Flexible Database Clusters with Oracle9i RAC
Fully Exploiting the Power of RAC

Technical Findings: A 16-node Intel-based cluster running Linux

An IBM-PolyServe White Paper

Abstract

Oracle9*i* RAC is the cornerstone for building flexible, high performance, highly-available, clustered database solutions on Linux. This paper features a case study of three different applications running on a 16-node cluster. While a cluster of this magnitude comprised of modern Intel-based servers may rival the established UNIX server, is it manageable? How does it affect TCO? These are the questions that trouble IT administrators. This paper addresses these issues by sharing the experience, tips and techniques, and lessons learned from the case study.

The case study gave special focus to issues of load balance, monitoring methodology, and minimizing single points of failure, as well as to the unique ability to shift servers dynamically among applications. The case study also included several forced server failures during peak load.

Availability is a key attribute of this system architecture; hence, we will share the measured performance impact at the application level, as well as recovery times experienced during the case study.

Introduction

Since the release of Oracle9*i* Real Application Clusters (RAC) during Oracle Open World Europe in June 2001, Oracle Corporation has made astounding advancements on a great architecture.

Countless benchmarks and customer testimonies have documented the capability of RAC to scale a single application horizontally to clusters as large as 8 or 10 nodes—without application modification! This proven trend of scalability puts RAC in a class of its own in the clustered database world.

Another trend has been quickly gaining momentum. That trend is Server Consolidation with large, flexible Intel-based clusters running Linux under Oracle9*i* RAC. Emerging clustered architectures such as powerful 1U rack-mounted servers and blade servers lend themselves to configuring large clusters in reasonable physical space at low cost. Connecting such clusters to a fault-resilient FibreChannel SAN lays the basic foundation for the computing infrastructure known in this paper as Flexible Database Clusters (FDC).

Flexible Database Clusters are the perfect combination of Server Consolidation and Oracle9*i* RAC. Compared to several small and separate clusters each running RAC, a large FDC offers management, availability, and performance characteristics that are difficult to ignore.

Significant testing and proof of concept was required to further the FDC concept. To that end, IBM and PolyServe joined forces to build a 16-node cluster running SuSE Linux and then attached it to a formidable SAN configured with 206 physical disk drives. This large cluster was the target of a series of tests that took an in-depth look at running and managing not just a single application, but three separate applications.

Unlike most RAC studies to date, this testing was much more than the typical exercise of measuring throughput for a single application as nodes are added to the cluster. While those studies are valuable proof points, they lack information about the management aspects of large clusters, what happens when things go "wrong," and how applications can benefit from dynamically shifting servers from under-utilized applications to applications where they are more useful. To that end, the tests included powering off servers while running large numbers of connected Oracle users and dynamically shifting servers from one application to another. Critical measurements such as recovery times and throughput impact were also analyzed. The tests and measurements are described in detail later in this paper.

The paper also includes performance information, but in a slightly different light than the norm. The FDC serves as a platform on which nodes can be shifted between applications without halting service to clients! Therefore, tests were conducted to measure speedup to applications that were granted additional nodes dynamically.

This case study provided invaluable lessons, tips and techniques that are described later in this paper.

Flexible Database Cluster at a Glance

The concept of a Flexible Database Cluster is based on building a sufficiently large cluster to support several databases. Instead of using one cluster for a "PROD" database and two other clusters for "DSS" and "DEV," a Flexible Database Cluster can support all three of these databases.

For many years large SMP systems have supported more than one Oracle database in what is routinely referred to as *Server Consolidation*—taking the workloads from several small SMPs and concentrating them into one large SMP. A trend started in the late 1990s, Server Consolidation was seen as a way to increase efficiency in the datacenter by reducing administrative overhead.

If consolidation adds value in an SMP environment, why wouldn't it add value in a cluster environment? The answer is not simply that it does add value, but in fact consolidation delivers more value in clustered environments than in SMP environments, particularly when supporting Oracle9*i* RAC!

A report provided by Meta Group[1] in 2002 makes the point that the "Hard Partitions" technology brought to market by Sun and Sequent has been recognized

[1] See the Meta Group report on ZDNet at:
http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2878371-1,00.html

as valuable in consolidation. Hard Partitions offer fixed system provisioning to given applications within a single SMP. With these systems, administrators can dedicate, say, 16 processors to application A and another 16 processors to application B. The report also focuses on the value of dynamic partitioning (e.g., LPAR, VPAR) and suggests that it is critical for supporting consolidation. That is, flexibility is key.

These SMP-oriented Server Consolidation systems support the ability to dedicate CPUs to important workloads. Likewise, less important workloads (e.g., untested applications still under development) are cordoned off to other CPUs to limit what perturbation they might cause to other applications. Arguably, clusters do this even better than SMPs with domains or partitions. After all, with SMP systems, there are usually many system components such as memory controllers, I/O adapters and so on that are shared among partitions or domains. Clusters, however, confirm that an application running on nodes 1 and 2 does not share server-level resources with applications on nodes 3 and 4. The SAN is shared, but partitionable.

So, Server Consolidation is a good thing and, because of architectural differences, clusters perform even better than single SMPs. Why then are so many IT organizations supporting a small dedicated cluster for each application? If several different clusters all support Oracle9*i* RAC based applications, why not consolidate?

The Meta Group report of 2002, along with many others, further credits Server Consolidation as enabling IT savings through reductions in the cost of support and skills, capital equipment, and system management. Fewer systems, less management.

If consolidating simple, single servers into one large server yields IT savings, how much more so does consolidating complex clusters into one large cluster?

The Flexible Database Cluster concept lowers administrative overhead and offers higher availability and on-demand scalability beyond that of several small clusters. It is an architecture well worth consideration. A Flexible Database Cluster is more than just a big cluster. The prime ingredients are systems software and deployment methodology.

Large Database Clusters with Oracle9*i* RAC—At a Glance

The thought of assembling a 16-node cluster, much less managing one, conjures up visions of cabling and OS configuration nightmares. This mentality is likely rooted in the UNIX-based clustered systems of the 1990s. Clusters of that era were configured with very few nodes primarily due to limitations in Oracle Parallel Server.

Oracle9*i* RAC has changed all of that. The scalability and availability characteristics of RAC are compelling reasons to build large clusters. The economic benefit of combining powerful Intel-based servers running Linux into a large cluster makes the idea even more palatable. The question is why make a large database cluster, and what management considerations are there?

The answer hinges on the technology that is being assembled into the cluster.

The base requirement for an Oracle9*i* RAC cluster is a set of servers with shared disk (e.g., JBOD) access and LAN/interconnect connectivity. Strictly speaking, nothing more is required. For reasons explored throughout this paper, it is unlikely that such a stripped down cluster will be as manageable at a large node count. Nor would such a cluster be configured to offer the management and flexibility attributes of the Flexible Database Cluster (FDC) model studied during this project.

Concerns over building and maintaining large clusters for Oracle9*i* RAC generally fall into six general categories, although not necessarily in this order:

- Storage Configuration/Management
- Storage Connectivity
- OS Configuration/Management
- Oracle Product Installation, Configuration and Maintenance
- Database File Locations and Space Management
- Operational Methodology

Storage Configuration/Management

Today, building large clusters such as the Flexible Database Cluster in this proof of concept is actually quite simple, due in part to FibreChannel SAN technology. Even simpler are large clusters with the emerging "blade server" technology.

Storage management is much simpler and more powerful with today's technology. A great example is the FastT intelligent storage array from IBM, which was configured with 206 disk drives for the FDC proof of concept. This storage subsystem completely handles RAID issues and reduces administrative overhead dramatically. Simplicity is critical in a large cluster environment; therefore, virtualizing a large disk farm at the storage level is key. In the FDC test, all 206 disks were virtualized into just a few large LUNs configured with RAID 1+0. From a manageability perspective, the most important factor is that the RAID characteristics are maintained by the storage array. This is much simpler than the administrative overhead of configuring and managing software RAID.

Storage management at the OS level in a cluster can become problematic as well. In Linux, it is possible for one node in the cluster to attribute a device path of */dev/sdc* to a drive that is known as */dev/sdf* on another node. This is called "device slippage" and occurs because of a potentially different boot-time probing order of devices. Concerns of this sort are not an issue when using PolyServe Matrix Server. All disk devices (LUNS) are imported and given a cluster-global name and are tracked based upon their unique device ID (World Wide Name). The Matrix Server software was important in the proof of concept because a significant portion of the testing included powering off nodes simultaneously throughout the cluster. It would have been quite difficult to script the validation of all */dev* entries for the disk farm to ensure parity of logical to physical mappings among nodes of the cluster.

Storage Connectivity

Modern technology is also making it easier to configure SAN switches. In the FDC proof-of-concept system, the switch was easily configured via the IBM TotalStorage SAN FibreChannel Switch Specialist management tool. The switch features an embedded Web browser interface for configuration, management, and troubleshooting.

OS Configuration/Management

Configuring Linux to support a large cluster is much simpler than legacy UNIX. Enhanced configuration tool kits are available with Linux distributions such as SuSE, which provides the KDE Konsole[2] utility. This utility feeds key input to all or some nodes of a cluster, which is useful for redundant administrative tasks. This is but one example of the value add SuSE has to offer.

SuSE also provides a helpful package in RPM form called **orarun**. This package assists administrators with node-level RAC requirements such as environment variables, kernel parameters, and automated startup/shutdown of key Oracle components such as OCMS, SQL*Net listeners and GSD (global services daemon).

Every small improvement that makes a cluster feel more like a single system is critically important, even more so in the case of the Flexible Database Cluster.

Oracle Product Installation, Configuration and Maintenance

On standard clusters, the Oracle9*i* Enterprise Edition software must be installed on each node where instances of RAC will be executed. During installation, the Oracle Universal Installer prompts for a list of nodes upon which to install the product. It then copies the files for Oracle Home, all 100,000+ of them, to each node in the list.

Installing Oracle is not difficult. The difficulty arises when Oracle is installed on, say, 16 nodes[3]. Configuration must occur in 16 different locations, one for each node. For instance, configuring *init.ora* requires logging in to 16 different systems. Applying patches is difficult as well. For example, the 9.2.0.3 upgrade patch for Linux is over 280 MB in size. This is a lot of software to apply to a lot of Oracle Home locations.

With a general-purpose cluster filesystem[4], it is quite simple to set up a “Shared Oracle Home.” First, the Oracle Universal Installer is instructed to install on only one node (a single-node cluster install). Then, after Oracle is fully installed on the

[2] For information about using KDE Konsole when configuring large clusters for Oracle9*i* RAC, refer to this paper on the SuSE website: http://www.suse.com/en/business/certifications/certified_software/oracle/docs/9iR2_RAC_sles8_polyserve.pdf

[3] The Oracle Universal Installer provides for a list of eight nodes for installing Oracle, while the maximum supported node count on Linux is 32. Installing Oracle on a 16-node cluster requires manual file propagation if a shared Oracle Home is not available.

[4] Such as the PolyServe Matrix Server CFS used in this test. For more information, visit www.polyserve.com.

cluster filesystem as a single-node install, it is very simple to "convert" that Oracle Home to be shared by any number of nodes.

Converting to a shared Oracle Home is merely the concept of providing for like-name directories and files with node-specific directories and files. For example, it may be advantageous to have the directory *$ORACLE_HOME/network/admin* set up as a different physical directory when you log into node 1 than on node 2, yet this indirection needs to be automagic. The PolyServe Matrix Server cluster filesystem provides Context Dependent Symbolic Links (CDSL) to do this. Only a few objects require a CDSL so this is really quite simple.

With a shared Oracle Home, all nodes in the cluster can use the same executables. Also, configuration files are located in the cluster filesystem and can be edited from any node in the cluster. In fact, all configuration files for all nodes can be edited from any node in the cluster. $ORACLE_HOME points to the same directory on all nodes, and that directory is in the cluster filesystem.

Database File Locations and Space Management

Managing a great number of Oracle datafiles can become difficult when several databases are housed in one large cluster, such as those in the FDC proof of concept. Perhaps the most significant technology to assist in this area is a new Oracle9*i* feature known as Oracle Managed Files (OMF). Using OMF with Oracle9*i* RAC requires a cluster filesystem such as the PolyServe Matrix Server cluster filesystem.

Oracle Managed Files are preferred in a large FDC because they simplify administration. OMF[5] provides the following benefits:

- Simplifies deployment of databases. OMF minimizes the time spent making decisions regarding file structure and naming and reduces file management tasks overall.
- Reduces corruption that can be caused by administrators specifying the wrong file. All OMF files are provided a unique name by Oracle.
- Reduces wasted disk space consumed by obsolete files. OMF removes unused files automatically.

Forcing a Database Administrator to think about tablespaces as a collection of files takes time away from actual database administration. The "contents" of the files are the actual database. Databases consist of rows in blocks grouped into extents and tracked as segments. Those objects reside in datafiles. Deploying with OMF reduces the need for physical administration and allows concentration on logical administration—exactly where the biggest return on DBA effort lies. For instance, why fuss over what sectors of disk a filesystem uses? That particular detail is the responsibility of the Operating System. Likewise, Database Administrators can relinquish datafile management to OMF. More time can be dedicated to query optimization and other such tasks that actually improve performance.

[5] For in-depth information about Oracle Managed Files, please refer to the Oracle documentation.

Examples showing how the FDC case study used OMF are provided later in this paper.

Operational Methodology

Monitoring is perhaps the most grave concern in a large cluster environment. If administrators must execute numerous commands to get the status of many nodes, the value proposition quickly dwindles.

Monitoring cluster operations with the type of technology used in the Flexible Database Cluster proof of concept is vastly superior to the technology of the recent past. Comprehensive I/O monitoring at the storage level is possible through modern storage management software. Oracle9*i* RAC and Enterprise Manager offer a great deal of instance and global database monitoring. PolyServe Matrix Server offers monitoring of the entire cluster from a central console that can be executed either remotely or on any system in the datacenter, and finally, PolyServe's implementation of the Oracle Disk Manager offers unprecedented Oracle-centric cluster-aware I/O monitoring.

Beyond monitoring, other issues have traditionally affected cluster operations. For instance, administrators have had to connect to specific nodes in the cluster to perform operations such as configuring files in Oracle Home. Also, requiring administrators to remember what node they used to perform an export or what node has a SQL*Plus report on it can be bothersome in today's hectic IT environment.

Concerns such as these have served as roadblocks to deploying the sort of large cluster that can be used as a Flexible Database Cluster.

Infrastructure for Flexible Database Clusters

This section describes how the Flexible Database Cluster was set up for testing. The core technology for this aspect of the proof of concept was the PolyServe Matrix Server cluster filesystem. Two key Matrix Server product features were germane to this testing: the cluster filesystem and the MxODM I/O monitoring package.

Matrix Server Cluster Filesystem

The Matrix Server cluster filesystem is both general purpose and Oracle optimized. These attributes proved quite valuable in the following ways.

General Purpose:

- A single Shared Oracle Home was configured for all 16 nodes.
- Archived redo logging was performed into a cluster filesystem location and compressed.
- Some of the data was loaded with External Tables (an Oracle9*i* feature supported only on CFS with RAC).

Oracle Optimized:

- All Datafiles, Control Files, Online Logs, and so on were located in filesystems mounted with the Matrix Server "DBOPTIMIZED" mount option, which implements Direct I/O.
- Oracle Disk Manager[6] (ODM) was used for async I/O and improved clusterwide I/O monitoring.

Shared Oracle Home

As described earlier in this paper, a general purpose cluster filesystem such as the Matrix Server CFS supports setting up a single directory for Oracle Home. This functionality is key to the Flexible Database Cluster architecture.

Context Dependent Symbolic Links

Context Dependent Symbolic Links (CDSL) are a feature of the Matrix Server CFS. They are links to files or directories that are resolved based on the hostname of the node on which the current process is executing. It is an essential feature in enabling a shared Oracle Home, and also offers ease of management for several areas such as setting up SQL*Net, archived redo logging, instance logging (e.g., alert logs and trace files) and so on.

CDSL—Oracle Cluster Management Services Setup

Using a shared Oracle Home and CDSLs made it simple to set up Oracle Cluster Management Services for 16 nodes on the FDC.

In Figure 1, the Oracle Cluster Management Services subdirectory (*oracm*) is actually a CDSL linking the directory *.oracm.${HOSTNAME}* to *oracm*. As such, logging in to, say, rac6 will automatically link *oracm* to *oracm.rac6*. It is easy, however, to modify the contents of any of the *oracm* CDSL directories because they are just simple directories.

[6] For in-depth information regarding Oracle Disk Manager, please refer to this white paper on the Oracle Technology Network:
http://otn.oracle.com/deploy/availability/pdf/odm_wp.pdf.

```
Linux
$ hostname
rac1
$ cd $ORACLE_HOME
$ ls -l oracm
lrwxrwxrwx    1 oracle   dba            17 2003-07-10 13:10 oracm -> .oracm.{HOSTNAME}
$ ls -ld .oracm*
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:10 .oracm.rac1
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac10
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac11
drwxr-xr-x    5 oracle   dba           120 2003-07-10 11:12 .oracm.rac12
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:11 .oracm.rac13
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:10 .oracm.rac14
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:11 .oracm.rac15
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:10 .oracm.rac16
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:11 .oracm.rac2
drwxr-xr-x    5 oracle   dba           120 2003-07-10 13:10 .oracm.rac3
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac4
drwxr-xr-x    5 oracle   dba           120 2003-07-10 11:12 .oracm.rac5
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:10 .oracm.rac6
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac7
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac8
drwxr-xr-x    5 oracle   dba           120 2003-07-10 12:11 .oracm.rac9
$
```

Figure 1

Figure 2 shows a shell process on rac1 that lists the contents of the *cmcfg.ora* file in its CDSL-resolved *$ORACLE_HOME/admin/cmcfg.ora* file. It also shows that, while executing on rac1, the *cmcfg.ora* file is visible via the physical directory path of *$ORACLE_HOME/.oracm.rac6/admin/cmcfg.ora*. Finally, using the **rsh**(1) command, the *cmcfg.ora* file for rac6 is listed as a simple path because the CDSL resolves it properly once the shell is spawned on rac6.

```
Linux
$ hostname
rac1
$ cd $ORACLE_HOME
$ cat oracm/admin/cmcfg.ora
HeartBeat=15000
ClusterName=TEST
PollInterval=1000
MissCount=210
ServicePort=9998
KernelModuleName=hangcheck-timer
PrivateNodeNames= int-rac1 int-rac2 int-rac3 int-rac4 int-rac5 int-rac6 int-rac7 int-rac8 int-rac9 int-r
ac10 int-rac11 int-rac12 int-rac13 int-rac14 int-rac15 int-rac16
PublicNodeNames= rac1 rac2 rac3 rac4 rac5 rac6 rac7 rac8 rac9 rac10 rac11 rac12 rac13 rac14 rac15 rac16
HostName=rac1
CmDiskFile=/mnt/ps/db/quorum
$
$ cat .oracm.rac6/admin/cmcfg.ora
HeartBeat=15000
ClusterName=TEST
PollInterval=1000
MissCount=210
ServicePort=9998
KernelModuleName=hangcheck-timer
PrivateNodeNames= int-rac1 int-rac2 int-rac3 int-rac4 int-rac5 int-rac6 int-rac7 int-rac8 int-rac9 int-r
ac10 int-rac11 int-rac12 int-rac13 int-rac14 int-rac15 int-rac16
PublicNodeNames= rac1 rac2 rac3 rac4 rac5 rac6 rac7 rac8 rac9 rac10 rac11 rac12 rac13 rac14 rac15 rac16
HostName=rac6
CmDiskFile=/mnt/ps/db/quorum
$
$ rsh rac6 cat oracm/admin/cmcfg.ora
HeartBeat=15000
ClusterName=TEST
PollInterval=1000
MissCount=210
ServicePort=9998
KernelModuleName=hangcheck-timer
PrivateNodeNames= int-rac1 int-rac2 int-rac3 int-rac4 int-rac5 int-rac6 int-rac7 int-rac8 int-rac9 int-r
ac10 int-rac11 int-rac12 int-rac13 int-rac14 int-rac15 int-rac16
PublicNodeNames= rac1 rac2 rac3 rac4 rac5 rac6 rac7 rac8 rac9 rac10 rac11 rac12 rac13 rac14 rac15 rac16
HostName=rac6
CmDiskFile=/mnt/ps/db/quorum
```

Figure 2

Using a shared Oracle Home and CDSLs reduces administrative overhead. The administrator can view and edit all of the configuration files for all nodes while logged in on any node in the cluster. Without a shared Oracle Home, the administrator would need to log in to each of the 16 nodes and edit the *cmcfg.ora* file to set up OCMS.

It is important point to note that if a given node name does not appear in the list assigned to the *PrivateNodeNames* and *PublicNodeNames* parameters in the *cmcfg.ora* file, it cannot join the Oracle cluster. That is, it cannot join until all instances of OCMS have been shut down—a task that requires all database instances clusterwide to be shut down. For example, if there is another node called rac17 in the cluster, it cannot dynamically join the Oracle RAC cluster (OCMS) until all database instances are shut down, all copies of the *cmcfg.ora* file edited to add rac17 in the *PrivateNodeNames*/ *PublicNodeNames* parameter assignments and then OCMS rebooted on each node.

This fact impacts sites that rely on a cold-standby server in the event of a node failure. In a four-node cluster scenario, losing node 4 and replacing it with the cold standby node means that *cmcfg.ora* needs to be configured in advance to accommodate its node name in the active cluster.

The FDC architecture is much more flexible. All nodes in the cluster are included in the OCMS member list. They all run Oracle9*i* RAC; the only thing that varies is what instances run on the various nodes.

CDSL—SQL*Net Setup

Here again, CDSLs and shared Oracle Home make administration quite simple. A CDSL was used to set up *listener.ora* files for the 16 SQL*Net **tnslsnr** processes.

Figure 3 shows that *listener.ora* in $TNS_ADMIN is a CDSL. While executing on rac13, a listing of *$TNS_ADMIN/listener.ora* shows specific detail for rac13. Of course the contents of *listener.ora* for rac9 are stored in the file *listener.ora.rac9*, but to show the functionality, the **rsh**(1) command is used to execute a shell on rac9 to list the contents of the file with the simple path name *$TNS_ADMIN/listener.ora*.

```
Linux
$ hostname
rac13
$ cd $TNS_ADMIN
$ ls -l list*ora*
lrwxrwxrwx    1 oracle   dba           23 2003-07-25 19:02 listener.ora -> list
ener.ora.{HOSTNAME}
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac1
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac10
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac11
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac12
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac13
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac14
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac15
-rw-r--r--    1 oracle   dba          356 2003-07-25 19:02 listener.ora.rac16
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac2
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac3
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac4
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac5
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac6
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac7
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac8
-rw-r--r--    1 oracle   dba          353 2003-07-25 19:02 listener.ora.rac9
$ cat listener.ora
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = rac13)(PORT = 1521))
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = prod)
      (ORACLE_HOME = /usr1/oracle)
      (SID_NAME = prod13)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = dss)
      (ORACLE_HOME = /usr1/oracle)
      (SID_NAME = dss13)
    )
  )
$ rsh rac9 "cat $TNS_ADMIN/listener.ora"
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = rac9)(PORT = 1521))
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = prod)
      (ORACLE_HOME = /usr1/oracle)
      (SID_NAME = prod9)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = dss)
```

Figure 3

The FDC also used a shared *tnsnames.ora* file. Figure 4 shows the definition of the PROD service. This service supports dynamically adding or removing servers without interruption to incoming clients. Remember that TNS_ADMIN points to the same directory on all 16 nodes because it is a shared Oracle Home. In this case TNS_ADMIN is set to *$ORACLE_HOME/config*.

```
Linux
rac1 oracle  $ cd $TNS_ADMIN
rac1 oracle  $ more tnsnames.ora
prod=
 (description=
  (load_balance=on)
  (failover=on)
  (ADDRESS_LIST =
  (address=  (protocol=tcp)  (host=rac1)  (port=1521))
  (address=  (protocol=tcp)  (host=rac2)  (port=1521))
  (address=  (protocol=tcp)  (host=rac3)  (port=1521))
  (address=  (protocol=tcp)  (host=rac4)  (port=1521))
  (address=  (protocol=tcp)  (host=rac5)  (port=1521))
  (address=  (protocol=tcp)  (host=rac6)  (port=1521))
  (address=  (protocol=tcp)  (host=rac7)  (port=1521))
  (address=  (protocol=tcp)  (host=rac8)  (port=1521))
  (address=  (protocol=tcp)  (host=rac9)  (port=1521))
  (address=  (protocol=tcp)  (host=rac10)  (port=1521))
  (address=  (protocol=tcp)  (host=rac11)  (port=1521))
  (address=  (protocol=tcp)  (host=rac12)  (port=1521))
  (address=  (protocol=tcp)  (host=rac13)  (port=1521))
  (address=  (protocol=tcp)  (host=rac14)  (port=1521))
  (address=  (protocol=tcp)  (host=rac15)  (port=1521))
  (address=  (protocol=tcp)  (host=rac16)  (port=1521))
  )
  (connect_data=
     (service_name=prod)
     (failover_mode=
       (type=select)
       (method=preconnect))))
```

Figure 4

SRVCTL for Flexibility

The **srvctl** utility is a helpful tool for starting database instances and querying their status. Although **sqlplus** can be used to start an instance, the administrator must be executing on the specific node where the instance will be started. Some administrators use scripts with **rsh**(1) or start instances during system bootup via **rc** scripts. While such scripts may be adequate for small cluster environments, the Flexible Database Architecture draws upon the features of **srvctl**. With **srvctl**, instances can be started on any node from any node, which better fits the FDC model.

Figure 5 shows how the FDC was configured to use a cluster filesystem location for the *srvconfig* file. Also shown is the quick step-by-step command to configure support for all 16 instances of the PROD database. Finally, Figure 5 shows a clusterwide startup of all 16 instances.

```
Linux
rac1 oracle  $ ls -l $SRVM_SHARED_CONFIG
-rwxrwxrwx    1 oracle   dba      209715200 2003-07-15 17:43 /mnt/ps/oradata1/srvconf
rac1 oracle  $ srvconfig -init -f
rac1 oracle  $ gsdctl start
Successfully started GSD on local node
rac1 oracle  $ srvctl add database -d prod -o $ORACLE_HOME
rac1 oracle  $
rac1 oracle  $ for node in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
> do
> srvctl add instance -d prod -i prod${node} -n rac${node}
> done
rac1 oracle  $ time srvctl start  database -d prod 2>/dev/null
  144.96s real     1.81s user     1.00s system
rac1 oracle  $
rac1 oracle  $ srvctl status database -d prod
Instance prod1 is running on node rac1
Instance prod2 is running on node rac2
Instance prod3 is running on node rac3
Instance prod4 is running on node rac4
Instance prod5 is running on node rac5
Instance prod6 is running on node rac6
Instance prod7 is running on node rac7
Instance prod8 is running on node rac8
Instance prod9 is running on node rac9
Instance prod10 is running on node rac10
Instance prod11 is running on node rac11
Instance prod12 is running on node rac12
Instance prod13 is running on node rac13
Instance prod14 is running on node rac14
Instance prod15 is running on node rac15
Instance prod16 is running on node rac16
rac1 oracle  $
```

Figure 5

Although the test plans called for only 12-node support for OLTP, the FDC infrastructure provides for instances of any database starting on any node. To that end, OCMS and **srvctl** need to be set up accordingly as was done in this example. The same methodology was also used for the DSS database under test.

When instances are booted with **srvctl**, it searches the *$ORACLE_HOME/dbs* directory for *init.ora* and expects to find the file *init${ORACLE_SID}.ora*. If a shared Oracle Home is used, the *dbs* directory is, of course, a single location. Instead of making the *dbs* directory a CDSL itself, the preferred approach is to use links. In this case, however, not Context Dependent Symbolic Links but simple symbolic links. (CDSLs rely on hostname context for resolution, not instance name.) Figure 6 shows how an *init.or*a called *initprod.ora* is linked with symbolic links bearing the various 16 instance names for the PROD database.

```
$ cd $ORACLE_HOME/dbs
$ ls -l initprod*
-rw-r--r--    1 oracle   dba         2879 2003-08-11 16:07 initprod.ora
-rw-r--r--    1 oracle   dba         2966 2003-07-14 14:06 initprod.ora.old
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod1.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod10.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod11.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod12.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod13.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod14.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod15.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod16.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod2.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod3.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod4.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod5.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod6.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod7.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod8.ora -> initprod.ora
lrwxrwxrwx    1 oracle   dba           12 2003-07-14 13:32 initprod9.ora -> initprod.ora
$
```

Figure 6

All of the symbolic links point to the same file because the *init.ora* processing feature that prefaces parameters with instances names was used. This is a very convenient feature. Although there are up to 16 instances accessing the PROD database, there is only one simple *init.ora* file to edit and, because it resides on the shared Oracle Home, it can be edited from any node in the cluster. (The Appendix contains a full listing of the *initprod.ora* file from the FCB proof of concept.)

A single *init.ora* in a single location for 16 instances is a great administrative help.

Shared Oracle Home—Improved Support for GSD (Global Services Daemon)

The GSD processes are used by **srvctl** and Enterprise Manager and occasionally require troubleshooting. Doing this on a 16-node cluster can be difficult without a shared Oracle Home. First, 16 copies of *$ORACLE_HOME/bin/gsd.sh* would need to be edited. Once tracing began, the administrator would need to monitor the *$ORACLE_HOME/srvm/log* directory for trace output.

Figure 7 shows how, with a shared Oracle Home, starting GSD tracing is a simple edit of one *gsd.sh* file. The resultant trace file for each instance is visible from any node since it is in the cluster filesystem. (GSD tracing is turned on by adding the –**DTRACING_ENABLED** and –**DTRACING_LEVEL** command-line options to the Java runtime.)

```
Linux
rac1 oracle  $ cd $ORACLE_HOME/bin
rac1 oracle  $ grep TRACING gsd.sh
exec $JRE -DTRACING.ENABLED=true -DTRACING.LEVEL=2 -DPROGRAM=gsd -classpath $CLA
SSPATH oracle.ops.mgmt.daemon.OPSMDaemon $MY_OHOME &
rac1 oracle  $ cd $ORACLE_HOME/srvm/log
rac1 oracle  $ ls -l
total 3516
-rw-r--r--    1 oracle   dba       155758 2003-07-21 15:53 gsdaemon_rac1.log
-rw-r--r--    1 oracle   dba       135582 2003-07-21 14:54 gsdaemon_rac10.log
-rw-r--r--    1 oracle   dba       135836 2003-07-21 14:54 gsdaemon_rac11.log
-rw-r--r--    1 oracle   dba       135806 2003-07-21 13:55 gsdaemon_rac12.log
-rw-r--r--    1 oracle   dba       135629 2003-07-21 15:54 gsdaemon_rac13.log
-rw-r--r--    1 oracle   dba       135818 2003-07-21 15:54 gsdaemon_rac14.log
-rw-r--r--    1 oracle   dba       135641 2003-07-21 15:54 gsdaemon_rac15.log
-rw-r--r--    1 oracle   dba       135886 2003-07-21 15:54 gsdaemon_rac16.log
-rw-r--r--    1 oracle   dba      1489137 2003-07-21 15:54 gsdaemon_rac2.log
-rw-r--r--    1 oracle   dba       135839 2003-07-21 15:53 gsdaemon_rac3.log
-rw-r--r--    1 oracle   dba       135638 2003-07-21 14:54 gsdaemon_rac4.log
-rw-r--r--    1 oracle   dba       135713 2003-07-21 13:55 gsdaemon_rac5.log
-rw-r--r--    1 oracle   dba       135685 2003-07-21 14:54 gsdaemon_rac6.log
-rw-r--r--    1 oracle   dba       135568 2003-07-21 14:54 gsdaemon_rac7.log
-rw-r--r--    1 oracle   dba       135840 2003-07-21 14:54 gsdaemon_rac8.log
-rw-r--r--    1 oracle   dba       135792 2003-07-21 14:54 gsdaemon_rac9.log
rac1 oracle  $
```

Figure 7

CDSL—Support for Archived Redo Logging

Using a cluster filesystem for archived redo logging is one of the most significant advantages of FDC architecture.

Having all threads of archived logging available to all nodes offers improved availability. In the case of a database recovery scenario, any node in the cluster can replay the logs because they are stored in a shared location. Also, because the PolyServe Matrix Server cluster filesystem is general purpose, standard filesystem tools can be used to compress or move the logs.

Figure 8 depicts the convenience of using a cluster filesystem as an archived redo log destination. This is, in fact, the exact set of simple steps taken during the proof of concept to ready a 16-instance database for archived redo logging. In stark contrast, without a general purpose cluster filesystem, the *log_archive_dest* must be located on internal disk instead of the SAN.

Also, notice in Figure 8 that on rac1 the archived redo logs for instance 7 are visible by changing directories to the CDSL-linked directory *arch.rac7*.

```
$ hostname
rac1
$ pwd
/mnt/ps/oradata7
$ echo $ALL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$ for node in $ALL
> do
> mkdir arch.rac$node
> done
$ cd $ORACLE_HOME
$ ln -s /mnt/ps/oradata7/arch.{HOSTNAME} arch
$ # Comment: Place database in archivelog mode and run some OLTP
$ cd $ORACLE_HOME/dbs
$ grep archive initprod.ora
log_archive_start = TRUE
log_archive_dest = /usr1/oracle/arch
$ cd /usr1/oracle/arch
$ ls
12_61.dbf  14_26.dbf  16_26.dbf  1_478.dbf  1_481.dbf  5_90.dbf  6_89.dbf
13_28.dbf  14_27.dbf  1_476.dbf  1_479.dbf  1_482.dbf  5_91.dbf  7_69.dbf
13_31.dbf  15_26.dbf  1_477.dbf  1_480.dbf  1_483.dbf  5_93.dbf  8_68.dbf
$ rsh rac7 "ls -C $ORACLE_HOME/arch"
13_42.dbf  14_49.dbf  16_34.dbf  5_117.dbf  7_74.dbf  7_78.dbf  7_82.dbf
14_29.dbf  15_36.dbf  16_45.dbf  7_71.dbf   7_75.dbf  7_79.dbf  7_83.dbf
14_31.dbf  15_47.dbf  16_54.dbf  7_72.dbf   7_76.dbf  7_80.dbf
14_38.dbf  15_56.dbf  5_108.dbf  7_73.dbf   7_77.dbf  7_81.dbf
$ cd /mnt/ps/oradata7/arch.rac7
$ ls -C
13_42.dbf  14_49.dbf  16_34.dbf  5_117.dbf  7_74.dbf  7_78.dbf  7_82.dbf
14_29.dbf  15_36.dbf  16_45.dbf  7_71.dbf   7_75.dbf  7_79.dbf  7_83.dbf
14_31.dbf  15_47.dbf  16_54.dbf  7_72.dbf   7_76.dbf  7_80.dbf
14_38.dbf  15_56.dbf  5_108.dbf  7_73.dbf   7_77.dbf  7_81.dbf

$
```

Figure 8

Any node in the cluster can service tasks such as compressing archive logs on behalf of the whole cluster. For best performance, it may prove beneficial to select a dedicated node in the cluster to process all compression of redo logs. This feature improves not only manageability, but availability.

All nodes in the cluster have direct access to the archived redo logs. As such, any node can perform total database recovery should the need arise. In contrast, without a general purpose cluster filesystem, the administrator must log into each node and push the archived redo logs via file transfer to the node that was performing database recovery. This method is not optimal when minutes count. Also, the private internal storage of a given server may not be accessible when database recovery is needed[7].

[7] It is acknowledged that non-CFS filesystems can be located in the SAN and used for private mounted archived redo logging. That approach has its pitfalls as well.

Matrix Server Oracle Disk Manager

PolyServe Matrix Server provides an ODM implementation called MxODM to support the Oracle Disk Manager interface. Although MxODM offers improved datafile surety through cluster-wide file keys for access, its main benefit in the FDC architecture is monitoring. MxODM also enables Oracle9*i* with asynchronous I/O on the direct I/O mounted filesystems where it stores datafiles and other database files such as redo logs.

The MxODM I/O statistics package provides the following basic I/O performance information. These reported items are referred to as the Core Reporting Elements.

- Number of File Read and Write Operations
- Read and Write throughput per second in Kilobytes
- Count of synchronous and asynchronous I/O operations
- I/O service times
- Percentages

The Core Reporting Elements can be provided at the following levels:

- **Cluster-Wide Level.** Provides aggregate information for all database instances on all nodes.
- **Database Global Level.** Limits information to a named database (e.g., PROD, DEV, FIN, DSS).
- **Instance Level.** Limits information to a named instance (e.g., PROD1, PROD8, DEV1, FIN4,DSS_6).
- **Node Level.** Limits information to a named node (e.g., rhas1.acme.com, rhas6.acme.com). This information is the aggregate of all instance activity on the named node. If a node hosts instances accessing different databases (e.g., $ORACLE_SID=PROD1, $ORACLE_SID=DEV1), the Core Reporting Elements will reflect the combined information for all instances on the named node.

Because MxODM understands Oracle file, process, and I/O types, the **mxodmstat**(8) command offers very specialized reporting capabilities. On complex clustered systems, it is nearly impossible to take a quick look at the cluster-wide or per-instance activity for a given subsystem of the Oracle Server.

For instance, on an 8-node cluster with six PROD instances, two DEV instances, and Parallel Query Slaves active only on nodes 1 through 4 on the PROD database, a DBA will find it extremely difficult to associate cluster-wide impact to the PQO activity. Likewise, quickly determining the DBWR activity for only the PROD instances on nodes 1 through 6 is nearly impossible—without MxODM.

MxODM offers “canned” reporting that focuses on the following key Oracle “subsystems.”

Parallel Query Option (PQO). This query returns the Core Reporting Elements for only the Parallel Query Option slaves (e.g., ora_p000_PROD1, ora_p001_PROD3, etc.). This is an extremely beneficial set of information as it allows DBAs to get a top-level view of the impact PQO is having on the cluster, either as a whole or at the node level.

Log Writer. This query focuses on only the **lgwr** processes and their activity at the cluster level, database level, or node level. Because all of the Core Reporting Elements can be returned in this query, it is beneficial for DBAs to maintain streaming output of this query showing **lgwr** activity at either the cluster level or broken down by database, instance, or node.

Database Writer. This query is of the utmost value. It too can return all Core Reporting Elements at all Reporting Levels; however, it can also limit reporting to only **dbwr** process activity. DBAs can glance at **mxodmstat**(8) output and easily determine the average **dbwr** I/O service times for all databases cluster-wide, or can focus on specific databases, nodes, or instances.

Although there is too much functionality in the **mxodmstat** package to describe in this paper[8], the following examples show some of the helpful monitoring used during the FDC project.

Monitoring three databases that have instances spread across a 16-node cluster is a daunting task. A bird's-eye view can be obtained with **mxodmstat**, as was done on the FDC test system. The first command in Figure 9 shows top-level I/O activity for all databases in aggregate broken out into reads and writes. The second command shows top-level I/O by database for all three databases. This is aggregate total I/O with breakout for the count of synchronous and asynchronous I/O as well as I/O service times. Note that without the **-a op** argument/option pair, the I/O is not broken out by reads and writes.

```
Linux
$ mxodmstat -i5 -a op
              Read                             Write
 Sync  Async     KB/s  Ave ms    Sync  Async     KB/s  Ave ms
 5018   6087  798799      14      16   2251   14933      15
 2448   2743  361485      17       7    981    6298      14
 1997   2114  279438      13       5    775    5067      12
 2048   2123  281085      12       5    782    5046      16
 1408   1786  233820      15       4    709    4802      17

$ mxodmstat -i5 -D
             dss                          dev                          prod
 Sync Async    KB/s Ave ms  Sync Async   KB/s Ave ms  Sync Async    KB/s Ave ms
    9  5939 740196     10    10   341   5267     51  3957  1410  36687     15
    3  2160 269139     10     7   192   2635     60  1701   553  15682     17
    3  2151 268134     10     6   168   2222     44  1966   552  17877     13
    3  2093 260989     10     3   321   3495     65  1932   528  17481     14
    4  2081 259287     10     3   182   2468     46  1890   544  17133     12
    3  2154 268570     10     4    93   1737     43  1923   535  17441     16
    3  2116 263581     10     4   186   2663     47  1958   626  18427     13

$
```

Figure 9

[8] For complete usage documentation for the PolyServe MxODM I/O monitoring package, please refer to the *MxODM User Guide*. For general information about MxODM, see www.polyserve.com.

The **mxodmstat** output in Figure 10 shows a flurry of DSS database activity.

```
Linux
$ mxodmstat -i10 -D prod dss dev
               prod                         dss                          dev
 Sync Async   KB/s Ave ms  Sync Async   KB/s Ave ms  Sync Async   KB/s Ave ms
 3160  2714  34339     15     4  0.40     66      1     6    38   1624      4
 2735  2775  33268     18     3  0.40     53      1     3    30   1364      5
 2566  2301  29048     17     3  0.30     51      1     3   228   2917     53
 2596  2119  27663     16     3  0.30     51      2     4   110   1808     53
 2295  2110  25998     15     3  0.36     47      1     3   398   4200     59
 2304  2085  25625     16     3  0.30     51      1     6    69   1469     38
 1978  1932  21946     22     3  0.30     51      1     3    28   1212      5
 2393  2541  29718     15     3   745  92698     12     4    28   1273      4
 2584  2176  28052     16     3  2111 263050     10     4   224   2796     59
 2604  2203  28431     16     3  2130 265467     10     3   142   2223     82
 2490  2262  28155     16     6  2102 262051     10     4   436   4529     66
 2537  2140  27485     16     3  2060 256738     10     6   117   1923     55
 2658  2277  29227     16     3  1618 201694     10     3   148   2340     54
 2581  2342  29259     15     3  0.40     48      1     4    82   5052     41
 2596  2284  28791     15     3  0.40     51      1     4   118   4972     44
 2565  2313  28975     16     3  0.20     54      1     3   404   4338     56

$
```

Figure 10

Figure 11 shows a drill-down command to see what processes are performing the I/O on the DSS database. The output shows that throughout all of the DSS instances, the I/O is almost entirely performed by Parallel Query Processes.

```
Linux
$ mxodmstat -i10 -D dss -s proc
                                       dss
   Background       DB Writer       Log Writer          PQO           Foreground
Sy  As KB/s  Ave  Sy  As KB/s  Ave  Sy  As KB/s  Ave  Sy  As  KB/s  Ave  Sy  As KB/s  Ave
 2   0   38    1   0   0    0    0   0 0.20   0    1   0    0      0    0   0   0    0    0
 3   0   51    2   0   0    0    0   0 0.30   0    8   0    0      0    0   0   0    0    0
 3   0   51    1   0   0    0    0   0 0.30   0    1   0    0      0    0   0   0    0    0
 3   0   50    5   0   0    0    0   0 0.40   0    6   0 1754 218363 10   0   0    0    0
 3   0   51    6   0   0    0    0   0 0.30   0    4   0 2113 263240 10   0   0    0    0
 3   0   48    6   0   0    0    0   0 0.30   0    8   0 2114 263435 10   0   0    0    0
 3   0   48    7   0   0    0    0   0 0.40   0    6   0 2132 265597 10   0   0    0    0
 3   0   54    6   0   0    0    0   0 0.20   0    5   0 2124 264634 10   0   0    0    0
 3   0   51    8   0   0    0    0   0 0.40   0    9   0 2106 262446 10   0   0    0    0
 4   0   61    7   0   0    0    0   0 0.50   0    6   0 2118 263824 10   0   0    0    0
 3   0   54    8   0   0    0    0   0 0.20   0    5   0 1726 215011 10   0   0    0    0
 3   0   53    1   0   0    0    0   0 0.40   0    1   0    0      0    0   0   0    0    0
 3   0   48    1   0   0    0    0   0 0.30   0    1   0    0      0    0   0   0    0    0
$
```

Figure 11

Figure 12 shows a focused monitoring of instances DSS1 and DSS2 only, broken out by reads and writes. Note that after some six lines of output (60 seconds), the query plan being serviced by DSS1 and DSS2 switched from 100% asynchronous large reads evenly by both instances to only DSS1 performing small synchronous reads along with just a few asynchronous writes.

```
Linux
$ mxodmstat -i10 -I dss1 dss2 -a op
                  dss1                                        dss2
        Read                  Write                   Read                  Write
Syn Asy  KB/s Ave  m Syn Asy  KB/s Ave  m Syn Asy  KB/s Ave  m Syn Asy  KB/s Ave  m
  1 1233 153846  10    0    0     0     0   1 1240 154440  10    0    0     0     0
  1 1046 130483  10    0    0     0     0   1 1050 130728  10    0    0     0     0
  1 1060 132339  10    0    0     0     0   1 1058 131645  10    0    0     0     0
  1 1049 130899  10 0.30 0.10     5     6   1 1052 131013  10 0.30 0.20     5     6
  1 1043 130131  10 0.40 0.30     7     7   1 1050 130677  10 0.40 0.20     6     9
  8 321 40171    10 0.30  7      60     2  13 322 40291     9 0.30 0.10     5     2
529   0  8464     2 0.30  4     324     9   1   0     21    1 0.30 0.20     5     1
669   0 10698     1 0.30  2     517     6   1   0     16    1 0.30 0.20     5     3
720   0 11522     1 0.30  3     697     6   1   0     21    1 0.30 0.20     5     1
$

$
```

Figure 12

Cluster Management

Matrix Server is more than a cluster filesystem; it is also clusterware and SAN management. Furthermore, Matrix Server offers multipath I/O, which is a pivotal component in eliminating single points of failure.

Figure 13 shows the Matrix Server console. This console provides a birds-eye view of cluster status. Aside from a command-line interface for cluster configuration, the console can be used with simple point and click to set up filesystems, mount options and advanced Hi-Av process and service monitoring for failover.

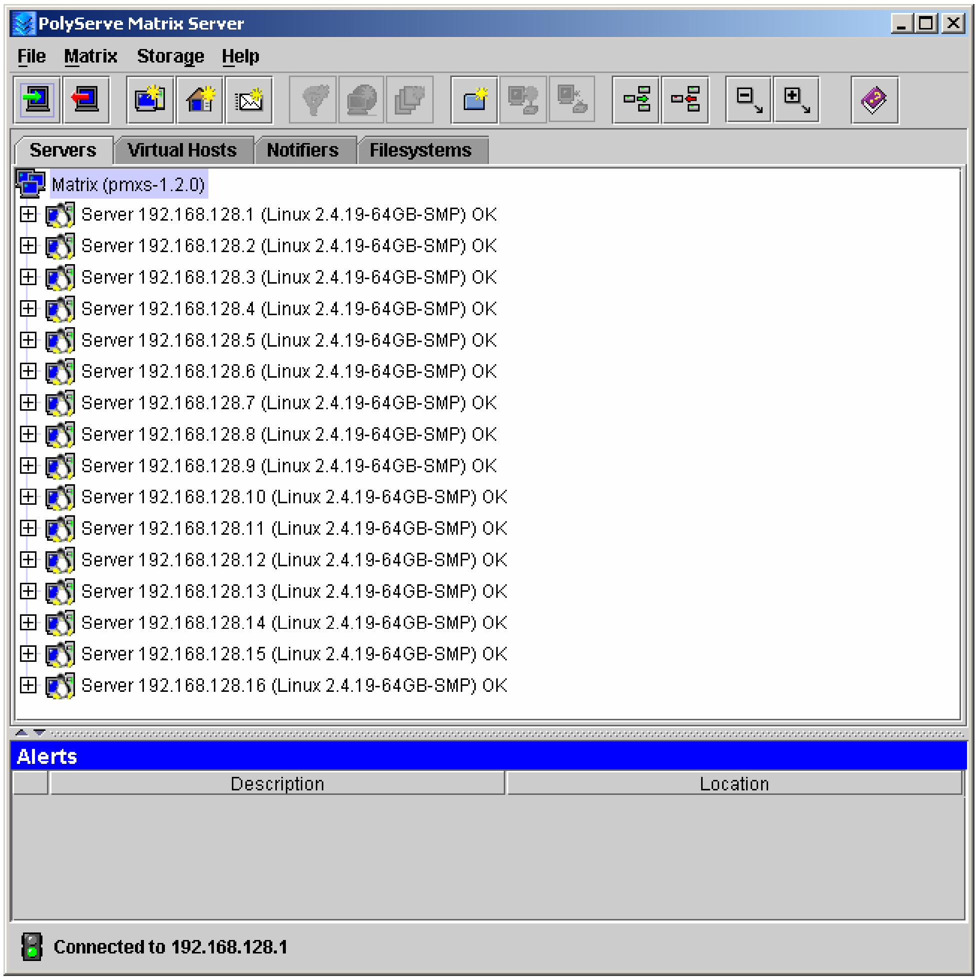

PolyServe Matrix Server
File Matrix Storage Help
Servers
Virtual Hosts
Notifiers
Filesystems
Matrix (pmxs-1.2.0)
Server 192.168.128.1 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.2 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.3 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.4 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.5 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.6 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.7 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.8 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.9 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.10 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.11 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.12 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.13 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.14 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.15 (Linux 2.4.19-64GB-SMP) OK
Server 192.168.128.16 (Linux 2.4.19-64GB-SMP) OK
Alerts
Description
Location
Connected to 192.168.128.1

Figure 13

Figure 14 presents another view of the Matrix Server console, showing drill-down capability for obtaining status of filesystems with a cluster-wide view.

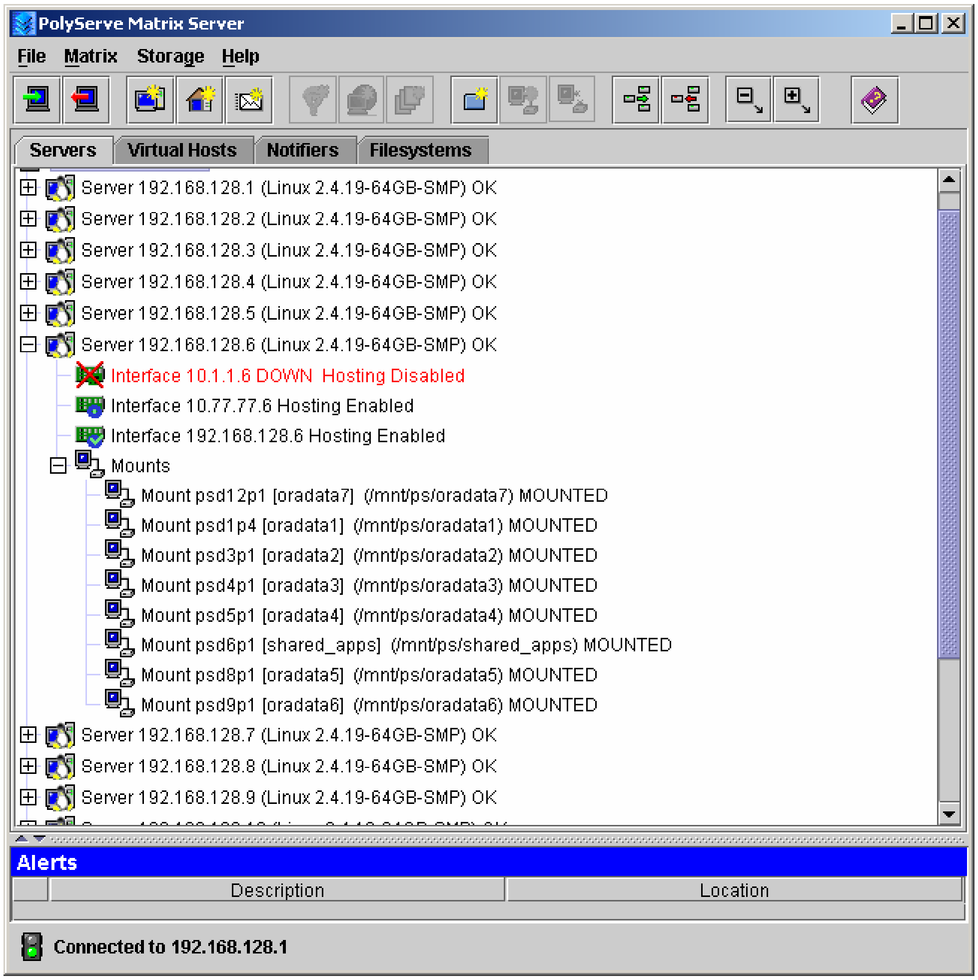
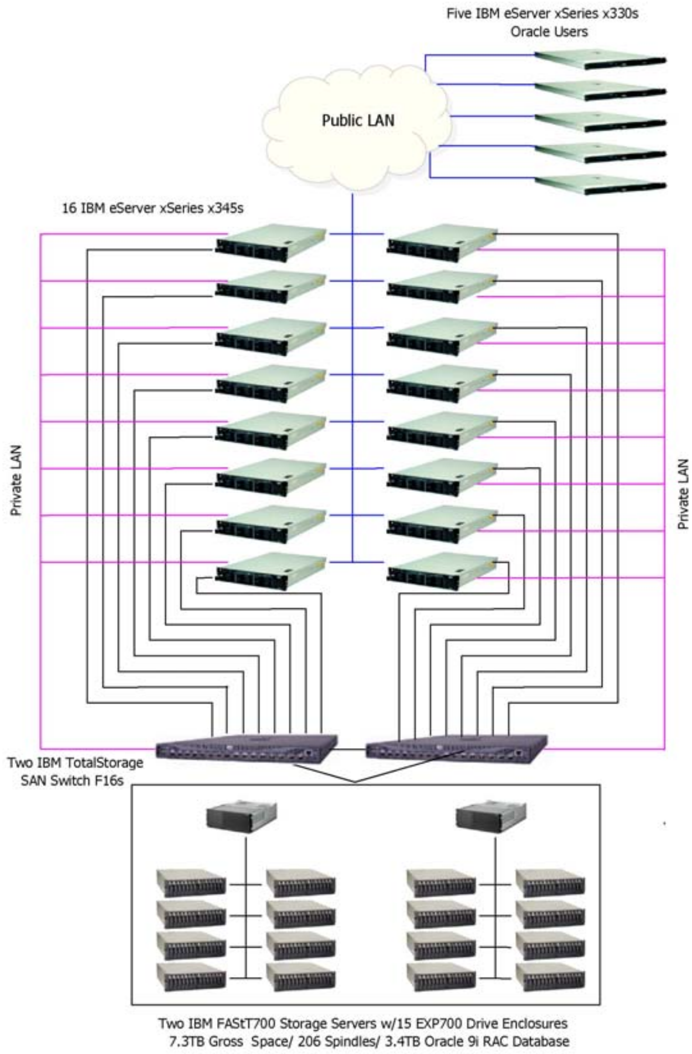


Figure 14

Proof of Concept

To create a suitable test system to prove Flexible Database Cluster architecture, it was key to use state-of-the-art and robust technologies. The following illustration shows the cluster system components used for the FCD proof of concept.



System Overview

The following sections describe the cluster system components.

Client Nodes

Five IBM xSeries 330 1U rack-optimized servers were used to drive the workload of the simulated Oracle users. The servers were configured with:

- Dual Intel 1.266Ghz Pentium III processors with 133MHz front-side bus speed
- 2 GB of SDRAM memory
- One EIDE 30GB Hard drive
- Integrated dual 100Mbps Ethernet
- Red Hat AS 2.1

Server Nodes

IBM's highly available 2U, 2-way application server, the eServer xSeries x345, was a key component in the 16-node cluster. The x345s were configured with:

- Dual Intel 2.4GHz Xeon processors with 533MHz front-side bus speed
- 2 GB of DDR memory
- One Ultra320 hot-swap 18-GB HDD
- Integrated dual Gigabit Ethernet for high-speed redundant network connectivity
- Two additional Intel Pro1000 XT NICs
- IBM TotalStorage FAStT FC2-133 Host Bus adapter
- SuSE Linux Enterprise Server 8 (SLES8)

A goal of this exercise was to show the high availability of the different components in the environment by reducing as many single points of failure as possible. The architecture of the x345 made it easy to incorporate high availability into the environment. One of the ways this was accomplished at the node level was by bonding the onboard gigabit interfaces. The dual onboard GigE interfaces were dedicated to the Oracle 9*i* RAC interconnect network. Bonding these interfaces provided fault tolerance capabilities for the Oracle interconnect.

The host bus adapter used was the IBM FAStT FC2-133 Adapter. This is a 2-GB high-performance, direct memory access (DMA), bus master, FibreChannel host adapter designed for high-end systems derived from the ISP2312 chip. The adapter is a half-length PCI-X adapter, which runs at 66MHz on a 64-bit PCI-X bus. With two PCI-X slots available in the system, dual adapters can be configured for high availability requirements. When the adapters are combined with host-level multipath I/O such as that offered with PolyServe Matrix Server, a SAN subsystem free of single points of failure can be built. The adapter auto-senses the data transfer rate of 1GB/s to 2GB/s, depending on the IBM storage server to which it is connected.

Storage

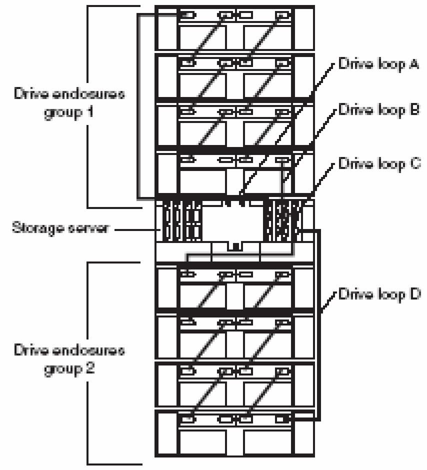
The storage subsystem for this environment consisted of two IBM TotalStorage FAStT700 Storage Servers and 15 FAStT EXP700 Storage Expansion Units with 206 36.4GB HDDs, providing a total of over 7 TB of storage space.

The IBM TotalStorage FAStT700 Storage Server is based on the latest 2-GB fibre. To avoid single points of failure, it also provides high availability features such as dual hot-swappable RAID controllers, two dual redundant FC disk loops, write cache mirroring, redundant hot-swappable power supplies, fans and dual AC line cords.

Management of such complex environments is easily accomplished with the IBM FAStT Storage Manager software. The FAStT Storage Manager eases tasks such as configuring arrays and logical drives, assigning logical drives into storage partitions, replacing and rebuilding failed disk drives, expanding the size of arrays and logical volumes, and converting from one RAID level to another.

SAN

The first FAStT700 had eight EXP700 drive enclosures attached with a total of 110 15K rpm 36.4-GB HDDs. The Storage Manager Software was used to automatically lay out each of the arrays across the multiple controllers and drive loops. Because the FAStT Storage Server can support two redundant drive loops, the drive enclosures were set up to take advantage of this redundancy. If one data path fails, the controller uses the other data path to maintain the connection to the drive group.



This back view of the storage server and drive enclosures shows two redundant drive loops. Loop A and Loop B make up one redundant pair of drive loops. Loop C and Loop D make up a second redundant pair.

96 of the drives were automatically distributed across four arrays of 24 drives each with a RAID 1 configuration and a 64K stripe size. The remaining 14 drives were dedicated to storing the Oracle product software, DSS filesystem workspace for ETL testing, workload application binaries, and other general purpose uses such as work space for SQL*Loader, import, export and External Tables.

Figure 15 shows the IBM FAStT Storage Manager 8 (Subsystem Management) screen.

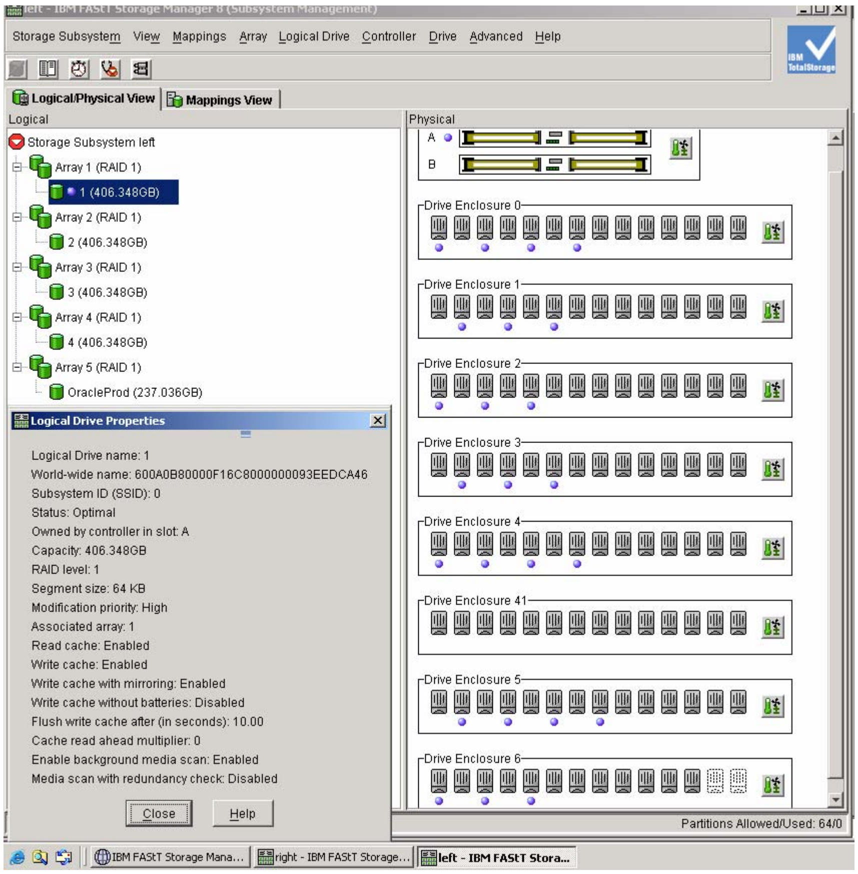


Figure 15

The second FASTt700 had seven EXP700 drive enclosures attached. Again, the Storage Manager Software was used to automatically lay out four RAID 1 arrays of 24 drives each across the multiple controllers and drive loops, as shown in Figure 16.

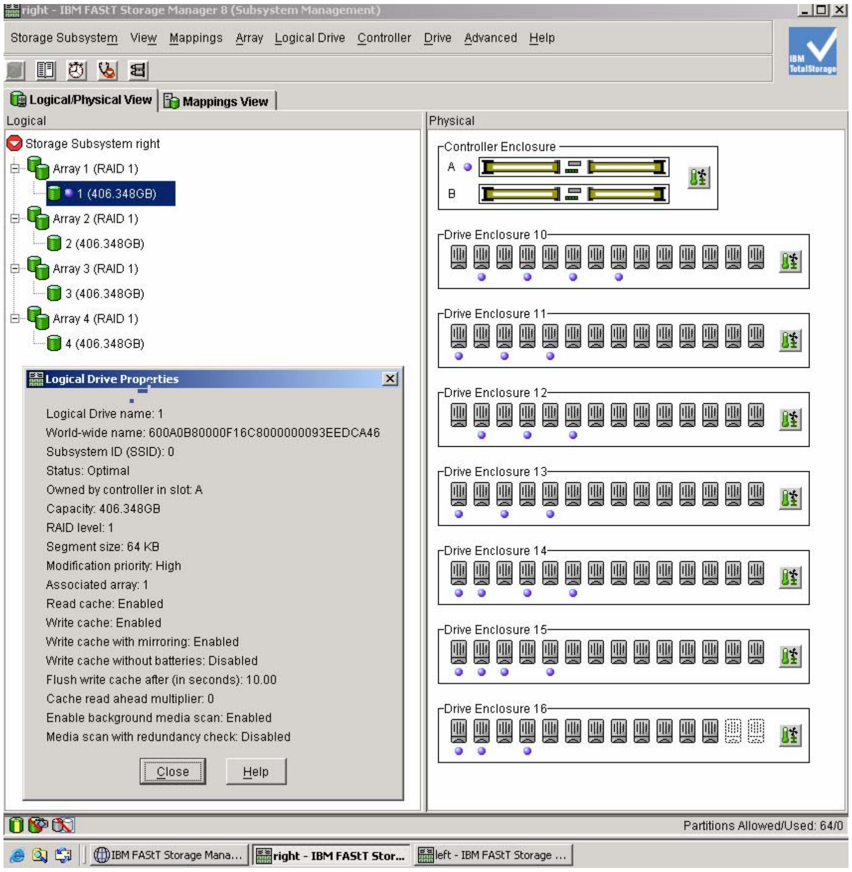


Figure 16

LAN

All public LAN traffic between clients and cluster nodes as well as Oracle interconnect traffic was exchanged through a Catalyst 6509 switch. Additional traffic on the switch included Ethernet access to the FAStT700 Storage Server for configuration and management of the storage subsystem. The Catalyst 6509 was configured with seven 16-port Gigabit Ethernet modules. The bonded Oracle interconnect was easily implemented by using Cicso's implementation of the EtherChannel protocol. The Gigabit EtherChannel port bundles allow a group of ports to be defined as a single logical transmission path between the switch and a host. When a link in an EtherChannel fails, the traffic previously carried over the failed link is switched to the remaining segments within the EtherChannel.

SAN Switch

Two IBM TotalStorage SAN Switch F16s were used to interconnect the cluster nodes and the SAN. Each switch port provides bandwidth of up to 200MB/s with maximum latency of two microseconds. The non-blocking architecture allows multiple simultaneous connections. Switch ports can operate in any of these modes: F, FL, or E.

Features of the F16 switch include *self-learning*, which allows the fabric to automatically discover and register host and storage devices, and *self-healing,* which gives the fabric the capability to isolate problem ports and reroute traffic onto alternate paths.

For performance and flexibility, an internal processor provides services such as name serving, zoning and routing. For availability, a second hot-plug power supply is supported.

Configuration of the switch is easily achieved with the use of the IBM TotalStorage SAN Fibre Channel Switch Specialist management tool. The switch features an embedded Web browser interface for configuration, management, and troubleshooting.

Overview of Databases

To test the Flexible Database Cluster architecture, three databases were created in the Matrix Server Cluster Filesystem using Oracle9*i* Release 2 version 9.2.0.3. The main databases were called OLTP and DSS. The third, smaller database was called DEV.

These databases were not intended to convey a best-practice for operations or performance—with the notable exception of the use of Oracle Managed Files (OMF). OMF is Oracle9*i* file management simplification, which deserves close consideration in any deployment scenario. The databases were intended only to be of realistic size and structure and usable to test the functionality and value add of FDC architecture.

OLTP Database (PROD)

The OLTP database schema is based on an order entry system similar to but not compliant with that defined in the TPC-C[9] specification. At a high level, the database schema contains the following application tables:

Customers. The database contains over 120 million customer rows in the "customer" table. This table contains customer-centric data such as a unique customer identifier, mailing address, email contact information and so on. The customer table is indexed with a unique index on the custid column and a non-unique index on the name column.

Orders. The database contains an orders table with 149 million rows of data. The orders table has a unique composite index on the custid and order id columns.

Line Items. Simulating a customer base with complex transactions, the Line item table contains as many as 15 line items per order. The item table has a three-way unique composite index on custid, ordid and itemid.

Historical Line Items. This table is maintained for OLAP purposes based on customer activity. The Historical Line Item table contains over 1.2 billion rows and is partitioned into 14 tablespaces.

[9] The Flexible Database Cluster proof of concept was in no fashion intended to comply with any specification of the Transaction Processing Council. For more information on TPC-C, please refer to www.tpc.org.

Product. This table describes products available to order. Along with such attributes as price and description, there are up to 140 characters available for a detailed product description. There are over 140 million products. The product table is indexed with a unique index on its prodid column.

Warehouse. This table maintains product levels at the various warehouse locations as well as detailed information about warehouses. This table is crucial in order fulfillment. The warehouse table is indexed with a unique composite index of two columns.

The database was created using the simplified Oracle Managed Files method. The FDC proof of concept was meant to prove manageability in a complex environment so it made little sense to use complex tablespace definitions. In fact, OMF works extremely well. Tablespaces created by OMF are optimized for the norm; however, specialized tuning may still be required in certain cases.

The full database create statement for the DSS database is supplied in the Appendix of this paper as reference to the simplicity of OMF. The PROD listing is lengthy and only notable sections, mostly pertaining to manageability, are discussed herein.

Figure 17 shows the "home" directory for all of the PROD datafiles. Using traditional symbolic links, a structure is set up so that a simple **ls**(1) command can be used to view all of the directories that can be assigned to the OMF command ALTER SESSION SET DB_CREATE_FILE_DEST. This give the database a "container" feel. All of these filesystems are, of course, PolyServe cluster filesystems mounted with the direct I/O mount option.

```
Linux
rac2 oracle   $ cd $APPHOME
rac2 oracle   $ ls -ld DATA* INDEX*
lrwxrwxrwx    1 oracle   dba          21 2003-07-02 20:40 DATA -> /mnt/ps/oradata1/DATA
lrwxrwxrwx    1 oracle   dba          22 2003-08-10 15:27 DATA1 -> /mnt/ps/oradata1/DATA1
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA2 -> /mnt/ps/oradata2/DATA2
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA3 -> /mnt/ps/oradata3/DATA3
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA4 -> /mnt/ps/oradata4/DATA4
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA5 -> /mnt/ps/oradata5/DATA5
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA6 -> /mnt/ps/oradata6/DATA6
lrwxrwxrwx    1 oracle   dba          22 2003-07-09 01:14 DATA7 -> /mnt/ps/oradata7/DATA7
lrwxrwxrwx    1 oracle   dba          23 2003-07-10 14:00 INDEX -> /mnt/ps/oradata6/INDEX6
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX1 -> /mnt/ps/oradata1/INDEX1
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX2 -> /mnt/ps/oradata2/INDEX2
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX3 -> /mnt/ps/oradata3/INDEX3
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX4 -> /mnt/ps/oradata4/INDEX4
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX5 -> /mnt/ps/oradata5/INDEX5
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX6 -> /mnt/ps/oradata6/INDEX6
lrwxrwxrwx    1 oracle   dba          23 2003-08-09 14:45 INDEX7 -> /mnt/ps/oradata7/INDEX7
rac2 oracle   $
```

Figure 17

Figure 18 shows the simple set of statements used to create the 60-GB customer tablespace. This style of tablespace creation was carried thoughout the entire database, and, as shown in the Appendix, throughout the DSS database as well. Here again, OMF eliminates all of the traditional complexity of the "railroad diagram" CREATE TABLESPACE statement.

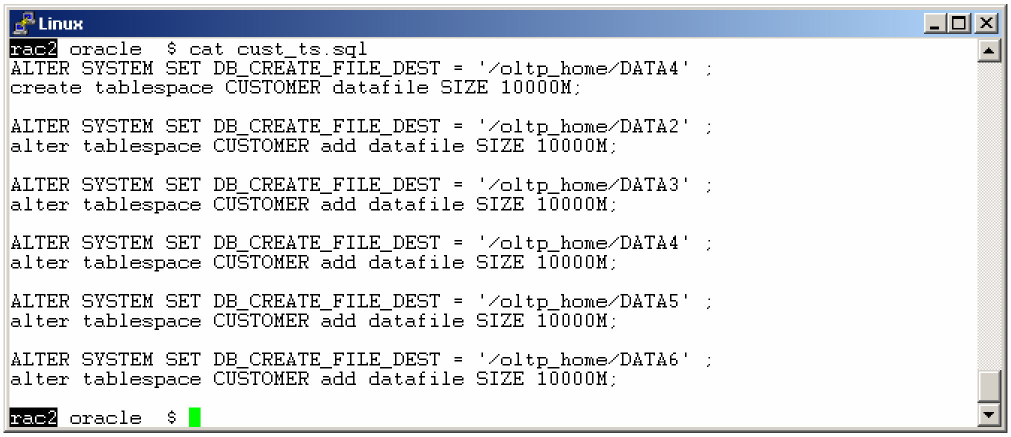


Figure 18

Figure 19 shows the mappings within Oracle for the OMF files that comprise the CUSTOMER tablespace.

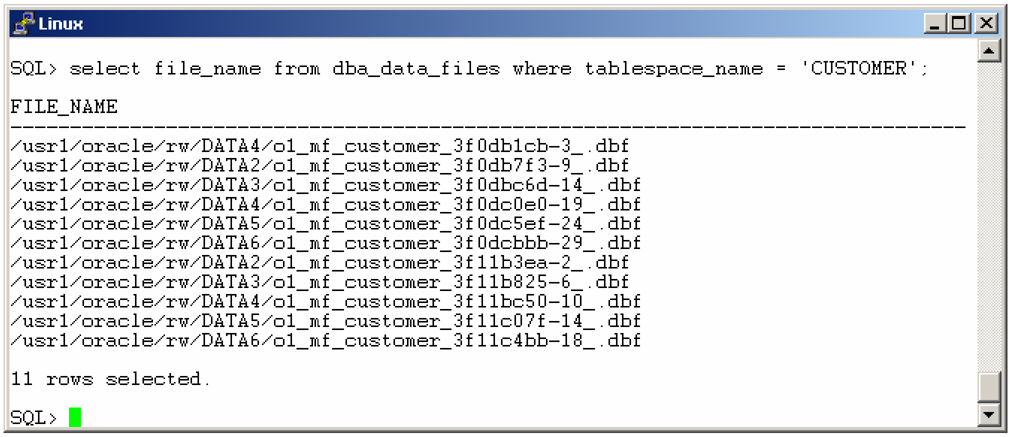
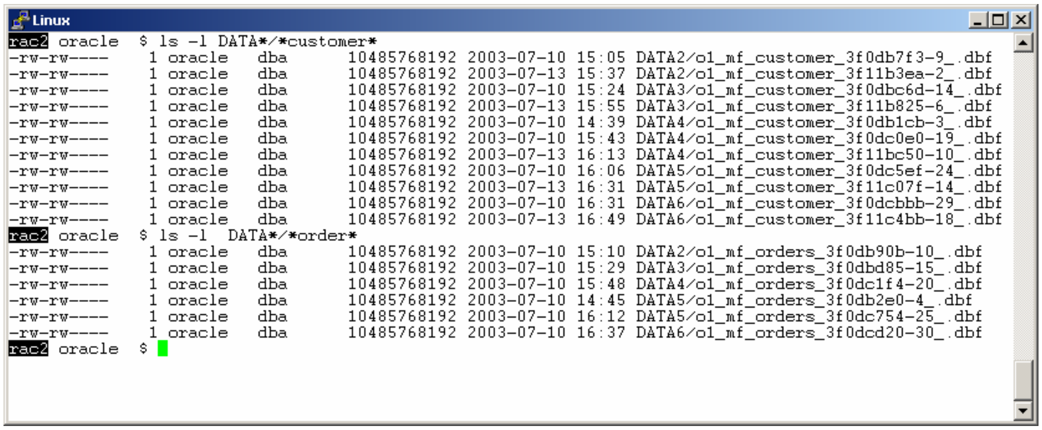


Figure 19

The **select** from DBA_DATA_FILES in Figure 19 is actually a long way to determine the files that comprise the CUSTOMER tablespace. When a sound OMF methodology is used, administrators can simply navigate to the $APPHOME top directory and use **ls**(1) under the DATA directories as shown in Figure 20. This file location method aids in backup operations as well. After placing the CUSTOMER tablespace in backup mode, a simple backup command of **$APPHOME/DATA*/*customer*** can capture all of the associated datafiles.



```
Linux
rac2 oracle  $ ls -l DATA*/*customer*
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:05 DATA2/o1_mf_customer_3f0db7f3-9_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-13 15:37 DATA2/o1_mf_customer_3f11b3ea-2_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:24 DATA3/o1_mf_customer_3f0dbc6d-14_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-13 15:55 DATA3/o1_mf_customer_3f11b825-6_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 14:39 DATA4/o1_mf_customer_3f0db1cb-3_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:43 DATA4/o1_mf_customer_3f0dc0e0-19_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-13 16:13 DATA4/o1_mf_customer_3f11bc50-10_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 16:06 DATA5/o1_mf_customer_3f0dc5ef-24_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-13 16:31 DATA5/o1_mf_customer_3f11c07f-14_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 16:31 DATA6/o1_mf_customer_3f0dcbbb-29_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-13 16:49 DATA6/o1_mf_customer_3f11c4bb-18_.dbf
rac2 oracle  $ ls -l  DATA*/*order*
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:10 DATA2/o1_mf_orders_3f0db90b-10_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:29 DATA3/o1_mf_orders_3f0dbd85-15_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 15:48 DATA4/o1_mf_orders_3f0dc1f4-20_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 14:45 DATA5/o1_mf_orders_3f0db2e0-4_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 16:12 DATA5/o1_mf_orders_3f0dc754-25_.dbf
-rw-rw----    1 oracle   dba      10485768192 2003-07-10 16:37 DATA6/o1_mf_orders_3f0dcd20-30_.dbf
rac2 oracle  $
```

Figure 20

The total database size was on the order of 810 GB as seen in Figure 21. The *space.sql* script tallies total database space, which can be seen in the used and free columns.

```
Linux
SQL> @space

TS_NAME                 TOTAL_MB    FREE_MB    USED_MB PCT_FREE MAX_CONTIG FRAGS
-------------------- ---------- ---------- ---------- -------- ---------- -----
CSTATE_IDX               100.00      99.94        .06    99.94      99.94     1
CUSTOMER              110000.00   62551.38   47448.63    56.86    3712.00    24
CUS_IDX                10000.00    7655.00    2345.00    76.55    2960.94   357
DATE_IDX                 100.00      99.94        .06    99.94      99.94     1
DELCUST                10000.00    9989.94      10.06    99.90    3968.00     3
ITEM                  228600.31     390.00  228210.31      .17      44.94    16
ITEM_DATA1_1           30000.00   16986.75   13013.25    56.62    3968.00     5
ITEM_DATA1_2           40000.00   26986.19   13013.81    67.47    3968.00     9
ITEM_DATA2_1           40000.00   26991.50   13008.50    67.48    3968.00     9
ITEM_DATA2_2           40000.00   26969.81   13030.19    67.42    3968.00     9
ITEM_DATA3_1           40000.00   26977.44   13022.56    67.44    3968.00     9
ITEM_DATA3_2           40000.00   26983.19   13016.81    67.46    3968.00     9
ITEM_DATA4_1           40000.00   13973.50   26026.50    34.93    3968.00     5
ITEM_DATA4_2           46912.00   20895.13   26016.88    44.54    3968.00     7
ITEM_DATA5_1           40000.00   26990.44   13009.56    67.48    3968.00     9
ITEM_DATA5_2           40000.00   26988.50   13011.50    67.47    3968.00     9
ITEM_DATA6_1           40000.00   26985.06   13014.94    67.46    3968.00     9
ITEM_DATA6_2           40000.00   26990.63   13009.38    67.48    3968.00     9
ITEM_DATA7_1           30000.00   16974.06   13025.94    56.58    3968.00     5
ITEM_DATA7_2           30000.00   16985.25   13014.75    56.62    3968.00     5
ITM_IDX                70000.00   69999.56        .44   100.00    3968.00    21
NAME_IDX               10000.00    4805.31    5194.69    48.05    2645.75    47
ORDERS                 60000.00   11170.69   48829.31    18.62    1935.94     8
ORD_IDX                10000.00    6537.56    3462.44    65.38    3968.00    51
POL1                   10240.00    2367.81    7872.19    23.12    1067.00     5
POL2                   10240.00    2367.81    7872.19    23.12    1067.00     5
POL3                   10240.00    2367.81    7872.19    23.12    1067.00     5
POL4                   10240.00    2367.81    7872.19    23.12    1067.00     5
PRD_IDX                 1290.00     238.19    1051.81    18.46     217.00    44
PRODID_IDX              1000.00     999.94        .06    99.99     999.94     1
PRODUCT                56000.00   41742.13   14257.88    74.54    3968.00    22
QTY_IDX                  671.00     579.19      91.81    86.32     579.19     1
ROLL_1                   300.00     299.91        .09    99.97     299.91     1
ROLL_2                   300.00     299.91        .09    99.97     299.91     1
STATE_IDX                100.00      99.94        .06    99.94      99.94     1
STRMTEST                1024.00    1019.44       4.56    99.55    1019.44     1
SYSTEM                  3183.38    3046.27     137.12    95.69    2874.88    18
SYS_UNDOTS               170.38       1.19     169.19      .70        .56     3
TEMP                   61000.00   60999.95        .05   100.00    7463.16 #####
WAREHOUSE             109272.00     200.50  109071.50      .18      29.50    14
WHR_IDX                15566.00   15565.94        .06   100.00    3968.00     4

41 rows selected.

SQL>
```

Figure 21

DSS Database

The DSS database schema simulates a sales productivity decision support system. It contains space for both an extraction of the CUSTOMER table from the PROD database and a number of smaller datamarts.

Figure 22 shows the table definition for the main table in the warehouse.

```
Linux

SQL> desc cust_analysis
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 HASH                                      NOT NULL NUMBER(9)
 CUSTID                                             NUMBER
 NUMORDS                                            NUMBER(4)
 NAME                                               VARCHAR2(60)
 ADDRESS                                            VARCHAR2(60)
 CITY                                               VARCHAR2(40)
 STATE                                              CHAR(3)
 ZIP                                                CHAR(6)
 AREA                                               CHAR(4)
 PHONE                                              CHAR(8)
 BUSINESSTYPE                                       VARCHAR2(40)
 SALESMAN                                           VARCHAR2(40)
 CUSTDETAILS1                                       VARCHAR2(60)
 CUSTDETAILS2                                       VARCHAR2(60)
 CREDITLIM                                          NUMBER(9)
 BUSINESS_CODE                                      NUMBER(9)

SQL> select count(*) from cust_analysis;

  COUNT(*)
----------
 286570835
```

Figure 22

DEV Database

The DEV database is a simple insert engine designed to test scalability while inserting records 2 KB in size. The database is approximately 10 GB total. It has only two threads defined; therefore, only two instances can access this database at one time.

Workload Descriptions

The workloads chosen for the Flexible Database Cluster proof of concept were not as important as the fact that there were three of them. As stated earlier, the databases were called PROD, DSS and DEV. Following is a description of the type of processing each database sustained during the test. The goal was to have a realistic mix of processing running on the system while testing the manageability of FDC architecture.

OLTP Workload

Simulating an Order Entry system, the application accessing the PROD database consists of connecting 100 users per node via the PROD service defined in the *tnsnames* service definition shown earlier. The nodes under test are evenly loaded due to the load balancing attribute of the PROD SQL*Net service. Each user cycles through a set of transactions described below. At the end of each transaction the client process sleeps for a small random period of time to simulate human interaction. To that end, this testing is not the typical benchmark style where all processors are 100% utilized[10]. Such a condition is not desirable in a datacenter scenario, therefore testing manageability of a proposed architecture under those conditions was not deemed realistic.

A key attribute of this testing is that it was completely void of traditional cluster-aware tuning. Most cluster-centric database tests have possessed at least some form of application-level partitioning. For example, nearly all Transaction Processing Council Specification C (TPC-C) benchmarks executed on a cluster use a method called “data dependant request routing” at a bare minimum. This method uses a transaction monitor to route all requests for a given transaction to a node in the cluster provisioned for servicing requests that modify data within a certain key range. For instance, node 1 in the cluster accepts new order transaction requests only from customers with customer identifiers in the 1-1000000 range, node 2 services the 1000001-2000000 range, and so on.

The pitfall of such partitioning schemes is that they require changes to the application. Applications should not have to change in order to scale horizontally in a clustered environment, and with Oracle9*i* Real Application Clusters, they don’t. Oracle9*i* Real Application Clusters is a radical departure from typical clustered database technology. With its Cache Fusion Technology and shared disk architecture, “off the shelf” applications can fully exploit clustered systems—without cluster-centric tuning or application code modifications. To that end, the Flexible Database Cluster proof of concept used an application test that possessed no cluster-centric tuning.

The pseudo-users of the test application connect to any Oracle Instance in the cluster and execute transactions as if they were running on a legacy SMP system. In fact, this test application has been used in the past to test SMP scalability.

The transaction details are as follows:

Orders Query. This transaction accounts for 16% of the activity. It provides top-level detail on existing orders for the customer and provides such detail in a most-recent to least-recent order.

Customer Attribute Update. This transaction represents 28% of the workload. It offers the ability to update such information as phone number, address, credit card information, etc.

Orders Report. This transaction differs from Orders Query in that it offers full order detail for a customer to include shipment status. This transaction is executed 4% of the time.

Product Update. This transaction occurs as the result of a customer service change to a product description. 36% of all transactions are a Product Update.

New Items. This transaction accounts for 16% of the mix. It adds items into stock on hand.

The I/O mix for the workload is 75% read and 26% write. On average, each transaction has the following cost associated with it:

[10] The value of traditional benchmarks is not being questioned. Hardware vendors need a fair playing field to establish the capability of their offerings.

Oracle Statistics	Average Per Transaction
SGA Logical Reads	209
SQL Executions	58
Physical I/O	6
Block Changes	7
User Calls	61

DSS Workload

The DSS workload is set up as a stream of queries in a queue that is serviced in serial order by the Parallel Query Option(PQO). Changes to the number of instances do not affect the current query being executed, but at the beginning of each query PQO considers how many instances there are. An increase from, say, 2 to 4 nodes will cause a speed-up on the very next query that is executed.

The majority of the queries pushed through the queue had an access method of full table scan. A few had index range scans in their plans. The goal of the testing was to keep the PQO busy as instances were dynamically added, and these queries generated sufficient demand on the I/O subsystem.

Figure 23 lists some of the queries that funnel through the queue. The WHERE predicate is largely randomized in a program that generates the *q_input.sql* file.

```
Linux
racl3 oracle  $ cat q_input.sql
select salesman, BUSINESSTYPE
from ca
where
BUSINESS_CODE not in ( select BIZTYPE from BUSINESS_TYPE_CODES where BIZTYPE < 315 )
and
creditlim between 1000 and 18000
order by custdetails1 , CITY ;

select salesman, BUSINESSTYPE
from ca
where
salesman  IN ('Lilly Kovacs','Larry Tate','Harry Pike' )
order by custdetails1 , CITY ;

select salesman, BUSINESSTYPE
from ca
where custdetails1 LIKE '%PROMO CODE BETA%'
and
creditlim between 1000 and 18000
order by custdetails1 , CITY ;

select SALESMAN,sum(numords)
from cust_analysis ca
where to_number(businesstype) NOT IN
( select   biztype from business_type_codes
where biztype_desc NOT LIKE '%RETAIL%')
group by SALESMAN ;

racl3 oracle  $
```

Figure 23

DEV Workload

The DEV workload is simply a zero think time program that inserts 2K rows via pipe to SQL*Loader. The streams of loader processes execute on up to two nodes (rac15 and rac16) when DEV is being tested along with the other workloads.

Test Results

Stress Testing

The Flexible Database Cluster proof of concept was not a benchmark per se. The goal was to test and prove the architecture and to ascertain value add in key areas such as "on-demand" scalability and higher availability that are attributable to the architecture.

That said, the point would seem moot if the cluster was not capable of sustaining a high stress workload, or if the scalability of the primary database on the system was poor.

A test suite was set up with 16 instances of the PROD database. To stress test the entire cluster with a single horizontally-scaled application, an equal number of users were specifically targeted to each node via simple single instance SQL*Net services. Since the workload[11] is not a traditional benchmark, where the test method is to utilize 100% of system resources before adding a node, the performance metric is slightly non-traditional. The workload contains think time pauses between transactions. Therefore, processor utilization is not 100% even at 500 users per node. The peak at 500 users per node was nearly 97% processor utilization on average across the 16 nodes. Datacenter managers do not like their production systems running at 100% processor utilization, so it seemed unrealistic to measure system performance that way during this proof of concept.

The test scenario consisted of connecting 100, 250 and 500 users per node, each executing the OLTP workload described above. This was a great capacity testing workload. The graph in Figure 24 shows how the cluster and Oracle9*i* RAC performed nicely as the user count increased from 1600 clusterwide to 4000 and then 8000. In fact, when the workload increased from 100 to 250 users per node, the throughput increased from 19,140 transactions per minute (tpm) to 47,160 tpm—98% of linear scalability.

Increasing the user count to 500 users per node did not, however, result in near linear increase in throughput. The reason for this is not attributable to Oracle9*i* RAC, the hardware or PolyServe Matrix Server. Instead, examination of the **vmstat**(1) data collected revealed a modicum of swapping. These were dedicated connections (shadow processes), so their PGA and process stack did account for a good portion of the physical memory available once the Kernel and SGA were subtracted from the 2-GB total per node. Throughput did increase to 70,740 tpm, representing 74% scalability when ramping up users from 1600 clusterwide to 8000.

[11] Please refer to the description of the workload in the "Workload Description" section of this paper.

More importantly however, even in conditions of virtual memory shortage, the workload progressed and the 30-minute measured tests completed.

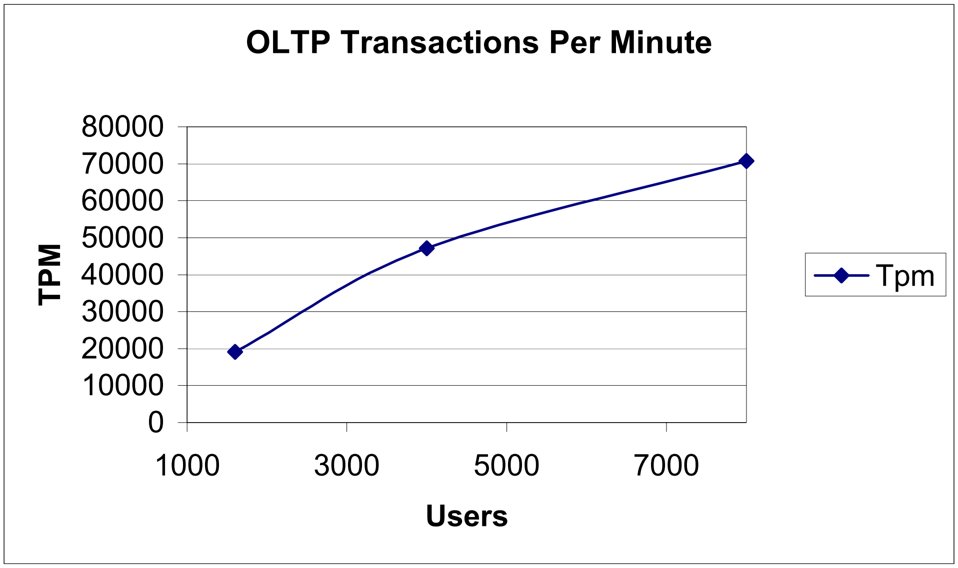


Figure 24

Advanced Testing

The advanced testing consisted of two test suites. Test Suite One was oriented to determining how the FDC architecture can aid Oracle9*i* RAC in offering even better availability than it naturally offers. Test Suite Two was designed to measure performance increase while dynamically adding servers to a DSS application.

Test Suite One: Flexible Database Cluster for Higher Availability with RAC

While Oracle9*i* RAC is renown for scalability, it also offers unprecedented availability. When combined with the power of SQL*Net, RAC offers a nearly bullet-proof guarantee that there will always be an instance of a database to connect to. The full capability of RAC for high availability is limited, however, in small clustered environments. Once again the Flexible Database Cluster adds tremendous architectural and operational value.

Consider an application that is sized to require a 4-node cluster. If a node suffers hardware failure, the application is now serviced at no better than 75%. Until the hardware is either repaired or replaced, this condition will persist. Of course there is still 100% uptime for the application due to the power of RAC, but there are likely users that can detect the performance hit related to the reduced node count.

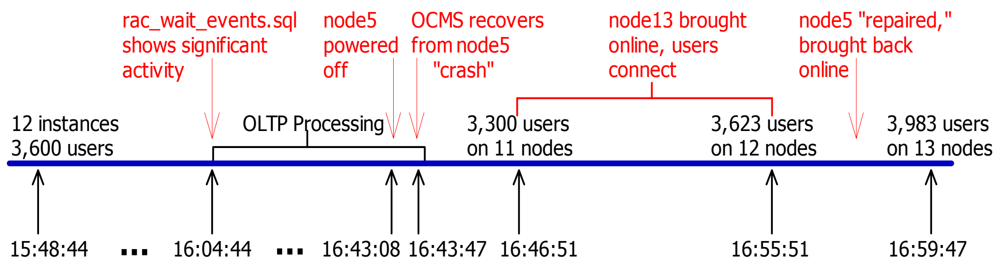
Having a spare node fully loaded with Oracle[12] and ready to be cabled and booted is one form of protection. The question is whether it will have the right “personality”

[12] A requirement in the absense of a Cluster File System-based Shared Oracle Home such as was configured for the Flexible Database Cluster testing.

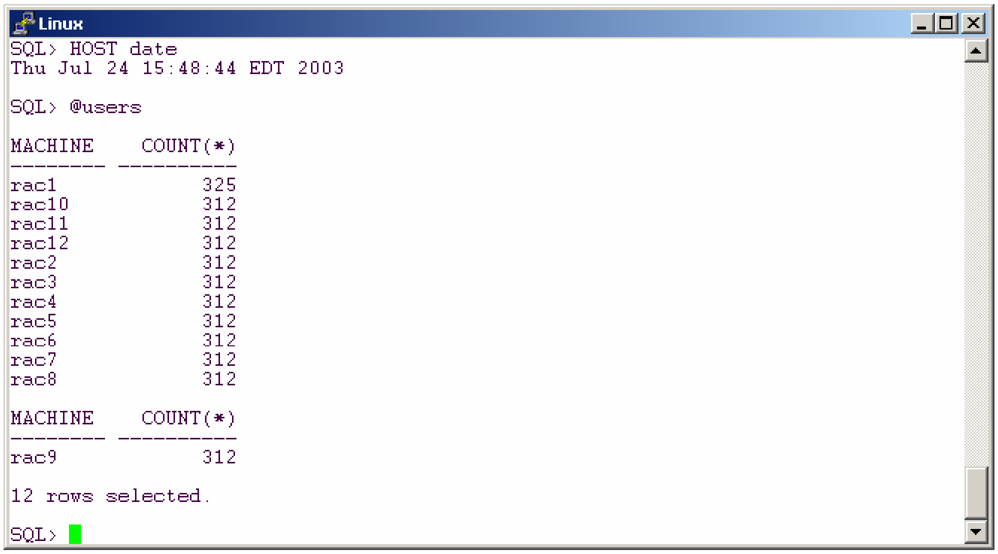
to replace the failed node. That is, with Oracle Cluster Management Services (OCMS), the *cmcfg.ora* parameter called *PrivateNodeNames* contains a list of hostnames or IP addresses that cannot be changed without rebooting OCMS on all nodes in the cluster. Of course all database instances must be down to reboot OCMS. Likewise, *tnsnames.ora, listener.ora,* and many other configuration files likely expect the replacement node to possess at a minimum the same IP address and hostname. While these issues are not insurmountable, they do tax the continuity of operations in the face of node failure. Every minute counts when your database is running at 75% capacity.

In stark contrast, this same application serviced by four nodes in a FDC environment can rapidly ramp back up to 100% capacity in light of not one, but potentially several concurrent hardware failures. We should never be so unlucky as to suffer multiple hardware failures, but it can happen and this is one of the reasons RAC is the right choice for mission critical deployments. However, to fully harness the ability of RAC to sustain operations in light of multiple node failures, the FDC architecture is arguably the only option. After all, the difficulty of ramping back up to 100% from a single failure is likely more then double should more than one server fail in a small window of time.

The architecture of the FDC clearly lends itself to much improved handling of node failure over that of small node count clusters. This point became quite evident during the proof of concept testing. The following timeline provides a bird's-eye view of what transpired during this availability testing.



A test was set up with PROD executing on nodes 1 through 12, DSS on nodes 13 and 14 and DEV on nodes 15 and 16. The PROD session count was ramped up to more than 3600 users spread evenly across the 12 nodes and brought to a steady state processing transactions at 15:48:44 EDT, as depicted in Figure 25 via the *users.sql* script.



```
SQL> HOST date
Thu Jul 24 15:48:44 EDT 2003

SQL> @users

MACHINE    COUNT(*)
--------- ----------
rac1             325
rac10            312
rac11            312
rac12            312
rac2             312
rac3             312
rac4             312
rac5             312
rac6             312
rac7             312
rac8             312

MACHINE    COUNT(*)
--------- ----------
rac9             312

12 rows selected.

SQL>
```

Figure 25

Using the *rac_wait_event.sql* script, global activity was monitored to ensure that the instances were quite active as was the case at 16:04 EDT. Figure 26 shows the amount of transactional locking (enqueue waits) the sessions were sustaining, which indicates a good deal of contention/activity[13].

[13] If this were a tuning exercise, action would be warranted to track down the source of enqueue waits. This was, however, a very active and contentious workload and valid for testing instance recovery.

```
SQL> HOST date
Thu Jul 24 16:04:29 EDT 2003

SQL> @rac_wait_event

EVENT                                                                 COUNT(*)
--------------------------------------------------------------------- ----------
SQL*Net message from client                                                 2003
enqueue                                                                      958
row cache lock                                                               444
rdbms ipc message                                                             71
db file sequential read                                                       45
latch free                                                                    29
global cache open x                                                           24
gcs remote message                                                            22
global cache cr request                                                       20
log file sync                                                                 20
ges remote message                                                            12

EVENT                                                                 COUNT(*)
--------------------------------------------------------------------- ----------
smon timer                                                                    12
pmon timer                                                                    12
global cache null to x                                                        11
global cache s to x                                                            3
buffer busy global cache                                                       2
PX Deq: reap credit                                                            1
global cache cr disk request                                                   1
buffer busy global CR                                                          1

19 rows selected.

SQL>
```

Figure 26

After allowing the workload to run for about 40 minutes, node 5 was abruptly powered off to simulate a node failure. As seen in Figure 27, Oracle Cluster Management Services logged the failure in the *cm.log* file on rac1. As nodes are counted from 0 in OCMS, it can be seen that the fifth node (node 4) is no longer in the cluster. The log also shows that OCMS was able to reconfigure its view of node membership in just a matter of seconds (from 16:43:47 to 16:43:51 EDT). This is quite amazing, given the size of the cluster and the cluster-wide activity at the time.

```
Linux
ener:108557 file = unixinc.c, line = 754 {Thu Jul 24 16:43:29 2003 }
NMEVENT_SUSPEND [00][00][00][00][00][00][ff][ef] {Thu Jul 24 16:43:47 2003 }
HandleUpdate(): SYNC(5) from node(0) completed {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(0) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(1) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(2) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(3) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(5) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(6) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(7) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(8) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(9) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(10) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(11) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(12) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(13) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(14) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
HandleUpdate(): NODE(15) IS ACTIVE MEMBER OF CLUSTER {Thu Jul 24 16:43:50 2003 }
NMEVENT_RECONFIG [00][00][00][00][00][00][ff][ef] {Thu Jul 24 16:43:51 2003 }
Successful reconfiguration,  15 active node(s) node 0 is the master, my node num
 is 0 (reconfig 6) {Thu Jul 24 16:43:51 2003 }
rac1 oracle  $
rac1 oracle  $
```

Figure 27

A further look into *alert_prod1.log* reveals that instance recovery on node 1 commenced nearly immediately after OCMS membership reconfiguration and was completed in a mere 10 seconds as seen in Figure 28. Keep in mind that everything involving Oracle was stored in the Matrix Server cluster filesystem, including Oracle Home (executables) and all datafiles, control files, log file and OCMS quorum disk. Since RAC cannot complete recovery without access to its files, this Oracle-level recovery time is also a testament to the rapid recovery times of the Matrix Server cluster filesystem.

```
Linux
Thu Jul 24 16:43:53 2003
Reconfiguration started
List of nodes: 0,1,2,3,5,6,7,8,9,10,11,
 Global Resource Directory frozen
 Communication channels reestablished
 Master broadcasted resource hash value bitmaps
 Non-local Process blocks cleaned out
 Resources and enqueues cleaned out
 Resources remastered 11043
 38831 GCS shadows traversed, 296 cancelled, 2280 closed
 17182 GCS resources traversed, 3 cancelled
 24955 GCS resources on freelist, 40734 on array, 40734 allocated
 set master node info
 Submitted all remote-enqueue requests
 Update rdomain variables
 Dwn-cvts replayed, VALBLKs dubious
 All grantable enqueues granted
 38831 GCS shadows traversed, 2459 replayed, 2576 unopened
 Submitted all GCS remote-cache requests
 1 write requests issued in 17057 GCS resources
 214 PIs marked suspect, 0 flush PI msgs
Thu Jul 24 16:43:54 2003
Reconfiguration complete
 Post SMON to start 1st pass IR
Thu Jul 24 16:44:02 2003
Undo Segment 943 Onlined
Thu Jul 24 16:44:03 2003
"alert_prod1.log" 1210L, 45091C written                        1114,1        91%
```

Figure 28

Logging into Oracle and executing the *instance_status.sql* and *users.sql* scripts indicated that there were 11 active instances, all with their respective users attached as seen in Figure 29.

```
Linux
SQL> HOST date
Thu Jul 24 16:46:51 EDT 2003

SQL> @instance_status

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac1                1 ACTIVE
rac2                2 ACTIVE
rac3                3 ACTIVE
rac4                4 ACTIVE
rac6                6 ACTIVE
rac7                7 ACTIVE
rac8                8 ACTIVE
rac9                9 ACTIVE
rac10              10 ACTIVE
rac11              11 ACTIVE
rac12              12 ACTIVE

11 rows selected.

SQL> @users

MACHINE    COUNT(*)
-------- ----------
rac1            336
rac10           312
rac11           312
rac12           312
rac2            312
rac3            312
rac4            312
rac6            312
rac7            312
rac8            312
rac9            312

11 rows selected.
```

Figure 29

What took place next is the perfect example of the value FDC architecture adds to RAC. As seen in Figure 30, by 16:55:51 EDT, the choice had been made to incorporate node 13 into the OLTP mix and 327 users had already connected. That is, during a period of 12 minutes, the following events occurred:

- The OLTP portion of the FDC suffered a server failure on node rac5.
- The user community suffered a momentary pause in service during PolyServe Matrix Server, OCMS and Oracle instance recovery.
- All users on the remaining 11 nodes maintained their connections as indicated in Figure 30, which shows that 11 of the original 12 instances were still quite active in spite of the failure of node 5.
- Node 13 was chosen to replace node 5 and since *tnsnames.ora* was set up appropriately, users were able to connect to the instance on that node just as soon as it was brought online.

All told, there was at most a period of 12 minutes where the user community was serviced by roughly 92% capacity!

```
Linux
SQL> HOST date
Thu Jul 24 16:55:51 EDT 2003

SQL> @instance_status

HOST_NAME      THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac1                1 ACTIVE
rac2                2 ACTIVE
rac3                3 ACTIVE
rac4                4 ACTIVE
rac6                6 ACTIVE
rac7                7 ACTIVE
rac8                8 ACTIVE
rac9                9 ACTIVE
rac10              10 ACTIVE
rac11              11 ACTIVE
rac12              12 ACTIVE

HOST_NAME      THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac13              13 ACTIVE

12 rows selected.

SQL> @users

MACHINE    COUNT(*)
-------- ----------
rac1            325
rac10           312
rac11           312
rac12           312
rac13           327
rac2            312
rac3            320
rac4            312
rac6            312
rac7            320
rac8            312

MACHINE    COUNT(*)
-------- ----------
rac9            312

12 rows selected.

SQL>
```

Figure 30

Part of the 12-minute duration of reduced instance count for OLTP was intended to simulate the time required to determine that node 5 was fully out of commission and to decide which of the remaining Flexible Database Cluster nodes should be repurposed from its primary database. The node chosen was the first node of the DSS database instance set. Indeed, since node 13 is a defined node in Oracle Cluster Management Services (*cmcfg.ora*) and is an added thread of the PROD database, an instance could have easily been started there within moments of OCMS recovery completion.

Finally, to completely establish the value of FDC architecture, node 5 was brought back into the OLTP mix. An instance was started and 298 sessions had connected to the database by 16:59:47 EDT as seen in Figure 31—only four minutes since ramping back up to 12 nodes. This portion of the test simulated incrementally adding instances to OLTP to add processing power in the event that any work was back-logged during the 12-minute reduced node-count period. Although it is unlikely that processing at 8% reduced capacity could make a dramatic impact, growing the OLTP portion of the cluster to 13 nodes in four minutes while the cluster, and OLTP in particular, are otherwise quite busy is possibly the best proof point for the power of RAC in a Flexible Database Cluster configuration.

```
Linux
SQL> HOST date
Thu Jul 24 16:59:47 EDT 2003

SQL> @instance_status

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac1                1 ACTIVE
rac2                2 ACTIVE
rac3                3 ACTIVE
rac4                4 ACTIVE
rac5                5 ACTIVE
rac6                6 ACTIVE
rac7                7 ACTIVE
rac8                8 ACTIVE
rac9                9 ACTIVE
rac10              10 ACTIVE
rac11              11 ACTIVE

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac12              12 ACTIVE
rac13              13 ACTIVE

13 rows selected.

SQL> @users

MACHINE    COUNT(*)
-------- ----------
rac1            330
rac10           316
rac11           316
rac12           312
rac13           327
rac2            316
rac3            324
rac4            316
rac5            298
rac6            316
rac7            324

MACHINE    COUNT(*)
-------- ----------
rac8            316
rac9            316

13 rows selected.

SQL>
```

Figure 31

No total outage, no long duration service brown-outs, no troublesome action required on behalf of the administrative and operational staff in spite of a server failure in this large cluster environment establishes the FDC architecture as a natural fit for today's demanding IT requirements.

Test Suite Two: Flexible Database Cluster for Scalability On Demand!

There are many Oracle9*i* RAC scalability benchmark results that prove the power of RAC. These benchmarks range from audited TPC and Oracle Application Standard benchmarks to many customer-driven proof points.

For the Flexible Database Cluster proof of concept, scalability was looked at from a very non-standard viewpoint. With RAC, SQL*Net and the Parallel Query Option, the ability to add instances to active databases is functionality not found in any other commercial database. It is truly amazing that nodes can be added to a mix of OLTP or DSS and throughput increases seamlessly—with little or no user interruption. This is precisely one of the types of tests that was conducted on the FDC.

Modeled after a query queuing system, the DSS database sustains a non-stop stream of requests that vary greatly from an intensity standpoint. That is, one query that performs a very quick index range scan may come through the query queuing system, and the next may be a rather non-optimal anti-join that scans hundreds of millions of rows. Fact is, the queries for this test were purposefully not tuned and therefore lack perfect query plans. It was our viewpoint that every datacenter has at least a few queries that are not tuned, for one reason or another, and often the only choice at hand is to give it more hardware. That is not such a horrible ordeal in the case of RAC and Parallel Query and even simpler when deployed with FDC architecture. Indeed, right in the midst of a stream of queries processing through a query queue, instances can be booted and the next query serviced by the Parallel Query Option will take advantage of the added hardware resources. Since database instances can be stopped and started on the fly, and the Parallel Query Option can immediately recognize and use the resources, what value can FDC architecture possible add?

The answer lies in just how many nodes comprise the cluster and whether the database knows about the instances that may open the database on the additional nodes. Simply put, one cannot dynamically add resources that are not physical members of the cluster. More importantly, however, is that nodes and threads must be defined and added to both OCMS and the database beforehand, as described in the sections above.

The lifeblood of DSS is table scan throughput and, to a lesser degree, processor bandwidth for filter, sort/join, grouping and aggregation. Parallel Query processing is essentially a set of pipeline operations. Scanning feeds filtering (e.g., application of the WHERE predicate in the SQL statement), filtering feeds sorting/joining and so on. Any query system based on the Parallel Query Option cannot push through more work unless there is more data coming into the bottom of the pipeline[14]. To that end, Parallel Query throughput is directly tied to the rate of I/O.

True, query completion time is the true metric of performance on a DSS query. However, most datacenters are not able to time each query that comes through a system. In general, DBAs have a “feel” for how much “work” their DSS system performs. Through the use of Oracle Enterprise Manager, Statspack, simple OS level monitoring tools and third party tools, a DBA can sense when throughput is likely not acceptable without even discussing it with the end users.

[14] Given the query plans remain static.

For example, consider a DBA who has monitored six months of I/O processing associated with a given query system and found that the average I/O load is on the order of 100 Megabytes per second. The application development team members and users indicate that queries are completing in less than three minutes through a query queue. The developers and users are happy with the query plans and response times. This would be a general “work” level and “feel” for such a system.

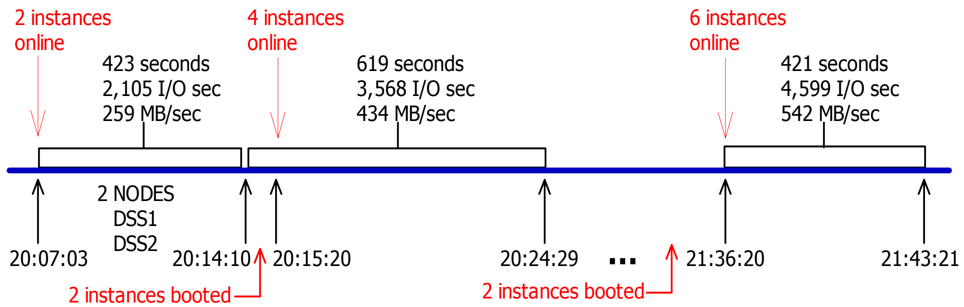
In this example, it is fair to surmise that, provided there were no new poorly tuned queries introduced to the query mix, a substantial increase in I/O throughput would likely result in a reduced query complete time. The DBA does not need to benchmark the whole system to discover that an increase in monitored I/O throughput of, say, 100MB/s to 300MB/s is going to result in happier end users.

This example sets the stage for one of the dynamic server provisioning tests performed during the proof of concept. As described above, the DSS database and workload simulate a continual stream of sales-productivity queries. The measurements taken were that of cluster-wide I/O throughput available in Oracle9*i*’s internal global performance views (e.g., gv$) while incrementally adding nodes over a given period. The measurement of success is simply more I/O throughput by the DSS database instances as measured by Oracle.

On-Demand Scalability Test Suites

Test Scenario One

The following timeline shows what transpired during Test Scenario One.



The first phase of the test was to service queries from the queue while only the DSS1 and DSS2 instances were running on rac13 and rac14 respectively. Figure 32 shows that from 20:07:03 EST to 20:14:10, the cumulative internal gv$ performance views[15] increased from 4,191,242 to 5,081,848 physical disk transfers and the total cumulative Megabytes transferred increased 109,577 MB from 482,858 to 592,435. At 423 seconds, this was roughly 259 MB/sec throughput. As described above, this could have been any number of queries or a few large queries. They are serviced through the queue first come first served.

[15] See the Appendix of this paper for a code listing of the scripts used in this section.

```
Linux
SQL> @instance_status

HOST_NAME      THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac13               1 ACTIVE
rac14               2 ACTIVE

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   4191242 482858.969        511   7.984375

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 20:07:03

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   5081848 592435.578        514    8.03125

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 20:14:10

SQL>
```

Figure 32

The next phase was to start two more instances of the DSS database. The DSS3 and DSS4 instances were started on nodes rac15 and rac16 respectively. These instances were ready by 20:15:20 EST. Of course, these instances were booted without interruption to the query stream processing. To the contrary, Figure 33 also reveals that by 20:24:29 EST, Oracle's cumulative internal counters had incremented to 7,290,944 physical reads and 268,833 MB transferred. The prior reading of the gv$ tables was at 20:14:10 EST, so over this 619 second period, the throughput rate jumped to roughly 434 MB/sec.

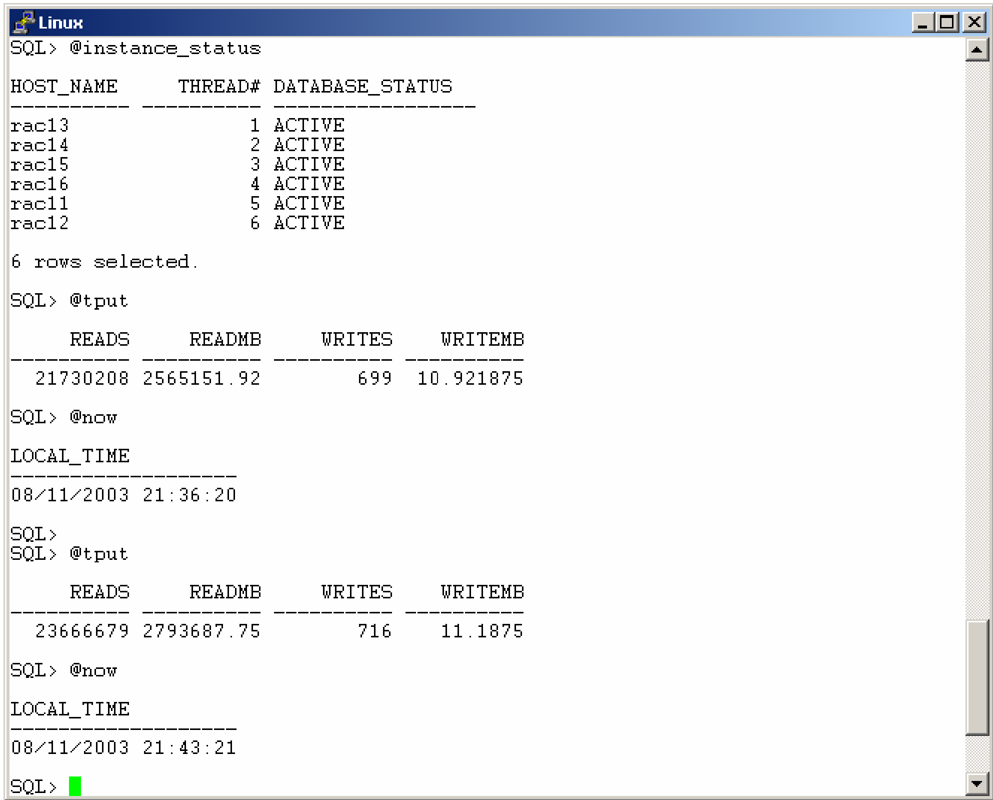
Oracle9*i* RAC and the power of the Parallel Query Option were able to seamlessly increase throughput to a running application by 84% of linear scalability! A very impressive result. This was only possible, however, because all resources were on hand in a manageable form in the Flexible Database Cluster.

```
Linux

SQL>
SQL> @instance_status

HOST_NAME      THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac13               1 ACTIVE
rac14               2 ACTIVE
rac15               3 ACTIVE
rac16               4 ACTIVE

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 20:15:20

SQL>
QL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   7290944 861268.938        522    8.15625

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 20:24:29
```

Figure 33

The test continued to the next phase, which was to start two more instances and add them to DSS. This took a little “slight of hand” since DSS was now executing on nodes 13 through 16. Further demonstrating the flexibility of this architecture, instances DSS5 and DSS6 were started on nodes rac11 and rac12 respectively. At the time of the test, PROD instances were executing on rac11 and rac12. Those instances were quiesced and stopped, and then DSS instances were booted.

Figure 34 shows that there were six instances and that between 21:36:20 and 21:43:21 EST there were 1,936,471 physical disk reads totaling 228,536 MB transferred. For a period of 421 seconds, that breaks down to 542 Mb/sec. Given the initial two-node reading of 259MB/sec, this test showed 70% scalability. Keep in mind that OLTP was initially running on nodes 1 through 12 and instances PROD11 and PROD12 where quiesced to accommodate DSS5 and DSS6. Remarkable flexibility and impressive scalability—on demand!

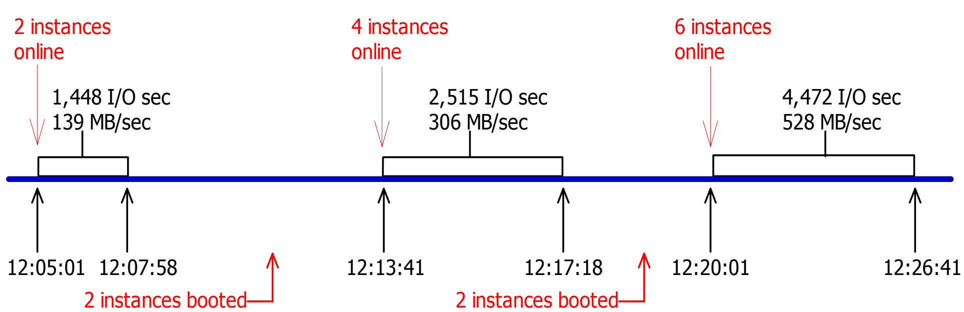


```
Linux
SQL> @instance_status

HOST_NAME      THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac13                1 ACTIVE
rac14                2 ACTIVE
rac15                3 ACTIVE
rac16                4 ACTIVE
rac11                5 ACTIVE
rac12                6 ACTIVE

6 rows selected.

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
  21730208 2565151.92        699  10.921875

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 21:36:20

SQL>
SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
  23666679 2793687.75        716    11.1875

SQL> @now

LOCAL_TIME
-------------------
08/11/2003 21:43:21

SQL>
```

Figure 34

Test Scenario Two

Test Scenario Two was modeled differently than Test Scenario One. The following timeline shows what transpired during this test.



In this test, DSS was initially set up on nodes rac11 and rac12. PROD was contained to nodes 1 through 10. Nodes 13 through 16 were considered a hot standby “pool” of server resources that could be dedicated to either PROD or DSS on demand. At the beginning of the test there were no Oracle instances booted on these reserved nodes.

To enhance the monitoring of this test, the PolyServe Oracle Disk Manager I/O monitoring package MxODM was used. The **mxodmstat** command can monitor I/O at all cluster and Oracle levels (e.g., cluster-wide all instances, cluster-wide database, named instances, node level, etc). The monitoring performed during this test scenario was of the instance level on DSS1 through DSS6 specifically. At the start of the test, only instances DSS1 and DSS2 are booted, hence **mxodmstat** shows no activity for the other instances in the monitoring list. A monitoring interval of 60 seconds was chosen to capture the "ramp-up" effect intended in this test.

Once again, phase one of the test was to monitor throughput while executing queries on only two nodes. Figure 35 shows that over a 177-second period, the instances on rac11 and rac12 serviced the query queue with an I/O rate of 139 MB/sec.

```
Linux
SQL> @instance_status

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac11               1 ACTIVE
rac12               2 ACTIVE

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   2419053 286272.984        230    3.59375

SQL> @now

08/12/2003 12:05:01

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   2620452 311060.578        230    3.59375

SQL> @now

08/12/2003 12:07:58

SQL>

rac11 oracle  $
```

Figure 35

Phase two of the test was measured between 12:13:41 and 12:17:18 EST (267 seconds). Figure 36 shows that Oracle's cumulative global physical read counters increased by
81,746 MB, yielding an I/O throughput rate of 306 MB/sec during the sampling. As is common for the Parallel Query Option, this particular stream of queries has enjoyed a speed-up of 100%, going from 2 to 4 nodes!

```
Linux
SQL> @instance_status

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac11               1 ACTIVE
rac12               2 ACTIVE
rac13               3 ACTIVE
rac14               4 ACTIVE

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   3649598 436263.469        233   3.640625

SQL> @now

08/12/2003 12:13:41

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   4321309  518009.25        235   3.671875

SQL> @now

08/12/2003 12:17:18
rac11 oracle  $
```

Figure 36

Phase 3 of the test was measured between 12:20:01 and 12:26:41 EST (400 seconds). In Figure 37, the gv$ tables show that six instances delivered 211,622 MB of scan throughput for a rate of 529 MB/sec. Note that the randomness of the queries in the query queue can vary the I/O demand slightly. Although it appears as though the cluster delivered super-scalability (140MB/s @ 2 nodes and 528MB/s @ 6), it is merely a reflection of the realistic nature of these queries. They vary in weight just like queries in the real world. What better workload to test the on-demand power of RAC in a Flexible Database Cluster?

```
Linux
SQL> @instance_status

HOST_NAME     THREAD# DATABASE_STATUS
---------- ---------- -----------------
rac11               1 ACTIVE
rac12               2 ACTIVE
rac13               3 ACTIVE
rac14               4 ACTIVE
rac15               5 ACTIVE
rac16               6 ACTIVE

6 rows selected.

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   4889800 586271.828        240       3.75

SQL> @now

08/12/2003 12:20:01

SQL> @tput

     READS     READMB     WRITES    WRITEMB
---------- ---------- ---------- ----------
   6683188 797893.922        247   3.859375

SQL> @now

08/12/2003 12:26:41

rac11 oracle  $
```

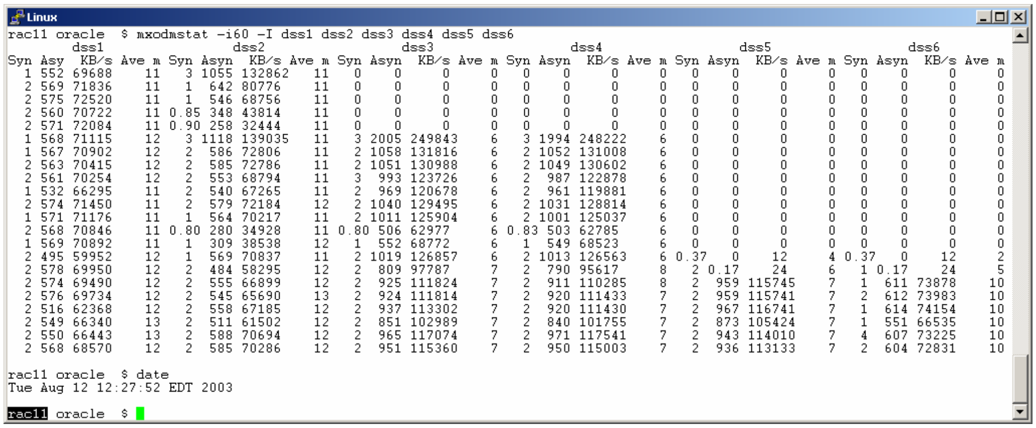
Figure 37

Notice again just how “on demand” the resources of the Flexible Database Cluster are. At 12:07:58 EST (Figure 35), the query queue was being serviced by two nodes at a rate of 139 MB/sec and just 12 minutes later at 12:20:01 EST (Figure 37) the instance count was six. All the while the query queue was being serviced non-stop.

During Test Scenario 2 of the “On-demand” test suite, **mxodmstat** was used to monitor all instances of the DSS database intended for this test. The only reporting element chosen for this command was “total I/O.” (The **mxodmstat** command can break out the I/O into reads and writes, but for the sake of capturing screen output, only total I/O was reported.)

Figure 38 shows the dynamic nature of adding instances to service the query. This is a nice way to monitor Oracle I/O occurring on various nodes in the cluster. Notice how **mxodmstat** accounts for the I/O operations as either synchronous or asynchronous. More importantly, however, is that MxODM attributes async I/O with service times. Without ODM, asynchronous I/Os are never attributed complete-times by Oracle, as can be readily seen in Statspack reports for such events as *direct path read* or log file *parallel write*. When Oracle links with an ODM library such as the MxODM library, Oracle is given asynchronous I/O complete times and therefore reports such as Statspack benefit from that added value.

This suite of “on-demand” scalability tests proved the invaluable value-add of the Flexible Database Cluster architecture.



```
Linux
racl1 oracle  $ mxodmstat -i60 -I dss1 dss2 dss3 dss4 dss5 dss6
         dss1                      dss2                     dss3                     dss4                     dss5                     dss6
Syn Asy  KB/s Ave m  Syn Asyn   KB/s Ave m  Syn Asyn   KB/s Ave m  Syn Asyn   KB/s Ave m  Syn Asyn   KB/s Ave m  Syn Asyn   KB/s Ave m
  1 552 69688     11    3 1055 132862    11    0    0      0      0    0    0      0      0    0    0      0      0    0    0      0      0
  2 569 71836     11    1  642  80776    11    0    0      0      0    0    0      0      0    0    0      0      0    0    0      0      0
  2 575 72520     11    1  546  68756    11    0    0      0      0    0    0      0      0    0    0      0      0    0    0      0      0
  2 560 70722     11 0.85  348  43814    11    0    0      0      0    0    0      0      0    0    0      0      0    0    0      0      0
  2 571 72084     11 0.90  258  32444    11    0    0      0      0    0    0      0      0    0    0      0      0    0    0      0      0
  1 568 71115     12    3 1118 139035    11    3 2005 249843      6    3 1994 248222      6    0    0      0      0    0    0      0      0
  1 567 70902     12    2  586  72806    11    2 1058 131816      6    2 1052 131008      6    0    0      0      0    0    0      0      0
  2 563 70415     12    2  585  72786    11    2 1051 130988      6    2 1049 130602      6    0    0      0      0    0    0      0      0
  2 561 70254     12    2  553  68794    11    3  993 123726      6    2  987 122878      6    0    0      0      0    0    0      0      0
  1 532 66295     11    2  540  67265    11    2  969 120678      6    2  961 119881      6    0    0      0      0    0    0      0      0
  2 574 71450     11    2  579  72184    12    2 1040 129495      6    2 1031 128814      6    0    0      0      0    0    0      0      0
  1 571 71176     11    1  564  70217    11    2 1011 125904      6    2 1001 125037      6    0    0      0      0    0    0      0      0
  2 568 70846     11 0.80  280  34928    11 0.80  506  62977      6 0.83  503  62785      6    0    0      0      0    0    0      0      0
  1 569 70892     11    1  309  38538    12    1  552  68772      6    1  549  68523      6    0    0      0      0    0    0      0      0
  2 495 59952     12    1  569  70837    11    2 1019 126857      6    2 1013 126563      6 0.37    0     12      4 0.37    0     12      2
  2 578 69950     12    2  484  58295    12    2  809  97787      7    2  790  95617      8    2 0.17     24      6    1 0.17     24      5
  2 574 69490     12    2  555  66899    12    2  925 111824      7    2  911 110285      8    2  959 115745      7    1  611  73878     10
  2 576 69734     12    2  545  65690    13    2  924 111814      7    2  920 111433      7    2  959 115741      7    2  612  73983     10
  2 516 62368     12    2  558  67185    12    2  937 113302      7    2  920 111430      7    2  967 116741      7    1  614  74154     10
  2 549 66340     13    2  511  61502    12    2  851 102989      7    2  840 101755      7    2  873 105424      7    1  551  66535     10
  2 550 66443     13    2  588  70694    12    2  965 117074      7    2  971 117541      7    2  943 114010      7    4  607  73225     10
  2 568 68570     12    2  585  70286    12    2  951 115360      7    2  950 115003      7    2  936 113133      7    2  604  72831     10

racl1 oracle  $ date
Tue Aug 12 12:27:52 EDT 2003

racl1 oracle  $
```

Figure 38

Summary

The Flexible Database Cluster architecture clearly enhances the value of Oracle9*i* Real Applications Clusters. This proof of concept tested the functional benefit of having a very large cluster of dynamic server resources. Oracle9*i* RAC not only endured the rigorous testing but also proved to handle nicely such issues as node failure and dynamically adding instances. Oracle9*i* RAC stands alone as the only database product that can offer such functionality. When deployed using FDC architecture, datacenter deployments of RAC can clearly add to the overall value IT plays in the enterprise.

We feel enterprises will increasingly consider not only the adoption of Oracle9*i* RAC, but also the deployment methodology of Flexible Database Clusters running Linux if they deem that:

- RAC is beneficial overall
- RAC gives better availability and scalability without making changes to their application code
- RAC is easy to manage and flexible enough to grow (both planned and dynamic growth)
- RAC gives favorable TCO

The testing conducted during this proof of concept takes positive steps toward showing that these criteria are indeed met with today's Oracle9i RAC product when combined with very robust clustered Intel-based servers running Linux.

Appendix

cr_dss.sql

```
REM – This script is a great example of what a database needs to consist of to
REM - be used in a Flexible Database Cluster environment. Since this database
REM - has 16 defined threads, any node in a 16 node cluster can boot an
REM - instance of this database.

REM  - This is a really good example of how simplistic, yet correct a database
can be created with Oracle Managed Files.

REM – Note: This will not function with Real Application Clusters unless the
REM - path assigned to the DB_CREATE_FILE_DEST is a cluster filesystem
directory

REM – Put the database in archivelog mode later if desired.

ALTER SYSTEM SET DB_CREATE_FILE_DEST =  'PUT YOUR PATH HERE' ;
create database dss
maxinstances 24
maxdatafiles 16384
maxlogfiles 48
noarchivelog
logfile size 100M, SIZE 100M;
@?/rdbms/admin/catalog
@?/rdbms/admin/catproc
connect / as sysdba
@?/sqlplus/admin/pupbld

ALTER DATABASE ADD LOGFILE THREAD 2 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 2 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 2;

ALTER DATABASE ADD LOGFILE THREAD 3 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 3 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 3;

ALTER DATABASE ADD LOGFILE THREAD 4 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 4 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 4;

ALTER DATABASE ADD LOGFILE THREAD 5 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 5 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 5;

ALTER DATABASE ADD LOGFILE THREAD 6 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 6 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 6;

ALTER DATABASE ADD LOGFILE THREAD 7 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 7 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 7;

ALTER DATABASE ADD LOGFILE THREAD 8 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 8 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 8;

ALTER DATABASE ADD LOGFILE THREAD 9 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 9 SIZE 100M ;
```

```
ALTER DATABASE ENABLE PUBLIC THREAD 9;

ALTER DATABASE ADD LOGFILE THREAD 10 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 10 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 10;

ALTER DATABASE ADD LOGFILE THREAD 11 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 11 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 11;

ALTER DATABASE ADD LOGFILE THREAD 12 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 12 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 12;

ALTER DATABASE ADD LOGFILE THREAD 13 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 13 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 13;

ALTER DATABASE ADD LOGFILE THREAD 14 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 14 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 14;

ALTER DATABASE ADD LOGFILE THREAD 15 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 15 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 15;

ALTER DATABASE ADD LOGFILE THREAD 16 SIZE 100M ;
ALTER DATABASE ADD LOGFILE THREAD 16 SIZE 100M ;
ALTER DATABASE ENABLE PUBLIC THREAD 16;

create undo tablespace rb1 ;
create undo tablespace rb2 ;
create undo tablespace rb3 ;
create undo tablespace rb4 ;
create undo tablespace rb5 ;
create undo tablespace rb6 ;
create undo tablespace rb7 ;
create undo tablespace rb8 ;
create undo tablespace rb9 ;
create undo tablespace rb10 ;
create undo tablespace rb11 ;
create undo tablespace rb12 ;
create undo tablespace rb13 ;
create undo tablespace rb14 ;
create undo tablespace rb15 ;
create undo tablespace rb16 ;

CREATE TABLESPACE TEMP DATAFILE SIZE 10000M TEMPORARY ;
CREATE TABLESPACE TEST DATAFILE SIZE 1000M ;
shutdown
disconnect
exit
```

initprod.ora

```
*.control_files                  = (  /usr1/oracle/rw/DATA/cntlprod_1 )
*.db_name                               = prod
*.CLUSTER_DATABASE = TRUE
*.CLUSTER_DATABASE_INSTANCES = 16
*.remote_listener      = PROD

prod1.INSTANCE_NUMBER = 1
```

```
prod1.THREAD = 1
prod1.INSTANCE_NAME=prod1
prod1.UNDO_TABLESPACE=rb1

prod2.INSTANCE_NUMBER = 2
prod2.THREAD = 2
prod2.INSTANCE_NAME=prod2
prod2.UNDO_TABLESPACE=rb2

prod3.INSTANCE_NUMBER = 3
prod3.THREAD = 3
prod3.INSTANCE_NAME=prod3
prod3.UNDO_TABLESPACE=rb3

prod4.INSTANCE_NUMBER = 4
prod4.THREAD = 4
prod4.INSTANCE_NAME=prod4
prod4.UNDO_TABLESPACE=rb4

prod5.INSTANCE_NUMBER = 5
prod5.THREAD = 5
prod5.INSTANCE_NAME=prod5
prod5.UNDO_TABLESPACE=rb5

prod6.INSTANCE_NUMBER = 6
prod6.THREAD = 6
prod6.INSTANCE_NAME=prod6
prod6.UNDO_TABLESPACE=rb6

prod7.INSTANCE_NUMBER = 7
prod7.THREAD = 7
prod7.INSTANCE_NAME=prod7
prod7.UNDO_TABLESPACE=rb7

prod8.INSTANCE_NUMBER = 8
prod8.THREAD = 8
prod8.INSTANCE_NAME=prod8
prod8.UNDO_TABLESPACE=rb8

prod9.INSTANCE_NUMBER = 9
prod9.THREAD = 9
prod9.INSTANCE_NAME=prod9
prod9.UNDO_TABLESPACE=rb9

prod10.INSTANCE_NUMBER = 10
prod10.THREAD = 10
prod10.INSTANCE_NAME=prod10
prod10.UNDO_TABLESPACE=rb10

prod11.INSTANCE_NUMBER = 11
prod11.THREAD = 11
prod11.INSTANCE_NAME=prod11
prod11.UNDO_TABLESPACE=rb11

prod12.INSTANCE_NUMBER = 12
prod12.THREAD = 12
prod12.INSTANCE_NAME=prod12
prod12.UNDO_TABLESPACE=rb12

prod13.INSTANCE_NUMBER = 13
prod13.THREAD = 13
prod13.INSTANCE_NAME=prod13
prod13.UNDO_TABLESPACE=rb13
```

```
prod14.INSTANCE_NUMBER = 14
prod14.THREAD = 14
prod14.INSTANCE_NAME=prod14
prod14.UNDO_TABLESPACE=rb14
prod15.INSTANCE_NUMBER = 15
prod15.THREAD = 15
prod15.INSTANCE_NAME=prod15
prod15.UNDO_TABLESPACE=rb15

prod16.INSTANCE_NUMBER = 16
prod16.THREAD = 16
prod16.INSTANCE_NAME=prod16
prod16.UNDO_TABLESPACE=rb16

*.UNDO_MANAGEMENT = AUTO

*.parallel_automatic_tuning=true
*.parallel_min_servers=16
*.parallel_max_servers=16

*.cursor_space_for_time          = TRUE  # pin the sql in cache
*.log_parallelism=2
*.pre_page_sga = TRUE

*.compatible = 9.2.0.0.0
*.db_block_size                  = 8192
*.db_files                       = 300
*.db_writer_processes = 1
*.db_16K_cache_size = 30M
*.db_file_multiblock_read_count = 32
log_checkpoint_interval                 = 999999999
optimizer_mode                  = choose

*.processes                      = 600

*.sort_area_size                   = 40000000
*.shared_pool_size               = 300000000
log_archive_start = TRUE
log_archive_dest = /usr1/oracle/arch

*.open_cursors = 255
*.transactions_per_rollback_segment = 1
*.java_pool_size = 8M
```

users.sql

```
column machine format a8
select machine,count(*) from gv$session
group by machine ;
```

instance_status.sql

```
column  host_name format a10
select host_name,thread#,database_status from gv$instance
order by thread# ;
```

tput.sql

```
select sum(PHYRDS) reads,sum(PHYBLKRD * 16 )/1024 readMB,
sum(PHYWRTS) writes,sum(PHYBLKWRT * 16 )/1024 writeMB
from dba_data_files,gv$filestat
```

```
where dba_data_files.file_id = gv$filestat.file#;
```

rac_wait_event.sql

```
select event,count(*) from gv$session_wait where wait_time = '0'
group by event order by 2 desc ;
```

now.sql

```
select to_char(sysdate,'MM/DD/YYYY HH24:MI:SS') "LOCAL_TIME" from dual
```

space.sql

```
column    ts_name format a20
column    total_mb format 999999.99
column    free_mb format 999999.99
column    used_mb format 999999.99
column    pct_free format 999.99
column    max_contig format 9999.99
column    frags format 9999
REM omit rollback tablespaces

 select name ts_name,
    sum(t_bytes)/1024/1024 total_mb,
    sum(f_bytes)/1024/1024 free_mb,
    (sum(t_bytes)-sum(f_bytes))/1024/1024 used_mb,
    (sum(f_bytes)/sum(t_bytes))*100 pct_free,
    sum(mf_bytes)/1024/1024 max_contig,
    sum(nfrags) frags
from
    (
        select        df.tablespace_name name,
                      sum(df.bytes) t_bytes,
                      0 f_bytes,
                      0 mf_bytes,
                      0 nfrags
        from          sys.dba_data_files df
        where         df.tablespace_name not like 'RB%'
        group by      df.tablespace_name
        union all
        select        fs.tablespace_name,
                      0,
                      sum(fs.bytes),
                      max(fs.bytes),
                      count(*)
        from          sys.dba_free_space fs
        where         fs.tablespace_name not like 'RB%'
        group by      fs.tablespace_name
    )
group by name
order by name;
```

Author: Kevin Closson, Principal Engineer, PolyServe Inc.

Acknowledgements: A very special thanks Madhu Tumma for his insightful consultation and broad I.T. knowledge. Thanks to Martha Centeno and Phil Horwitz of IBM for their system support and understanding of the IBM test cluster. No project this big is possible without Project Management so a hearty thanks to David Dougherty. Thanks to Mike Zuhl for his fine work on MxODM. Thanks to Janelle Adams for her technical writing efforts. And finally, thanks to IBM corporation for sponsoring this proof of concept.

PolyServe, Inc.
20400 NW Amberwood Drive
Suite 150
Beaverton, OR 97006

(877) 476-5973 (within U.S.)
(503) 617-7592 (fax)
(503) 617-7574 (outside U.S.)
www.polyserve.com

FDC_paper 091093 IBM markup