATE dump may take an hour or more and often fails to dump the entire SGA and lock structures. Moreover, a SYSTEMSTATE dump has the following limitations when dealing with hang issues:

- Reads the SGA in a "dirty" manner, so it may be inconsistent when the time to dump all the process is long.
- Usually dumps a lot of information (most of which is not needed to determine the source of the hang), which makes it difficult to determine quickly the dependencies between processes.
- Does not identify "interesting" processes on which to perform additional dumps (ERRORSTACK or PROCESS STATE).
- Often very expensive operation in for databases that have large SGAs. A SYSTEMSTATE dump can take hours, and taking few continuous dumps within a few minutes interval is nearly impossible.

To overcome the limitations of the SYSTEMSTATE dump, a new utility called hanganalyze was introduced in Oracle 8i. In Oracle 9i, the `hanganalyze` command was enhanced to provide clusterwide information in RAC environments on a single shot. This uses the DIAG daemon process in the RAC process to communicate between the instances. Clusterwide, hanganalyze will generate information for all the sessions in the cluster regardless of the instance that issued the command.

Hanganalyze can be invoked from SQL *Plus or through oradebug (which is available when you connect as *SYS* in the SQL *Plus utility). The following syntax can be used to get a hanganalyze trace when connected to SQL *Plus:

```
alter session set events 'immediate trace name hanganalyze level <level>';
```

Or when logged in as *SYS*:

```
oradebug hanganalyze <level>
```

Clusterwide, hanganalyze can be obtained like so:

```
oradebug setmypid
oradebug setinst all
oradebug -g def hanganalyze <level>
```

The `<level>` sets the amount of additional information that will be extracted from the processes found by hanganalyze (ERROSTACK dump) based on the STATE of the node. The following table describes the various levels and the trace information emitted when they are set:

Level	**Trace Information**
1-2	Only hanganalyze output, no process dump at all
3	Level 2 + Dump only processes thought to be in a hang (IN_HANG state)
4	Level 3 + Dump leaf nodes (blockers) in wait chains (LEAF, LEAF_NW, IGN_DMP state)
5	Level 4 + Dump all processes involved in wait chains (NLEAF state)
10	Dump all processes (IGN state)

Hanganalyze uses internal kernel calls to determine whether a session is waiting for a resource and reports the relationships between blockers and waiters. In addition, it determines which processes are "interesting" to be dumped, and it may perform automatic PROCESS STATE dumps and ERRORSTACKs on those processes, based on the level used while executing hanganalyze.

NOTE
Hanganalyze is not intended to replace a SYSTEMSTATE dump, but it may serve as a road map to interpret a system state while diagnosing complex issues. Performance issues related to row cache objects, enqueues, and latches can be analyzed only with hanganalyze and/or a SYSTEMSTATE dump. The process state information in these dumps provides an accurate snapshot of the locks/latches held by each process or session. It also tells us which event the process was waiting for and if any TM locks were held for a transaction. Problems associated with DDL statement locks and row cache lock issues can be debugged using only these dumps.

Debugging Node Eviction Issues

One of the most common and complex issues in RAC is performing the root cause analysis (RCA) of the node eviction issues. A node is evicted from the cluster after it kills itself because it is not able to service the applications. This generally happens during the communication failure between the instances, when the instance is not able to send heartbeat information to the control file and various other reasons.

During failures, to avoid data corruption, the failing instance evicts itself from the cluster group. The node eviction process is reported as Oracle error ORA-29740 in the alert log and LMON trace files. To determine the root cause, the alert logs and trace files should be carefully analyzed; this process may require assistance from Oracle Support. To get into deeper levels of the node eviction process, you need to understand the basics of node membership and Instance Membership Recovery (IMR), also referred to as Instance Membership Reconfiguration.

Instance Membership Recovery

When a communication failure occurs between the instances, or when an instance is not able to issue the heartbeat information to the control file, the cluster group may be in danger of possible data corruption. In addition, when no mechanism is present to detect the failures, the entire cluster will hang. To address the issue, IMR was introduced in Oracle 9i and improved in Oracle 10g. IMR removes the failed instance from the cluster group. When a subset of a cluster group survives during failures, IMR ensures that the larger partition group survives and kills all other smaller groups.

IMR is a part of the service offered by Cluster Group Services (CGS). LMON is the key process that handles many of the CGS functionalities. As you know, cluster software (known as Cluster Manager, or CM) can be a vendor-provided or Oracle-provided infrastructure tool. CM facilitates communication between all nodes of the cluster and provides information on the health of each node—the node state. It detects failures and manages the basic membership of nodes in the cluster. CM works at the cluster level and not at the database or instance level.

Inside RAC, the Node Monitor (NM) provides information about nodes and their health by registering and communicating with the CM. NM services are provided by LMON. Node membership is represented as a bitmap in the GRD. A value of 0 denotes that a node is down and a value of 1 denotes that the node is up. There is no value to indicate a "transition" period such as during bootup or shutdown. LMON uses the global notification mechanism to let others know of a change in the node membership. Every time a node joins or leaves a cluster, this bitmap in the GRD has to be rebuilt and communicated to all registered members in the cluster.

Node membership registration and deregistration is done in a series of synchronized steps—a topic beyond the scope of this chapter. Basically, cluster members register and deregister from a group. The important thing to remember is that NM always communicates with the other instances in the cluster about their health and status using the CM. In contrast, if LMON needs to send a message to LMON on another instance, it can do so directly without the help or involvement of CM. It is important to differentiate between cluster communication and RAC communication.

A simple extract from the alert log file about member registration is provided here:

```
Thu Jan 1 00:02:17 1970
alter database mount
Thu Jan 1 00:02:17 1970
lmon registered with NM - instance id 1 (internal mem no 0)
Thu Jan 1 00:02:17 1970
Reconfiguration started
List of nodes: 0,
 Global Resource Directory frozen
```

Here you can see that this instance was the first to start up and that LMON registered itself with the NM interface, which is a part of the Oracle kernel.

When an instance joins or leaves the cluster, the LMON trace of another instance shows the reconfiguration of the GRD:

```
kjxgmpoll reconfig bitmap: 0 1 3
*** 1970-01-01 01:20:51.423
kjxgmrcfg: Reconfiguration started, reason 1
```

You may find these lines together with other lines asking SMON to perform instance recovery. This happens when any instance crash occurs or when an instance departs the cluster without deregistering in a normal fashion:

```
Post SMON to start 1st pass IR
*** 1970-01-01 01:20:51.423
kjxgmpoll reconfig bitmap: 0 1 3
*** 1970-01-01 01:20:51.423
kjxgmrcfg: Reconfiguration started, reason 1
kjxgmcs: Setting state to 2 0.
*** 1970-01-01 01:20:51.423
     Name Service frozen
```

The CGS is present primarily to provide a coherent and consistent view of the cluster from an OS perspective. It tells Oracle that *n* number of nodes are in the cluster. It is designed to provide a synchronized view of the cluster instance membership. Its main responsibility involves regular status checks of the members and measures whether they are valid in the group, and very importantly, it detects split-brain scenarios in case of communication failures.

Specific rules bind together members within the cluster group, which keeps the cluster in a consistent state:

- Each member should be able to communicate without any problems with any other registered and valid member in the group.
- Members should see all other registered members in the cluster as valid and have a consistent view.
- All members must be able to read from and write to the control file.

So, when a communication failure occurs between the instances, or when an instance is not able to issue the heartbeat information to the voting disk, IMR is triggered. Without IMR (there is no mechanism to detect the failures), the entire cluster could hang.

Member Voting

The CGS is responsible for checking whether members are valid. To determine periodically whether all members are alive, a voting mechanism is used to check the validity of each member. All members in the database group vote by providing details of what they presume the instance membership bitmap looks like. As mentioned, the bitmap is stored in the GRD. A predetermined master member tallies the vote flags of the status flag and communicates to the respective processes that the voting is done; then it waits for registration by all the members who have received the reconfigured bitmap.

How Voting Happens The CKPT process updates the control file every 3 seconds in an operation known as the *heartbeat*. CKPT writes into a single block that is unique for each instance, thus intra-instance coordination is not required. This block is called the *checkpoint progress record*.

All members attempt to obtain a lock on a control file record (the *result record*) for updating. The instance that obtains the lock tallies the votes from all members. The group membership must conform to the *decided (voted)* membership before allowing the GCS/GES reconfiguration to proceed. The *control file vote result record* is stored in the same block as the heartbeat in the control file checkpoint progress record.

In this scenario of dead instances and member evictions, a potentially disastrous situation could arise if pending I/O from a dead instance is to be flushed to the I/O subsystem. This could lead to potential data corruption. I/O from an abnormally departing instance cannot be flushed to disk if database integrity is to be maintained. To “shield” the database from data corruption in this situation, a technique called *I/O fencing* is used. I/O fencing implementation is a function of CM and depends on the clusterware vendor.

I/O fencing is designed to guarantee data integrity in the case of faulty cluster communications causing a split-brain condition. A split-brain occurs when cluster nodes hang or node interconnects fail, and as a result, the nodes lose the communication link between them and the cluster. Split-brain is a problem in any clustered environment and is a symptom of clustering solutions and not RAC. Split-brain conditions can cause database corruption when nodes become uncoordinated in their access to the shared data files.

For a two-node cluster, split-brain occurs when nodes in a cluster cannot talk to each other (the internode links fail) and each node assumes it is the only surviving member of the cluster. If the nodes in the cluster have uncoordinated access to the shared storage area, they would end up overwriting each other’s data, causing data corruption because each node assumes ownership of shared data. To prevent data corruption, one node must be asked to leave the cluster or should be forced out immediately. This is where IMR comes in, as explained earlier in the chapter.

Many internal (hidden) parameters control IMR and determine when it should start. If a vendor clusterware is used, split-brain resolution is left to it and Oracle would have to wait for the clusterware to provide a consistent view of the cluster and resolve the split-brain issue. This can potentially cause a delay (and a hang in the whole cluster) because each node can potentially think it is the master and try to own all the shared resources. Still, Oracle relies on the clusterware for resolving these challenging issues.

Note that Oracle does not wait indefinitely for the clusterware to resolve a split-brain issue, but a timer is used to trigger an IMR-based node eviction. These internal timers are also controlled using hidden parameters. The default values of these hidden parameters are not to be touched as that can cause severe performance or operational issues with the cluster.

An obvious question that pops up in your mind might be, Why does a split-brain condition take place? Why does Oracle say a link is down, when my communication engineer says its fine? This is easier asked than answered, as the underlying hardware and software layers that support RAC are too complex, and it can be a nightmare trying to figure out why a cluster broke in two when things seem to be normal. Usually, configuration of communication links and bugs in the clusterware can cause these issues.

As mentioned time and again, Oracle completely relies on the cluster software to provide cluster services, and if something is awry, Oracle, in its overzealous quest to protect data integrity, evicts nodes or aborts an instance and assumes that something is wrong with the cluster.

Cluster reconfiguration is initiated when NM indicates a change in the database group, or IMR detects a problem. Reconfiguration is initially managed by the CGS and after this is completed, IDLM (GES/GCS) reconfiguration starts.

Cluster Reconfiguration Steps

The cluster reconfiguration process triggers IMR, and a seven-step process ensures complete reconfiguration.

1. Name service is frozen. The CGS contains an internal database of all the members/instances in the cluster with all their configuration and servicing details. The name service provides a mechanism to address this configuration data in a structured and synchronized manner.
2. Lock database (IDLM) is frozen. The lock database is frozen to prevent processes from obtaining locks on resources that were mastered by the departing/dead instance.
3. Determination of membership and validation and IMR.
4. Bitmap rebuild takes place, instance name and uniqueness verification. CGS must synchronize the cluster to be sure that all members get the reconfiguration event and that they all see the same bitmap.
5. Delete all dead instance entries and republish all names newly configured.
6. Unfreeze and release name service for use.
7. Hand over reconfiguration to GES/GCS.

Now that you know when IMR starts and node evictions take place, let's look at the corresponding messages in the alert log and LMON trace files to get a better picture. (The logs have been edited for brevity. Note all the lines in boldface define the most important steps in IMR and the handoff to other recovery steps in CGS.)

Problem with a Node Assume a four-node cluster (instances A, B, C, and D), in which instance C has a problem communicating with other nodes because its private link is down. All other services on this node are assumed to be working normally.

Alter log on instance C:

```
ORA-29740: evicted by member 2, group incarnation 6
Thu Jun 30 09:15:59 2005
LMON: terminating instance due to error 29740
Instance terminated by LMON, pid = 692304
…
…
…
```

Alter log on instance A:

```
Thu Jun 30 09:15:59 2005
Communications reconfiguration: instance 2
Evicting instance 3 from cluster
Thu Jun 30 09:16:29 2005
Trace dumping is performing id=[50630091559]
Thu Jun 30 09:16:31 2005
Waiting for instances to leave:
3
Thu Jun 30 09:16:51 2005
Waiting for instances to leave:
3
Thu Jun 30 09:17:04 2005
Reconfiguration started
List of nodes: 0,1,3,
 Global Resource Directory frozen
 Communication channels reestablished
 Master broadcasted resource hash value bitmaps
Thu Jun 30 09:17:04 2005
Reconfiguration started
```

LMON trace file on instance A:

```
*** 2005-06-30 09:15:58.262
kjxgrgetresults: Detect reconfig from 1, seq 12, reason 3
kjxgfipccb: msg 0x1113dcfa8, mbo 0x1113dcfa0, type 22, ack 0, ref 0,
stat 3
kjxgfipccb: Send timed out, stat 3 inst 2, type 22, tkt (10496,1496)
*** 2005-06-30 09:15:59.070
kjxgrcomerr: Communications reconfig: instance 2 (12,4)
Submitting asynchronized dump request [2]
kjxgfipccb: msg 0x1113d9498, mbo 0x1113d9490, type 22, ack 0, ref 0,
stat 6
kjxgfipccb: Send cancelled, stat 6 inst 2, type 22, tkt (10168,1496)
kjxgfipccb: msg 0x1113e54a8, mbo 0x1113e54a0, type 22, ack 0, ref 0,
stat 6
kjxgfipccb: Send cancelled, stat 6 inst 2, type 22, tkt (9840,1496)
```

Note that `Send timed out, stat 3 inst 2` is LMON trying send message(s) to the broken instance.

```
kjxgrrcfgchk: Initiating reconfig, reason 3 /* IMR Initiated */
*** 2005-06-30 09:16:03.305
kjxgmrcfg: Reconfiguration started, reason 3
kjxgmcs: Setting state to 12 0.
*** 2005-06-30 09:16:03.449
     Name Service frozen
kjxgmcs: Setting state to 12 1.
*** 2005-06-30 09:16:11.570
Voting results, upd 1, seq 13, bitmap: 0 1 3
```

Note that instance A has not tallied the vote; hence it has received only the voting results.
Here is an extract from the LMON trace file on instance B, which managed to tally the vote:

```
Obtained RR update lock for sequence 13, RR seq 13
*** 2005-06-30 09:16:11.570
Voting results, upd 0, seq 13, bitmap: 0 1 3
```

…
…
Here's the LMON trace file on instance A:

```
Evicting mem 2, stat 0x0007 err 0x0002
kjxgmps: proposing substate 2
kjxgmcs: Setting state to 13 2.
     Performed the unique instance identification check
kjxgmps: proposing substate 3
kjxgmcs: Setting state to 13 3.
     Name Service recovery started
     Deleted all dead-instance name entries
kjxgmps: proposing substate 4
kjxgmcs: Setting state to 13 4.
     Multicasted all local name entries for publish
     Replayed all pending requests
kjxgmps: proposing substate 5
kjxgmcs: Setting state to 13 5.
     Name Service normal
     Name Service recovery done
*** 2005-06-30 09:17:04.369
kjxgmrcfg: Reconfiguration started, reason 1
kjxgmcs: Setting state to 13 0.
*** 2005-06-30 09:17:04.371
     Name Service frozen
kjxgmcs: Setting state to 13 1.
```

GES/GCS recovery starts here:

```
Global Resource Directory frozen
node 0
node 1
node 3
res_master_weight for node 0 is 632960
res_master_weight for node 1 is 632960
res_master_weight for node 3 is 632960
…
…
…
```

Death of a Member For the same four-node cluster (A, B, C, and D), instance C has died unexpectedly:

```
kjxgrnbrisalive: (3, 4) not beating, HB: 561027672, 561027672
*** 2005-06-19 00:30:52.018
kjxgrnbrdead: Detected death of 3, initiating reconfig
kjxgrrcfgchk: Initiating reconfig, reason 2
*** 2005-06-19 00:30:57.035
kjxgmrcfg: Reconfiguration started, reason 2
kjxgmcs: Setting state to 6 0.
*** 2005-06-19 00:30:57.037
     Name Service frozen
kjxgmcs: Setting state to 6 1.
*** 2005-06-19 00:30:57.239
Obtained RR update lock for sequence 6, RR seq 6
*** 2005-06-19 00:33:27.261
Voting results, upd 0, seq 7, bitmap: 0 2
Evicting mem 3, stat 0x0007 err 0x0001
kjxgmps: proposing substate 2
kjxgmcs: Setting state to 7 2.
     Performed the unique instance identification check
kjxgmps: proposing substate 3
kjxgmcs: Setting state to 7 3.
     Name Service recovery started
     Deleted all dead-instance name entries
kjxgmps: proposing substate 4
kjxgmps: proposing substate 4
kjxgmcs: Setting state to 7 4.
     Multicasted all local name entries for publish
     Replayed all pending requests
kjxgmps: proposing substate 5
kjxgmcs: Setting state to 7 5.
     Name Service normal
     Name Service recovery done
*** 2005-06-19 00:33:27.266
kjxgmps: proposing substate 6
…
…
…
kjxgmps: proposing substate 2
```

GES/GCS recovery starts here:

```
Global Resource Directory frozen
node 0
node 2
res_master_weight for node 0 is 632960
res_master_weight for node 2 is 632960
 Total master weight = 1265920
 Dead  inst 3
Join  inst
 Exist inst 0 2
…
…
```

Debugging CRS and GSD Using DTRACING

Oracle Server management configuration tools include a diagnostic and trace facility. For verbose output for SRVCTL, GSD, GSDCTL, or SRVCONFIG, tracing can be enabled to provide additional screen output. The following steps explain the process of setting and tracing the other programs:

1. `vi` the gsd.sh/srvctl/srvconfig file in the $ORACLE_HOME/bin directory. In Windows, right-click the OraHome\bin\gsd.bat file and choose Edit.
2. At the end of the file, look for the following line:

   ```
   exec $JRE -classpath $CLASSPATH oracle.ops.mgmt.daemon.OPSMDaemon $MY_OHOME
   ```

3. Add the following just before the `-classpath` in the `exec $JRE` line:

   ```
   -DTRACING.ENABLED=true -DTRACING.LEVEL=2
   ```

4. At the end of the gsd.sh file, the string should now look like this:

   ```
   exec $JRE -DTRACING.ENABLED=true -DTRACING.LEVEL=2 -classpath.....
   ```

NOTE
Beginning with Oracle Database 10gd, setting the environment variable `SRVM_TRACE` to `true` traces all the SRVM files such as GSD, SRVCTL, and OCRCONFIG.

In a Nutshell

In this chapter, you have seen the basic to advanced methods of collecting diagnostic information when a hang situation occurs. You also studied the steps in node reconfiguration and IMR internals. Most of the complex problems may require assistance from Oracle Support, and this chapter will help you when dealing with Oracle Support personnel.