Oracle RAC Best Practices on Linux

An Oracle White Paper
November 2003

ORACLE

TABLE OF CONTENTS

EXECUTIVE OVERVIEW

Oracle Real Application Clusters (RAC) is a multi-node extension to the Oracle database server and a key component in Oracle's Grid strategy.

Oracle Real Application Clusters (RAC) is a multi-node extension to the Oracle database server and a key component in Oracle's Grid strategy. Commodity hardware, such as blade servers, and commodity OS, such as Linux offer the most compelling cost benefits for Grids. Grids must leverage these commodity components to achieve maximum utilization. Oracle RAC runs real applications on the commodity components and enables high utilization on commodity blade farms. Applications running on RAC can dynamically leverage more blades provisioned to them. Similarly, these applications can easily relinquish these blades when they no longer need them.

This paper will present some of the best practices that have been implemented at more than 500 customer sites that you can use in your own environments to ultimately realize the benefits of moving your IT infrastructure towards a Grid architecture.

Much has been written and presented to date on the RAC architecture, and the business benefits of implementing this form of clustering. However, very little has been written regarding strategies and best practices for actually deploying applications on RAC. Oracle's RAC Pack is a team of RAC implementation specialists that are part of the RAC Development organization. To date, this global team has assisted over 500 customers successfully deploy their business applications on top of RAC, with a significant percentage of these deployments being on Linux. These applications cover the spectrum - from in-house custom applications, to off-the-shelf applications, and from DSS systems to operational OLTP systems. This paper will present some of the best practices that have been implemented at these customer sites, and that you can use in your own environments to ultimately realize the benefits of moving your IT infrastructure towards a Grid architecture.

PLANNING

Many of the challenges that have been encountered at customer sites have been related to unreasonable expectations, misunderstandings about what RAC is, and what it isn't, and inadequately defined objectives for the RAC implementation.

A successful RAC implementation is not all about the mechanics of installing the software, building the database, and tweaking the knobs to make it perform the way you want it to. Many of the challenges that have been encountered at customer sites have been related to unreasonable expectations, misunderstandings about what RAC is, and what it isn't, and inadequately defined objectives for the RAC implementation. Having these important issues addressed up front allows you to put together a structured implementation plan that gives the best opportunity to realize the desired business benefits.

Understand RAC Architecture

There is a great deal of confusion and misunderstanding in the marketplace around clustering in general, and this easily creates some fundamental misconceptions about what RAC is, and how to implement it.

Understanding the architecture is an obvious initial step prior to implementing any technology, but there is a great deal of confusion and misunderstanding in the marketplace around clustering in general, and this easily creates some fundamental misconceptions about what RAC is, and how to implement it. It is beyond the scope of this paper to provide a complete primer on clustering architectures, and the detailed hardware and software components that make up those architectures, but some things to consider in reviewing cluster-related literature include:

- Distinctions between hardware clustering and database clustering.
- Shared nothing vs. shared data vs. shared cache DB architectures.
- Active-Passive vs. Active-Active.
- Hot standby vs. mutual takeover vs. distributed load clustering
- Cold failover vs warm failover vs hot failover

.

Terminology aside, it is important to note that all clustering architectures provide some degree of high availability through redundancy. However, only RAC, and the hardware cluster architecture on which it is based, provides scalability benefits in addition to HA benefits.

Getting past the terminology is the first step. The next is understanding the hardware and software components that comprise the architecture.

Getting past the terminology is the first step. The next is understanding the hardware and software components that comprise the architecture. In the Lintel environment, both IA32 and IA64 architectures are supported, NAS, SCSI, and fiber storage are all compatible, and Gigabit Ethernet is currently the best choice for the private interconnect. Each cluster configuration can be a little bit different, so it is important to work closely with the hardware vendor to identify specific driver and/or patch requirements for the cluster being implemented.

The cluster software is the key enabling technology for RAC, and also the key differentiator with respect to various cluster architectures and platform-specific implementations of those architectures.

At a minimum, RAC requires a cluster software infrastructure that can provide concurrent access to the same storage and the same set of data files from all nodes in the cluster, a communications protocol for enabling IPC across the nodes in the cluster, and a mechanism for monitoring and communicating the status of the nodes in the cluster. On Linux, since no hardware or OS vendor supplies a cluster manager with all the required functionality, the majority of this functionality is provided by the Oracle-supplied cluster manager.

Set Your Expectations Appropriately

If your application will scale transparently on SMP, then you can expect it to scale well on RAC, without having to make any code changes to the application.

It is not hard to understand how the marketing messages of "RAC enables your application to scale with no code changes" and "RAC makes your system Unbreakable" could easily get watered down to something like "RAC makes any application scalable" or even "RAC is all you need for High Availability". In terms of practical expectations, the following much more accurately represents reality:

- If your application will scale transparently on SMP, then you can expect it to scale well on RAC, without having to make any code changes to the application.
- RAC eliminates the database instance, and the node itself as a single point of failure, and ensures database integrity in the case of such failures.

Expectations need to come from an understanding of the technology, not from an interpretation of the marketing buzz. Many challenges with deployments originate from a misunderstanding of exactly what RAC can do, and what it doesn't do.

The two facts above represent the two key aspects of RAC. The first is scalability. RAC is not magic. It cannot make a non-scalable application scale. Scalability is a direct function of the degree of contention introduced by the application for system resources and data. If the application won't scale in moving from a 4 processor to an 8 processor SMP configuration, then it certainly won't scale going from a single 4 processor box to a cluster of 2 4 – way boxes.

This does not mean that you do not have to do anything in terms of configuring the database connection mechanisms, or turning on facilities in the database that are designed specifically in the current release to improve performance, scalability, availability, or manageability.

In terms of application changes, while it may be true that logic in the application need not be changed in order to deploy successfully on RAC, this does not mean that you do not have to do anything in terms of configuring the database connection mechanisms, or turning on facilities in the database that are designed specifically in the current release to improve performance, scalability, availability, or manageability. Additionally, there is no substitute for good design. If there are known points of contention or serialization in the application code, and these can be changed reasonably easily, then you could expect the scalability realized in both the SMP and the clustered environments to be that much better.

From a high availability perspective, RAC is a database failover technology only. It ensures that in the event of node or instance failure, the failed instance will be recovered, the database will be returned to a transactionally consistent state automatically, and that there is an alternate access path (via a surviving node in the cluster) for the application to access the data. RAC provides a highly available database service. What isn't often understood is that RAC by itself currently does nothing to preserve application connections, login context, and session state. It provides no protection from network failures, human errors, data

errors, or site disasters. As such it is not an end-to-end high availability solution. RAC is a critical piece of the technology stack when trying to achieve high availability service level objectives, but much more is required if those service levels are to be fully realized.

Define Your Objectives

Without clear objectives in mind, you have no way of determining whether or not you have actually achieved what you set out to do.

Without clear objectives in mind, you have no way of determining whether or not you have actually achieved what you set out to do. Objectives need to be categorized, quantified, and measurable in some way. High availability, incremental scalability, server consolidation, database consolidation, reduced TCO, improved capacity utilization, etc. are all possible categories of objectives.

- High availability objectives may be defined in terms of a Service Level Agreement with specific tolerances for downtime associated with various types of outages, e.g., Recovery time after node failure < 2 minutes.
- Scalability objectives are often a bit more difficult to nail down as they may vary depending on workload type – online vs. batch, DSS vs. OLTP, etc. Regardless, scalability represents the relationship between workload, system capacity, and response/processing time. E.g. a fully loaded single system supports 1000 users with Average response time < 2 secs. By doubling system capacity, and assuming scalability of 80%, the objective might be to support 1600 users and maintain average response time < 2 secs.
- Server or database consolidation objectives are conceptually much easier, and are often tied to TCO objectives, but are typically dependant on more subjective measures like ease of use, or ease of management, making them more difficult to actually measure success.

Different organizations, and even different applications within an organization have different primary business drivers. Whatever the business drivers are, the objectives need to be clearly defined with the drivers in mind. These objectives then help define the project plan, the testing plan, and help maintain a focus throughout the implementation.

Build a Project Plan

There are several key elements that may easily be missed in a project plan that are particularly important in a RAC environment. Partnering well with your vendors is one of these

This is perhaps the most obvious component of the planning phase. However, there are several key elements that may easily be missed in a project plan that are particularly important in a RAC environment.

- Partner with your vendors. This is important for 2 reasons:
  - You have the best opportunity of acquiring and staging a cluster configuration that has been well tested and proven.
  - You have a stakeholder in the problem diagnosis and resolution process.

By limiting the number of vendors you minimize the points of integration and the number of places to turn in trying to diagnose and resolve a problem.

- Simplify. In a best-case scenario, you have a single systems vendor, Oracle as your software infrastructure vendor, and your custom or 3rd party application. Integration is the single most difficult task for IT professionals. With only three vendors, you minimize the points of integration, and you have relatively few places to turn in trying to diagnose and resolve a problem. In a more typical scenario, particularly in the Lintel environment, you will have different vendors for the machines themselves, for the I/O subsystem, for the network, for the OS, possibly for the clusterware, Oracle, and one or more application providers. This has the potential to be a diagnostic nightmare if/when a problem arises. Fortunately, many of the technology vendors have worked very closely to provide a well-integrated and tested technology infrastructure. By effectively partnering with your chosen vendors, you will quickly learn what technology components have been tested and are known to work well together. You should also be seriously considering using one of the 2 flavours of Oracle-supported "Unbreakable" Linux: RedHat AS, or UnitedLinux as this improves/simplifies both the ease with which the clusters can be built, and supported. A preinstalled, pre-integrated, and fully regression tested cluster is a real fast track to a successful RAC deployment, and can significantly improve mean time to resolution of any problems.

Use proven strategies for building mission critical systems. Eliminating single points of failure, maximizing Mean Time Between Failures (MTBF), minimizing Mean Time to Recovery (MTTR), using proven technology components, avoiding x.0 releases, etc. are all pertinent to achieving the objectives.

- Use proven strategies for building mission critical systems. Minimizing the number of integration points, eliminating single points of failure, maximizing Mean Time Between Failures (MTBF), minimizing Mean Time to Recovery (MTTR), using proven technology components, avoiding x.0 releases (in production), etc. are all pertinent to achieving the objectives. Many of the strategies in successfully deploying mission critical systems boil down to minimizing risk. It is also worth noting that there is inherent risk in being too cautious. New technology allows you to do things that you couldn't do before. And technology changes rapidly. By being too cautious, you risk not only not achieving the objectives you set out

to achieve, but also being obsolete in a very short period of time with no opportunity to realize a return on the investment. The strategy cannot be risk avoidance, but rather must be risk mitigation and management.

Address knowledge gaps and training requirements. Clusters are new to many organizations. Clustered databases are new to most. High availability is non-trivial to implement. Scalability is not often well understood.

- Address knowledge gaps and training requirements. Clusters are new to many organizations. Clustered databases are new to most. High availability is non-trivial to implement. Scalability is not often well understood. These issues can easily work against you. Formal training on specific technologies like the storage subsystem, the volume manager, the clusterware, or RAC is always useful. There is, however, no substitute for experience, so leveraging skills of people who have done it before, to help you in your own environment has proven to be immensely beneficial. There is not shame in not knowing. There is only shame in not identifying what you don't know. Both Oracle and our hardware partners have packaged service offerings to assist specifically with this, and a large percentage of the 500+ customers currently running RAC in production have taken advantage of this. Hiring someone to simply do a cluster install and configuration, or a RAC install/config is a low value proposition. But when you combine that effort with well-defined technology/knowledge transfer objectives, you have a truly worthwhile investment.

Establish support mechanisms and escalation procedures. It is inevitable that some kind of problem will occur that will require vendor assistance to resolve.

- Establish support mechanisms and escalation procedures. While a great deal of the design effort that goes into building a clustered environment is an attempt to mitigate problems in the technology stack when they occur, it is inevitable that some kind of problem will occur that will require vendor assistance to resolve. As a result, it is critical to understand how to maximize the effectiveness of vendor Support organizations to your advantage. Understanding the organizational structure, the diagnostic mechanisms and processes, the routing mechanisms, the escalation procedures, and expected turn around times for various types of issues are all critical to minimizing mean time to resolution. Again, partnering and joint ownership of issue resolution, with an expectation that some back and forth work may be required to get to root cause, are critical overall success factors.

IMPLEMENTATION

Implementation involves acquiring and configuring the necessary hardware and OS, implementing the cluster software and Oracle software, building a RAC database, deploying your applications to the RAC environment, and doing some structured testing and tuning.

Assuming the planning has been done with the above considerations in mind, implementation can be reasonably straightforward. Conceptually, implementation involves acquiring and configuring the necessary hardware and OS, implementing the cluster software and Oracle software, building a RAC database, deploying your applications to the RAC environment, and doing some structured testing and tuning.

RAC is a technology that is really only possible due to the convergence of several disparate technology advancements:

- Intelligent I/O subsystems (SANs)
- Sophisticated high bandwidth hardware interconnects
- Efficient low latency communication protocols

It turns out you can actually run RAC on 2 laptops with a dual-ported shared SCSI drive, and a 10Mb/s ethernet network between them. This is instructive in terms of understanding the absolute minimum requirement to allow RAC to function, but is of use for little more than demonstration purposes. This points out the need to design and build your RAC cluster environment from the ground up with your objectives clearly in mind:

The I/O subsystem itself needs to be able to handle the required throughput, and be able to scale to handle anticipated growth.

- The I/O subsystem itself needs to be able to handle the required throughput, and be able to scale to handle anticipated growth. SAN technology can virtually assure the I/O subsystem is not a scalability bottleneck.

The interconnect technology must be scalable to handle the amount of traffic generated by the cache synchronization mechanism.

- The interconnect technology must be scalable to handle the amount of traffic generated by the cache synchronization mechanism (which is directly related to the amount of contention created by the application). It is advisable to implement the highest bandwidth, lowest latency interconnect that is available for a given platform. The volume of synchronization traffic directly impacts the bandwidth requirement, and messaging delays are highly dependant on the IPC protocol. The interconnect is not something that you should risk having under configured, assuming scalability is a key objective. The current recommendation for Linux environments is 1GB Ethernet, and UDP is the only available IPC protocol. 10g will see the introduction of support for emerging technologies like Infiniband which will greatly improve interconnect scalability (and standardization) for large numbers of nodes, as well as provide a choice of interconnects under Linux.

The cluster infrastructure needs to scale to handle the number of nodes required.

- The cluster infrastructure needs to scale to handle the number of nodes required to provide sufficient processing capacity for anticipated growth in workload. Most cluster frameworks today differ in terms of the maximum number of nodes they can handle. Most can support a minimum of 4 nodes today, and some can support hundreds. This will also be driven by your scalability objectives and capacity requirements.

The nodes themselves may or may not need to be scalable to provide additional capacity in terms of CPU or memory.

- The nodes themselves may or may not need to be scalable to provide additional capacity in terms of CPU or memory. The ability to scale both within a machine as well as across machines is often desirable. Linux machines claim to be able to scale up to 8 CPUs, but the majority of systems are 2 or 4 CPU nodes. SMP scalability in Linux beyond 4 CPU's is not well proven, so the current recommendation is to stick with 4-CPU machines, or else 2-way Blade servers since 4-ways are not currently available.

Single Points of Failure (SPOFs) need to be eliminated in the hardware and software stack.

- Single Points of Failure (SPOFs) need to be eliminated in the hardware and software stack. Power supplies, HBA's, NIC's, switches, interconnects, OS images, software trees, etc, all represent potential SPOFs. The degree to which you attempt to eliminate these will be a function of your high availability Service Level Agreements (SLAs). Remember that if you have only a single interconnect, or a single path from each node to the storage, a NIC or HBA failure renders that node unusable, with a proportional decrease in system-wide processing capacity. An interconnect switch failure could render all but a single node unusable, and a SAN switch failure could render the entire cluster unusable. Redundancy needs to be built into these components as dictated by the SLAs. The tradeoff here is the cost factor. High Availability can be as expensive as you allow it to be. Building internal redundancy into the IO subsystem, networking infrastructure, and the machines themselves will greatly increase the cost per node. The investment in eliminating SPOFs must be made with an understanding of the overall cost and exposure of downtime. Many customers do not bother with building the internal redundancy into the nodes themselves if the cost to the business of losing a node in the cluster is tolerable.

The workload distribution (load balancing) strategy must be determined.

- The workload distribution (load balancing) strategy must be determined so it is clear how the application is actually going to connect to the various nodes and instances. Workload distribution could be on a per-connection basis, or based on server CPU-load, perhaps on an application basis. Different strategies may be more appropriate than others depending on the implementation

A management infrastructure must be set up.

- A management infrastructure must be set up and configured to monitor the system, keep management costs under control, and to manage to the targeted service levels.

Common Linux "Issues"

LVM

Lack of a cluster aware logical volume manager has not yet proven to be a show-stopper at any customer.

RHAS 2.1 does not ship with a logical volume manager. RHAS 3.0 and UnitedLinux do ship with an LVM, but it is not cluster aware. While this issue does reflect on the overall maturity of Linux, it has not yet proven to be a show-stopper at any customer. As a best practice, more and more storage configuration is being done in the storage subsystem itself, including the striping and mirroring, volume grouping, etc., leaving relatively little that needs to be managed at the OS level. While tools like fdisk, or unclustered LVM's are relatively primitive, and will require some manual intervention, this is compensated for by the sophistication of the storage subsystems themselves.

Large Memory

There are several ways to extend the Oracle SGA beyond the default of approx 1.7GB on 32-bit Linux.

There are several ways to extend the Oracle SGA beyond the default of approx 1.7GB on 32-bit Linux, including lowering the Oracle base address, configuring Linux BIGPAGES, and configuring Oracle's VLM. If there is a known requirement for a large memory configuration, then best practice would dictate starting with an environment that naturally supports that requirement – namely IA64 and 64-bit Linux, rather than attempting to use these other techniques that are effectively workarounds for 32-bit addressing limitations.

Cluster Installation/Configuration

Validating the cluster infrastructure, prior to installing the Oracle software, will save you considerable time and potential headaches.

Validating the cluster infrastructure, prior to installing the Oracle software, will save you considerable time and potential headaches. The Oracle Universal Installer is cluster-aware, and in fact dependent on the cluster being operational in order to install RAC and build a cluster database. Ensuring full read/write access to the shared storage (either raw devices or cluster file system), and ensuring the cluster communication is occurring over the correct private interconnect, should be the minimal validation checks. Specific patch levels for both the OS and clusterware are generally documented in the Oracle installation/configuration guide for your specific platform.

In addition to validating the cluster, there are several additional best practices related to the physical configuration of the system in preparation for RAC:

- Avoid the use of a cross-over cable for the interconnect (use a switch).
- Use Asynch IO

- Use S.A.M.E. methodology [1] for storage configuration
- If using Oracle Clustered File System (OCFS), ensure you have the latest release from http://oss.oracle.com/projects/ocfs.
- If using raw volumes:
  - Pre-allocate extra raw volumes, and make available to DBA to add to database as necessary
  - Minimize the number of different sizes of raw volumes – 4-6 different sizes are generally sufficient
  - Do not embed tablespace name in device name. On some systems, it may make sense to embed the database name.
- Set UDP packet send and receive parameters to 256K.
- Use the latest Linux Kernel RPM
- Cross check recommended OS patches with Oracle, Linux vendor, AND system vendor

Oracle Cluster File System Installation/Configuration

OCFS is designed to provide an alternative to using raw devices for Oracle RAC.

OCFS is designed to provide an alternative to using raw devices for Oracle RAC. Managing raw devices can be cumbersome, particularly in Linux where the lack of a clustered volume manager is problematic. Also many DBAs and system administrators are more familiar with filesystems. Also, with raw devices on Linux, you are limited to only 255 raw devices - there cannot be more than 255 /dev/raw entries. So for any reasonably large application deployment, like the default Oracle eBusiness Suite implementation for example, this restriction forces a workaround involving tablespace consolidation to minimize the number of distinct raw devices required. OCFS provides a better solution.

It should be noted that at this time OCFS only supports Oracle data files. OCFS is expected to also support Shared Oracle Home installs in the not too distant future.

It should be noted that at this time OCFS (V1.0.x) only supports Oracle data files. This includes Redo Log files, Archive log files, Control files and database Data files. The shared quorum disk file for the cluster manager and the shared init file (srvconfig) are also supported. OCFS 2.0 is expected to also support Shared Oracle Home installs in the not too distant future.

---

[1] See "Optimal Storage Configuration Made Easy", by Juan Loaiza, Oracle Corporation.

http://otn.oracle.com/deploy/availability/pdf/oow2000_same.pdf

There are some specific recommendations relating to OCFS configuration and use including selection of block size, placement of archive log files, and use of partitions.

Following are some specific recommendations relating to OCFS configuration and use:

- Format with 128KB block size. Sizes between 2KB and 1MB are supported. The smaller block sizes will have a performance penalty, but will be useful for the future when OCFS supports regular files. With 128KB block size, every file created with content uses a minimum of 128KB space on disk, even if there is only 1 byte of data in the file. The 128KB recommendation has thus far proven to be a reasonable tradeoff between space utilization and performance.
- Place archive log files on a separate disk partition from the data files, and if possible a separate archive log partition for each node, for performance reasons. Mount all these on every node in the cluster, even though only one node will be writing out archive log files to its assigned partition. This reduces contention on space allocation if the database is heavily used and lots of archive logs are created on each node.
- All data files can reside on the same partition. An Oracle block size of 8KB is usually recommended and on OCFS this does not change. Use an extent size that is the same on every tablespace in order to prevent disk fragmentation. OCFS requires contiguous space on disk for every oracle extension, or for the initial data file creation. E.g. if you create a 1GB datafile in 1 command, it will require 1GB contiguous space on disk. If you then extend this data file by 100MB, it will need another 100MB contiguous chunk on disk, but which need not be contiguous with the original 1GB allocation.
- OCFS will support partition sizes up to 1TB (tested) configurations. Since there is no volume management built into OCFS it is recommended to have hardware raid support and create a logical disk volume that is large enough. If this is not possible, depending on the Linux distribution you run, lvm or md can be used.
- Avoid large numbers of mount points as this tends to create a performance bottleneck (typically < 50)
- OCFS has support for mount-by-label. If you create your OCFS volumes with a unique ocfs LABEL, mount will support this. RHAS2.1's mount errata release has updates for OCFS and it is possible to simply do: mount -t ocfs -L mylabel /ocfs. This will be of great help if the device naming is variable.
- For performance reasons, ensure updatedb is not running on the OCFS partitions. (E.g. add OCFS to PRUNEFS= in /etc/updatedb.conf). updatedb runs on OCFS by default

- It is useful to note that for troubleshooting purposes, OCFS will print debug and error information into the system log. /var/log/messages or dmesg will show OCFS related errors. If a problem is suspected to be related to OCFS, a few things can be checked quickly:
    - type 'dmesg' and see if there are messages starting with OCFS.
    - run 'ps aux' and find out which process is hanging, (D state). If this does not change, strace -p <pid> will show if there is any process activity. Please also provide this information to Oracle Support.
- Update /etc/fstab and /etc/modules.conf to automatically load OCFS correctly, as per the documentation.
- Sample configuration and layout

  Install the ocfs-kernel module, depending on your kernel

  uname -a will provide you with the current running kernel

  Install the UI tool and the format tools.

  ocfs-tools-XX.rpm and ocfs-support-XX.rpm

  Sample filesystem layout (same mountpoint on every node in the RAC cluster)

  /ocfs-data

  /ocfs-index

  /ocfs-ctrl-redo

  /ocfs-ctrl-redo2

  /archive1

  /archive2

  /archive3

  /archive4

Oracle Cluster Manager Installation/Configuration

The hangcheck-timer kernel module is not required for oracm operation, but its use is highly recommended.

There is considerable (and understandable) confusion relating to the Linux implementation of the Oracle Cluster Manager. The original 9iR1 implementation included a component called watchdog. The "watchdogd" daemon impacted system availability as it initiated system reboots under heavy workloads. As of Oracle 9.2.0.2, the watchdogd implementation in oracm has been removed, and replaced with a kernel-based implementation known as hangcheck-timer. The hangcheck-timer kernel module is not required for oracm operation, but its use is highly recommended. This module monitors the Linux kernel for long operating system hangs that could affect the reliability of a RAC node and potentially cause a corruption of a RAC database. When such a hang occurs, this module reboots the node. This approach offers the following three advantages over the watchdogd approach:

- Node resets are triggered from within the Linux kernel making them much less effected by system load,
- oracm on a RAC node can easily be stopped and reconfigured because its operation is completely independent of the kernel module,
- The features provided by the hangcheck-timer module closely resemble features available in the implementation of the Cluster Manager for RAC on the Windows platform, on which the Cluster Manager on Linux was based.

A fresh install involves first installing 9.2.0.1, and then upgrading to 9.2.0.3 or 9.2.0.4. As a result, close attention must be paid to the documented parameter changes in the configuration file.

The change in implementation causes confusion both in upgrading from earlier releases of 9iRAC, and with new installs, since a fresh install involves first installing 9.2.0.1, and then upgrading to the latest patch level (currently 9.2.0.4). As a result, close attention must be paid to the documented parameter changes in the configuration file:

1. Remove the following parameters from cmcfg.ora on all nodes in the cluster:

   WatchdogTimerMargin
   WatchdogSafetyMargin

2. Ensure the parameter *KernelModuleName* (introduced by Hangcheck timer) is set properly in cmcfg.ora on all nodes in the cluster. This will allow the oracm to know the name of the hangcheck-timer kernel module so it can determine if it is correctly loaded: If the module in KernelModuleName is either not loaded but correctly specified or incorrectly specified, the oracm will produce a series of error messages in the syslog system log (/var/log/messages). However, it will not prevent the oracm process from running. The module must be loaded prior to oracm startup.

3. Ensure the parameter *CMDiskFile* has been set properly. Hangcheck timer changes the use of this configuration parameter from optional to mandatory. This is done to ensure that a CM quorum partition is used and allows the oracm

to more reliably handle certain kinds of hardware and software errors that affect cluster participation.

4. The inclusion of the hangcheck-timer kernel module also introduces two new configuration parameters to be used when the module is loaded:

hangcheck_tick - the hangcheck_tick is an interval indicating how often the hangcheck-timer checks on the health of the system.

hangcheck_margin - certain kernel activities may randomly introduce delays in the operation of the hangcheck-timer. hangcheck_margin provides a margin of error to prevent unnecessary system resets due to these delays.

Taken together, these two parameters indicate how long a RAC node must hang before the hangcheck-timer module will reset the system. A node reset will occur when the following is true:

(system hang time) > (hangcheck_tick + hangcheck_margin)

Example of loading the hangcheck-timer. Put this into the rc.local script on RH or the /etc/init.d/oracle script on UL

```
# load hangcheck-timer module for ORACM 9.2.0.3
/sbin/insmod /lib/modules/2.4.19-4GB/kernel/drivers/char/hangcheck-timer.o
hangcheck_tick=30 hangcheck_margin=180
```

Oracle RAC Installation/Configuration

If the cluster has been installed and configured correctly, the Oracle and RAC install will go very smoothly.

If the cluster has been installed and configured correctly, the Oracle and RAC install will go very smoothly. It is a good practice to return to the platform-specific Oracle Installation and Configuration Guide and confirm all documented prerequisites have been met, including disk space for the software, memory requirements, /tmp and /swap space, and OS and clusterware patch levels. It is also useful to check Metalink for “Step-by-Step Installation of RAC” technical notes, and/or OTN for additional RAC Quickstart guides for your specific platform (IA32 vs. IA64, RedHat vs. UnitedLinux).

The current version recommendation is 9.2.0.4. It has proven advantageous to use a staging area on disk to both avoid having to swap CD’s during the install, and to improve the speed of the install. The Oracle Universal Installer (OUI) will automatically detect the cluster, and install the Oracle software, including RAC, on the selected nodes of the cluster. Using NFS mounts for ORACLE_HOME are not recommended due to the single point of failure this

introduces, as well as the fact that some node-specific configuration information is still stored in ORACLE_HOME.

Database Creation

The Database Configuration Assistant (DBCA) is the recommended way to create and initially configure RAC databases.

The Database Configuration Assistant (DBCA) is the recommended way to create and initially configure RAC databases. DBCA simplifies database creation, automatically configures some important 9i features, and is fully cluster aware. At a minimum, it is recommended to use it to build the database creation scripts, which can subsequently be executed manually. DBCA performs a number of functions in the creation process that would otherwise have to be dealt with manually.

Before using DBCA to build a cluster DB, you will need to make sure gsd is running. OUI will take care of this for you if running DBCA as part of the install process, but otherwise, you must do it manually. DBCA will also:

- Properly set MAXINSTANCES, MAXLOGFILES/MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES to avoid downtime in having to recreate the control file.
- Create all tablespaces as locally managed (LMT's: `EXTENT MANAGEMENT LOCAL`)
- Create all tablespaces with Automatic Segment Space Management (ASSM: `SEGMENT SPACE MANAGEMENT AUTO`)
- Configure Automatic UNDO Management
- Use SPFILE instead of multiple init.ora's

Note that the use of LMT's, ASSM, and automatic UNDO are also single instance recommendations, so are not specific to RAC. DBCA will configure these important features in the creation of non-cluster databases as well, as they greatly simplify space management and reduce contention on the data dictionary. ASSM is particularly useful in RAC databases as it eliminates the need for setting of freelists and freelist groups, and allows for dynamic affinity of space to instances.

After running DBCA, you should have a fully configured RAC database up and running on all nodes of the cluster.

The actual interconnect (and protocol) Oracle is using for GCS/GES traffic can be validated as follows:

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

The information written to a trace file in the user_dump_dest will indicate the IP address:

```
SSKGXPT 0x2ab25bc flags        info for network 0
    socket no 10    IP  10.0.0.1        UDP 49197
    sflags SSKGXPT_UP
    info for network 1
    socket no 0     IP 0.0.0.0      UDP 0
    sflags SSKGXPT_DOWN
```

Application Deployment

In general, the deploying your application to a RAC environment follows the same guidelines as for single instance. As such, single instance tuning recommendations still very much apply.

In general, the deploying your application to a RAC environment follows the same guidelines as for single instance. As such, single instance tuning recommendations still very much apply:

- SQL Tuning more often than not yields the greatest benefits. Ensuring an optimal execution plan, making use of shared SQL and bind variables, minimizing parse failures, and minimizing full table scans all contribute greatly to scalability in a single instance environment.
- Sequence caching yields benefits in a single instance environment. Since sequence number generation is a point of serialization, sequence caching benefits are magnified in a RAC environment.
- Partitioning large objects has an obvious benefit in terms of manageability and potentially with performance as well. This will have magnified benefits in a RAC environment for DML intensive objects.
- Tune Instance Recovery (`FAST_START_MTTR_TARGET`) to balance performance vs. availability
- Avoid DDL as this can create contention issues in the data dictionary that get magnified in a RAC environment.

There are no special application design or coding considerations required for RAC. All applications that run well in a single instance environment should run well on RAC, and all applications that scale well on SMP should scale well on RAC.

There are no special application design or coding considerations required for RAC. All applications that run well in a single instance environment should run well on RAC, and all applications that scale well on SMP should scale well on RAC. This actually highlights two of the biggest challenges that have been encountered in deploying applications on RAC:

- The application has never before been run or thoroughly tested on single instance, so performance problems observed in the RAC environment are automatically attributed to RAC.

- The application doesn't scale well on SMP, so scalability observed in RAC is very poor.

Experience has shown that it actually makes sense to deploy applications first to single instance, and then to RAC. There are a number of optimizer changes in 9i, as well as numerous feature enhancements and performance improvements that can change application behavior from previous releases. If you take the time to get a handle on how your application behaves first in single instance, and take steps to optimize it there first, you will save yourself a great deal of time and aggravation, when you turn on RAC. Likewise, making an effort to determine how your application will scale on SMP, where those scalability bottlenecks occur, and where contention or hot spots are in a single instance, will go a long way towards helping understand how it should behave when deployed on RAC. In a few cases, we have seen some performance/scalability improvements by implementing some kind of workload partitioning, since arbitrarily balancing workload created artificial, or self-induced contention that is easily avoided. And in a couple of cases, primarily in configurations involving a large number of CPUs, we have seen RAC actually scale better than SMP.

OPERATIONS

Operationally, managing a RAC database is essentially the same as managing a single instance, with a few, mostly mechanical differences.

Operationally, managing a RAC database is essentially the same as managing a single instance, with a few, mostly mechanical differences. Starting and stopping instances using the srvctl and gsdctl management infrastructure provided with RAC, managing mulitple threads of redo, and monitoring RAC traffic across the interconnect are really the only differences. We still see people raise some initial concerns around the managability of the RAC environment, but once they understand the few differences from the single instance environment most are already familiar with, RAC is simply an incremental step in the Oracle knowledge base.

Production Migration

The majority of successful deployments are a direct result of adherence to strong systems development and deployment disciplines.

Planning and successfully executing the final deployment of the application, and the transition into an operational phase transcends the technology itself. The majority of successful deployments are a direct result of adherence to strong systems development and deployment disciplines, and the vast majority of problems at production rollout and post-production are a result of lapses in these processes and procedures. Structured test plans, detailed and rehearsed production cutover plans, tight system and application change control procedures, standards and systems integrity disciplines, detailed and rehearsed backup and recovery procedures, comprehensive security and auditing plans, etc. are all key components in a successful final production rollout.

These disciplines have been around for years, and in most cases form the operational backbone in larger IT shops – particularly those that have legacy mainframe environments. Large Unix shops now generally have reasonably well established processes, largely adapted from mainframe environments, although historically, virtually anything that was Unix-based was, by definition, undisciplined. Smaller shops have not had the benefit of the same experience, and of course tend to have fewer resources to establish the same level of structure and discipline.

Due to the nature of RAC deployments – largely into mission critical-type environments, adherence to these systems disciplines is truly a critical success factor. Recognizing that not all organizations have the resources necessary to address all these issues in a comprehensive manner, there are 3 items that have proven to be the most critical in customer environments to date:

- Separate environments for Development, Test, QA/UAT, and Production. It is critical that test systems closely mirror production systems in that they maintain a consistent set of patches to match what is deployed in production.

- System-level change control. Changes should only be applied to one system element at a time. This allows for speedy problem identification and reversal should problems occur. Changes should always be applied first to test systems and impact should be well understood prior to implementing in production.
- Application change control. This is particularly important for e-commerce companies whose business is essentially their website. To be competitive frequent changes to the website application must be made. Nevertheless, it is critical that these changes be tracked. An untracked change should never be introduced. If failures occur it is critical to be able to quickly and accurately identify the most recent series of changes that were made, and be able to back them out if necessary. In addition, it is critical that the impact of these application changes on underlying system components be monitored.

Without these key issues addressed, even simple configuration changes or patch applications to any of the technology stack run the risk of creating a cluster-wide outage and virutally eliminating the benefit of having a fully redundant processing environment.

Backup and Recovery (B&R)

RMAN makes backup and recovery easy for RAC.

The only real complexity associated with B&R that is specific to RAC vs. single instance is the fact that there are mulitple threads of redo, and that the redo logs and archive logs from all instances must be backed up to ensure full recoverability. RMAN is the easiest mechanism for doing this. It has been designed to be RAC aware, and understands these minor differences. The configuration of RMAN to manage a RAC environment is well documented in the Backup and Recovery documentation. The process is further simplified under Linux if you are using OCFS.

Database Monitoring and Tuning

There is very little to monitor and tune that is specific to RAC. Never ignore the basics of system monitoring.

Database monitoring and tuning is an entire paper unto itself. The most important observation is that there is very little to monitor and tune that is specific to RAC. Never ignore the basics of system monitoring. Long periods of not monitoring will inevitably lead to potentially serious problems.

- Monitor CPU utilization, it should never be 100%
- Monitor disk I/O service times, never over acceptable thresholds
- Monitor run queue to ensure it is at optimal level.

The following Linux OS monitoring tools will inevitably prove useful:

- Overall tools *sar, vmstat*
- CPU */proc/cpuinfo, mpstat, top*
- Memory */proc/meminfo, /proc/slabinfo, free*
- Disk I/O *iostat*
- Network */proc/net/dev, netstat, mii-tool*
- Kernel Version and Release *cat /proc/version*
- Types of I/O Cards *lspci -vv*
- Kernel Modules Loaded *lsmod, cat /proc/modules*
- List all PCI devices (HW) *lspci –v*
- Startup changes */etc/sysctl.conf, /etc/rc.local*
- Kernel messages */var/log/messages, /var/log/dmesg*
- OS error codes */usr/src/linux/include/asm/errno.h*
- OS calls */usr/sbin/strace-p*

Additionally, familiarity with single instance performance monitoring and tuning is essential:

- Identify/tune contention using v$segment_statistics to identify objects involved
- Concentrate on bad SQL if CPU bound
- Tune I/O – if the storage subsystem is saturated, adding more nodes/instances won't help

There are a few key techniques that are important in the context of successfully deploying an application on RAC:

- Maintain a balanced load on underlying systems (e.g, database, operating system, storage system). This is not only important for optimal performance, but can become critical for reliability.
    - Excessive load on individual components, e.g., a single node in a cluster, or an individual I/O subsystem, can invoke aberrant behavior and failures that would not normally be experienced.
    - Avoid any load that goes beyond the red line.
    - Periodically address imbalances thru constant monitoring.

- Make use of statspack reports – there is a wealth of information in them
    - Use a separate large tablespace for statspack reports
    - Automatic snapshots every 10-20 mins during stress testing, every hour during normal operations
    - Run statspack on all instances, staggered a couple of minutes apart
    - Concentrate on the top 5 Statspack wait events if majority of time is spent waiting
- Use scripts and tracing
    - Statspack does not currently provide the exact details of which blocks or segments are incurring contention.
    - A simple way to check which blocks are involved in wait events is to monitor V$session_wait, and use that information to trace to the corresponding segments.
    - Trace events, like 10046/8, can provide additional details on wait events
    - Alert logs and trace files should be monitored for events, as on single instance.

CONCLUSION

The above discussion encompasses observations and best practices that have been derived from experiences at over 500 different customers currently running their applications in production on RAC. The promise of Real Application Clusters is REAL, and allows you to take an important step towards Grid computing.

Oracle Real Application Clusters is a breakthrough in cluster database technology, and a key enabler of the Grid. Grid computing is poised to change the economics of computing. Grid computing enables efficient utilization of enterprise resources and thus drastically reduces enterprise computing costs. Fortune 100 companies have already started reaping Grid benefits, by using Oracle technology that is available today. Oracle is committed to supporting Grid standards and making it easier for you to utilize the Grid, and product directions are aligned with the Grid.

The above discussion encompasses observations and best practices that have been derived from experiences at over 500 different customers currently running their applications in production on RAC. It is worth noting that of these 500+ customers, the majority are also referencable. Although no piece of software is perfect, the promise of Real Application Clusters is REAL, and allows you to take an important step towards Grid computing. Following these best practices will enable you run your applications successfully on Oracle Real Application Clusters on Linux.

REFERENCES

Note.241114.1 Step-By-Step Installation of RAC on Linux - Single Node (Oracle9i with OCFS)
Note.240575.1 RAC on Linux Best Practices
Note.184821.1 Step-By-Step Installation of 9.2.0.4 RAC on Linux

Oracle RAC Best Practices on Linux
November 2003
Author: Kirk McGowan, Roland Knapp

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software that powers the internet.