Session 5

Oracle Clusterware

Advanced RAC
Auckland - May 2008

Agenda

- Failover
- Oracle Clusterware
- Oracle Clusterware Utilities
- High Availability Framework

Failover
Instance Recovery

- The following steps are performed when a node fails:
  1. Enqueue resources are remastered
  2. Cache resources are remastered
  3. LMON recovers Global Resource Directory
  4. SMON recovers the database
  5. Recovery set is built
     - Failed redo threads are merged
  6. Resources are claimed
  7. Recovery set is rolled forward

- Database is
  - unavailable until resource claim is complete
  - partially available while recovery set is rolled forward
  - fully available thereafter

The slide describes the steps that are performed when an instance fails. Note that a similar set of steps must be performed when an instance leaves the cluster. Some of the above steps can overlap or occur in parallel.

The database will be completely unavailable from the time when the instance failure is detected until the resource claim is complete. The database will then be partially available while the recovery set is rolled forward and then rolled back. When this step is complete, the database will be fully available again.

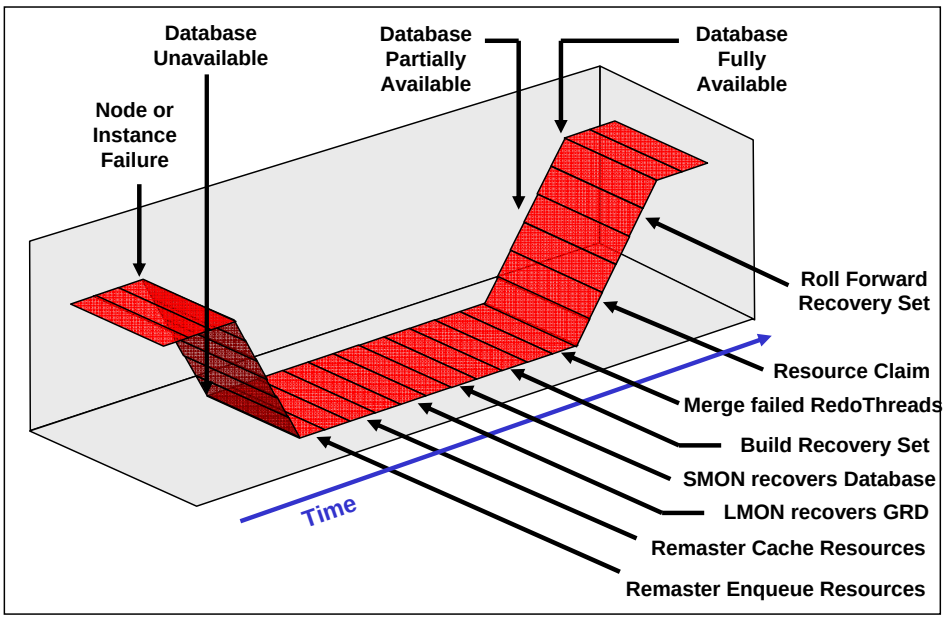
The time required to perform an instance recovery can be reduced by using parallel instance recovery, in particular setting the PARALLEL_MIN_SERVERS parameter. You can also reduce recovery times by

• increasing the PARALLEL_EXECUTION_MESSAGE_SIZE parameter

• using asynchronous I/O

• increasing the size of the default buffer cache

Concepts
Failover



This slide shows the steps performed by the recovery node following a node failure. When a node failure is detected, the database becomes unavailable on all instances in the cluster. Sessions will be unable to continue until all the resources managed by the failed instance have been re-mastered.

The instance detecting the failure manages the recovery process. Recovery consists of a number of steps, some of which can be performed in parallel. These include:

- Remastering enqueue resources (locks)
- Remastering cache resources (blocks)
- Identifying the set of blocks to be recovered
- Reserving the resources required for recovery

At this point the database can be made partially available. Block recovery will continue in the background. Any session attempting to read a block that has not yet been recovered will need to perform that recovery itself as part of the read operation.

The database continues to be partially available until all blocks have been recovered at which point the database is fully available again.

The recovery instance reads the online redo log of the failed instance and applies all changes to the affected data blocks. Any changes made by uncommitted transactions are subsequently rolled back.

Failover Parameters

- FAST_START_MTTR_TARGET
  - Introduced in Oracle 9.0.1
  - Specifies number of seconds database takes to perform crash recovery of a single instance
  - Range of values 0 - 3600 seconds
  - Can have different values on each instance
  - Overridden by LOG_CHECKPOINT_INTERVAL
  - Affects frequency of background checkpoints
  - Affects amount of time database is unavailable following an instance failure

The FAST_START_MTTR_TARGET parameter specifies the number of seconds that the database should take to perform crash recovery of a single instance. Setting this parameter specifies how aggressively you wish Oracle to perform background checkpoints during normal processing. Setting a lower value for this parameter will cause Oracle to perform more frequent background checkpoints. In the event of an instance failure, there will therefore be less redo to apply since the last checkpoint and consequently the time required to recover the transactions for the failed instance will be reduced.

FAST_START_MTTR_TARGET has a range of values between 0 and 36000 seconds. The default value is 0 seconds. It can have a different value on each instance, which might be appropriate if you are running an asynchronous workload.

Failover Parameters

- FAST_START_PARALLEL_ROLLBACK
  - Introduced in Oracle 8.1.5
  - Specifies maximum number of processes that can be used to perform parallel rollback
  - Useful on systems where some transactions are long-running
  - Values can be FALSE, LOW or HIGH
    - FALSE - parallel rollback disabled
    - LOW (default) - number of rollback processes limited to 2 * CPU_COUNT
    - HIGH - number of rollback processes limited to 4 * CPU_COUNT
  - Affects time required to rollback transactions following instance failure

The FAST_START_PARALLEL_ROLLBACK parameter specifies the maximum number of processes that can be used for performing parallel rollback.

Following an instance failure, transactions for the failed instance will be rolled forward and then any uncommitted transactions will be rolled back.

While the rollback is being performed, the database will only be partially available. The FAST_START_PARALLEL_ROLLBACK parameter specifies how many background processes will be used to perform the rollback. Additional processes will result in the rollback being completed more quickly, but more resources will be required to perform the rollback and therefore other activity on the instance will be affected.

If you have long-running transactions then setting FAST_START_PARALLEL_ROLLBACK to HIGH may improve recovery times. Otherwise the default value of LOW will probably be sufficient.

Failover Parameters

- _FAST_START_INSTANCE_RECOVERY_TARGET
  - Introduced in Oracle 10.1
  - Specifies cluster availability target time in a RAC environment
  - Integer value
  - Currently unsupported and undocumented
  - Discuss with Oracle Support before specifying a non-default value for this parameter on a production database

The _FAST_START_INSTANCE_RECOVERY_TARGET parameter was introduced in Oracle 10.1 and specifies the target time for cluster availability in a RAC environment. My understanding is that this parameter specifies the target time from instance failure to when the database is fully available again. In other words it specifies the time from instance failure to completion of transaction rollback.

The parameter is currently unsupported and undocumented and therefore you should contact Oracle Support before specifying a non-default value on a production database.

Although Oracle sometimes recommends the use of _FAST_START_INSTANCE_RECOVERY_TARGET in RAC environments in Oracle 10.2.0.2 and below setting FAST_START_MTTR_TARGET to 1 can yield faster instance recovery times. This is due to bug 4933038 which is fixed in Oracle 10.2.0.3

Oracle Clusterware Overview

- Oracle Clusterware has two main functions
  - Provide node membership services
  - Provide inter-node communications
- Oracle RAC supports two types of clusterware
  - Oracle Clusterware
  - Third-Party Clusterware

Clusterware has two main functions; to provide node membership services and to provide internode communications.

Oracle RAC provides two types of clusterware; Oracle Clusterware and Third Party Clusterware

Oracle Clusterware was originally licensed from Compaq. It was originally known as Oracle Cluster Management Services (OCMS) and was released on Linux and Windows platforms in Summer 2001. OCMS was renamed to CRS in Oracle 10.1 and again to Oracle Clusterware in Oracle 10.2

Third-party clusterware. In Oracle 9i, third-party clusterware was mandatory for RAC on all platforms other than Linux and Windows. In Oracle 10.1 and above, third-party clusterware is optional because Oracle Clusterware can run with or without it.

Examples of supported third-party clusterware include:

• Sun Cluster,

• HP Serviceguard and TruCluster,

• IBM High Availability Cluster Multiprocessing (HACMP) for AIX,

• Fujitsu PRIMECLUSTER

• Veritas Cluster Server.

Oracle Clusterware Overview

- Introduced in Oracle 10.1 (Cluster Ready Services - CRS)
- Renamed in Oracle 10.2 to Oracle Clusterware
- Cluster Manager providing
  - Node membership services
  - Global resource management
  - High availability functions
- On Linux
  - Configured in /etc/inittab
  - Implemented using three daemons
    - CSS - Cluster Synchronization Service
    - CRS - Cluster Ready Service
    - EVM - Event Manager
- In Oracle 10.2 includes High Availability framework
  - Allows non-Oracle applications to be managed

Oracle Clusterware includes a number of background processes that are implemented in Linux as daemons, including the following:

•Cluster Synchronization Service (CSS)

•Cluster Ready Services (CRS)

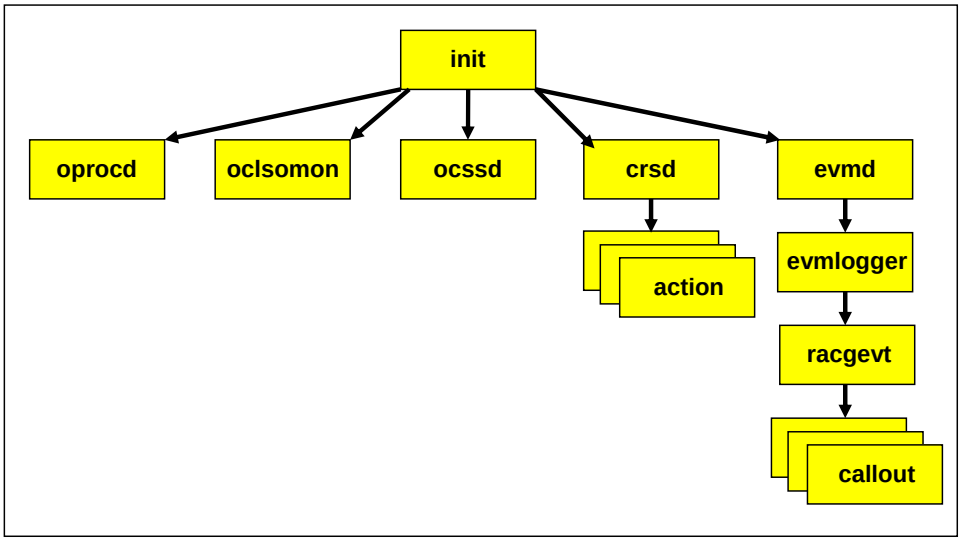
•Event Management (EVM)

These background processes communicate with similar components on other instances in the same database cluster. They also enable communication between Oracle Clusterware and the Oracle database. Under Linux, each daemon can have multiple threads, each of which appears as a separate operating system process.

On Unix platforms an additional daemon process called OPROCD is configured. This process is locked into memory to monitor the cluster and provide I/O fencing. It provides similar functionality to the hangcheck timer on Linux. OPROCD performs its check, stops running, and if the wake up is beyond the expected time, then OPROCD reboots the node. An OPROCD failure results in Oracle Clusterware restarting the node.

Oracle Clusterware Process Structure

- Unix-based systems



10 © 2008 Julian Dyke juliandyke.com

The OCLSOMON process was introduced in Oracle 10.2.0.2.

The OPROCD process was introduced on Linux platforms in Oracle 10.2.0.4

On Unix (not Linux) platforms OPROCD provides I/O fencing capabilities on clusters that do not use vendor clusterware.

Oracle Clusterware
Cluster Synchronization Service (CSS)

- **Manages cluster configuration by controlling which nodes are members of the cluster**
- **Consists of two parts:**
  - **Node Monitor**
  - **Group Manager**
- **When a node joins or leaves the cluster, CSS notifies the other nodes of the change in configuration**
- **Under Linux, CSS is implemented by the ocssd daemon which is started by the root user**
- **In Oracle 10.2.0.2 and above ocssd daemon is monitored by OCLSOMON process**

This component manages the cluster configuration by controlling which nodes are members of the cluster. When a node joins or leaves the cluster, CSS notifies the other nodes of the change in configuration. If this process fails, then the cluster will be restarted. Under Linux, CSS is implemented by the ocssd daemon, which runs as the root user.

The OCSSD process is spawned in init.cssd. It runs as the Oracle user. in both vendor Clusterware and non-vendor Clusterware environments. Its primary job is inter-node health monitoring and RDBMS instance endpoint discovery.

In Unix and Linux environments, the init daemon spawns init.cssd which in turn spawns the OCSSD daemon. If OCSSD dies or is kills then node kill functionality within the init script will kill the node. When a failure occurs, init actually respawns a second init.cssd script which attempts to start another OCSSD daemon and then on discovering it is already running, the second init.cssd will kill the node.

In Oracle 10.2.0.2 and above there is an additional process called OCLSOMON which monitors the CSS daemon for hangs or scheduling issues and can reboot a node if there is a perceived hang. OCLSOMON is spawned in init.cssd and runs as the Oracle user.

Oracle Clusterware Process Monitor (OPROCD)

- Process Monitor Daemon
  - Provides Cluster I/O Fencing
  - Implemented on Unix systems
    - Not required with third-party clusterware
  - Implemented in Linux in 10.2.0.4 and above
    - In 10.2.0.3 and below hangcheck timer module is used
  - Provides hangcheck timer functionality to maintain cluster integrity
    - Behaviour similar to hangcheck timer
    - Runs as root
    - Locked in memory
    - Failure causes reboot of system
    - See /etc/init.d/init.cssd for operating system reboot commands

OPROCD is spawned in any non-vendor Clusterware environment, except on Windows and earlier versions of Linux.

In Windows uses a kernel driver to perform the same actions

In Linux in Oracle 10.2.0.3 and below the hangcheck timer is used. In Oracle 10.2.0.4 and above and also in Oracle 11.1.0.6 and above Linux uses OPROCD.

OPROCD is spawned by init.cssd and runs as the root user. It is used todetect hardware and driver freezes on the local node. If the node has been frozen for long enough that the other nodes have evicted it from the cluster, OPROCD will kill the local node to prevent any further IO being issued to the disk after the rest of the cluster has been remastered. OPROCD kills the node using C code

Prior to Oracle 10.2.0.4, the OPROCD log file was cleared during a reboot. This made problem diagnosis very difficult. The problem has been fixed in 10.2.0.4 and above.

Oracle Clusterware Process Monitor (OPROCD)

- OPROCD takes two parameters
  - -t - Timeout value
    - Length of time between executions (milliseconds)
    - Normally defaults to 1000
  - -m - Margin
    - Acceptable margin before rebooting (milliseconds)
    - Normally defaults to 500
- Parameters are specified in /etc/init.d/init.cssd
  - OPROCD_DEFAULT_TIMEOUT=1000
  - OPROCD_DEFAULT_MARGIN=500
- Contact Oracle Support before changing these values

The OPROCD executable is intended to detect potential node hangs. When it detects a potential node hang, it will cause a node reboot, to ensure that, if it has been evicted by other cluster nodes, none of the processes can issue an I/O after the hang clears.

When I installed Oracle 10.2.0.4 on my test cluster, I was able to successfully install Oracle Clusterware which includes the new OPROCD daemon. However, I could not install the Oracle RDBMS patch set because OPROCD was rebooting the node running the OUI. I temporarily fixed this problem by increasing the values in /etc/init.d./init.cssd to

• OPROCD_DEFAULT_TIMEOUT=10000

• OPROCD_DEFAULT_MARGIN=5000

Restarting OPROCD with these values allowed me to complete the upgrade.

Note that this is not a supported fix, but it does give an insight into the way OPROCD operates.

Oracle Clusterware Process Monitor (OPROCD)

- /etc/init.d/init.cssd can increase OPROCD_DEFAULT_MARGIN based on two CSS variables
  - reboottime (mandatory)
  - diagwait (optional)
- Values can for these be obtained using

```
[root@server3]# crsctl get css reboottime
[root@server3]# crsctl get css diagwait
```

- Both values are reported in seconds
- The algorithm is

```
If diagwait > reboottime then
   OPROCD_DEFAULT_MARGIN := (diagwait - reboottime) * 1000
```

- Therefore increasing diagwait will reduce frequency of reboots e.g

```
[root@server3]# crsctl set css diagwait 13
```

14 juliandyke.com

The information on this slide is based on observation and bug reports. It is not official Oracle Support policy.

However in a test or unsupported environment it may help you to investigate timeout issues with OPROCD.

The CSS reboottime parameter defaults to 3 seconds and specfies the amount of time allowed for a node to complete a reboot after the CSS daemon has been evicted. (how long does it take for the machine to completely shutdown when it is rebooted). It can be set using

crsctl set css reboottime <value>

The diagwait environment variable allows you to increase the value of the OPROC_DEFAULT_MARGIN variable

The value of OPROCD_DEFAULT_MARGIN is 500 (milliseconds). Therefore setting diagwait to 4 or more will cause the OPROC_DEFAULT_MARGIN to be increased.

Setting diagwait to 13 appeass to be a popular workaround for a lot of bugs:

crsctl set css diagwait 13

Again this information is unofficial and you should consult Oracle Support before modifying this variable on a production system.

See Metalink Note 294430. MISSCOUNT Definition and Default Values

Oracle Clusterware
Event Management (EVM)

- Event Management daemon publishes events created by Oracle Clusterware
- In Linux implemented as the evmd daemon which is started by the root user
- You can specify callout scripts which will be executed by the Event Manager when a specified event occurs
  - Callouts are managed by the racgevt process

The Event Management daemon publishes events created by Oracle Clusterware. Under Linux, the Event Manager is implemented as the evmd daemon which runs as the root user.

You can specify callout scripts, which will be executed by the Event Manager when a specified event occurs. These callouts are managed by the racgevt process.

In addition to the background processes, Oracle Clusterware also communicates with the Oracle Notification Service (ONS), which is a publish and subscribe service that communicates FAN events to clients.

Oracle Clusterware
Cluster Ready Services (CRS)

- Manages high availability operations within the cluster
- Managed objects are called resources
- Resources include:
  - databases
  - instances
  - database services
  - listeners
  - virtual IP addresses
  - application processes

The CRS component manages high availability operations within the cluster. Objects managed by CRS are known as *resources* and can include databases, instances, services, listeners, virtual IP addresses, and application processes. Configuration information about each resource is stored in the OCR When the status of a resource changes, CRS generates an event.

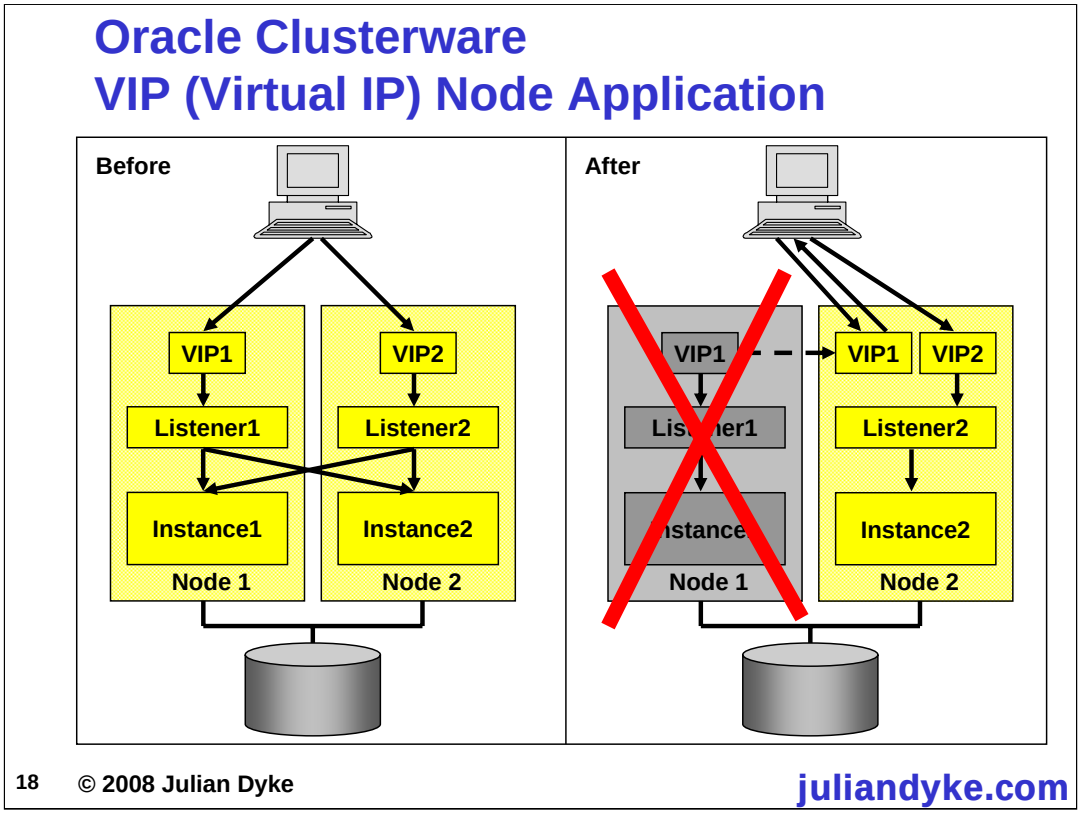
CRS monitors resources, such as instances and listeners. In the event of the failure of a resource, CRS will attempt to automatically restart the component. By default, CRS will attempt to restart the resource five times before giving up.

Under Linux, CRS is implemented as the crsd daemon, which runs as the root user. In the event of a failure, this process restarts automatically

Oracle Clusterware
Cluster Ready Services (CRS)

- By default CRS manages four application process resources:
  - Oracle Net listener
  - Virtual IP addresses
  - Global Services Daemon (GSD)
  - Oracle Notification Service (ONS)
- Also known as node applications

By default, CRS manages four application process resources: Oracle Net listeners, virtual IP addresses, the Global Services Daemon (GSD), and the Oracle Notification Service (ONS).



Oracle VIP addresses were introduced in Oracle 10.1 to solve problems caused by excessively long waits due to TCP/IP timeouts for connecting and reconnecting sessions following a node failure. This slide demonstrates how Oracle Clusterware manages virtual IP addresses.

During normal operation, each node will have a unique VIP address. This is different from the public IP address, but must be on the same subnet.

Applications should connect using the VIP address and not the public IP address. In Oracle 10.1 and early versions of 10.2, DBCA incorrectly configured TNSNAMES.ORA to use the public network address rather than the VIP address. This has been corrected in Oracle 10.2.0.3 and above.

The VIP address maps to the MAC address of the adapter. It is possible to use Oracle VIPs with bonded/teamed NICs.

Oracle Clusterware manages VIPs as a node application. When Oracle Clusterware discovers that a node has failed, it will relocate the VIP to one of the remaining nodes. A promiscuous ARP packet is broadcast to inform clients that the MAC address for the VIP has changed.

The relocated VIP does not forward incoming connections to the listener; instead it immediately sends back a failure message to the client. The client can then immediately failover to an alternate address. In the above slide, node 1 has crashed. Oracle Clusterware relocates VIP1 to node2. When the client attempts to connect using VIP1 it receives an error message; when it attempts to connect using VIP2 the connection is forwarded to the listener.

Oracle Clusterware
VIP (Virtual IP) Node Application

- On Linux during normal operation, each node will have one VIP address. For example:

```
[root@server3]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:11:D8:58:05:99
          inet addr:192.168.2.103  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::211:d8ff:fe58:599/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6814 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10326 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:684579 (668.5 KiB)  TX bytes:1449071 (1.3 MiB)
          Interrupt:217 Base address:0x8800

eth0:1    Link encap:Ethernet  HWaddr 00:11:D8:58:05:99
          inet addr:192.168.2.203  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:217 Base address:0x8800
```

- The resource for VIP address for 192.168.2.203 is initially running on server3

Oracle Clusterware
VIP (Virtual IP) Node Application

- **If Oracle Clusterware on server3 is shutdown, the VIP resource is transferred to another node (in this case server11)**

```
[root@server11]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1D:7D:A3:0A:55
          inet addr:192.168.2.111  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::21d:7dff:fea3:a55/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2792 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4097 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:329891 (322.1 KiB)  TX bytes:593615 (579.7 KiB)
          Interrupt:177 Base address:0x2000

eth0:1    Link encap:Ethernet  HWaddr 00:1D:7D:A3:0A:55
          inet addr:192.168.2.211  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:177 Base address:0x2000

eth0:2    Link encap:Ethernet  HWaddr 00:1D:7D:A3:0A:55
          inet addr:192.168.2.203  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:177 Base address:0x2000
```


After Oracle Clusterware has been shut down on server3, the eth0:1virtual interface will no longer exist.

As shown above, a new device called eth0:2 will be created on a [random] node for the new address.

Oracle Clusterware
VIP Failover

- **VIP addresses can occasionally be failed over incorrectly.**
- **For example:**

```
HA Resource                  Target       State
-----------                  ------       -----
ora.server11.vip             application  ONLINE on server11
ora.server12.vip             application  ONLINE on server12
ora.server3.vip              application  ONLINE on server11
ora.server4.vip              application  ONLINE on server4
```

```
[root@server3]# ./crs_relocate ora.server3.vip -c server3
Attempting to stop `ora.server3.vip` on member `server11`
Stop of `ora.server3.vip` on member `server11` succeeded.
Attempting to start `ora.server3.vip` on member `server3`
Start of `ora.server3.vip` on member `server3` succeeded.
```

```
HA Resource                  Target       State
-----------                  ------       -----
ora.server11.vip             application  ONLINE on server11
ora.server12.vip             application  ONLINE on server12
ora.server3.vip              application  ONLINE on server3
ora.server4.vip              application  ONLINE on server4
```


When Oracle Clusterware is restarted on server3, the VIP address should automatically be failed back. However, this does not always happen (Oracle 11.1.06) and it may be necessary to fail back the VIP manually.

The example shows the output from crs_stat -t. Initially the VIP for server3 has been failed over to server11.

The VIP is manually relocated using the crs_relocate command.

Subsequently crs_stat -t shows the VIP as being located on server3.

Oracle Clusterware Dependencies

- Oracle Clusterware dependencies changed in 10.2.0.3
- If a public network adapter fails
  - VIP resource is relocated to another node
  - Listener resource is taken offline
  - Prior to 10.2.0.3:
    - RAC database instance is taken offline
    - ASM instance is taken offline
  - In 10.2.0.3 and above:
    - RAC database stays up blocked for general operations
    - ASM instance stays up
  - RAC and ASM instances deregister with other listeners in the cluster

Prior to 10.2.0.3 the RAC database instance, the ASM instance (if present) and the listener depend on the node application VIP resource. If the public network adapter fails then all three resources will fail and the node applicaiton VIP will be relocated to another node in the cluster.

In 10.2.0.3 and above, the ASM instance, if present, and RAC database instance are no longer dependent on the node applications VIP Any existing dependencies are removed automatically. Dependencies will still exist

- between the Listener and the VIP
- between the RAC database instance and, if present, the ASM instance.

.

Oracle Clusterware
Oracle Cluster Registry (OCR)

- The Oracle Cluster Registry (OCR) stores the configuration information for Oracle Clusterware / CRS
- Introduced in Oracle 10.1
  - Replaced Server Management (SRVM) disk/file
- Similar to Windows Registry
- Located on shared storage
- In Oracle 10.2 and above the OCR can be mirrored
  - Maximum two copies

The OCR maintains cluster and database configuration information for RAC and Oracle Clusterware resources including information about nodes, databases, instances, services, applications, and listeners.

The OCR is similar in many ways to the Windows registry. Information is stored in a hierarchy of key-value pairs within a directory tree structure. The OCR can be updated using EM, the Server Control Utility (SRVCTL) and DBCA. The OCR is also updated by SQL*Plus and a number of other Oracle Clusterware tools and utilities.

Oracle Clusterware
Oracle Cluster Registry (OCR)

- **In Linux the location of OCR is specified in /etc/oracle/ocr.loc**

```
ocrconfig_loc=/dev/raw/raw1
local_only=FALSE
```

- **In Oracle 10.2 and above, if the OCR is mirrored a second location is included in ocr.loc**

```
ocrconfig_loc=/dev/raw/raw1
ocrmirrorconfig_loc=/dev/raw/raw2
local_only=FALSE
```


The OCR must be located on shared storage and must be accessible from all nodes. On each node, the location of the OCR is specified in the file /etc/oracle/ocr.loc:

```
ocrconfig_loc=/u02/oradata/RAC/OCRFile
local_only=FALSE
```

In Oracle 10.2 and above, the OUI can optionally create a mirrored copy of the OCR File. This file should also be located on shared storage. We recommend creating a mirror even if your storage system already implemented external redundancy, as maintaining two copies at the Oracle Clusterware level reduces the chances of inadvertently deleting or overwriting the OCR file.

The following is an example of the contents of /etc/oracle/ocr.loc in Oracle 10.2 with a mirrored OCR:

```
ocrconfig_loc=/u02/oradata/RAC/OCRFile1
ocrmirrorconfig_loc=/u03/oradata/RAC/OCRFile2
local_only=FALSE
```

Oracle Clusterware
Voting Disk

- Known as Quorum Disk / File in Oracle 9i
- Located on shared storage accessible to all instances
- Used to determine RAC instance membership
- In the event of interconnect failure voting disk is used to determine which instance takes control of cluster
  - Avoids split brain
- In Oracle 10.2 and above can be mirrored
  - Odd number of copies (1, 3, 5 etc)

The voting disk, which must reside on shared storage, manages cluster membership information. It is used by RAC to determine instances that are members of the cluster. It is also used to arbitrate cluster ownership between the remaining instances in the event of a network failure.

In Oracle 10.2 and above, it is possible to create multiple voting disks. Oracle recommends configuring an odd number of voting disks (e.g. three or five). By default in Oracle 10.2, the OUI will create three voting disks, although you can specify a single voting disk if your storage provides mirroring at the hardware level.. The voting disk is critical to the operation of Oracle Clusterware and of the database. Therefore, if you choose to define a single voting disk or you are still using Oracle 10.1, you should use external mirroring to provide redundancy.

Oracle Clusterware Fencing

- Relates to how a cluster handles nodes that should no longer have access to shared resources
- Critical for data integrity and application availability
- If a cluster node has access to shared disk, but not to the interconnect, fencing ensures that node cannot make further changes to shared disk
- Types of fencing include:
  - STONITH - Shoot The Other Node In The Head healthy nodes determine another node should not be part of cluster
  - Self-fencing - nodes fence themselves by executing the reboot command from /etc/init.d/init.cssd

Fencing relates to how a cluster handles nodes that should no longer have access to shared resources. Fencing is critical for data integrity and application availability

If a cluster node has access to shared disk, but not to the interconnect, fencing ensures that node cannot make further changes to shared disk

Types of fencing include:

• STONITH - Shoot The Other Node In The Head - This is the most common type of fencing in which healthy nodes determine another node should not be part of cluster.

• Self-fencing - Oracle Clusterware implements self-fencing. in which nodes fence themselves by executing the reboot command from /etc/init.d/init.cssd. Self fencing can be ineffective if the node is suffering from complete memory depletion.

Oracle Clusterware Heartbeats

- CSS maintains two heartbeats
  - Disk heartbeat to voting device
  - Network heartbeat across interconnect
- Disk heartbeat has an internal I/O timeout (in seconds)
  - Varies between releases
  - In Oracle 10.2.0.2 and above disk heartbeat timeout can be specified by CSS disktimeout parameter
    - Maximum time allowed for a voting file I/O to complete
    - If exceeded file is marked offline
    - Defaults to 200 seconds

```
crsctl get css disktimeout
crsctl set css disktimeout <value>
```


The synchronization services component (CSS) of Oracle Clusterware maintains two heartbeats.

1 - The disk heartbeat to the voting device

2 - The network heartbeat across the interconnect which establishes and confirms node membership in the cluster.

The disktimeout parameter defaults to 200 seconds and specifies the maximum amount of time allowed for a voting file I/O to complete. If this time is exceeded the voting disk will be marked as offline. This is also the amount of time that will be required for initial cluster formation i.e. when no nodes have previously been up and in a cluster.

The disktimeout parameter is available in

10.1.0.4 + CRS II Merge Patch

10.1.0.6

10.2.0.1 + bug 4896338

10.2.0.2 and above

See Metalink Note 294430. MISSCOUNT Definition and Default Values

Oracle Clusterware Heartbeats

- **Network heartbeat timeout can be specified by CSS misscount parameter**
- **Default values (Oracle Clusterware 10.1 and 10.2) are:**

Linux	60 seconds
Unix	30 seconds
Windows	30 seconds

- **Default value for vendor clusterware is 600 seconds**

```
crsctl get css misscount
crsctl set css misscount <value>
```


The CSS miscount parameter represents the maximum time in seconds that a heartbeat can be missed before entering into a cluster reconfiguration to evict the node.

The CSS misscount default value when using vendor Clusterware is 600 seconds. This is to allow the vendor Clusterware sufficient time to resolve split brain scenarios.

The MISSCOUNT parameter affects cluster membership reconfigurations and therefore directly affects the availability of the cluster. In most cases, the default settings for MISSCOUNT should be acceptable. Modifying the default value of MISSCOUNT not only influences the timeout interval for the I/O to the voting disk, but also influences the tolerance for missed network heartbeats across the interconnect.

See Metalink Note 294430. MISSCOUNT Definition and Default Values

Oracle Clusterware Heartbeats

- **Relationship between internal I/O timeout (IOT), MISSCOUNT and DISKTIMEOUT varies between releases**

Version	Description
10.1.0.3	IOT = MISSCOUNT - 15 seconds
10.1.0.4	IOT = MISSCOUNT - 15 seconds
10.1.0.5	IOT = MISSCOUNT - 3 seconds
10.1.0.6	IOT = DISKTIMEOUT during normal operations IOT = MISSCOUNT during initial cluster formation or reconfiguration
10.2.0.1	IOT = MISSCOUNT - 3 seconds
10.2.0.2	IOT = DISKTIMEOUT during normal operations IOT = MISSCOUNT during initial cluster formation or reconfiguration

The table is an oversimplification because the relationships are also affected by patches. For example

10.1.0.4 + bug 3306964

IOT = MISSCOUNT - 3 seconds

10.1.04 + CRS II merge patch

IOT = DISKTIMEOUT during normal operations

IOT = MISSCOUNT during initial cluster formation or reconfiguration

10.2.0.1 + bug 4896338

IOT = DISKTIMEOUT during normal operations

IOT = MISSCOUNT during initial cluster formation or reconfiguration

In Oracle 10.2.0.2 and above the algorithm appears to have stabilised.

See Metalink Note 294430. MISSCOUNT Definition and Default Values

Oracle Clusterware Heartbeats

- **If disktimeout supported CSS will not evict a node from the cluster when I/O to voting disk takes more than MISSCOUNT seconds unless during**
  - **during initial cluster formation**
  - **slightly before reconfiguration**
- **Nodes will not be evicted as long as voting disk operations are completed within DISKTIMEOUT seconds**

Network Heartbeat	Disk Heartbeat	Reboot
Completes within MISSCOUNT seconds	Completes within MISSCOUNT seconds	No
Completes within MISSCOUNT seconds	Takes more than MISSCOUNT seconds but less than DISKTIMEOUT seconds	No
Completes within MISSCOUNT seconds	Takes more than DISKTIMEOUT seconds	Yes
Takes more than MISSCOUNT seconds	Completes within MISSCOUNT seconds	Yes

One of the main reasons for these changes is to reduce the number of evictions caused by slow I/O. Storage components identified as causing problems in previous releases include

• HBA cards with Link Down timeouts > MISSCOUNT

• Bad cables in storage fabric increasing IO latencies

• SAN switch failover latency > MISSCOUNT

• SAN trespassing SP to backup SP taking more than MISSCOUNT

• Multipathing error detection and redirection time > MISSCOUNT

• NAS cluster failover latency > MISSCOUNT

• Sustained CPU load affecting CSSD disk ping monitoring thread

Most common problems are caused by multi-pathing drivers and their failover times.

Oracle does not recommend changing MISSCOUNT in older releases, but upgrading to a more recent version of Oracle Clusterware.

See Metalink Note 294430. MISSCOUNT Definition and Default Values

Oracle Clusterware Libraries

- Oracle Clusterware links with two clusterware libraries:
  - libskgxn.so
    - maintains node membership services
    - tells Oracle which nodes are currently in cluster
  - libskgxp.so
    - contains routines used by Oracle for communication between clusters
    - default version supplied by Oracle uses UDP over ethernet
    - other vendors have provided versions for Infiniband, Hyperfabric and SCI

Oracle RAC Clusterware links with two clusterware libraries.

libskgxn.so : contains the routines used by the Oracle server to maintain node membership services. These routines let Oracle know what nodes are in the cluster. When integrated with host clusterware, the Oracle server makes calls to the vendor supplied skgxn library which in turn provides information about which nodes are active in the cluster.

libskgxp.so: contains the Oracle server routines used for communication between instances (e.g., Cache Fusion CR sends, lock converts, etc). Most production RAC implementations use the default libskgxp supplied by Oracle which implements internode communication via UDP over ethernet. However, this clusterware component has been also implemented high-speed, low-latency interconnects such as InfiniBand, HyperFabric and SCI.

The same clusterware libraries were also used by Oracle Parallel Server (OPS).

Oracle Clusterware Libraries

- Oracle executable must be relinked to include RAC libraries
- Relinked automatically during RAC installation
- Can be relinked manually using

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk <option>
make -f ins_rdbms.mk ioracle
```

- where <option> is
  - rac_on - include RAC
  - rac_off - exclude RAC
- Useful for evaluating overhead of running RAC on single-instance

On Linux

- RAC_ON
  - copies libskgxpu.so to libskgxp10.so (ipc_udp)
  - copies libskgxns.so to libskgxn2.so (nm_auto)
  - adds kcsm.o to libknlopt.a (rac_on)
- RAC_OFF
  - copies libskgxpd.so to libskgxp10.so (ipc_none)
  - copies libskgxns.so to libskgxn2.so (nm_none)
  - adds ksnkcs.o to libknlopt.a (rac_off)

Oracle Clusterware Logging

- In Oracle 10.2, Oracle Clusterware log files are created in the $CRS_HOME/log directory
  - can be located on shared storage
- $CRS_HOME/log directory contains
  - subdirectory for each node e.g. $CRS_HOME/log/server6
- $CRS_HOME/log/<node> directory contains:
  - Oracle Clusterware alert log e.g. alertserver6.log
  - client - logfiles for OCR applications including CLSCFG, CSS, OCRCHECK, OCRCONFIG, OCRDUMP and OIFCFG
  - crsd - logfiles for CRS daemon including crsd.log
  - cssd - logfiles for CSS daemon including ocssd.log
  - evmd - logfiles for EVM daemon including evmd.log
  - racg - logfiles for node applications including VIP and ONS

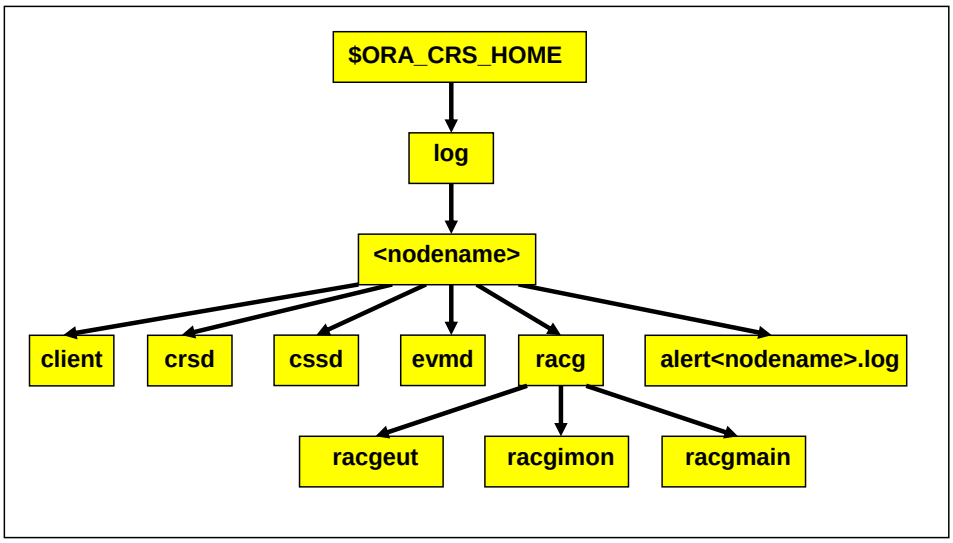
The log files in these directories grow quite rapidly, so you may wish to back them up and truncate them periodically. In addition, check for core dumps and delete them when they are no longer required

In Oracle 10.1, the logfiles can be found in different locations. There is no equivalent to the client directory; client applications such as *ocrconfig* and *oifcfg* generally create logfiles in the current working directory. The remaining daemons create logfiles in the following directories

crsd	$ORA_CRS_HOME/crs/log
cssd	$ORA_CRS_HOME/css/log
evmd	$ORA_CRS_HOME/evm/log
racg	$ORA_CRS_HOME/racg/log $ORA_CRS_HOME/racg/dump

Oracle Clusterware Log Files

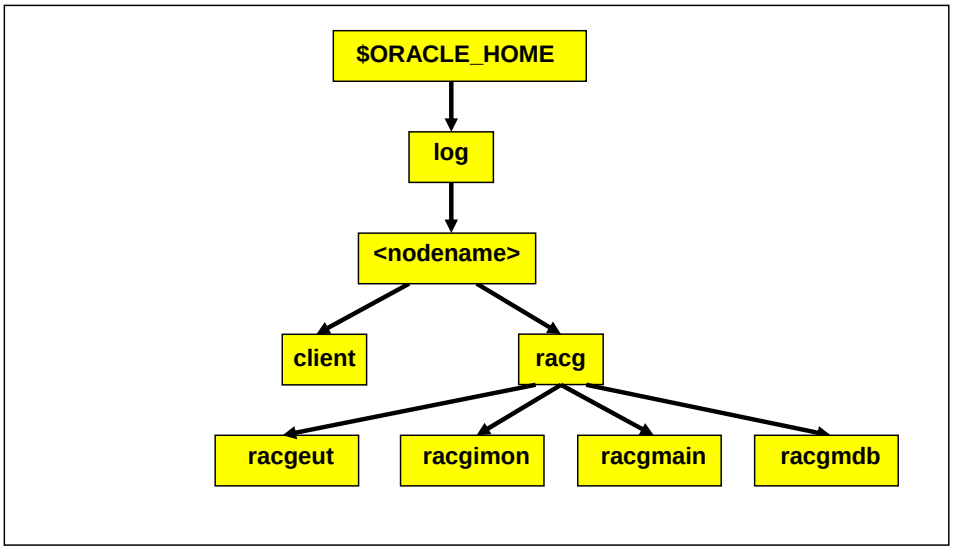
- Log File locations in $ORA_CRS_HOME



Where configured OPROCD logs are written to /etc/oracle/oprocd or /var/opt/oracle/oprocd

Oracle Clusterware Log Files

- Log File locations in $ORACLE_HOME (RDBMS and ASM)



Oracle Clusterware racgwrap

- Script called on each node by SRVCTL to control resources
  - Copy of script in each Oracle home
    - $ORA_CRS_HOME/bin/racgwrap
    - $ORA_ASM_HOME/bin/racgwrap
    - $ORACLE_HOME/bin/racgwrap
  - Sets environment variables
  - Invokes racgmain executable
- Generated from racgwrap.sbs
  - Differs in each home
- Sets $ORACLE_HOME and $ORACLE_BASE environment variables for racgmain
- Also sets $LD_LIBRARY_PATH
- Enable trace by setting _USR_ORA_DEBUG to 1

racgwrap is a shell script. It is invoked on each node by SRVCTL when starting and stopping databases, instances, services an d applications.

racgwrap sets values for the ORACLE_HOME and ORACLE_BASE environment variables.

The setting of the library path environment variable is operating system specific

These values are passed to racgmain which invokes sqlplus to attach to the local instance.

Oracle Clusterware
racgwrap

- In Unix systems the Oracle SGA is located in one or more operating system shared memory segments
- Each segment is identified by a shared memory key
  - Shared memory key is generated by the application
- Each shared memory key maps to a shared memory ID
  - Shared memory ID is generated by operating system
- Shared memory segments can be displayed using ipcs -m

```
[root@server3] #
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x8a48ff44 131072     oracle    640        94371840   20
0x17d04568 163841     oracle    660        2099249152 246
```

- Oracle generates the shared memory key from the values of
  - $ORACLE_HOME
  - $ORACLE_SID

In a single instance environment, the values of $ORACLE_HOME and $ORACLE_SID are passed to sqlplus from the calling shell

In a RAC environment the values of $ORACLE_HOME and $ORACLE_SID are passed to sqlplus from racgwrap.

If you are running sqlplus in a RAC environment, it is essential that your $ORACLE_HOME and $ORACLE_SID environment variables match those used in the racgwrap script. If they do not you will get the following message when attempting to connect

Connected to idle instance

Note that the pathnames are different if has a final / appended to it and the other does not. For example

/u01/app/oracle/product/10.2.0/db_1

and

/u01/app/oracle/product/10.2.0/db_1/

will generate different shared memory keys even with identical values for $ORACLE_SID.

See Metalink 372692.1 Unable to connect to a running instance which was started via SRVCTL

Oracle Clusterware
racgmain

- racgmain is an executable invoked by racgwrap
- It is effectively an Oracle Clusterware action program and normally takes one of three parameters
  - start
  - check
  - stop
- There is one copy of racgmain in each Oracle home
  - $ORA_CRS_HOME/bin/racgmain
  - $ORA_ASM_HOME/bin/racgmain
  - $ORACLE_HOME/bin/racgmain
- Actions performed depend on Oracle home
- racgmain normally calls racgeut to perform actions

The syntax for racgmain is

```
[oracle@server3]$ racgmain -help
Usage racgmain [resource name] start|stop|check
racgmain startorp|failsrvsa dbname instname [srvname]
racgmain startorp|failsrvsa nodename
racgmain cond_resname cond_state func [args...]
```

racgmain normally calls the racgeut executable in the same Oracle home

Oracle Clusterware
racgeut

- racgeut is an executable called by racgmain
- racgmain supplies parameters which racgeut uses to execute a child
- For example:

```
/u01/app/oracle/product/11.1.0/db_1/bin/racgeut
-e _USR_ORA_DEBUG=0 \
-e ORACLE_SID=+ASM2 810 \
/u01/app/oracle/product/11.1.0/db_1/bin/racgimon stop ora.server4.ASM2.asm
```

- Parameters prefixed with -e are environment variables
- Remaining parameters are program and arguments to be forked
- In this example
  - racgimon stop ora.server4.ASM2.asm
- This will stop the ASM2 instance on server4

The syntax for racgeut is:

[oracle@server4 bin]$ racgeut -help

Usage racgeut [-e ...=...] <timeout> <prog_exe> <param_list>

racgeut is used to start the following programs

• racgimon

• racgmdb

• racgvip

• onsctl

and possibly others

Oracle Clusterware
racgimon

- racgimon is invoked by racgeut
- racgimon can operate as
  - a child process of racgeut
  - a daemon
- racgimon can perform start, stop or check operations on
  - instance resources e.g.
    - start ora.server4.ASM2.asm
    - stop ora.server4.ASM2.asm
  - databases e.g.
    - startd TEST
- racgimon invokes child processes such as racgmdb using racgeut

The syntax for racgimon is:

racgimon start|stop|check|daemon [inst_res]

racgimon startd|stopd|checkd [db_name]

Oracle Clusterware
racgmdb

- racgmdb is invoked by racgeut
- racgmdb is an executable that is used to start, stop and monitor
  - database instances
  - ASM instances
- Options include
  - -s mount — startup mount (ASM instance)
  - -s open — startup open (RDBMS instance)
  - -d immediate — shutdown immediate
  - -q — check status
- Alternative start and stop options can be specified
- Start option can be specified by _USR_ORA_OPEN_MODE environment variable
- Stop option can be specified by _USR_ORA_STOP_MODE environment variable

The full syntax for racgmdb is

```
[oracle@server3]$ racgmdb -help
Usage: racgmdb [-s [openmode]|-d [stopmode]|-q]
-s [openmode]  : start up a database instance
                 open mode can be [force] [restrict] open|mount|nomount
                 If openmode is not specified, value of environment variable
                 _USR_ORA_OPEN_MODE will be used
-d [stopmode]  : shut down a database instance
                 stopmode can be [normal|transactional|transactional
local|immediate|abort]
                 If stopmode is not specified, value of environment variable
                 _USR_ORA_STOP_MODE will be used
-q             : Query for current status of database instance
```

Oracle Clusterware racgvip

- Shell script
- Invoked by racgeut
- Controls Oracle Clusterware VIP node application
- Only included in Oracle Clusterware home
  - Not included in Oracle ASM/RDBMS home

```
/u01/app/11.1.0/crs/bin/racgeut -e _USR_ORA_DEBUG=0 54 \
/u01/app/11.1.0/crs/bin/racgvip check server4
```

- Syntax is

```
racgvip start  vip_name
racgvip stop   vip_name
racgvip check  vip_name
racgvip create vip_name IP=vip_address NODE=preferred_node
         [MASK=netmask IF="interfaces"]
racgvip delete vip_name
```


The VIP implementation on Linux only supports ifconfig on IPv4.

The script includes a couple of environment variables:

• FAIL_WHEN_ALL_LINK_DOWN - Set to 1 for VIP failover when all links are down (default action) otherwise set to 0

• FAIL_WHEN_DEFAULTGW_NOT_FOUND - Defaults to 1. Set to 0 to ignore errors if default gateway has not been configured

Oracle Clusterware
onsctl

- Executable
- Invoked by racgeut
- Controls Oracle Notification Service
- Invokes ons executable
- For example check action is performed by onsctl ping

```
/u01/app/11.1.0/crs/bin/racgeut -e _USR_ORA_DEBUG=0 540
/u01/app/11.1.0/crs/bin/onsctl ping
```

- onsctl ping calls ons to perform ping

```
/u01/app/11.1.0/crs/opmn/bin/ons -a ping
```


/u01/app/11.1.0/crs/bin/racgeut -e _USR_ORA_DEBUG=0 540
/u01/app/11.1.0/crs/bin/onsctl ping

/u01/app/11.1.0/crs/opmn/bin/ons -a ping

Oracle Clusterware
Cluster Incarnations

- **Every time a node joins or leaves the cluster, the cluster incarnation is incremented**
- **For example:**

Event	# Nodes	Incarnation
Start Node 1	1	1
Start Node 2	2	2
Start Node 3	3	3
Start Node 4	4	4
Stop Node 2	3	5
Start Node 2	4	6
Stop Node 1	3	7
Stop Node 3	2	8
Stop Node 4	1	9

Changes to the cluster membership are logged in

$ORA_CRS_HOME/log/<node>/cssd/ocssd.log

For example:

```
[    CSSD]CLSS-3000: reconfiguration successful, incarnation 1 with 1 nodes
[    CSSD]CLSS-3001: local node number 4, master node number 4
[    CSSD]CLSS-3000: reconfiguration successful, incarnation 2 with 2 nodes
[    CSSD]CLSS-3001: local node number 4, master node number 4
[    CSSD]CLSS-3000: reconfiguration successful, incarnation 3 with 3 nodes
[    CSSD]CLSS-3001: local node number 4, master node number 4
[    CSSD]CLSS-3000: reconfiguration successful, incarnation 4 with 4 nodes
[    CSSD]CLSS-3001: local node number 4, master node number 4
```

Oracle Clusterware
Master Node

- At any time a cluster has a master node
- The master node is the only node which can update the OCR
- Probably the oldest surviving node in the cluster
- For example:

Event	Nodes	Master
Start Node 1	1	1
Start Node 2	1,2	1
Start Node 3	1,2,3	1
Start Node 4	1,2,3,4	1
Stop Node 2	1,3,4	1
Start Node 2	1,2,3,4	1
Stop Node 1	2,3,4	3
Stop Node 3	2,4	4
Stop Node 4	2	2

Although it is conventional wisdom that the master node is the only node that can update the OCR, recent tests have shown that occasionally other nodes appear to be able to update the OCR, for example when a service is enabled or disabled on a non-master node.

Oracle Clusterware
Clusterware Files

- In Linux the lsof utility reports which files are currently open
- e.g.

```
[root@server9 cssd]#  lsof -p `pgrep ocssd.bin` | grep raw
ocssd.bin 6889 oracle    4r   CHR      162,1             9993 /dev/raw/raw1
ocssd.bin 6889 oracle    9u   CHR      162,2            10042 /dev/raw/raw2
ocssd.bin 6889 oracle   10u   CHR      162,2            10042 /dev/raw/raw2
```

- The following table summarizes access to the Clusterware files by the Clusterware daemons

	OCR	Voting Disk
ocssd.bin	READ	UPDATE
crsd.bin	UPDATE	-
evmd.bin	READ	-
oclsomon.bin	READ	-

Oracle Clusterware Utilities
CRSCTL

- Command-line utility used to manage Oracle Clusterware
- Introduced in Oracle 10.1
  - Limited functionality
- Enhanced in Oracle 10.2
  - Functionality includes
    - start / stop crs
    - enable / disable crs
    - check crs / cssd / crsd / evmd
    - get / set / unset css <parameter>
    - query / add / delete css votedisk
    - start/stop resources
    - query crs softwareversion/activeversion
- Additional options can be listed using:

```
# crsctl help
```


In Oracle 10.1 a limited subset of the functionality shown in the slide is supported.

CRSCTL can also be used to take statedumps and to enable debug trace and log functionality for css, crs, evm and resources. The syntax is as follows:

crsctl debug statedump evm - dumps state info for evm objects
crsctl debug statedump crs - dumps state info for crs objects
crsctl debug statedump css - dumps state info for css objects
crsctl debug log css [module:level]{,module:level} ...
- Turns on debugging for CSS
crsctl debug trace css - dumps CSS in-memory tracing cache
crsctl debug log crs [module:level]{,module:level} ...
- Turns on debugging for CRS
crsctl debug trace crs - dumps CRS in-memory tracing cache
crsctl debug log evm [module:level]{,module:level} ...
- Turns on debugging for EVM
crsctl debug trace evm - dumps EVM in-memory tracing cache
crsctl debug log res <resname:level> turns on debugging for resources

You can also list the modules that can be used for debugging in CRS, CSS and EVM using:

crsctl lsmodules css - lists the CSS modules that can be used for debugging
crsctl lsmodules crs - lists the CRS modules that can be used for debugging
crsctl lsmodules evm - lists the EVM modules that can be used for debugging

Oracle Clusterware Utilities
CRSCTL

- **In Oracle 10.2 and above CRSCTL can be used to start and stop Oracle Clusterware**
  - **To start Oracle Clusterware use:**

```
# crsctl start crs
```

  - **To stop Oracle Clusterware use:**

```
# crsctl stop crs
```


In Oracle 10.2 and above, CRSCTL can be used to start and stop Oracle Clusterware using

$ crsctl start crs

$ crsctl stop crs

In Oracle 10.1, CRS can only be started by the init daemon when the node is booted. Starting CRS via any other mechanism in Oracle 10.1 is not supported.

However, in Oracle 10.1, CRS can be stopped manually using the service command. For example:

```
# service init.crs stop
Shutting down Oracle Cluster Ready Services (CRS):
Shutting down CRS daemon
Shutting down EVM daemon
Shutting down CSS daemon
Shutdown request successfully issued.
```

Oracle Clusterware Utilities
CRSCTL

- CRSCTL can also be used to enable and disable Oracle Clusterware
  - To enable Clusterware use:

```
# crsctl enable crs
```

  - To disable Clusterware use:

```
# crsctl disable crs
```

- These commands update the following file:
  - /etc/oracle/scls_scr/<node>/root/crsstart

In Oracle 10.1 and above CRSCTL can be used to enable and disable CRS using:

$ crsctl enable crs

$ crsctl disable crs

In Linux systems these commands modify the following file:

/etc/oracle/scls_scr/<node_name>/root/crsstart

e.g. for server6

/etc/oracle/scls_scr/server6/root/crsstart

The crsstart file contains either "enable" or "disable" depending on the current state of CRS.

Disabling CRS is a quick way of preventing Oracle Clusterware, ASM instances and any databases from restarting when the node is rebooted. This can be useful during maintenance sessions.

However, take care to enable CRS again on each node. It is easy to forget and difficult to trace when Oracle Clusterware fails to start.

Oracle Clusterware Utilities
CRSCTL

- **In Oracle 10.2, CRSCTL can be used to check the current state of Oracle Clusterware daemons**
- **To check the current state of all Oracle Clusterware daemons**

```
# crsctl check crs
CSS appears healthy
CRS appears healthy
EVM appears healthy
```

- **To check the current state of individual Oracle Clusterware daemons**

```
# crsctl check cssd
CSS appears healthy

# crsctl check crsd
CRS appears healthy

# crsctl check evmd
EVM appears healthy
```


On Unix systems, of course, you can also use the ps command to check the current state of Oracle Clusterware. However, due to the implementation of the daemons and supporting processes, the output can be confusing. For example:

```
ps -ef | grep crs
root   2915 1       /bin/su -l oracle -c sh -c 'ulimit -c unlimited; cd
       /u01/app/oracle/product/10.2.0/crs/log/server3/evmd; exec
       /u01/app/oracle/product/10.2.0/crs/bin/evmd '
root   2917 1       /u01/app/oracle/product/10.2.0/crs/bin/crsd.bin reboot
oracle       3350   2915       /u01/app/oracle/product/10.2.0/crs/bin/evmd.bin
root   3473 3322    /bin/su -l oracle -c /bin/sh -c 'ulimit -c unlimited; cd
       /u01/app/oracle/product/10.2.0/crs/log/server3/cssd;
       /u01/app/oracle/product/10.2.0/crs/bin/ocssd  || exit $?'
oracle       3474   3473       /bin/sh -c ulimit -c unlimited; cd
       /u01/app/oracle/product/10.2.0/crs/log/server3/cssd;
       /u01/app/oracle/product/10.2.0/crs/bin/ocssd  || exit $?
oracle       3510   3474       /u01/app/oracle/product/10.2.0/crs/bin/ocssd.bin
oracle       3894   3893
       /u01/app/oracle/product/10.2.0/crs/bin/evmlogger.bin -o
       /u01/app/oracle/product/10.2.0/crs/evm/log/evmlogger.info -l
       /u01/app/oracle/product/10.2.0/crs/evm/log/evmlogger.log
oracle       5156   1          /u01/app/oracle/product/10.2.0/crs/opmn/bin/ons -d
oracle       5157   5156       /u01/app/oracle/product/10.2.0/crs/opmn/bin/ons -d
```

(Output modified for readability)

The crsctl check commands return more user-friendly output as shown in the slide.

Oracle Clusterware Utilities
CRSCTL

- **CRSCTL can be used to manage the CSS voting disk**
- **To check the current location of the voting disk use:**

```
# crsctl query css votedisk
0.      0       /dev/raw/raw3
1.      0       /dev/raw/raw4
2.      0       /dev/raw/raw5
```

- **To add a new voting disk use:**

```
# crsctl add css votedisk <path_name>
```

- **To delete an existing voting disk use:**

```
# crsctl delete css votedisk <path_name>
```


Note - you should always have an odd number of voting disks e.g. 1 or 3, therefore you should always add new voting disks or delete existing voting disks in pairs.

In Oracle 10.2.0.1 I found that I could not add or delete voting disks while Oracle Clusterware was running. I had to stop Oracle Clusterware before I could add new disks. This problem may have been fixed in more recent patch sets.

Oracle Clusterware Utilities
CRSCTL

- In theory you can run multiple versions of Oracle Clusterware in the same cluster
- If different versions of Oracle Clusterware exist, the oldest will be used by all nodes until all nodes have been upgraded
- To list the version of Oracle Clusterware software installed on the local node use:

```
# crsctl query crs softwareversion
CRS software version on node [node1] is [10.2.0.1.0]
```

- To list the version of Oracle Clusterware software which is currently active on the local node use:

```
# crsctl query crs activeversion
CRS active version on the cluster is [10.2.0.1.0]
```


In the event that different versions of Oracle Clusterware have been installed on different nodes in the cluster, the oldest version will be used by all nodes in the cluster. When the last node has been upgraded then all nodes can start to use the new version.

Note this form of upgrade is not particularly well-known and therefore may not have been heavily tested in the field. If your availability requirements are so high that cannot upgrade Oracle Clusterware on all nodes in the cluster simultaneously, I recommend that you test this feature on a separate cluster first to verify that it will work as expected in your environment.

Oracle Clusterware Utilities
CRSCTL

- **CRSCTL can be used to start and stop all CRS resources configured to run in the cluster**
- **To start CRS resources use:**

```
# crsctl start resources
Starting resources.
Successfully started CRS resources
```

- **To stop CRS resources use:**

```
# crsctl stop resources
Stopping resources.
Successfully stopped CRS resources
```


CRS resources include all:

- node applications
- listeners
- ASM instances
- databases
- instances
- services

The actions performed by this command are automatically logged in:

$ORA_CRS_HOME/log/<node>/crsd/crsd.log

Oracle Clusterware Utilities
OLSNODES

- The olsnodes utility lists all nodes currently running on the cluster
- With no arguments olsnodes lists the nodes e.g.

```
$ olsnodes
london1
london2
```

- In Oracle 10.2 and above, with -p argument olsnodes lists node names and private interconnect

```
$ olsnodes -p
london1 london1-priv
london2 london2-priv
```

- In Oracle 10.2 and above, with -i argument olsnodes lists node names and VIP address

```
$ olsnodes -i
london1 london1-vip
london2 london2-vip
```


The olsnodes utility lists all nodes which are currently running in the cluster.

With no arguments olsnodes lists the node names. For example

```
$ olsnodes
london1
london2
```

With the -n argument, olsnodes prints the node name and the corresponding node number

```
$ olsnodes -n
london1 1
london2 2
```

With the -l argument, olsnodes prints the name of the current node

```
$ olsnodes -l
london1
```

In Oracle 10.2 and above, olsnodes has additional functionality. You can include the private interconnect name for each node using:

```
$ olsnodes -p
london1 london1-priv
london2 london2-priv
```

You can include the virtual IP (VIP) address for each node using:

```
$ olsnodes -i
london1 london1-vip
london2 london2-vip
```

In Oracle 10.2 you can also enable logging by including the -g option and enable verbose output using the -v option.

Oracle Clusterware Utilities
OCRCONFIG

- In Oracle 10.1 and above the OCRCONFIG utility performs various administrative operations on the OCR including:
  - displaying backup history
  - configuring backup location
  - restoring OCR from backup
  - exporting OCR
  - importing OCR
  - upgrading OCR
  - downgrading OCR
- In Oracle 10.2 and above OCRCONFIG can also
  - manage OCR mirrors
  - overwrite OCR files
  - repair OCR files

In Oracle 10.1 OCRCONFIG creates a log file called ocrconfig.log in the directory from which it is executed. In Oracle 10.2 ocrconfig creates a log file called ocrconfig_<pid>.log in the following directory:

```
$ORA_CRS_HOME/log/<node>/client
```

As this utility must be executed by the root user, you may find it convenient to add $ORA_CRS_HOME/bin to the PATH environment variable in the root profile. For example:

```
export ORA_CRS_HOME=/u01/app/oracle/product/10.1.0/crs/bin
export PATH=$PATH:$ORA_CRS_HOME/bin
```

During normal operations, you should never need to restore the OCR. In the event of an OCR failure on a production system, I recommend that you contact Oracle Support for the latest advice and assistance.

You should never need to manually upgrade or downgrade the OCR. These options are invoked by the root upgrade/downgrade scripts supplied with patch sets. However, it is useful to know that these options exist.

Oracle Clusterware Utilities
OCRCONFIG

- **Options include**

Option	Description	Version
-help	Display help message	10.1+
-showbackup	Display automatic OCR physical backup history	10.1+
-backuploc	Change OCR physical backup location	10.1+
-restore	Restore OCR from automatic physical backup	10.1+
-export	Export contents of OCR to operating system file	10.1+
-import	Import contents of OCR from operating system file	10.1+
-upgrade	Upgrade OCR from a previous version	10.1+
-downgrade	Downgrade OCR to a previous version	10.1+
-replace	Add/replace/remove OCR file or mirror	10.2+
-overwrite	Overwrite OCR configuration on disk	10.2+
-repair	Repair local OCR configuration	10.2+

The above slide shows the valid options for the OCRCONFIG utility in Oracle 10.1 and Oracle 10.2.

Oracle Clusterware Utilities
OCRCONFIG

- In Oracle 10.1 and above
  - OCR is automatically backed up every four hours
  - Previous three backup copies are retained
  - Backup copy retained from end of previous day
  - Backup copy retained from end of previous week
- Check node, times and location of previous backups using the showbackup option of OCRCONFIG e.g.

```
# ocrconfig -showbackup
london1  2005/08/04 11:15:29  /u01/app/oracle/product/10.2.0/crs/cdata/crs
london1  2005/08/03 22:24:32  /u01/app/oracle/product/10.2.0/crs/cdata/crs
london1  2005/08/03 18:24:32  /u01/app/oracle/product/10.2.0/crs/cdata/crs
london1  2005/08/02 18:24:32  /u01/app/oracle/product/10.2.0/crs/cdata/crs
london1  2005/07/31 18:24:32  /u01/app/oracle/product/10.2.0/crs/cdata/crs
```

- ENSURE THAT YOU COPY THE PHYSICAL BACKUPS TO TAPE AND/OR REDUNDANT STORAGE

57 juliandyke.com

In Oracle 10.1 and above, the OCR is backed up automatically by one instance every four hours. The previous three backup copies are retained. A backup is also retained for each full day and at the end of each week. The frequency of backups and the number of files retained cannot be modified.

If you have recently moved you OCR backup directory, this will be reflected in the path names.

```
# ls -l $ORA_CRS_HOME/crs/cdata/crs
-rw-r------  1 root    root    6901760 Aug  4 11:15 backup00.ocr
-rw-r------  1 root    root    6901760 Aug  3 22:24 backup01.ocr
-rw-r------  1 root    root    6901760 Aug  3 18:24 backup02.ocr
-rw-r------  1 root    root    6897664 Aug  2 14:24 day.ocr
-rw-r------  1 root    root    6807552 Jul 31 18:31 week.ocr
```

The backup files are renamed every time a backup is taken so that the most recent backup file is backup00.ocr. In addition a daily copy (day.ocr) and a weekly copy (week.ocr) are also maintained.

By default backup files are created in the $ORA_CRS_HOME/crs/cdata/crs directory. You can modify this location using

```
# ocrconfig -backuploc <directory_name>
```

The directory should preferably be on shared storage so that all nodes in the cluster can access it.

It is recommended that you manually copy the OCR backup files to another location so that you have at least two copies of each file.

Oracle Clusterware Utilities
OCRCONFIG

- To restore the OCR from a physical backup copy
  - Check you have a suitable backup using:

```
# ocrconfig -showbackup
```

  - Stop Oracle Clusterware on each node using:

```
# crsctl stop crs
```

  - Restore the backup file using

```
# ocrconfig -restore <filename>
```

  - For example:

```
# ocrconfig -restore $ORA_CRS_HOME/cdata/crs/backup00.ocr
```

  - Start Oracle Clusterware on each node using:

```
# crsctl start crs
```


To restore the Oracle Cluster Registry from a backup copy first check that you have a suitable backup using:

```
# ocrconfig -showbackup
```

Stop Oracle Clusterware on each node. For example:

```
# crsctl stop crs
Shutting down Oracle Cluster Ready Services (CRS):
Shutting down CRS daemon
Shutting down EVM daemon
Shutting down CSS daemon
Shutdown request successfully issued.
```

In Oracle 10.1 the CRSCTL utility cannot be used to start and stop CRS. In this version use:

```
# service init.crs stop
```

Restore the backup file using:

```
# ocrconfig -restore <file_name>
```

For example:

```
# ocrconfig -restore $ORA_CRS_HOME/cdata/crs/backup00.ocr
```

Restart the Oracle Clusterware using

```
# crsctl start crs
```

In Oracle 10.1 the CRSCTL utility cannot be used to start and stop CRS. In this version you will need to restart CRS by rebooting each node in the cluster.

Oracle Clusterware Utilities
OCRCONFIG

- To relocate the OCR
  - Check you have a suitable backup using:

```
# ocrconfig -showbackup
```

  - Stop Oracle Clusterware on each node using:

```
# crsctl stop crs
```

  - On each node edit /etc/oracle/ocr.loc and update the ocrconfig_loc parameter e.g.:

```
ocrconfig_loc=/dev/raw/raw3
local_only=FALSE
```

  - Move the OCR file to its new location:

```
dd if=/dev/raw/raw1 of=/dev/raw/raw3 bs=1M
```


It may be necessary to move the Oracle Cluster Registry. For example you may wish to reconfigure your shared storage to accommodate other applications or users.

The example in the slides shows an OCR being relocated from one raw device to another. The example in these notes shows an OCR being relocated from one shared filesystem (OCFS) to another.

To move the Oracle Cluster Registry, you must restore the registry from a backup copy. Before commencing this procedure check that you have a suitable backup using:

```
# ocrconfig -showbackup
```

Stop Oracle Clusterware on each node. For example:

```
# crsctl stop crs
```

On each node edit the /etc/oracle/ocr.loc file and update the ocrconfig_loc parameter. For example:

```
ocrconfig_loc=/u05/oradata/RAC/OCRFile
local_only=FALSE
```

Move the OCR file to its new location. For example:

```
# mv /u02/oradata/RAC/OCRFile /u05/oradata/RAC/OCRFile
```

Oracle Clusterware Utilities
OCRCONFIG

- To relocate the OCR (continued)
  - Restore the backup file using

```
# ocrconfig -restore <filename>
```

  - For example:

```
# ocrconfig -restore $ORA_CRS_HOME/cdata/crs/backup00.ocr
```

  - Check ocrconfig.log for errors. For example:

```
08/14/2005 22:34:59
ocrconfig starts...
Warning! cluster checking is disabled
Successfully restored OCR and set block 0
```

  - Restart Oracle Clusterware using:

```
# crsctl start crs
```


Restore the backup file using:

ocrconfig -restore <file_name>

For example:

ocrconfig -restore $ORA_CRS_HOME/cdata/crs/backup00.ocr

Check ocrconfig.log for errors.

08/14/2005 22:34:59

ocrconfig starts...

Warning!! cluster checking is disabled

Successfully restored OCR and set block 0

Note that the ocrconfig.log file will be created in the directory from which you executed OCRCONFIG. Any error messages will be written to this file.

On each node run the OCRCHECK utility to verify that the OCR location has been set correctly. Finally, restart the Oracle Clusterware by rebooting each node in the cluster.

Oracle Clusterware Utilities
OCRCONFIG

- Adding and Deleting an OCR Mirror
  - In Oracle 10.2 and above, the OCR can be mirrored by Oracle Clusterware
    - Recommended if you do not have hardware mirroring
  - Add an OCR mirror file using:

```
# ocrconfig -replace ocrmirror <filename>
```

  - Delete an OCR mirror file using:

```
# ocrconfig -replace ocrmirror
```


In Oracle 10.2 and above, I recommend that you allow Oracle to mirror the OCR file if you do not have hardware mirroring.

If you have migrated from a previous release or chose not to create an OCR mirror during installation, you can subsequently create an OCR file or mirror using:

```
# ocrconfig -replace ocr <filename>
# ocrconfig -replace ocrmirror <filename>
```

When we tested this option in Oracle 10.2.0.1 we found that OCRCONFIG incorrectly calculated the size of the target directory or device and would not allow OCR file creation to continue. We believe this is a bug and suggest that you test this procedure on a development system before attempting it in a production environment.

You can delete the OCR file or the OCR mirror using the following commands:

```
# ocrconfig -replace
# ocrconfig -replace ocrmirror
```

Oracle Clusterware Utilities
OCRCONFIG

- Repairing the OCR
  - If OCRCONFIG is used to add, replace or remove an OCR file or mirror, changes will be applied on all available nodes
  - If some nodes were unavailable changes will not have been propagated
  - Use -repair option of OCRCONFIG to manually fix remaining nodes

```
# ocrconfig -repair ocr <filename>
# ocrconfig -repair ocrmirror <filename>
```

  - If you have deleted one of the files, you can update the other nodes using:

```
# ocrconfig -repair ocr
# ocrconfig -repair ocrmirror
```


If you use the replace option of OCRCONFIG to add, replace or remove an OCR file or mirror, the changes will be applied on all available nodes. However, if some nodes are currently down, the changes will not be propagated. In this case you must manually apply the changes using the repair option:

```
# ocrconfig -repair ocr <filename>
# ocrconfig -repair ocrmirror <filename>
```

If you have deleted one of the files, you can update other nodes using:

```
# ocrconfig -repair ocr
# ocrconfig -repair ocrmirror
```

Oracle Clusterware Utilities
OCRCHECK

- **In Oracle 10.1 and above, you can verify the configuration of the OCR using the OCRCHECK utility**

```
# ocrcheck
Status of Oracle Cluster Registry is as follows :
        Version                  :          2
        Total space (kbytes)     :     262144
        Used space (kbytes)      :       7752
        Available space (kbytes) :     254392
        ID                       : 1093363319
        Device/File Name         : /dev/raw/raw1
                                   Device/File integrity check succeeded
                                   /dev/raw/raw2
                                   Device/File integrity check succeeded
        Cluster registry integrity check succeeded
```

- **In Oracle 10.1 this utility does not print the ID and Device/File Name information**

In Oracle 10.1 and above, you can verify the configuration of the Oracle Cluster Registry (OCR) using the ocrcheck utility.

The ocrcheck utility is located in $ORA_CRS_HOME/bin. It creates a log file called ocrcheck.log in the directory from which it is executed.

In Oracle 10.1, this utility does not print the ID or Device/File Name information

The OCRCHECK utility also creates a log file called ocrcheck.log. In Oracle 10.1, this file will be created in the directory in which you run OCRCHECK. In Oracle 10.2 logging information is written to the file ocrcheck_<pid>.log in the $ORA_CRS_HOME/log/<node>/client directory.

The contents of the log file should closely resemble the output of the utility. For example:

```
<timestamp> [OCRCHECK][3076427904]ocrcheck starts...
<timestamp> [OCRCHECK][3076427904]protchcheck: OCR status : total
=    [262144], used = [7752], avail = [254392]
<timestamp> [OCRCHECK][3076427904]Exiting [status=success]...
```

Any error messages will also be written to this file.

Oracle Clusterware Utilities
OCRDUMP

- In Oracle 10.1 and above, you can dump the contents of the OCR using the OCRDUMP utility
- For example:

```
# ocrdump
```

- This command writes its output to a file called OCRDUMPFILE in the current working directory
- You can specify an output file name using:

```
# ocrdump <dump_file_name>
```

- For example:

```
# ocrdump ocr_cluster1
```

64 © 2008 Julian Dyke juliandyke.com

In Oracle 10.1 and above you can dump the contents of the Oracle Cluster Registry (OCR) using the OCRDUMP utility which is located in $ORA_CRS_HOME/bin.

```
# ocrdump <dump_file_name>
```

The contents of the OCR will be written in an ASCII format to the specified file. If the command is executed with no parameters then it will create a file called OCRDUMPFILE in the current working directory.

```
# ocrdump
```

Note that you cannot import or restore the contents of the ASCII dump file back into the OCR. You can optionally specify a name for the output file. For example

```
# ocrdump ocr_cluster1
```

In Oracle 10.1 OCRDUMP creates a logfile called ocrdump.log in the current working directory; in Oracle 10.2 and above the logfile is called ocrdump_<pid>.log and is created in $ORA_CRS_HOME/log/<node>/client

Oracle Clusterware Utilities
OCRDUMP

- **In Oracle 10.2 and above, you can write OCRDUMP output to stdout**
- **For example:**

```
# ocrdump -stdout
```

- **In Oracle 10.2 and above, you can optionally restrict output by specifying a key**
- **For example:**

```
# ocrdump -stdout SYSTEM
# ocrdump -stdout SYSTEM.css
# ocrdump -stdout SYSTEM.css.misscount
```

- **In Oracle 10.2 and above, you can optionally format output in XML. For example:**

```
# ocrdump -stdout SYSTEM.css.misscount -xml
```


There are a number of enhancements to this utility in Oracle 10.2 and above. You can write output to stdout. For example

```
# ocrdump -stdout
```

In Oracle 10.2 and above you can optionally restrict the amount of output by specifying a key. For example

```
# ocrdump -stdout SYSTEM
# ocrdump -stdout SYSTEM.css
# ocrdump -stdout SYSTEM.css.misscount
```

Finally in Oracle 10.2 and above you can optionally format output in XML. For example

```
# ocrdump -stdout SYSTEM.css.misscount -xml
```

```
<OCRDUMP>
<COMMAND>$ORA_CRS_HOME/bin/ocrdump.bin -stdout -keyname
SYSTEM.css.misscount -xml </COMMAND>
```

```
<KEY>
<NAME>SYSTEM.css.misscount</NAME>
<VALUE_TYPE>UB4 (10)</VALUE_TYPE>
<VALUE><![CDATA[60]]></VALUE>
<USER_PERMISSION>PROCR_ALL_ACCESS</USER_PERMISSION>
<GROUP_PERMISSION>PROCR_READ</GROUP_PERMISSION>
<OTHER_PERMISSION>PROCR_READ</OTHER_PERMISSION>
<USER_NAME>root</USER_NAME>
<GROUP_NAME>root</GROUP_NAME>
</KEY>
</OCRDUMP>
```

Oracle Clusterware Utilities
OIFCFG

- Oracle Interface Configuration (OIFCFG) Utility
  - allows you to define and manage interfaces for Oracle database components
- Network interfaces identified by
  - interface name
  - subnet
  - interface type
- Supported interface types include
  - public
  - cluster_interconnect
- Network interface specification has the format:

```
<interface_name>/<subnet>:<interface_type>
```

- e.g.:

```
eth0/192.168.1.0:cluster_interconnect
```


The Oracle Interface Configuration (OIFCFG) utility allows you to define and manage network interfaces for Oracle database components. Although it can be used with single-instance Oracle databases it is primarily used in a RAC environment.

OIFCFG allows you to list and modify the current configuration of each network interface. A network interface is identified using the interface name, subnet mask and interface type. Supported interface types are:

Public - used to communicate externally using components such as Oracle Net and Virtual Internet Protocol (VIP) addresses

Private - used for inter-instance communication across the cluster interconnect.

A network interface specification has the following format:

<interface_name>/<subnet>:<interface_type>

For example

eth0/192.168.1.1:cluster_interconnect

Oracle Clusterware Utilities
OIFCFG

- Network interfaces can be global or node specific
- To display a list of current subnets use:

```
$ oifcfg iflist
eth0  147.43.0.0
eth1  192.168.2.0
```

- To include a description of the subnet specify the -p option:

```
$ oifcfg iflist -p
eth0  147.43.0.0      UNKNOWN
eth1  192.168.2.0     PRIVATE
```

- To include a the subnet mask append the -n option

```
$ oifcfg iflist -p -n
eth0  147.43.0.0      UNKNOWN      255.255.0.0
eth1  192.168.2.0     PRIVATE        255.255.255.0
```


Network interfaces can be stored as global interfaces or node-specific interfaces. An interface is stored as a global interface when all nodes in the cluster have the same interface connected to the same subnet. I recommend you use a symmetrical configuration whenever possible. An interface is stored as a node-specific interface when other nodes in the cluster have a different set of interfaces and subnets. This may be necessary if you have an asymmetric workload or if you have run out of slots in one or more servers. Node-specific interface definitions override global interface definitions.

You can display the current command syntax for OIFCFG using

```
$ oifcfg -help
```

To display a list of current subnets use:

```
$ oifcfg iflist
eth0 147.43.0.0
eth1 192.168.2.0
```

To include a description of the subnet specify the -p option

```
$ oifcfg iflist -p
eth0 147.43.0.0       UNKNOWN
eth1 192.168.2.0      PRIVATE
```

In the version of Oracle 10.2 that we tested all public interfaces had the description "UNKNOWN". To include the subnet mask append the -n option (to the -p option)

```
$ oifcfg iflist -p -n
eth0    147.43.0.0    UNKNOWN        255.255.0.0
eth1    192.168.2.0   PRIVATE        255.255.255.0
```

Oracle Clusterware Utilities
OIFCFG

- **To display a list of global networks use:**

```
$ oifcfg getif
eth0        147.43.0.0      global      public
eth1        192.168.2.0     global      cluster_interconnect
```

- **To display a list of node-specific configurations use the -n option:**

```
$ oifcfg getif -node london1
```

- **To display a details for a specific interface use the -if option:**

```
$ oifcfg getif -if eth0
eth0        147.43.0.0      global      public
```

- **To display a details for a specific interface type use the -type option:**

```
$ oifcfg getif -type cluster_interconnect
eth0        192.168.2.0     global      cluster_interconnect
```


To display a list of networks use:

```
$ oifcfg getif
eth0    147.43.0.0    global    public
eth1    192.168.2.0   global    cluster_interconnect
```

The above command is equivalent to specifying the -global option.

You can also display node-specific configurations by specifying the -node option. For example:

```
$ oifcfg getif -node london1
```

You can display details for a specific interface. For example:

```
$ oifcfg getif -if eth0
eth0    147.43.0.0    global    public
```

Alternatively you can display details for a specifc interface type. For example

```
$ oifcfg getif -type cluster_interconnect
eth1    192.168.2.0    global    cluster_interconnect
```

Oracle Clusterware Utilities
OIFCFG

- You can create a new interface using OIFCFG.
- In this example an interface is enabled on a new set of network cards

```
$ oifcfg setif -global eth2/147.44.0.0:public
```

```
$ oifcfg getif
eth0    147.43.0.0      global    public
eth1    192.168.2.0     global    cluster_interconnect
eth2    147.44.0.0      global    public
```

- To delete the interface again use:

```
$ oifcfg delif -global eth2
```

```
$ oifcfg getif
eth0    147.43.0.0      global    public
eth1    192.168.2.0     global    cluster_interconnect
```


You can create a new interface type. In the following example, we enabled the interface on a new set of network cards.

```
$ oifcfg setif -global eth2/147.44.0.0:public
$ oifcfg getif
eth0    147.43.0.0      global    public
eth1    192.168.2.0     global    cluster_interconnect
eth2    147.44.0.0      global    public
```

Note that you must specify a subnet (and not a subnet mask).

To delete the interface again use:

```
$ oifcfg delif -global eth2
$ oifcfg getif
eth0    147.43.0.0      global    public
eth1    192.168.2.0     global    cluster_interconnect
```

Take care not to omit the interface name from the delif option. You can delete all global interfaces using the command:

```
$ oifcfg delif -global
```

Oracle Clusterware Utilities
CRS_STAT

- The CRS_STAT utility reports the current status of resources managed by Oracle Clusterware
- Resources include:
  - databases
  - instances
  - services
  - ASM instances
  - node applications
    - gsd
    - ons
    - vip
  - listeners

The CRS_STAT utility reports on the current status of various resources managed by Oracle Clusterware. It is located in the $ORA_CRS_HOME/bin directory.

Oracle Clusterware Utilities
CRS_STAT

- With no arguments CRS_STAT lists all resources currently configured e.g.:

```
$ crs_stat
NAME=ora.RAC.RAC1.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london1

NAME=ora.RAC.RAC2.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london2

NAME=ora.RAC.SERVICE1.RAC1.srv
TYPE=application
TARGET=OFFLINE
STATE=OFFLINE

etc...
```

- If a node has failed, the STATE field will show which node the applications have failed over to

With no arguments CRS_STAT lists all resources currently configured. For example:

```
$ crs_stat
NAME=ora.RAC.RAC1.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london1
NAME=ora.RAC.RAC2.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london2
NAME=ora.RAC.SERVICE1.RAC1.srv
TYPE=application
TARGET=OFFLINE
STATE=OFFLINE
NAME=ora.RAC.SERVICE1.RAC2.srv
TYPE=application
TARGET=OFFLINE
STATE=OFFLINE
NAME=ora.RAC.SERVICE1.cs
TYPE=application
TARGET=OFFLINE
STATE=OFFLINE
NAME=ora.RAC.db
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london1
NAME=ora.london1.ASM1.asm
TYPE=application
TARGET=ONLINE
STATE=ONLINE on london1
```

Oracle Clusterware Utilities
CRS_STAT

- With the -t option, crs_stat lists resources together with their state and the current node

```
Name           Type           Target    State     Host
------------------------------------------------------------
ora....T1.inst application    ONLINE    ONLINE    server3
ora....T2.inst application    ONLINE    ONLINE    server4
ora....T3.inst application    ONLINE    ONLINE    server11
ora....T4.inst application    ONLINE    ONLINE    server12
ora.TEST.db    application    ONLINE    ONLINE    server3
ora....SM3.asm application    ONLINE    ONLINE    server11
ora....11.lsnr application    ONLINE    ONLINE    server11
ora....r11.gsd application    ONLINE    ONLINE    server11
ora....r11.ons application    ONLINE    ONLINE    server11
ora....r11.vip application    ONLINE    ONLINE    server11
ora....SM4.asm application    ONLINE    ONLINE    server12
ora....12.lsnr application    ONLINE    ONLINE    server12
ora....r12.gsd application    ONLINE    ONLINE    server12
ora....r12.ons application    ONLINE    ONLINE    server12
ora....r12.vip application    ONLINE    ONLINE    server12
```


Unfortunately CRS_STAT abbreviates the resource name in its output. You need to be careful about naming conventions if you intend to use this utility.

Oracle Clusterware Utilities
CRS_STAT

- **With the -ls option, crs_stat lists resources together with their owner, group and permissions.**

```
Name           Owner          Primary PrivGrp          Permission
------------------------------------------------------------------
ora....T1.inst oracle         oinstall                 rwxrwxr--
ora....T2.inst oracle         oinstall                 rwxrwxr--
ora....T3.inst oracle         oinstall                 rwxrwxr--
ora....T4.inst oracle         oinstall                 rwxrwxr--
ora.TEST.db    oracle         oinstall                 rwxrwxr--
ora....SM3.asm oracle         oinstall                 rwxrwxr--
ora....11.lsnr oracle         oinstall                 rwxrwxr--
ora....r11.gsd oracle         oinstall                 rwxr-xr--
ora....r11.ons oracle         oinstall                 rwxr-xr--
ora....r11.vip root           oinstall                 rwxr-xr--
ora....SM4.asm oracle         oinstall                 rwxrwxr--
ora....12.lsnr oracle         oinstall                 rwxrwxr--
ora....r12.gsd oracle         oinstall                 rwxr-xr--
ora....r12.ons oracle         oinstall                 rwxr-xr--
ora....r12.vip root           oinstall                 rwxr-xr--
```


Unfortunately CRS_STAT abbreviates the resource name in its output. You need to be careful about naming conventions if you intend to use this utility.

Oracle Clusterware Utilities
CRS_STAT

- CRS_STAT abbreviates resource names
- Oracle provides an AWK script that includes complete resource names
  - Metalink Note: 259301_1 CRS and 10g RAC

```
#!/bin/bash

RSC_KEY=$1
QSTAT=-u
AWK=/usr/bin/awk

$AWK \
 'BEGIN {printf "%-45s %-10s %-18s\n","HA Resource",  "Target", "State";
       printf "%-45s %-10s %-18s\n","-----------",  "------", "-----";}'
$ORA_CRS_HOME/bin/crs_stat $QSTAT | $AWK \
 'BEGIN { FS="="; state = 0; }
 $1~/NAME/ && $2~/'$RSC_KEY'/ {appname = $2; state=1};
 state == 0 {next;}
 $1~/TARGET/ && state == 1 {apptarget = $2; state=2;}
 $1~/STATE/ && state == 2 {appstate = $2; state=3;}
 state == 3 {printf "%-45s %-10s %-18s\n", appname,apptarget,appstate;state = 0;}'
```


The above AWK script produces output similar to the following:

```
HA Resource                                   Target     State
-----------                                   ------     -----
ora.TEST.TEST1.inst                           ONLINE ONLINE on server3
ora.TEST.TEST2.inst                           ONLINE ONLINE on server4
ora.TEST.TEST3.inst                           ONLINE ONLINE on server11
ora.TEST.TEST4.inst                           ONLINE ONLINE on server12
ora.TEST.db                                   ONLINE ONLINE on server3
ora.server11.ASM3.asm                         ONLINE ONLINE on server11
ora.server11.LISTENER_SERVER11.lsnr ONLINE ONLINE on server11
ora.server11.gsd                              ONLINE ONLINE on server11
ora.server11.ons                              ONLINE ONLINE on server11
ora.server11.vip                              ONLINE ONLINE on server11
ora.server12.ASM4.asm                         ONLINE ONLINE on server12
ora.server12.LISTENER_SERVER12.lsnr ONLINE ONLINE on server12
ora.server12.gsd                              ONLINE ONLINE on server12
ora.server12.ons                              ONLINE ONLINE on server12
ora.server12.vip                              ONLINE ONLINE on server12
ora.server3.ASM1.asm                          ONLINE ONLINE on server3
ora.server3.LISTENER_SERVER3.lsnr    ONLINE ONLINE on server3
ora.server3.gsd                               ONLINE ONLINE on server3
ora.server3.ons                               ONLINE ONLINE on server3
ora.server3.vip                               ONLINE ONLINE on server3
```

Oracle Clusterware Utilities
CRS_STAT

```
#!/usr/bin/perl
$s = ".";
if ($#ARGV >= 0) { $s = $ARGV[0]; chomp $s;}
printf ("%-45s %-12s %-18s\n","HA Resource","Target","State");
printf ("%-45s %-12s %-18s\n","-----------","------","-----");
open (CRS_STAT,"crs_stat -u|");
while ($line = <CRS_STAT>)
{
    if ($line =~ m/=/)
    {
        chomp $line;
        ($n,$v) = split (/=/,$line);
        if ($n eq "NAME") { $name = $v; }
        elsif ($n eq "TYPE") { $type = $v; }
        elsif ($n eq "STATE")
        {
            $state = $v;
            if ($name =~ m/$s/)
            {
                printf ("%-45s %-12s %-18s\n",$name,$type,$state);
            }
        }
    }
}
```


The above Perl script produces similar output to the Oracle-supplied AWK script, but is a little more readable and therefore simpler to modify. It produces similar output to the AWK script.

Oracle Clusterware Permissions

- The CRS_GETPERM and CRS_SETPERM utilities can be used to check and modify Oracle Clusterware permissions
- For example to change the owner of an instance to oracle and group to oinstall
  - Check the current permissions

```
[root@server11] crs_getperm ora.TEST.TEST3.inst
Name: ora.TEST.TEST3.inst
owner:root:rwx,pgrp:root:r-x,other::r--,
```

  - Set the new permissions

```
[root@server11] crs_setperm ora.TEST.TEST3.inst -o oracle
[root@server11] crs_setperm ora.TEST.TEST3.inst -g oinstall
```

  - Check the new permissions

```
[root@server11] crs_getperm ora.TEST.TEST3.inst
Name: ora.TEST.TEST3.inst
owner:oracle:rwx,pgrp:oinstall:r-x,other::r--,
```


Oracle Clusterware permissions can occasionally be corrupted by incomplete installations.

It is possible to rectify permission errors using the CRS_GETPERM and CRS_SETPERM utilities as shown in the example (from a production site) above.

The Oracle user was receiving the following error when attempting to start instance TEST3 for on node server11

[oracle@server11]$ srvctl start instance -d TEST -i TEST3
PRKP-1001 : Error starting instance TEST3 on node server11
CRS-0254: authorization failure

After making the above changes, the Oracle user was able to start the instance without error

[oracle@server11]$ srvctl start instance -d TEST -i TEST3

[oracle@server11]$ srvctl status instance -d TEST -I TEST3
Instance TEST3 is running on node server11

Note that Oracle Clusterware was stopped while the permissions were changed. It is not known whether this is mandatory or optional.

High Availability Framework Overview

- Oracle Clusterware High Availability Framework
- Introduced in Oracle 10.2
- Allows you to use Oracle Clusterware to manage applications or processes running in the cluster
- Provides high availability for custom applications
- Can detect failures and optionally restart critical user applications and processes
- Does not require a RAC licence

In Oracle 10.2 and above, Oracle Clusterware provides a high availability framework that you can use to enable Oracle Clusterware to manage applications or processes running on the cluster. The framework enables you to provide high availability for custom applications in addition to the RAC database.

In the event of the failure of the node on which the application is running, Oracle Clusterware automatically fails over the application to another available node. Existing connections are lost, and for short period during the failover, no connections are possible. However, within a few seconds the application should be available again on another node.

You can use Oracle Clusterware in conjunction with ASM to create a consolidated pool of storage to support both single-instance and RAC databases running within the cluster. Oracle Clusterware ensures that the ASM file system remains available by eliminating any single points of failure among the ASM instances.

One obvious benefit of the Oracle Clusterware High Availability Framework is that it saves you reinventing the wheel by writing your own clustering software. In addition, Oracle Clusterware does not require a RAC license. However, any Oracle databases that are supported by Oracle Clusterware must have appropriate licenses (RAC or single-instance).

High Availability Framework Implementation Options

- Can be implemented in two different ways:
  1. Register applications with High Availability Framework
  2. Extend applications to use High Availability Framework API
- Application profiles can be created to:
  - Monitor application
  - Relocate and restart it on another node

Oracle Clusterware components can detect failures and optionally restart critical user applications and processes. You can either register your applications with the High Availability Framework or extend your applications using the High Availability Framework API. You can also create application profiles that monitor your application and potentially relocate and restart it on another node.

On the minus side, you will need to write and maintain some additional code in order to use the High Availability Framework. The only restriction on the language you use is that it must be capable of returning an integer code on exit. Also, you should remember that your applications will be contributing to CPU consumption on nodes that have full RAC licenses, so in extreme situations, you may need to purchase additional cluster nodes or CPUs and licenses to maintain response times on your database cluster.

Potential uses include high availability monitoring software and real-time data feeds to and from other systems.

High Availability Framework Application Compatibility

- For an application to be compatible with Oracle Clusterware it must have two basic properties:
  - Node Independence:
    - Application can run on any node in the cluster
    - Application binaries available on all nodes in the cluster
    - Any directories or files required by the application should be symmetrically available on all nodes
  - Client Connectivity:
    - Clients use application virtual IP addresses access the application
    - If the node running the application fails, both the application and the virtual IP address fail over to another node
    - Clients specify VIP address, not public IP address.

The application must be capable of running on any node in the cluster. Therefore, application binaries should be available on all nodes in the cluster in local file systems or on shared storage. In addition, any other directories or files required by the application

Clients should be able to access the application on any node in the cluster. As the client does not initially know which node the application will be running on, Oracle uses application virtual IP addresses, which provide a method for consistently accessing the application from the network. If the node on which the application is running fails, both the application and the virtual IP address will fail over to another node. All clients should specify the virtual IP address , not the public IP address

High Availability Framework Commands

- In Oracle 10.2 and above, Oracle Clusterware commands that support the High Availability Framework include:
  - crs_profile
  - crs_register
  - crs_unregister
  - crs_getperm
  - crs_setperm
  - crs_start
  - crs_stop
  - crs_stat
  - crs_relocate

High Availability Framework Deployment

- **To use the High Availability Framework you must perform the following:**

1. **Supply an action program that Oracle Clusterware will use to start, stop and check the status of your application.**
2. **Create an application VIP if the application will be accessed by external clients over the network**
3. **Create an application profile describing the application and its protection attributes**
4. **Register the application with Oracle Clusterware**

The action program can be written in any appropriate language and is similar in concept to the service configuration scripts found in /etc/init.d.

The application profile is a text file describing the application and its protection attributes.

High Availability Framework Example

- The following example demonstrates use of the High Availability Framework to protect a single-instance Oracle database
- Based on Oracle White Paper "Using Oracle Clusterware to Protect a Single Instance Oracle database"
- Limitations of this example:
  - Using existing Oracle Clusterware
  - Using existing Oracle homes
  - Using existing VIP addresses
  - Using existing ASM instances
  - Using existing listeners
  - No dependencies

The purpose of this example is to demonstrate simple use of the High Availability Framework.

This example is based on those described in the Oracle white paper "Using Oracle Clusterware to Protect a Single Instance Oracle database". However, I have considerably simplified the example to demonstrate the functionality without becoming overwhelmed with the details.

I have also assumed that you might need to test this example on an existing cluster and have therefore used as much of the existing infrastructure as possible.

I have used the existing hardware and shared storage together with existing installations of Oracle Clusterware and the Oracle database software. I have also used existing ASM instances and listeners.

I have implemented the action scripts using Bash shell; the white paper uses Perl.

The examples in the white papers describe how to create new ASM instances and listeners for the single-instance database. They also describe how to set up dependencies and to manage these within resource groups (containers)

High Availability Framework Example

- Create a single-instance database on one node
  - Using ASM for shared storage
  - Using SPFILE for parameters
- Copy the following files to equivalent locations on the other nodes
  - $ORACLE_BASE/admin/<database_name>/*
  - $ORACLE_HOME/dbs/init<database_name>.ora
  - $ORACLE_HOME/dbs/orapw<database_name>
- Check that the database can be started and stopped on each node

For this example I used a 4-node cluster consisting of nodes called server6, server7, server8 and server9.

The single-instance database was created on server6 using DBCA. I used existing ASM shared storage. I also used an SPFILE for initialization parameters.

After creating the database, copy the administration, initialization parameter and password files to the other nodes as follows

For example if the database is called TEST then on server6:

```
scp -p -r  $ORACLE_BASE/admin/TEST server7:$ORACLE_BASE/admin
scp -p -r  $ORACLE_BASE/admin/TEST server8:$ORACLE_BASE/admin
scp -p -r  $ORACLE_BASE/admin/TEST server9:$ORACLE_BASE/admin

scp $ORACLE_HOME/dbs/initTEST.ora server7:$ORACLE_HOME/dbs
scp $ORACLE_HOME/dbs/initTEST.ora server8:$ORACLE_HOME/dbs
scp $ORACLE_HOME/dbs/initTEST.ora server9:$ORACLE_HOME/dbs

scp $ORACLE_HOME/dbs/orapwTEST server7:$ORACLE_HOME/dbs
scp $ORACLE_HOME/dbs/orapwTEST server8:$ORACLE_HOME/dbs
scp $ORACLE_HOME/dbs/orapwTEST server9:$ORACLE_HOME/dbs
```

Check you can start and stop the database on each node.

High Availability Framework Example

- On each node as the root user create the following shell scripts in $ORA_CRS_HOME/crs/public
  - check_database.sh
  - start_database.sh
  - stop_database.sh

- check_database.sh

```
#!/bin/bash
export ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
export ORACLE_SID=DATA
RES=`ps -ef | grep ora_pmon_$ORACLE_SID | grep -v grep | wc -l`
if [ $RES -eq 0 ]
then
  exit 1
else
  exit 0
fi
```


Note that the #!/bin/bash directive at the start of each script was essential in order to avoid placement errors when Oracle Clusterware attempted to start and stop the new resources.

High Availability Framework Example

- start_database.sh

```
#!/bin/bash
export ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
export ORACLE_SID=DATA
$ORACLE_HOME/bin/sqlplus /nolog << EOF
CONNECT / AS SYSDBA
STARTUP
EXIT
EOF
```

- stop_database.sh

```
#!/bin/bash
export ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
export ORACLE_SID=DATA
$ORACLE_HOME/bin/sqlplus /nolog << EOF
CONNECT / AS SYSDBA
SHUTDOWN IMMEDIATE
EXIT
EOF
```


The Oracle examples use a single Perl script to perform all actions. Whilst this is a more elegant solution I have chosen to split out the start, stop and check database scripts for two reasons:

1 - They are easier to test

2 - They are easier to incorporate into these slides.

High Availability Framework Example

- On each node as the root user create the following shell script in $ORA_CRS_HOME/crs/public
  - action_database.sh
- The action script must handle three parameters
  - start
  - stop
  - check
- action_database.sh

```
#!/bin/bash
export CRS_HOME=/u01/app/oracle/product/10.2.0/crs
export CRS_HOME_PUBLIC=$CRS_HOME/crs/public
export ORACLE_USER=oracle

if [ $# -lt 1 ]
then
  echo "Syntax is : start stop check "
  exit -1
fi
```


The action script is called by Oracle Clusterware. It must handle three parameters:

• start

• stop

• check

If successful, each option should return the value 0; if unsuccessful each option should return a non-zero value.

In this example, I have taken some short cuts with the error handling to keep the script simple.

High Availability Framework Example

- action_database.sh continued

```
case $1 in
start)  echo "Starting database"
    su - $ORACLE_USER -c "$CRS_HOME_PUBLIC/start_database.sh"
    exit 0
     ;;
stop)   echo "Stopping database"
    su - $ORACLE_USER -c "$CRS_HOME_PUBLIC/stop_database.sh"
    exit 0
    ;;
check)  echo "Checking database"
    su - $ORACLE_USER -c "$CRS_HOME_PUBLIC/check_database.sh"
    RES=$?
    exit $RES
    ;;
*)      echo Invalid argument
    exit -1
    ;;
esac
```


This section of the script contains a case statement which handles the three parameters. In each case I have created a new shell for the Oracle user using /bin/su and then executed one of the previously created commands.

High Availability Framework Example

- Create a profile for the action script

```
crs_profile -create testdb -t application \
-a /u01/app/oracle/product/10.2.0/crs/crs/public/action_database.sh \
-o ci=60,ra=5
```

- This creates a profile file called
  - $ORA_CRS_HOME/crs/profile/testdb.cap

```
NAME=testdb
TYPE=application
ACTION_SCRIPT=$ORA_CRS_HOME/crs/public/action_database.sh
ACTIVE_PLACEMENT=0
AUTO_START=restore
CHECK_INTERVAL=60
DESCRIPTION=testdb
PLACEMENT=balanced
RESTART_ATTEMPTS=5
....
```


In this example I have created a profile for an application called testdb with the following properties

-t application - type of OCR entry - must be application

-a <pathname> - pathname of action program used to start, stop and check the application

-o ci=60 - check interval every 60 seconds

-o ra-5 - maximum of 5 restart attempts

The profile file (testdb.cap) contains around 38 lines; I have only included the entries that are of interest in this example.

High Availability Framework Example

- **The next step is to register the application with Oracle Clusterware**

```
crs_register testdb
```

- **It is necessary to set permissions so that the testdb application can run as the Oracle user**

```
crs_setperm testdb -u user:oracle:r-x
```

- **Start the application on the local node**

```
crs_start testdb
Attempting to start `testdb` on member `server6`
Start of `testdb` on member `server6` succeeded
```

- **To stop the application again use:**

```
crs_stop testdb
Attempting to stop `testdb` on member `server6`
Stop of `testdb` on member `server6` succeeded
```


The application can be accessed by clients using the following TNS entry:

```
TEST =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = server6-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = server7-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = server8-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = server9-vip)(PORT = 1521))
  (LOAD_BALANCE = yes)
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = TEST)
  )
 )
```

In this example the TNSNAMES entry is using the VIP addresses e.g. server6-vip, server7-vip etc from the existing cluster infrastructure. In reality, you would probably want to create application VIPs specifically for the single-instance database.

High Availability Framework Example

- To check the current status of the application use crs_stat

```
crs_stat -t  testdb
Name           Type           Target    State     Host
------------------------------------------------------------
testdb         application    ONLINE    ONLINE    server6
```

- The application can be relocated to another node e.g.

```
[root@server6 public]# crs_relocate testdb
Attempting to stop `testdb` on member `server6`
Stop of `testdb` on member `server6` succeeded
Attempting to start `testdb` on member `server7`
Start of `testdb` on member `server7` succeeded.
crs_start testdb
```

```
crs_stat -t  testdb
Name           Type           Target    State     Host
------------------------------------------------------------
testdb         application    ONLINE    ONLINE    server7
```


It is possible to specify which node the application should be relocated to using the -c <cluster_member> parameter. For example:

crs_relocate testdb -c server8

will relocate testdb from the existing node to server8.

High Availability Framework Example

- If the application is currently located on server7

```
crs_stat -t  testdb
Name           Type           Target    State     Host
------------------------------------------------------------
testdb         application    ONLINE    ONLINE    server7
```

- and server7 fails e.g. init 0
- Oracle Clusterware will detect that the server has been shutdown

```
crs_stat -t  testdb
Name           Type           Target    State     Host
------------------------------------------------------------
testdb         application    ONLINE    OFFLINE
```

- and restart the application on another node:

```
crs_stat -t  testdb
Name           Type           Target    State     Host
------------------------------------------------------------
testdb         application    ONLINE    ONLINE    server8
```


This example shows the behaviour of the application following a node failure.

Oracle Clusterware does not immediately detect that the application has failed. Even after the node was completely shutdown, crs_stat -t testdb continued to show that the application was ONLINE on server7 for about another 60 seconds (check interval).

Once Clusterware has detected that the node has been shutdown, the state is shown as OFFLINE while the application is relocated to another node.

The application will be restarted on one of the remaining nodes after which the state will change to ONLINE and the new host will be shown.

High Availability Framework Dependencies

- **Dependencies can be specified in the profile file. For example:**

```
crs_profile -create asm -t application \
-a /u01/app/oracle/product/10.2.0/crs/crs/public/action_asm.sh \
-o ci=60,ra=5
crs_register asm
crs_setperm asm -u user:oracle:r-x
```

```
crs_profile -create db -t application -r asm \
-a /u01/app/oracle/product/10.2.0/crs/crs/public/action_db.sh \
-o ci=60,ra=5
crs_register db
crs_setperm db -u user:oracle:r-x
```

```
crs_profile -create listener -t application -r db \
-a /u01/app/oracle/product/10.2.0/crs/crs/public/action_listener.sh \
-o ci=60,ra=5
crs_register listener
crs_setperm listener -u user:oracle:r-x
```


In the above example, the listener is dependent on the database, which is, in turn, dependent on ASM.

Dependencies are specified using the -r <name> argument of crs_profile. The name specifies the name of the Oracle Clusterware managed resource that must be in status ONLINE for the application to start.

High Availability Framework
Application VIPs

- Oracle Clusterware can also support application VIPs
- Create a profile for the VIP

```
crs_profile -create testdbvip -t application
-a $ORA_CRS_HOME/bin/usrvip \
-o oi=eth0,ov=192.168.1.200,on=255.255.255.0
```

- Register the VIP with Oracle Clusterware using:

```
crs_register testdbvip
```

- Set the owner of the application VIP to root:

```
crs_setperm  testdbvip -o root
```

- Grant the oracle user permission to run the script:

```
crs_setperm testdbvip -u user:oracle:r-x
```

- Start the application vip

```
crs_start testdbvip
```


The VIP is used by the client to locate the application, irrespective of what node it is currently running on.

In the following example, we will create an application VIP with the IP address 192.168.1.200 and the subnet mask of 255.255.255.0. You may wish to add an entry for the application VIP address to /etc/hosts or your DNS database. The crs_profile command includes options to specify the interface (eth0), the VIP address (192.168.1.200) and the subnet mask (255.255.255.0)

The action program for the VIP is $ORA_CRS_HOME/bin/usrvip which is supplied with Oracle Clusterware.

The crs_profile command generates an application profile called testdb.cap in the $ORA_CRS_HOME/crs/public directory

You must configure the application VIP to run as the root user. Note that your own application can run as any user.

The application VIP can be defined as a dependency for the database application. For example:

```
 crs_profile -create testdb -t application -r testdbvip\
-a /u01/app/oracle/product/10.2.0/crs/crs/public/action_database.sh \
-o ci=60,ra=5
```

The -r testdbvip option specifies that testdb resource depends on the testdbvip resource.