"The Tens": Creating a Very Large Database on Intel-based Servers with Oracle9i Real Application Clusters

A PolyServe and XIOtech White Paper

June 2002

POLYSERVE™

XIOtech® A Seagate Company

This proof-of-concept was verified by Carl Olofson of IDC. IDC Analyze the Future

Abstract

The tremendous horizontal scaling characteristics of Oracle9i Real Application Clusters (RAC) redefines the term Very Large Database computing (VLDB). For years VLDB has referred mostly to the bulk of data stored in a database, but the number of users that could simultaneously access a VLDB was very seldom addressed. With RAC, extremely large databases can be concurrently accessed by a number of users that rivals that supported by a mainframe. This paper is based upon a real-world case study of a 10-node Intel-based server cluster running Oracle9i RAC on Linux serving 10 terabytes of data to 5,000 concurrent users.

"It would be fine putting 2 terabytes of data into Oracle if only 50 or 100 people are accessing the data, but you'd need something like the S/390 if you have 5,000 users all asking questions of the database."

The Meta Group Feb 2001

1 INTRODUCTION

The term "Very Large Database" (VLDB) has been redefined many times over the last several years, but as is suggested in the above quote by The Meta Group, the prevailing opinion is that a VLDB belongs in a proprietary environment. In its origin, the term VLDB was reserved for those database environments having at least 1 Terabyte of online data and at least 100 concurrent users. When combined, these qualities have traditionally narrowed the choice of hardware technology into a very small camp - Mainframes and Massively Parallel Processing systems. An alternative approach to deploying VLDBs – though no less proprietary - was specialized database/hardware combinations such as Teradata.

Oracle and Symmetric MultiProcessing (SMP) hardware changed the rules in the 1990's and Oracle has changed the rules again with Oracle9i Real Application Clusters. Oracle9i Real Application Clusters brings commodity-based server solutions into the VLDB arena, and in fact even expands the definition.

2 GETTING THERE FROM HERE

In the early 1990's, Oracle and several early pioneers of SMP systems changed the VLDB landscape.

Significant advancements in Oracle have, step-by-step, increased the viability of deploying VLDBs on SMP platforms. Perhaps most notable of VLDB-enabling Oracle releases was Oracle7 version 7.1, which featured the Parallel Query Option (PQO).

Without PQO, the execution time of a query was ultimately determined by the execution capability of only one processor in a large SMP. PQO, on the other hand, decomposes a query into portions that can be executed in parallel, even across nodes of a cluster in a Parallel Server environment[1]. The technology, however, was not quite complete. Indeed, on most platforms Oracle's maximum supported file size was 2GB. As such, a 1TB database would contain some 512 data files – more files than a process could simultaneously open on most platforms. At this release level of Oracle, it was also mandatory to perform backup operations on an entire table, which often produced an unacceptable performance impact.

Oracle8 was the first "real" VLDB release of Oracle. Gone with this release was the requirement for table-level backup and restore operations. Oracle8

[1] Intra-node Parallel Query (IPQO) requires Oracle Parallel Server (Oracle8i and older) or Oracle9i Real Application Clusters.

included table partitioning functionality, which greatly enhances management and query processing in a VLDB environment. With table partitioning, one can simply backup or restore a partition of a table as granular as the Database Administrator so desires. Table partitioning also enhanced query processing through dataset (partition) elimination. With data placed into table partitions, the cost based optimizer often produces a query plan that completely eliminates the need to read entire portions of a table. This was significant.

Oracle8i brought intrinsic Java technology and a rich set of tools that facilitated ease of development and deployment of Web-based applications. To that end, the user count for a given application could number in the tens, if not hundreds of thousands of users and with such user communities come even larger VLDBs.

With Oracle9i the VLDB bar has been raised again. Oracle9i introduced built-in OLAP functionality and enhanced ETL features such as External Tables and the MERGE statement.

Also new with Oracle9i is the data file management feature called Oracle Managed Files. This feature is by far the greatest Oracle enhancement to date in managing the physical files that compose an Oracle database.

The culmination of this technology coupled with SMP advancements has put to rest the question of Oracle's place in the VLDB environment. Not only does Oracle belong, but arguably offers the overall best software for a VLDB deployment.

One significant issue associated with deploying a VLDB remained, however – system cost.

3 CLUSTERED COMMODITY SERVERS – A LOW-COST VLDB APPROACH

In addition to the previously mentioned Oracle9i VLDB features, Oracle9i also includes Real Application Clusters (RAC) Technology. With RAC, several aspects of VLDB change. Most notably gone is the trade-off of users for database size. Historically, VLDB environments either had a very large data set or a very large number of users, but seldom both.

With Oracle9i RAC, with its breakthrough Cache Fusion technology, it is possible to scale a VLDB to many nodes of a clustered system. Scaling an application across nodes as such is commonly referred to as *horizontal scaling.* Horizontal scaling differs from the traditional SMP scaling, or *vertical scaling,* in many ways. Although the core of a horizontally scaled application is usually a small SMP system, adding processors is done by economically adding nodes to the cluster as opposed to adding processors to the system. Why is this approach economical? First, the ability to use standard 2 and 4-way Intel-based servers dramatically reduces hardware costs when compared with a single UNIX SMP server. Second, the horizontal scaling approach enables data center managers to deploy only the hardware needed to support the initial demand for the application. The application can be easily scaled later without bringing the application down.

Vertically scaling a system beyond its CPU capacity requires the replacement of the server chassis – sometimes referred to as a “forklift upgrade”. This issue can be avoided by purchasing a system large enough to handle predicted demand for the application. However, if the application initially only demands a few processors and the predicted demand is high it is extremely difficult to justify the cost of the required larger server chassis. A chassis upgrade also typically requires planned downtime to perform the operation. Horizontal scaling on the other hand enables the application to be scaled up as it grows (or scaled down as it shrinks). Nodes can be added or removed from the server cluster with no changes to the physical data layout or application required.

Above all, the relentlessly increasing performance of Intel processors and accompanying hardware has made standard servers remarkably less expensive than their legacy RISC counterparts per unit of performance delivered.

Oracle9i RAC harnesses the power of these clustered standard servers. Without database software that scales horizontally such as Oracle9i RAC, the economic and high-availability advantages of commodity-based cluster database computing would not be attainable. Furthermore, database clusters are an essential element of what is being called the “flexible server farm”. Such farms, based on cost-effective Intel-based servers and shared storage, are effective platforms for running 3-tier applications, mail servers, NFS and CIFS file servers in the corporate data center.

The question then shifts to whether the hardware and operating system of a commodity-based database clustering solution is sufficiently robust to handle high-end VLDB requirements. Just as important is the question of whether such a system is manageable.

4 “The Tens” – A VLDB PROOF OF CONCEPT ON A CLUSTER OF INDUSTRY STANDARD SERVERS

Manageability is the key to deploying a VLDB on a clustered system. It would be essentially impossible to deploy and manage a VLDB on a cluster of servers without a robust cluster file system into which the database files and application products can be placed. PolyServe Matrix Server is an example of such a cluster file system.

PolyServe Matrix Server is a highly reliable and scalable cluster file system that gives multiple servers scalable concurrent read and write access to data on a SAN. Matrix Server provides the manageability benefits of deploying Oracle9i RAC on a file system with the performance of running on raw devices. Matrix Server is tightly integrated with Matrix HA, PolyServe’s high availability software product that monitors application and hardware health and provides automated application failover for high application availability and reliability.

In addition to a robust cluster file system, the ability to effectively deploy and manage a VLDB requires storage which is also flexible and easy to manage. Virtualization and advanced storage management tools are essential for administering real-world VLDBs. XIOtech’s MAGNITUDE™ Hardware Platform and the REDI™ Software Family provide these features and are an excellent fit for a VLDB environment.

Because the distributed database and cluster file system technologies required for a VLDB implementation on clustered Intel-based servers have only recently become available, PolyServe, Inc and XIOtech Corporation, a Seagate Company, decided to create a proof of concept to show that such a VLDB deployment is possible. Using currently available products the two companies have partnered to perform a significant proof of concept showing that a cluster of standard servers running Linux and Oracle9i RAC can meet the requirements of a VLDB environment.

The test consisted of directly connecting 5,000 concurrent pseudo-users with online access to a VLDB – a 10TB Order Entry and Product Tracking application. A 10 node dual Intel CPU server cluster running Redhat Linux version 7.2 was used for the deployment. The storage system consisted of 5 XIOtech MAGNITUDE storage arrays with 4 expansion cabinets connected to a Brocade 2800 Fibrechannel switch. PolyServe Matrix Server cluster file system and Matrix HA software was used to manage the cluster. Because the proof-of-concept used ten servers and a 10TB database, the project became known as "The Tens".

5 "TENS" TEST OVERVIEW

The Tens test is described in five parts:

- **Database**. This section covers the key aspects of the database such as schema, row counts and index types. Oracle9i advanced file management features used are also covered.

- **Hardware**. This section details the XIOtech MAGNITUDE disk storage subsystem and the ten node Intel-based server cluster.

- **Platform Software**. This section describes the Operating System revision, and the PolyServe Matrix Server cluster file system.

- **Test Application Workload**. This section describes the types of transactions applied to the system under test.

- **Test Methodology and Performance Analysis**. This section describes the performance metrics and results of the various user counts tested.

6 DATABASE

6.1 Database Schema

VLDB case studies are generally based on data warehousing schemas. Historically, one of the only realistic sources of data for a VLDB was not transactional, but instead consisted of data extracted from the transactional schemas, transformed and loaded. Indeed, until the advent of the worldwide web and e-commerce, it was very unlikely that a transactional system would warrant the label VLDB. With the web, however, emerged retailers offering large numbers of products to tens, and sometimes, hundreds of millions of customers.

These millions of customers have products on order and a good deal of customer-oriented state saved in the database such as favorites lists, credit information, account history and so forth. Furthermore, these hundreds of millions of users “shop” at all hours of the day and their loyalty is only “a single click away”. For this reason, The Tens project was centered on a VLDB transactional system that added in yet another aspect of VLDB – very large user count. At a high level, the database schema contained the following application tables:

1. **Customers.** The database contained over 160 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 was indexed with a unique index on the custid column and a non-unique index on the name column.

2. **Orders.** The database contained an orders table with 840 million rows of data. The orders table had a unique composite index on the custid and order id columns.

3. **Line Items.** Simulating a customer base with complex transactions, the Line item table contains as many as 150 line items per order. The item table had a 3-way unique composite index on custid, ordid and itemid.

4. **Historical Line Items.** This table is maintained for OLAP purposes based on customer activity. The Historical Line Item table contained over 41 Billion rows.

5. **Product.** The product 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 were over 40 Million products. The product table was indexed with a unique index on its prodid column.

6. **Warehouse.** The warehouse 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 was indexes with a unique composite index of two columns.

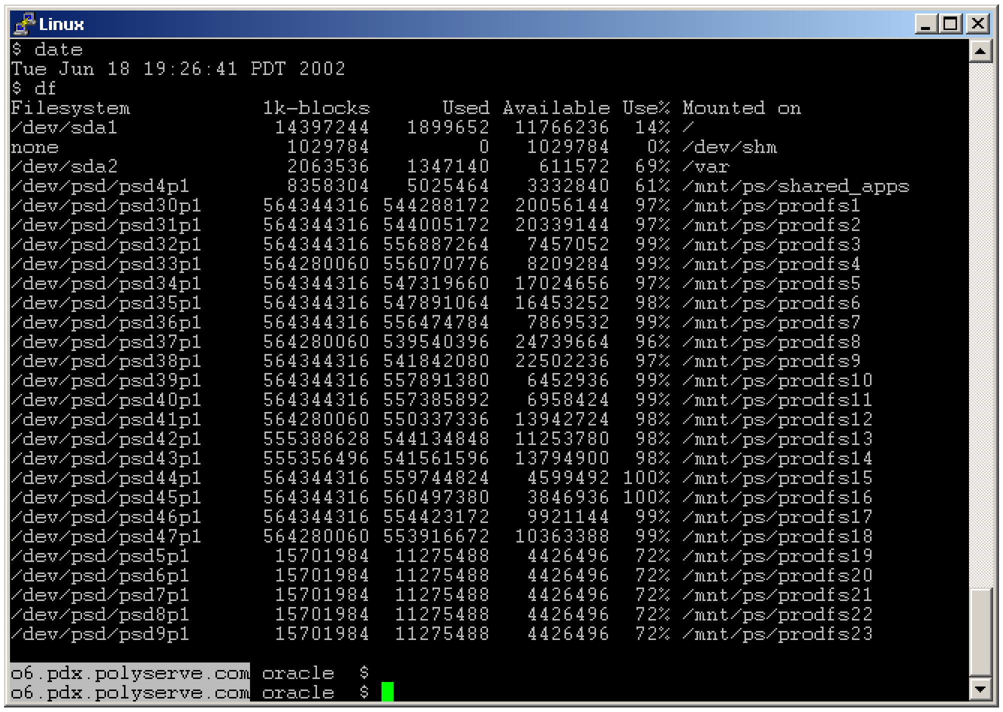
6.2 Table Partitioning and Oracle Managed Files

As is common practice in managing a VLDB, the test database was built with table partitioning[2]. Combining table partitioning with the Oracle Managed Files (OMF) feature greatly eases the management pains commonly endured in a VLDB environment. OMF is only supported on file systems – hence the use of the PolyServe Matrix Server cluster file system for this test.

Since the database was created in cluster file systems, it is fairly easy to visualize the physical layout. At the very top level, the physical disk space was virtualized and carved into 18 LUNs that were roughly 550GB each. There were

[2] Oracle partitioning is implemented through splitting tables or indexes across multiple tablespaces. Tablespaces can contain partitions of multiple objects.

5 additional small file systems of roughly 15GB used mainly for sort space. The main emphasis of this paper is upon the usage of the 18 large file systems. Figure 6.1 contains the output of the Linux df(1) command showing the file systems.



```
$ date
Tue Jun 18 19:26:41 PDT 2002
$ df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda1             14397244   1899652  11766236  14% /
none                   1029784         0   1029784   0% /dev/shm
/dev/sda2              2063536   1347140    611572  69% /var
/dev/psd/psd4p1        8358304   5025464   3332840  61% /mnt/ps/shared_apps
/dev/psd/psd30p1     564344316 544288172  20056144  97% /mnt/ps/prodfs1
/dev/psd/psd31p1     564344316 544005172  20339144  97% /mnt/ps/prodfs2
/dev/psd/psd32p1     564344316 556887264   7457052  99% /mnt/ps/prodfs3
/dev/psd/psd33p1     564280060 556070776   8209284  99% /mnt/ps/prodfs4
/dev/psd/psd34p1     564344316 547319660  17024656  97% /mnt/ps/prodfs5
/dev/psd/psd35p1     564344316 547891064  16453252  98% /mnt/ps/prodfs6
/dev/psd/psd36p1     564344316 556474784   7869532  99% /mnt/ps/prodfs7
/dev/psd/psd37p1     564280060 539540396  24739664  96% /mnt/ps/prodfs8
/dev/psd/psd38p1     564344316 541842080  22502236  97% /mnt/ps/prodfs9
/dev/psd/psd39p1     564344316 557891380   6452936  99% /mnt/ps/prodfs10
/dev/psd/psd40p1     564344316 557385892   6958424  99% /mnt/ps/prodfs11
/dev/psd/psd41p1     564280060 550337336  13942724  98% /mnt/ps/prodfs12
/dev/psd/psd42p1     555388628 544134848  11253780  98% /mnt/ps/prodfs13
/dev/psd/psd43p1     555356496 541561596  13794900  98% /mnt/ps/prodfs14
/dev/psd/psd44p1     564344316 559744824   4599492 100% /mnt/ps/prodfs15
/dev/psd/psd45p1     564344316 560497380   3846936 100% /mnt/ps/prodfs16
/dev/psd/psd46p1     564344316 554423172   9921144  99% /mnt/ps/prodfs17
/dev/psd/psd47p1     564280060 553916672  10363388  99% /mnt/ps/prodfs18
/dev/psd/psd5p1       15701984  11275488   4426496  72% /mnt/ps/prodfs19
/dev/psd/psd6p1       15701984  11275488   4426496  72% /mnt/ps/prodfs20
/dev/psd/psd7p1       15701984  11275488   4426496  72% /mnt/ps/prodfs21
/dev/psd/psd8p1       15701984  11275488   4426496  72% /mnt/ps/prodfs22
/dev/psd/psd9p1       15701984  11275488   4426496  72% /mnt/ps/prodfs23

o6.pdx.polyserve.com oracle  $
o6.pdx.polyserve.com oracle  $
```

Figure 6.1 – File Systems Used for the VLDB

6.3 Oracle9i Databases with OMF – Not Just a Huge Collection of Files

Combining table partitioning with Oracle Managed Files (OMF) allows a Database Administrator to think of a VLDB as a set of file groupings, or "segments", as opposed to a complex set of raw partition pathnames. Using the file system directory hierarchy, a very straightforward storage layout is easily achieved.

To illustrate such hierarchy, consider The Tens database which had each of the *small* tables and indexes partitioned into 18 partitions – one in each of the 18 file systems. As will be discussed in detail later, the Item and Line Item History tables were partitioned into 360 partitions – 20 in each of the 18 PolyServe Matrix Server file systems. Figure 6.2 illustrates the file system hierarchy used in this project.

Each file system, prodfs1-prodfs18, had a DATA and INDEX subdirectory. In the subdirectories DATA and INDEX, there was a subdirectory named according to the database object stored therein. For example, note in Figure 6.2 that under prodfs1/DATA there is a CUST_1 and ITEM_1 subdirectory holding 1/18th of their respective objects. As such, the file system path "/mnt/ps/prodfs1/DATA" can be thought of as a "segment" of the database under which 1/18th of all tables are stored. Likewise, under "/mnt/ps/prodfs1/INDEX" can be found exactly 1/18th of all indexes in the database. This methodology is very clean and manageable.

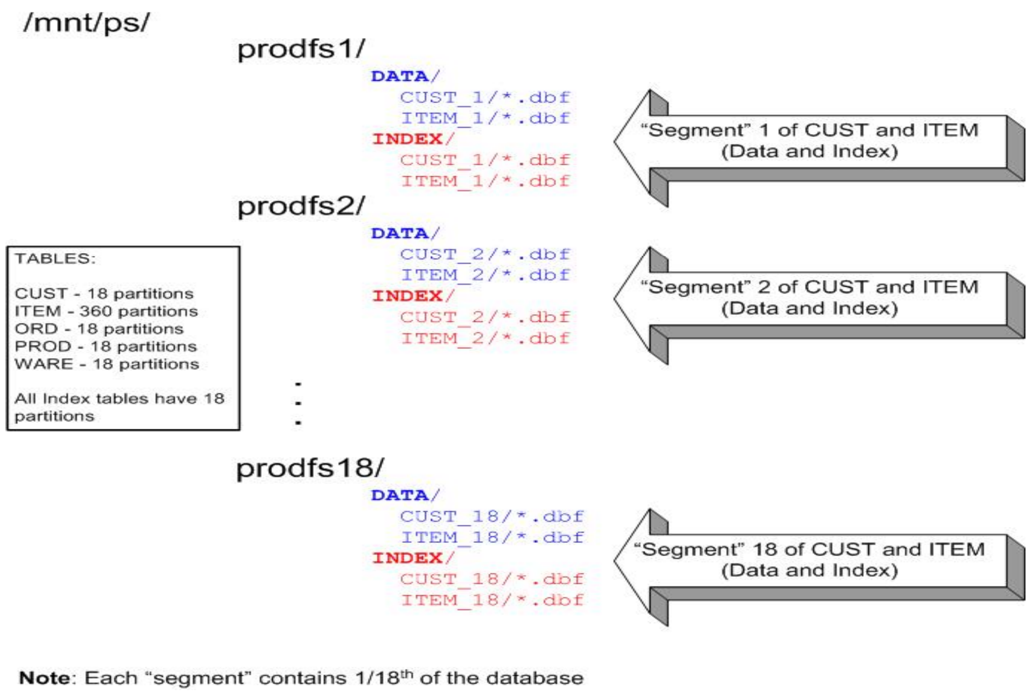


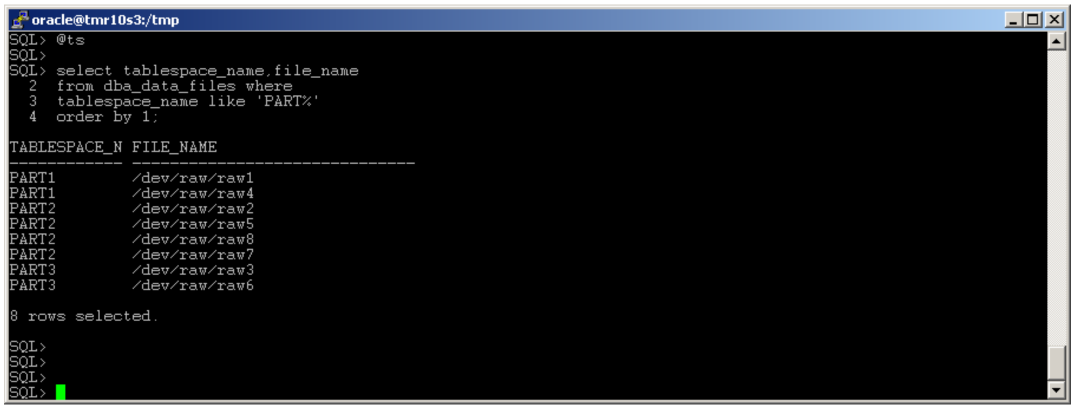
Figure 6.2 - Directory hierarchy using PolyServe Matrix Server cluster file system

This partitioning scheme bodes well for partition level operations such as backup and restore. For instance, the CUSTOMER, ORDERS, PRODUCT and WAREHOUSE tables and indexes were each partitioned into 18 file systems. To illustrate the simplification of a partition backup with OMF, consider first the normal process of backing up a major section of a large database that consists of a bunch of raw partitions or raw volumes.

After placing the partitions (tablespaces) into backup mode, the Database Administrator must perform a query of the database to get the list of full path names to the datafiles the partitions consist of – after all, the files are just slices of disk and do not really possess application attributes. That is, a filename such as "/dev/raw/raw37" is not intuitively associated with a tablespace, such as "ORDERS". From there, a series of commands must be executed to open the files and then transfer the data to tape with a tool such as the Linux dd(1) command[3] or a third party backup product.

Figure 6.3 and Figure 6.4 illustrate the commands required to backup the datafiles that comprise "Partition 2" of an example non-OMF database.

[3] Tools such as dd(1) are necessary for reading and writing raw partitions due to alignment and blocking requirements. Such simplified file utilities as tar(1) and cpio(1) cannot manipulate raw partitions or volumes.



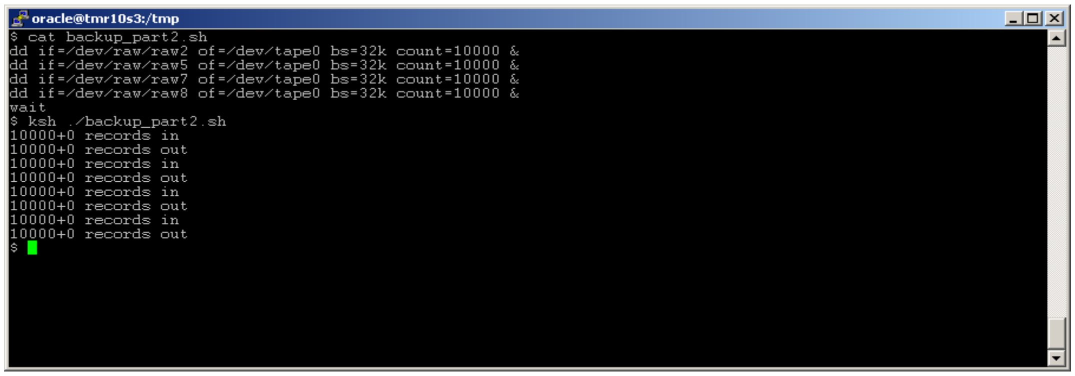
```
SQL> @ts
SQL>
SQL> select tablespace_name,file_name
  2  from dba_data_files where
  3  tablespace_name like 'PART%'
  4  order by 1;

TABLESPACE_N FILE_NAME
------------ ------------------------------
PART1        /dev/raw/raw1
PART1        /dev/raw/raw4
PART2        /dev/raw/raw2
PART2        /dev/raw/raw5
PART2        /dev/raw/raw8
PART2        /dev/raw/raw7
PART3        /dev/raw/raw3
PART3        /dev/raw/raw6

8 rows selected.

SQL>
SQL>
SQL>
SQL>
```

Figure 6.3 - How to determine what files are parts of an Oracle Tablespace



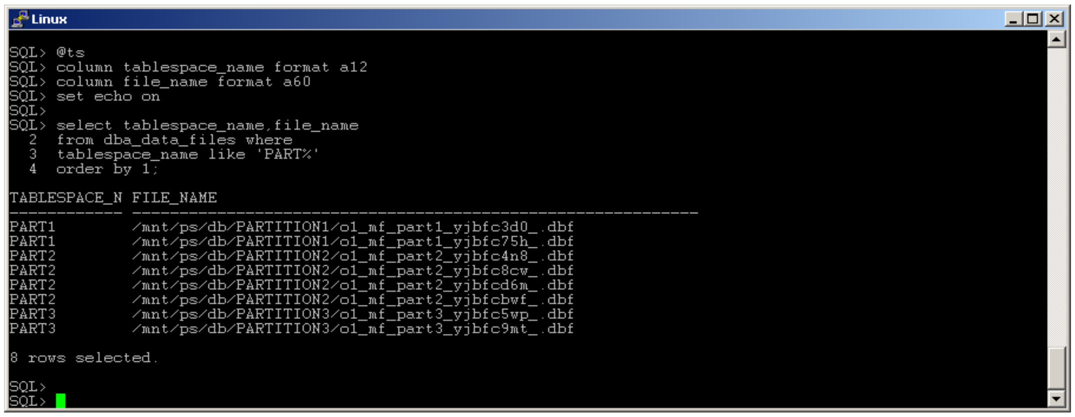
```
$ cat backup_part2.sh
dd if=/dev/raw/raw2 of=/dev/tape0 bs=32k count=10000 &
dd if=/dev/raw/raw5 of=/dev/tape0 bs=32k count=10000 &
dd if=/dev/raw/raw7 of=/dev/tape0 bs=32k count=10000 &
dd if=/dev/raw/raw8 of=/dev/tape0 bs=32k count=10000 &
wait
$ ksh ./backup_part2.sh
10000+0 records in
10000+0 records out
10000+0 records in
10000+0 records out
10000+0 records in
10000+0 records out
10000+0 records in
10000+0 records out
$
```

Figure 6.4 - Example of inflexibility of Raw partitions/Volumes and performing a backup

In contrast to the non-OMF case, which requires a data dictionary query to associate files to tablespaces, Figure 6.5 shows how an OMF database in a file system reduces complexity. Each of the partitions are located at the end of directory path names that are readily understood – at least much more so than “/dev/raw/raw3”. For example, finding all the datafiles that comprise “Partition 2” of a given table is illustrated with a simple ls(1) command.

Fundamentally, OMF allows the Database Administrator to think of directories as “containers” or “segments”, as it were, and the files in the containers as “components” of the database. In fact, with OMF it is not even required to name the files, Oracle can eliminate that unnecessary detail as well by supplying a file name guaranteed to be unique and associated, by name, with a particular database[4]. In a pinch, the datafiles can be found by simply navigating the file system pathnames. Figure 6.6 illustrates the ease of backing up “Partition 2” with OMF. A simple tar(1) command of the files in the /mnt/ps/db/PART2 directory is sufficient.

[4] OMF file naming is an excellent way to prevent accidentally associating a file with the wrong database. The OMF name itself prevents such mistakes.



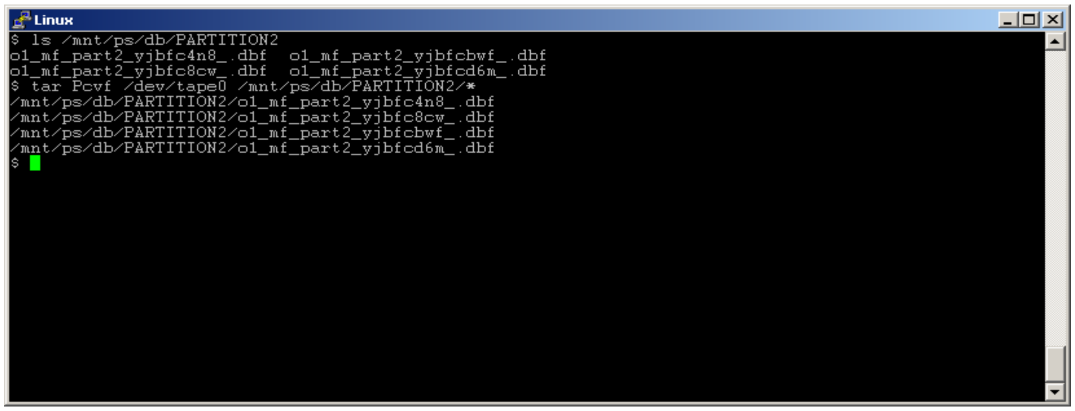
```
SQL> @ts
SQL> column tablespace_name format a12
SQL> column file_name format a60
SQL> set echo on
SQL>
SQL> select tablespace_name,file_name
  2  from dba_data_files where
  3  tablespace_name like 'PART%'
  4  order by 1;

TABLESPACE_N FILE_NAME
------------ ------------------------------------------------------------
PART1        /mnt/ps/db/PARTITION1/o1_mf_part1_yjbfc3d0_.dbf
PART1        /mnt/ps/db/PARTITION1/o1_mf_part1_yjbfc75h_.dbf
PART2        /mnt/ps/db/PARTITION2/o1_mf_part2_yjbfc4n8_.dbf
PART2        /mnt/ps/db/PARTITION2/o1_mf_part2_yjbfc8cw_.dbf
PART2        /mnt/ps/db/PARTITION2/o1_mf_part2_yjbfcd6m_.dbf
PART2        /mnt/ps/db/PARTITION2/o1_mf_part2_yjbfcbwf_.dbf
PART3        /mnt/ps/db/PARTITION3/o1_mf_part3_yjbfc5wp_.dbf
PART3        /mnt/ps/db/PARTITION3/o1_mf_part3_yjbfc9mt_.dbf

8 rows selected.

SQL>
SQL>
```

Figure 6.5 - OMF files in the Oracle data dictionary



```
$ ls /mnt/ps/db/PARTITION2
o1_mf_part2_yjbfc4n8_.dbf  o1_mf_part2_yjbfcbwf_.dbf
o1_mf_part2_yjbfc8cw_.dbf  o1_mf_part2_yjbfcd6m_.dbf
$ tar Pcvf /dev/tape0 /mnt/ps/db/PARTITION2/*
/mnt/ps/db/PARTITION2/o1_mf_part2_yjbfc4n8_.dbf
/mnt/ps/db/PARTITION2/o1_mf_part2_yjbfc8cw_.dbf
/mnt/ps/db/PARTITION2/o1_mf_part2_yjbfcbwf_.dbf
/mnt/ps/db/PARTITION2/o1_mf_part2_yjbfcd6m_.dbf
$
```

Figure 6.6 - OMF simplicity – using ordinary Linux file system utilities to backup an Oracle partition

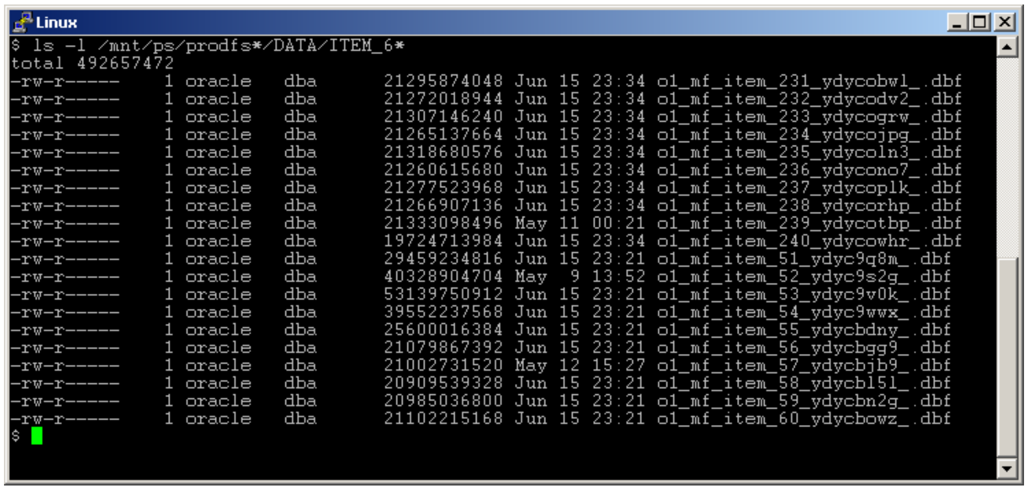
Forcing a Database Administrator to think about tablespaces as a collection of files takes time away from actual database administration. The "contents" of the files is the actual database. Deploying with OMF frees up some of the administration effort from physical administration and allows concentration on logical administration – exactly where the biggest return on DBA effort lies. For instance, why fuss over what sectors of disk a file system file consists of? That particular detail is the responsibility of the Operating System. Likewise, Database Administrators can relinquish data file management to OMF.

A better example of the benefits of OMF is The Tens database itself. The previous points made about OMF are amplified in a VLDB situation. The large tables in The Tens database had to be partitioned into 360 partitions – mainly due to the Oracle data file size limit of 64GB[5] on Linux.

[5] The maximum block size with Oracle9i Release 9.0.1.3 on Linux was 16KB. An Oracle9i data file on Linux is limited to 2^22 such blocks resulting in a 64GB limit. For most RISC-based systems, the file size limit is 128GB.

The combined Line Item and Item History tables required 7TB of gross storage. The goal of configuring space for these tables was equal placement into the LUNs for balanced I/O, dataset elimination[6] and the ability for the data files to grow through AUTOEXTEND. To that end, a scheme was chosen that established 20 table partitions in each file system for a grand total of 360 partitions. The tables were partitioned by Key Range; therefore this strategy nicely accommodates data skew. That is, it was not completely known beforehand whether all partitions would load evenly due to the randomness of the test-data generator. Data skew is yet another reason Oracle Manage Files are necessary.

A great example of such skew is a very active subset of customers each having inordinately large amounts of historical data. This might be caused when a set of new customers, all of whom place a large number of orders, is acquired during a store sale. Without any data skew, the Item and Item History tables for The Tens would have required 360 partitions at roughly 20GB each. In the actual test however there was substantial data skew in ITEM partitions 52 through 54. Whereas the average size of ITEM partitions was roughly 20GB, Figure 6.7 shows that the data files for partitions 52 through 54 ranged between 39GB and 53GB. This is a key motivator for deploying with Oracle Managed Files. Without OMF, the administrator would have had to know in advance that these partitions would contain twice as much data, or worse yet, suffer a space allocation failure and add space to the partition manually.



```
$ ls -l /mnt/ps/prodfs*/DATA/ITEM_6*
total 492657472
-rw-r-----    1 oracle   dba      21295874048 Jun 15 23:34 o1_mf_item_231_ydycobwl_.dbf
-rw-r-----    1 oracle   dba      21272018944 Jun 15 23:34 o1_mf_item_232_ydycodv2_.dbf
-rw-r-----    1 oracle   dba      21307146240 Jun 15 23:34 o1_mf_item_233_ydycogrw_.dbf
-rw-r-----    1 oracle   dba      21265137664 Jun 15 23:34 o1_mf_item_234_ydycojpg_.dbf
-rw-r-----    1 oracle   dba      21318680576 Jun 15 23:34 o1_mf_item_235_ydycoln3_.dbf
-rw-r-----    1 oracle   dba      21260615680 Jun 15 23:34 o1_mf_item_236_ydycono7_.dbf
-rw-r-----    1 oracle   dba      21277523968 Jun 15 23:34 o1_mf_item_237_ydycoplk_.dbf
-rw-r-----    1 oracle   dba      21266907136 Jun 15 23:34 o1_mf_item_238_ydycorhp_.dbf
-rw-r-----    1 oracle   dba      21333098496 May 11 00:21 o1_mf_item_239_ydycotbp_.dbf
-rw-r-----    1 oracle   dba      19724713984 Jun 15 23:34 o1_mf_item_240_ydycowhr_.dbf
-rw-r-----    1 oracle   dba      29459234816 Jun 15 23:21 o1_mf_item_51_ydyc9q8m_.dbf
-rw-r-----    1 oracle   dba      40328904704 May  9 13:52 o1_mf_item_52_ydyc9s2g_.dbf
-rw-r-----    1 oracle   dba      53139750912 Jun 15 23:21 o1_mf_item_53_ydyc9v0k_.dbf
-rw-r-----    1 oracle   dba      39552237568 Jun 15 23:21 o1_mf_item_54_ydyc9wwx_.dbf
-rw-r-----    1 oracle   dba      25600016384 Jun 15 23:21 o1_mf_item_55_ydycbdny_.dbf
-rw-r-----    1 oracle   dba      21079867392 Jun 15 23:21 o1_mf_item_56_ydycbgg9_.dbf
-rw-r-----    1 oracle   dba      21002731520 May 12 15:27 o1_mf_item_57_ydycbjb9_.dbf
-rw-r-----    1 oracle   dba      20909539328 Jun 15 23:21 o1_mf_item_58_ydycbl5l_.dbf
-rw-r-----    1 oracle   dba      20985036800 Jun 15 23:21 o1_mf_item_59_ydycbn2g_.dbf
-rw-r-----    1 oracle   dba      21102215168 Jun 15 23:21 o1_mf_item_60_ydycbowz_.dbf
$
```

Figure 6.7 - OMF accommodates data skew in Key-Range partitioned tables

7 HARDWARE DESCRIPTION

7.1 High-Level overview

The system created for this proof of concept was formed from several key technologies. A system as complex as The Tens requires state of the art

[6] Oracle partitioning by Key Range enables the query optimizer to eliminate partitions that *cannot* have data for a given query

technology such as intelligent RAID storage, advanced database management technology and clustering. Figure 7.1 illustrates the system at a high level.

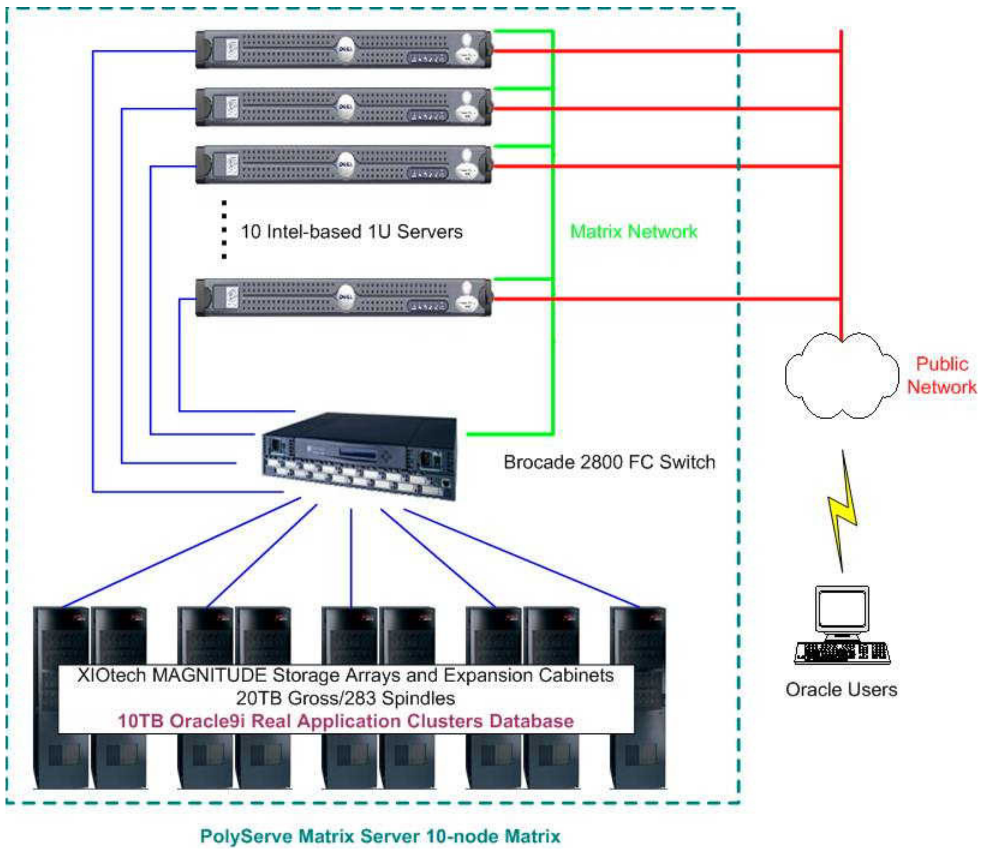

10 Intel-based 1U Servers
Matrix Network
Public Network
Brocade 2800 FC Switch
Oracle Users
XIOtech MAGNITUDE Storage Arrays and Expansion Cabinets
20TB Gross/283 Spindles
10TB Oracle9i Real Application Clusters Database
PolyServe Matrix Server 10-node Matrix

Figure 7.1 - Top-level overview of The Tens test system

7.2 Storage Simplicity: A VLDB Requirement

7.2.1 The Complexity of Software RAID

The discipline of creating Very Large Databases is relatively mature. VLDBs have been deployed on widely varying types of storage systems ranging from disk farms of JBOD (Just a Bunch of Disk) disks managed with software RAID to very high-end proprietary intelligent storage arrays. While the benefits and pitfalls of the various storage technology and methodology can be argued, one common aspect is considered irrefutable – the data must be redundant. To that end, the JBOD/Software RAID approach relies on software that executes on the database server itself to maintain a mirrored[7] strategy for writing data. With such software, disks are typically "imported" into a volume manager and known by their World Wide Name. The disk is then typically divided into "subdisks" that are mirrored for redundancy to "subdisks" on another spindle elsewhere in the storage subsystem. Collectively, both sides of the mirror are

[7] Non-mirrored RAID approaches are acknowledged, but not generally useful.

considered a “volume” and any write on one side of the mirror must also occur on the other side.

For greater availability of data, the typical selection criteria for subdisks mirroring policy is one that protects from losing both sides of a mirrored volume due to a failure in a disk enclosure, for instance a power loss. To alleviate such risk, volumes are typically mirrored to spindles in another disk enclosure and serviced by separate power and so on. The pitfall of creating a highly available RAID storage subsystem with JBOD components and software RAID is basically summed up in one word – complexity.

This complexity becomes apparent as Storage Administrators struggle to deal with the convention, policies and nomenclature required to maintain a very large software RAID system. Historically this complexity was an acceptable trade-off compared to the cost of intelligent hardware RAID arrays - and systems were generally quite small. To put this in perspective, in the early 1990’s, databases on the order of 100GB were considered quite large and seldom consisted of more than 50 data files.

To ensure balanced I/O, each volume would consist of a subdisk from perhaps as many as 50 disk drives. Each data file, therefore, required at least 50 volume manager configuration objects. With 50 files, the total configuration would consist of roughly 2500 objects. These configurations were cumbersome, but not terribly difficult for Storage Administrators to manage.

Times have changed and so has data volume and availability expectations. With advanced database technology such as table partitioning, database administrators are compelled to deploy databases that consist of hundreds upon hundreds of datafiles.

For example, in order to partition a hypothetical 1TB table containing 5 years of sales history data by month, the table would be partitioned into some 60 partitions (files) at a minimum. If all partitions were relatively the same size, it would require 60 files of roughly 16GB each to derive the 1TB. Historically this presented a problem because many system vendors limited file size support to 4GB - in which case each partition of the historical sales table would likely consist of 4 concatentated 4GB files. The product is a grand total file count of 240. Herein lies the difficulty of software RAID.

To achieve data availability and balanced I/O, the storage administrator faces a significant challenge – as does the software RAID product itself! Given 36GB disk drives, and mirroring overhead raising gross capacity to 2TB, the required physical disk count would be 60. In order to spread I/O operations over maximum spindles, each “side” of the mirrored volumes would consist of some 30 subdisks, therefore each volume would consist of some 60 configuration objects. With 240 datafiles, the total number of software RAID configuration objects would be on the order of 14,400 – essentially substituting complex raw disk management with complex volume management.

While there are software RAID products that “can do” this, the important question is why deal with the complexity? Moreover, the complexity of software RAID often times results in a near “Data Mining” level effort to associate what spindles are a part of what database object. Finally, a 1TB Historical Sales table is a good example to describe the problems associated with software RAID strategies, but what about a ***large*** database?

"The Tens" project, being tenfold larger than the Historical Sales table example above, was simply not a candidate for software RAID. As described previously in this paper, the table partitioning strategy employed in The Tens project yielded an Oracle9i data file count of 534[8]. Given the gross disk space requirement of 20TB, using state of the art 73GB drives would yield a spindle count of 280. To achieve availability and balanced I/O[9] for each of these files using Software RAID is just too combursome. Each of the 534 files would require a volume with 280 subdisks mirrored to another 280 subdisks producing an approximated volume configuration database of more then 299,040 objects!

Some systems are simply too large for old methodology - and old technology.

7.2.2 The Simplicity of Hardware Storage Virtualization

XIOtech, with their MAGNITUDE products, provides hardware volume management, removing the problems that are associated with software volume management. XIOtech's true virtualization technology stripes the data across all available spindles, utilizing all storage capacity in the system. This effectively removes the limitations imposed by fixed RAID sets, both in server-attached and aggregated storage technology.

The MAGNITUDE hardware platform conforms to open systems standards. The virtualization process is transparent to the system, which simply sees a large storage pool that can be carved up and allocated as needed into LUNS. This eliminates the physical aspects of the storage media and enables management of the storage pool instead of disk drives. In addition, striping the data across all available spindles effectively eliminates hot spots caused by applications accessing a particular set of spindles.

Oracle recognizes this in their paper[10] on storage optimization, "Optimal Storage Configuration Made Easy", where the author states "configuring storage subsystems for an Oracle database is an unnecessarily complex process. In the conventional methodology, a storage configuration is custom-designed for each site based on detailed knowledge of the application". This paper defines a stripe and mirror everything methodology (SAME). By the SAME methodology, the storage configuration can be simple, efficient and highly available. The MAGNITUDE's storage virtualization technology and the SAME methodology are founded on parallel objectives: simplify, optimize performance and maximize availability.

The MAGNITUDE and its resident software suite (REDI* Software Family) allows quick and simplified management, no downtime expansion and nearly limitless expandability. These and other storage management tasks can be

[8] Thankfully PolyServe Matrix Server supports files much larger than the Oracle 16K blocksize maximum. For this reason The Tens did not require 10 times as many files as the Historical Sales example.
[9] Balanced I/O is best achieved with the "Stripe and Mirror Everything" methodology as described later in this paper
[10] Paper number 295 in the Proceedings of OOW 2000. For convenience, the following URL is supplied: http://otn.oracle.com/deploy/performance/pdf/opt_storage_conf.pdf

performed from one central location without bringing down servers; including storage tuning, such as, RAID level changes.

The MAGNITUDE architecture fit the requirement of The Tens wonderfully. Offering both redundancy and performance without complexity. The MAGNITUDE and their expansion cabinets housed a total of 288 virtualized spindles (RAID 1+0). In each MAGNITUDE is a hot spare ready to be used without human intervention should any spindle fail. There were 5 MAGNITUDE arrays with 4 expansion cabinets as depicted in Figure 7.1.

The total storage capacity was nicely virtualized and presented to the operating system as 23 LUNs. Each LUN was turned over to PolyServe Matrix Server as a managed SAN object to be known throughout the Matrix with a common name. Matrix Server provides consistent unique cluster-wide device names for devices. Gone are the pitfalls of singleton disk management where any given spindle can have different names on different nodes of the cluster. These LUNs contained the prodfs1-prodfs18 file systems for The Tens database as depicted in Figure 6.1.

Combining XIOtech hardware virtualization and PolyServe file management proved a simplified, winning combination.

7.3 Server Hardware

The last 30 years have seen major advances in computing architectures. During the 1970's, the introduction of minicomputers, and even more so, clustered minicomputers from Digital Equipment Corporation, dealt a severe blow to the mainframe industry. The advent of the microprocessor brought profound change to the computer world. Pioneers in Symmetric Multiprocessing (SMP) technology pushed microprocessor based systems into the data center starting as early as the late 1980s. By the mid 1990's, SMP servers were firmly entrenched in the data center based on hardware from vendors such as Sequent Computer Systems, Silicon Graphics and Pyramid Technologies.

Those early pioneers resisted common theory such as the much-deprecated Amdahl's Law, which suggested fixed limits on scalability by sheer number of processors. The Earth, it seems, wasn't flat, and systems that scale well to as many as 30 or more processors were abundant.

This mid 1990's trend of scalable SMP systems did not go unchecked, however. It turns out that Moore's Law is much more difficult to overcome. "Moore's Law" observes that microprocessor speeds double roughly every 18 months. While microprocessors double in speed every 18 months, the processor interconnects (bus, switch, or other) do not. The resulting SMP systems may have scaled to, say, 30 processors when released but scalability suffered if later generation processors were added.

Small, tightly coupled Intel-based systems are another story entirely. These systems have nearly no scalability problems since they are coupled with very nicely integrated chipsets that are tuned to the scalability requirements of the processors. Generally fitted with 2 or 4 processors, and accommodating as much as 4GB main memory, these are very balanced small SMP systems – and very inexpensive. These are the building blocks of radically new enterprise systems. Building blocks require “glue”, as it were. The “glue” is clustering technology – in both platform software and the database server.

The system used for The Tens project was a 10 node cluster of dual Intel Pentium III Xeon (900MHz) processor nodes. Each node was configured with 2GB main memory, two 100baseT Ethernet NICs, 1 Gigabit Ethernet card and 1 PCI Fibre Channel host bus adaptor. All 10 nodes connected to a Brocade Fibre Channel switch and, in turn, connected to the XIOtech MAGNITUDE storage arrays. Figure 7.1 contains a graphical depiction of the Tens configuration.

8 PLATFORM SOFTWARE

8.1 PolyServe Matrix Server

As previously mentioned, the Tens Project system was a 10 node cluster that needed “glue”, as it were, to transform the collection of server “nodes” into a manageable entity that more resembles a single system. That “cement” is the PolyServe Matrix Server product.

PolyServe Matrix Server is a cluster file system that is tightly integrated with PolyServe Matrix HA high availability software. Matrix Server includes specific support for Oracle9i RAC deployment and management. The product has fully-integrated ODM support and enables users to get access to the full functionality of Oracle9i in a clustered environment including enhanced ETL[11] via External Tables and features such as Oracle Managed Files. PolyServe Matrix Server is currently available on Linux and a Windows version is in development. PolyServe Matrix HA is available on Linux and Windows.

8.2 PolyServe Matrix Server Cluster File System

8.2.1 DBOPTIMIZED File System Option

As discussed in the section on OMF above, the PolyServe Matrix Server cluster file system was used to store all Oracle data files, redo logs, control files, gcs file, SPFILE, and Oracle Cluster Management Services (OCMS) quorum disk. This is only possible with PolyServe’s DBOPTIMIZED file system mount option.

The DBOPTIMIZED mount option is a unique way to ensure Oracle is not hindered in anyway while accessing files, while at the same time offering the management ease of file system storage. Being a file system mount option, this feature is really of the “out of sight, out of mind” sort. Whereas other products that attempt to provide similar functionality, such as Veritas Quick I/O, suffer from compromising limitations, the DBOPTIMIZED mount option truly allows both high-performance database I/O and seamless access to files with normal utilities and mechanisms.

Unlike Quick I/O, files in the DBOPTIMIZED file system are simple files and can be accessed not only with Oracle, but Linux tools such as tar(1),cpio(1) compress(1) and most importantly, Oracle Managed Files. OMF requires simple file create, not special ioctl() file create, and more importantly, the ability to dynamically grow a file – two requirements not met by Quick I/O.

[11] Extract, Transformation and Load

Files in the DBOPTIMIZED file system are treated as direct I/O files provided the program adheres to the same requirements as necessary to access a raw device. That is, if a program performs read and write operations with the same buffer alignment and blocking requirements as required by a raw device[12], the I/O is completely unbuffered and not serialized through any OS locking. Oracle I/O operations fit this criteria, nicely, hence Oracle I/O operations are carried out by a special kernel code path that offers the same performance as raw partitions – the best of both worlds.

The unique aspect of the DBOPTIMIZED file system option lies in the fact that the files are just regular files. As such, programs that execute I/O operations lacking the Direct I/O criteria still succeed – through the buffered path. This means that Oracle I/O is rendered direct, and tools such as tar(1),cpio(1), gzip(1) are not – yet they work just fine. Being "Database Optimized", the support focus of this feature centers on Oracle I/O and tools that manipulate Oracle database files.

8.2.2 External Tables and Intra-node Parallel Query – CFS Required

Perhaps one of the most unique aspects of this project was the combination of two powerful new technologies – External Tables and the PolyServe Matrix Server cluster file system. As mentioned above, External Table functionality is new in Oracle9i. This feature allows the Database Administrator to define ordinary flat files in the file system to Oracle. In describing a flat file to Oracle, a data dictionary entry is made and the flat file is then accessible with the power of the SELECT statement – including Parallel Query. But what about External Tables in a clustered environment?

Without a cluster file system, the power of External Tables cannot be realized with Oracle9i RAC. To leverage the processing power of all CPUs in the cluster, intra-node Parallel Query (IPQO) is required. IPQO cannot however be used without a cluster file system to provide concurrent access by all nodes to the external table files. With the PolyServe Matrix Server cluster file system, IPQO on External Tables is a reality, and proved very useful in this project.

8.2.3 External Tables, 66% Reduction in Data Movement during ETL

Typically, building DSS tables with aggregate and other summation data is a three part process once the data is available in flat file form[13] that requires moving the data three times. These steps commonly consist of:

- **Splitting The File.** This phase is in preparation for parallel loading and is generally performed by an operating system tool, such as the Linux split(1) command, or a custom program.

[12] Read or Write system call buffer must be aligned on a page boundary and the I/O must be at a 512 byte or multiple offset. The size of the I/O must also be 512 bytes or a multiple thereof. Oracle I/O is sized as a db_block_size operation or a multiple thereof – both will always be a 512 byte multiple.

[13] Typically, flat file data is generated from the "Extract" phase of the ETL. The extraction might occur from foreign systems (IMS, etc) or even the same system, but from a different schema – such as the OLTP database.

- **Parallel Loading.** This phase executes multiple streams of SQL*Loader inserting the split files into temporary tables in the Database in preparation for aggregation or other transformation.

- **Transformation.** This phase generally consists of running PL/SQL queries against the temporary tables storing the results in the final DSS[14] tables. This is the phase that performs aggregation, summation and other data-intensive operations.

The first step is generally to split the flat file, the second is to load the files in parallel with SQL*Loader. Of course, the splitting of the file is a process that requires moving all the data once with a tool such as the Linux split(1) command. If the original file were split into, say, 16 parcels, parallel loading can be achieved by executing 16 streams of SQL*Loader – one for each file. This works nicely presuming there is enough processor bandwidth to accommodate the 16 streams. What about a clustered environment of commodity 1,2 or 4 processor servers? Without a cluster file system, engaging processors on other nodes in the cluster requires moving the data to file systems on the other nodes. This is typically done by pipelining a data "shipper" in the file splitting phase. This is network intensive but it works, though not optimally. With the split file data strewn about the cluster in local file systems, SQL*Loader can then be used.

Why split the data and place it into temporary tables? It isn't necessary with External Tables. Splitting and loading into temp tables is 66% of the data movement. Instead of splitting the file, it can be described as a single flat file and Oracle will parallelize access to it by tasking multiple Parallel Query Slaves to access it. Instead of loading the data into temporary tables, the same "transformation" SQL statements can be executed against the flat file – in parallel and across all nodes of the cluster. The data makes it from a single flat file to a final DSS schema in one operation.

This culmination of functionality was put to the test during this project. A large portion of the test-data loading was conducted by generating the data and accessing the data as an External Table.

9 TEST APPLICATION WORKLOAD

Simulating an Order Entry system, the application test consisted of connecting 500 users per node in the cluster and running a series of transactions while collecting response times. Transaction response times are a key metric when evaluating the ability to scale up an application.

A key attribute of this testing is that it was completely void of traditional cluster-aware tuning. Comparatively speaking, nearly every cluster-centric database test ever completed 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

[14] Decision Support System

requests only from customers with customer identifiers in the 1-1000000 range and node 2 services the 1000001-2000000 range on so on.

The pitfall of such partitioning schemes is they require changes to the application. Applications should not have to change in order to scale horizontally in a clustered environment, and with Oracle9i Real Application Clusters, they don't. Oracle9i 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 Tens 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 and the only attribute of the entire test that changed from the SMP scenario is the addition of Oracle freelist groups – a task that consisted of simply adding 5 words of text in just a small number of table and index create statements.

To the credit of Oracle9i, the Oracle Shared Global Area configuration parameters required were quite minimal. In fact, the total number of parameters explicitly set was a mere 18! For a test of this magnitude, most Oracle Database Administrators will find this fact quite interesting as in years past they have been accustomed to toiling over large numbers of parameters. Most notable was the setting for db_cache_size, which was 200MB on each Instance.

The bulk of the transaction mix centers around "Customer Service" transactions with a moderate number of New Orders taken. The transactions details are as follows:

- **Orders Query**. This transaction accounted for 63% of the activity. This query provides detail on existing orders for the customer and provides such detail in a most recent to least-recent order - but only top-level detail.

- **Customer Attribute Update**. This transaction speaks for 6% of the workload and offers the ability to update such information as phone number, address, credit card info, etc.

- **Orders Report**. This transaction differs from Orders Query in that it offers full orders detail for a customer to include shipment status. This transaction is executed 20% of the time.

- **New Orders**. New Orders are taken from customers at a rate of 5% of all transactions.

- **Product Update**. This transaction occurs as the result of a customer service change to a product. 6% of all transactions are a Product Update.

10 TEST METHODOLOGY AND PERFORMANCE ANALYSIS

Using Oracle's Shared Server Architecture, connecting 500 users per node was achieved through the use of Net9 Services – one for each node. Test users were routed to dispatchers based on the number of nodes involved in the test.

The testing levels were 1,2,4,6,8 and 10 nodes. At each level, 500 users were connected to each node and triggered. Once triggered, the users cycled through the set of transactions listed above. After the execution of a transaction, the users paused for a random amount of time between 1 and 15 seconds to simulate human and client-side software behavior.

The goal of the testing was not to achieve any given throughput *per se*, as is often the case with benchmarks. Instead, it was important to study the behavior and stability of the components involved at a realistic system load level. To that end, the user count was throttled to 500 users per node and the resultant processor utilization seldom peaked above 75% - at the node level. However, given the fact that this was a single database application serviced by many nodes, processor utilization was tracked at the cluster-global level as well.

One common critique of benchmarks is the fact that they are executed for very short periods of time. To that end, the workload was executed for 1 hour at each user count.

Oracle Statspack and a myriad of systems tuning data was collected, analyzed and summarized for this paper.

10.1 Test Findings

Although there was no predefined goal for transaction response times for this proof of concept, the very first performance finding was that the system overall had no bottlenecks as is evident in Figure 10.1. The key metric was response time as 500 users were added per node. Entering the graph at 500 users on 1 node with 2.27 second average response times and topping out at 2.9 seconds from 5000 users on 10 nodes was a very acceptable trend – only 27% response time degradation from the lightest to the heaviest load.

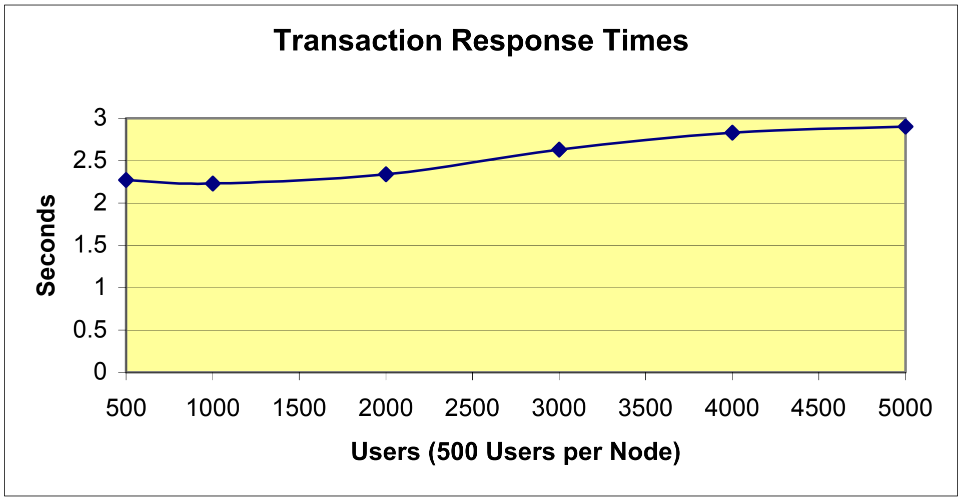


Figure 10.1 - Transaction Response Times for 500 through 5000 users with 500 Users per Node

Oracle9i RAC performed exceptionally well in terms of the number of transactions the varying user counts pushed through the system. The scalability reflected the response times; after all, if response times remain relatively flat the throughput should increase at a commensurate rate.

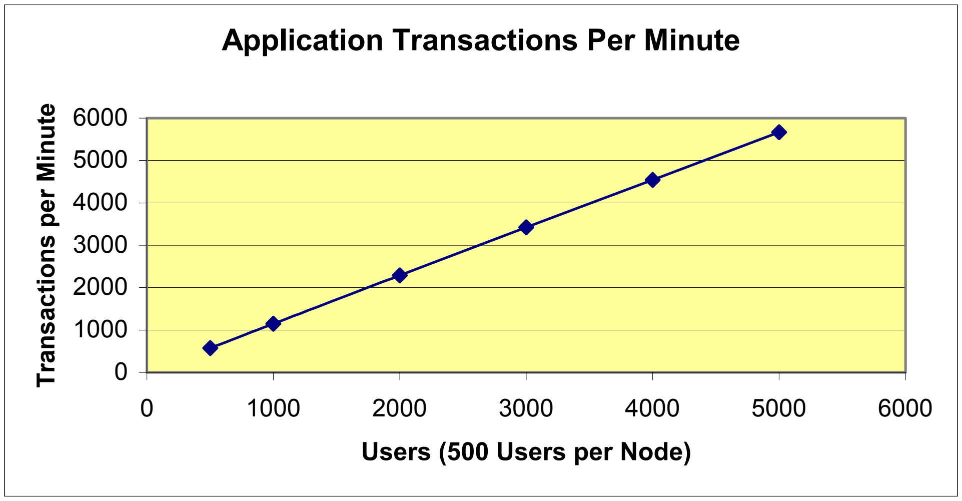


Figure 10.2 - Application Throughput with 500 Users per node – Transactions per Minute

Relying on Oracle9i Shared Server architecture was perhaps one of the single largest elements of success. Configuring 10 Shared Servers per processor and 2 MTS Dispatchers per Shared Server yielded a very predictable processor load. So even, in fact, that the highest peak on processor utilization was a mere 8% compared to that seen on the single node 500-user test. This predictability is extremely important in a clustered VLDB environment, and most certainly so when ramping up to 5,000 users!

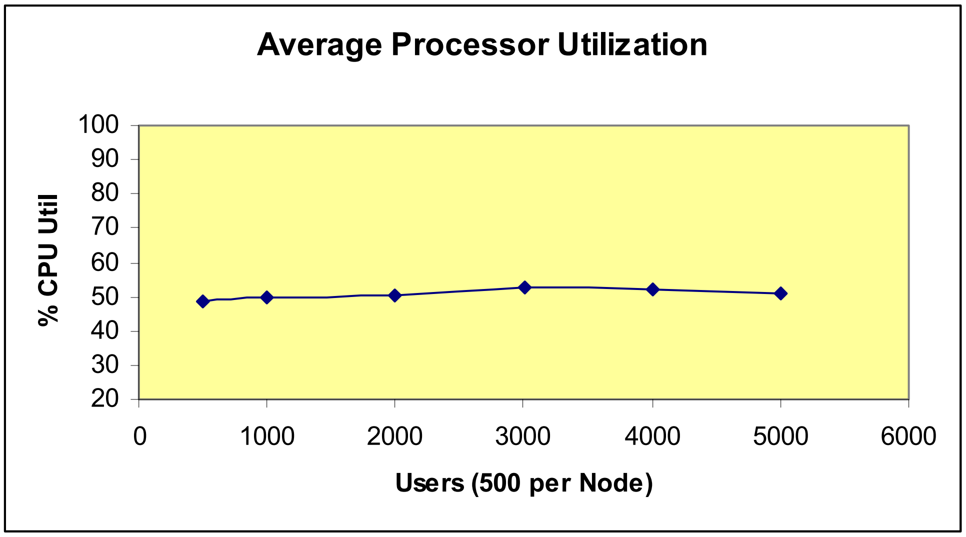


Figure 10.3 - Processor Utilization

A closer look at the Oracle Statspack data revealed that on average, each transaction incurred 30 User Calls, 154 Logical Reads in the SGA and 28 physical reads. Figure 10.4 depicts the physical read profile from 1 to 10 nodes.

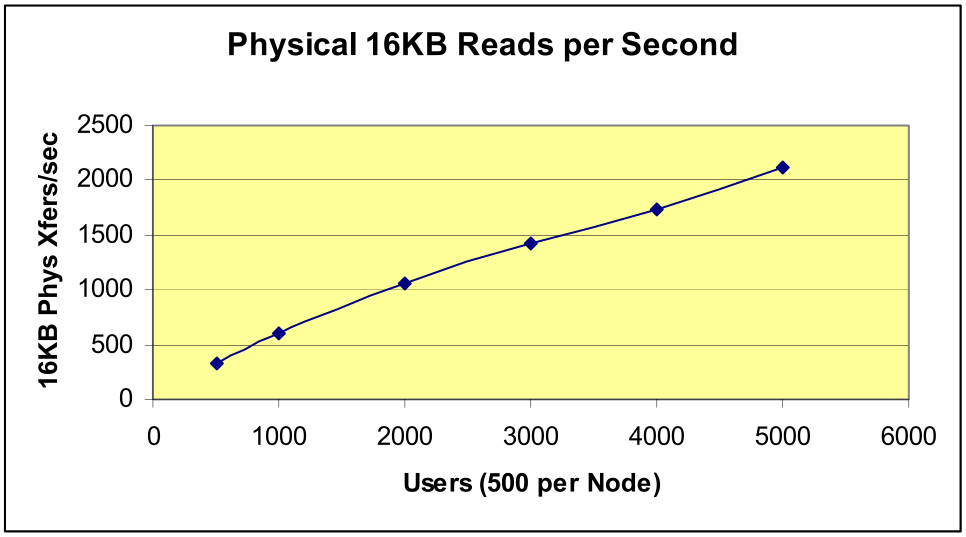


Figure 10.4 - Physical Reads

One important aspect that reflects the performance of the PolyServe clustered file system is measured in the Oracle wait event called *db file sequential read*. This server statistic is the total duration of an Oracle physical single block read. The time calculated in this statistic includes time spent in a runable state - once the I/O is complete. As the I/O rate increases, a poorly implemented clustered file system can result in execution overhead or serialization in the I/O code path. No such characteristic was witnessed and in fact, the wait times for *db file sequential read* only increased from 14.9 to 17.1 milliseconds when ramping up from 1 to 10 nodes.

10.1.1 Oracle9i Real Application Clusters – Scalable by Architecture

The workload scalability achieved in this testing reflects greatly on the architecture of Oracle9i RAC. Scaling a workload to 20 processors in an SMP environment is generally quite difficult. Most Database Administrators will attest to the fact that large SMPs spend a good amount of their bandwidth dealing with Oracle critical code sections – known as Latches.

Oracle Latches are spinlocks used to protect structures in memory through serialization. There are many serialization points in Oracle and they generally track application throughput. That is, on average there is a correlation between the number of executions on a given Oracle serialized code segment and the COMMIT rate of the application. For instance, every COMMIT by the application must perform a Redo Log entry. This code includes, in addition to a good deal of other processing, allocating space in the Oracle Redo Log buffer prior to copying the transaction information into it. Space allocation from this buffer is serialized, hence, the more COMMIT processing an application does, the more contentious this critical section becomes. A high rate of contention on Oracle Latches is reported in Oracle Statspack as *latch free* waits.

There seems to be no end to the online material, books, training documents and other sources of information available to assist DBAs in their dealings with Oracle Latch tuning. Oracle9i mitigates Latch contention by its very architecture.

Recalling the Redo Allocation latch, it is essential to point out that on a 20 processor SMP system there is but one of these serialization points. In contrast, the Tens test system was a 10 node cluster and therefore 10 instances of Oracle. For every usually troublesome Oracle serialization point, The Tens system had 10. For this reason, contention on all Oracle serialization points remained flat at .3 per transaction when scaling from 2 to 20 processors (1 to 10 nodes). Without Real Application Clusters, this efficiency cannot be achieved[15].

11 CONCLUSION

As the price-performance of Intel-based servers continues to increase and networked storage continues its growth, standard Intel-based servers will become pervasive in the enterprise data center. Software like Oracle9i Real Application Clusters makes these servers an increasingly attractive alternative to large SMP Unix platforms for applications ranging from web front ends to databases. The Tens study has shown that a Very Large Database can be implemented using a cluster of Intel-based Linux servers running Oracle9i Real Application Clusters. Furthermore, a robust cluster file system like PolyServe Matrix Server is essential in managing the complexity of such a deployment.

[15] In fact, this efficiency is so compelling that traditional SMP vendors have been compelled to utilize RAC within an SMP to achieve competative TPC results. For example, the following URL provides the full disclosure of an HP result using 8 Instances of Oracle9i RAC on a 32 CPU SMP: http://tpc.org/results/FDR/TPCC/compaq.gs320.v5.062501.fdr.pdf

Title: Very Large Databases: Creating a VLDB on Intel-based Servers with Oracle9i Real Application Clusters

Author: Kevin Closson, Sr. Staff Engineer,PolyServe

Acknowledgements: A very special thanks to XIOtech for generously providing such substantial hardware in support of this Proof of Concept and for having the vision that VLDB technology is no longer the domain of Legacy storage. Additionally, special thanks are given to James Lorenzen and Stan Glasoe of XIOtech for their expertise and effort in support for this project. Thanks to David Dougherty for excellent project management, and Tim Wright for being an absolute Linux Wizard. Finally, much thanks and recognition is given to the Core Engineers and Architects of the PolyServe Matrix Server for without this leading edge technology, this effort would not have been possible nor would the idea of commodity-based clusters in the Enterprise be feasible.

For sales, marketing or partnering inquiries, please contact:

PolyServe, Inc.
4 Embarcadero Center
Suite 540
San Francisco, CA 94111 USA
www.polyserve.com

XIOtech Corporate
6455 Flying Cloud Drive
Eden Prairie, MN 55344 USA
www.xiotech.com

VLDB wp 070102a kc skp