Oracle Database Acceleration

DELL

John M. Harper

Opportunities of the Information Age

1

Dell has created an integrated system that begins to satiate business thirst for data-processing, without breaking the bank. In this white paper, we benchmark the Dell Integrated System for Oracle Databases (DISOD) and show you how to provide extreme performance for extreme data.

The Current State of Data

Challenges & Opportunities

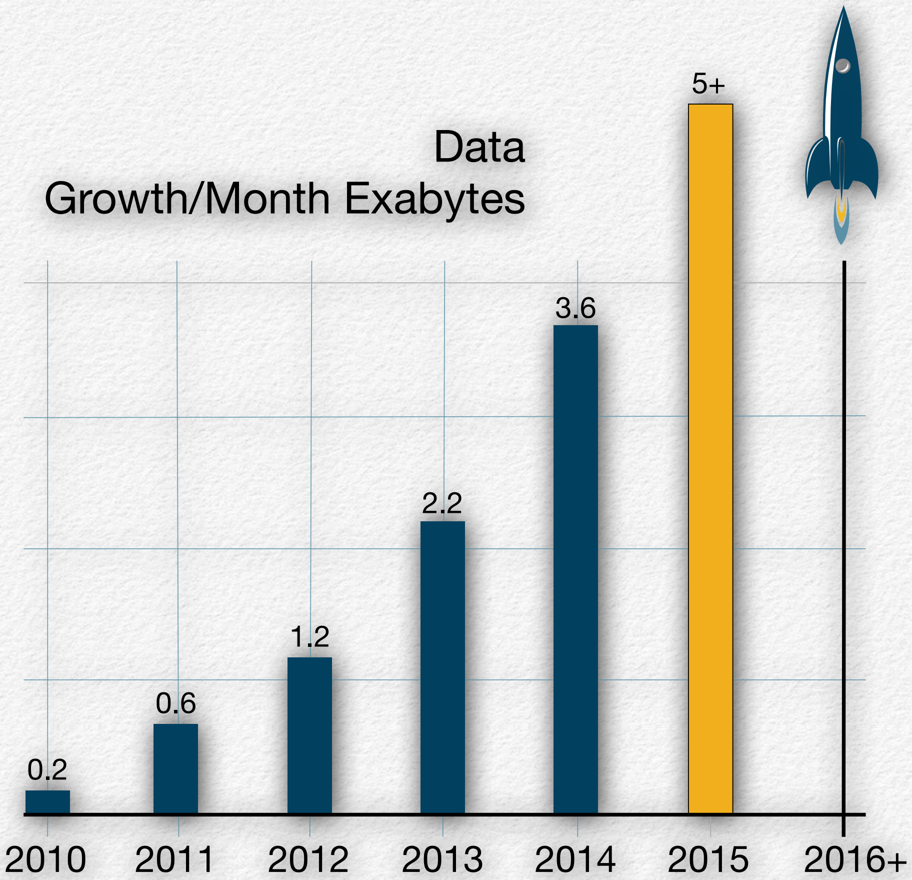
1. **Last year’s mobile data accounted for 4 Zettabytes of data.**
2. **The rate of date growth is expected to be 10 times greater by 2017.**
3. **Big-data and Relational database technologists need to work together.**
4. **While the rate of data growth has never been higher, the cost of ownership has never been more affordable.**

The Big-Data/Information Age

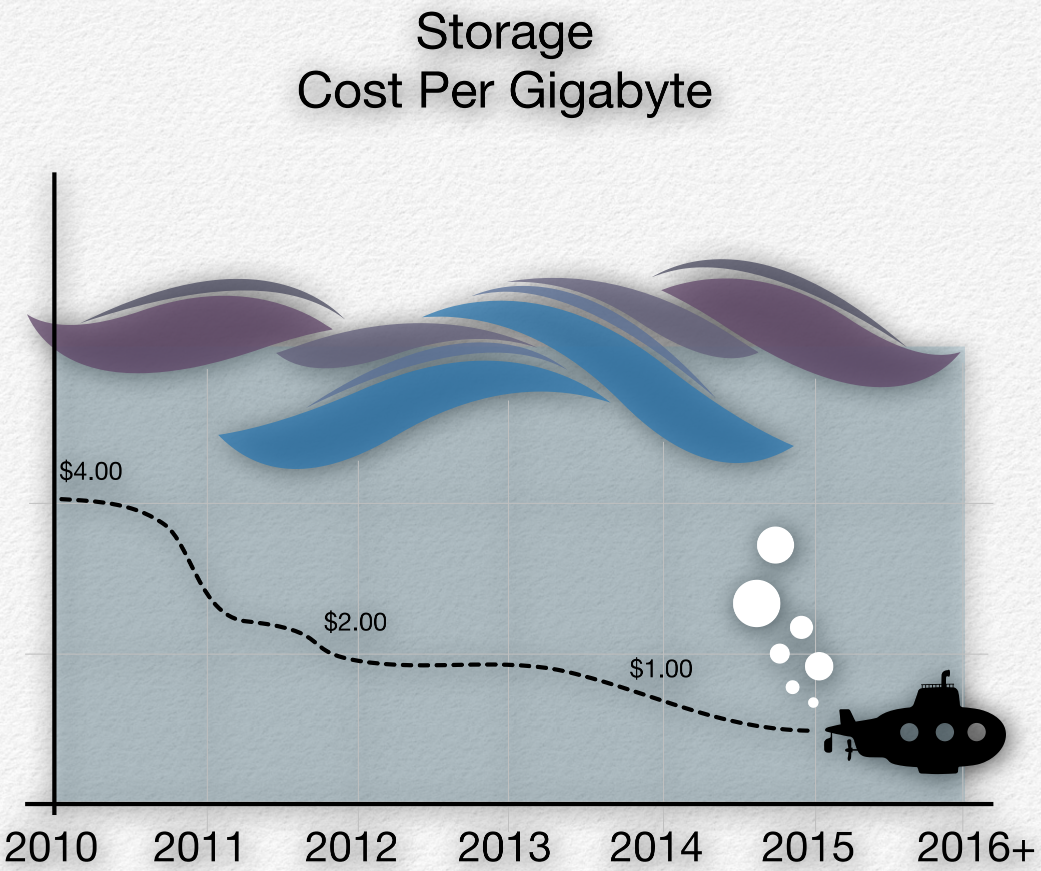
At no other time has data been more important to businesses, and in order to stay competitive in the global market place, they are required to sift through extreme amounts of information. In fact, last year’s mobile data accounted for 4 zettabytes, or 4 trillion gigabytes of data. It is predicted that our data may be more than 10 times that size by the year 2017. Without a doubt, companies must take full advantage of traditional and emerging technologies to stay viable.

In our scramble to manage the sheer volume and complexity of modern data, two camps have emerged. On one hand, traditional, Relational Database Management System (RDBMS) professionals promote that their methodologies are superior. They remind us of time-honored principles that keep data secure and of high quality. On the other hand, Big Data professionals are quick to point out that RDBMS technologies are pricy from a license point-of-view, and that they do not scale as well as modern clustered systems. This jockeying pits each camp against the other and is a waste. Instead of working together, professionals from each camp spend considerable time lobbing criticism at each other. A much better approach would be to create hybrid solutions that help businesses thrive in the Big-Data/Information age.

It is our opportunity to live in a time when the volume of data and its total cost of ownership have reached critical mass. Data volume continues to soar and the cost of data per Gigabyte has never been lower. In addition, new all-flash technologies continue to be faster, more affordable, and reliable. For example, the DELL Acceleration Appliance for Databases (DAAD) is an all-flash device that is capable of more than 7 gigabytes per second, with latencies in the microseconds, and millions of input-output operations per second. Furthermore, physical memory has never been cheaper allowing for gigantic memory configurations for sorting and aggregation operations. Lastly, chip manufacturers like Intel and AMD are creating CPUs with amazing speed and core counts not dreamed of in decades past.



Data Source: Cisco 2014 VNI



IDC's Digital Universe Study, Sponsored by EMC 2012

Big Data

The term Big Data is a generalization for data that initially does not fit well in predefined database structures. Good examples of Big-Data include telephone communication, Web-traffic logs, and Social/Political trends.

Big Data is difficult to place in typical RDBMS structures because of it size, variety, velocity, and complexity.

The size of Big Data is daunting because simple implementations of relational tables do no perform well when billions, trillions, and even quadrillions of rows are placed in them. It is possible to create relational tables that will perform well against this amount of data but careful architectural considerations are mandatory.

Typical Big Data is received in adhoc fashion. Telecommunication data is a prime example. From our smart phones we send and receive digital voice, texts, tweets, and twitters -- of which some contain digital images taken from our cameras or copied from online browsing. This variety changes from one moment to the next and is difficult to store in relational tables.

The rate and complexity of Big Data also makes it difficult to store in neatly organized transactional tables. It is true that Big Data elements eventually make it to relational structures but in its initial state, it is more efficient to leave the data in its original form. Big Data technologies attempt to filter and organize data elements based on statistical modeling and artificial intelligence. They use massive parallelization with thousands of CPU cores, terabytes of memory, and petabytes of hard drive storage to accomplish that end.

Transactional Data

In the early 1960’s hierarchal and network databases were used to gather and manage large sets of data. The Apollo rocket required the use of a hierarchal database that stored information in a rigid structure that looked something like an upside down tree. Child data elements were statically dependent on their parent components, in a complex one-to-many relationship. A modern data structure similar to these is XML; however, XML technology is much more flexible and query-able.

At the time, these databases were difficult to work with. They required complex programming to get data in and out of their databanks and were clunky when it came to manipulating the data itself. Often times, data would become orphaned or out of synch because a programmer forgot to update all of the places where data was stored.

Doctor Edgar Frank Codd, AKA E.F. Codd, invented a new method of storing data that did not contain the limitations of hierarchal and network technologies. Instead, he stored data in tables and related those tables to each other by way of key fields that uniquely identified each row. Relationships between tables were more flexible and could evolve versus the rigid parent-child requirements found in previous database technologies.

Codd designed a simplified language which he called *Alpha*, which was founded on relational calculus; however, this language was not fully developed as SEQUEL had already gained support from IBM executives. SEQUEL was not a full implementation of Codd's relational models but is was far superior to languages of the time and quickly became the de-facto language for all relational database technologies.

Codd's theories proved effective for storing large amounts of data in neatly organized tables. When developers and architects used his normalization processes, they auto-magically protected their data against possible transactional anomalies. However, many of architects had difficulty grasping his theories because they were steeped in advanced relational algebra and calculus. The result was many chaotic designs that claim to be relational because they have primary and foreign keys, when in actuality they closely resemble network and hierarchal databases instead.

Relational databases are very good at storing event information like financial transactions and should continue to be the database of choice for that kind of data.

Our Objective

This white paper focuses on the Oracle ORDBMS and its capabilities in transactional and warehouse scenarios. We plan to write several follow-on articles, white papers, and books that detail both relational and Big Data methodologies. It is our hope that the results we found will properly illustrate what modern systems are capable of and that they will inspire you.

Transactional Databases

Transactional Database Key Points

1. **Transactional and warehouse databases require different levels of the same resources**
2. **Transactional systems require high IOPS rates**
3. **Transactional systems require moderate CPU and Memory resources**
4. **Transactional systems require moderate Networking between the application server and database but high networking between nodes and storage**
5. **Parallel CPU utilization and fast IO empowered the DELL DISOD to perform 1.2 million web transactions per minute**

Transactional Database Needs

It has been the privilege of the authors to work in large environments consisting of almost 1000 Oracle and 300 MSSQL instances. Data centers of this size are costly to setup and maintain, what's more, their sponsors have specific expectations that require the ability to scan hundreds of gigabytes in a very short period of time. Our experiences give us insight on what transactional and warehouse databases need to operate at high levels of performance and reliability.

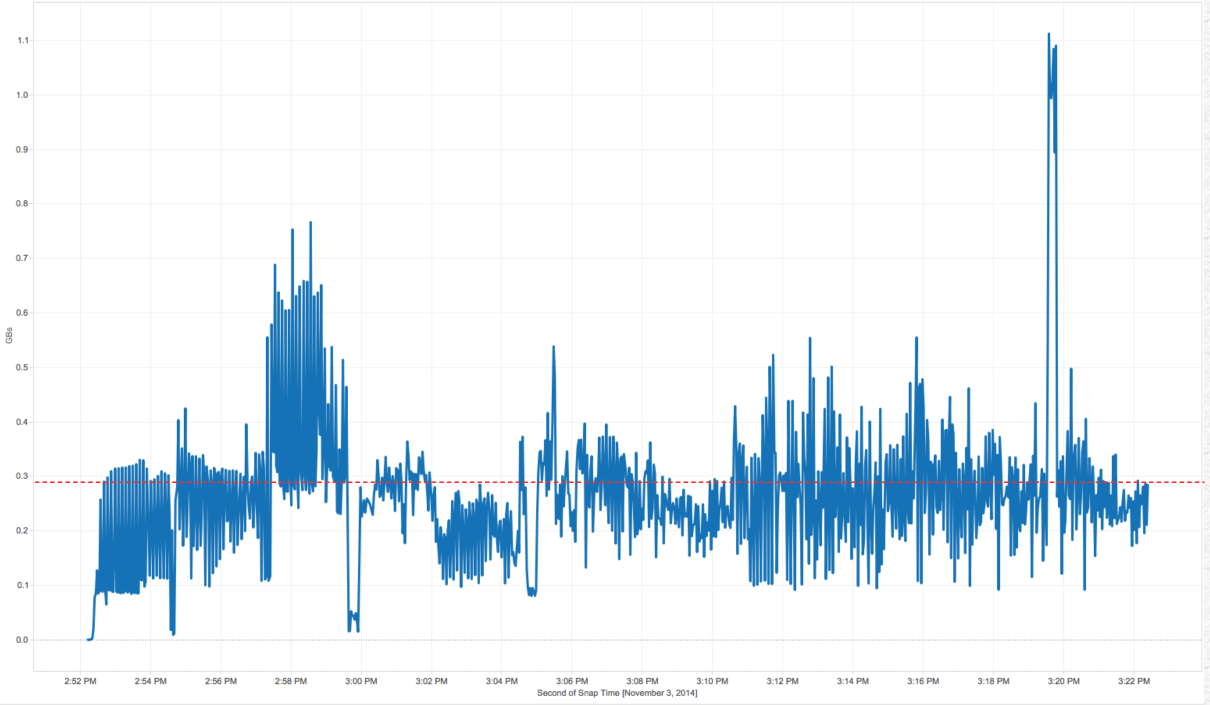
IO Requirements

It comes as no surprise that transactional and warehouse systems have a different profile. That's because they solve two very different business needs. The most important factor in transactional systems is their ability to complete many concurrent transactions. Moderately sized systems may only require hundreds or thousands of transactions per second; however, large systems can easily require the ability to complete millions of transactions per second. Because of this requirement, transactional systems lean heavily on Input Output Operations Per Second (IOPS).

It is a mistake to think that IOPS are the only decisive factor. Highly concurrent systems also require a large amount of bandwidth and very low latency in their storage-subsystem. We suggest that your design focus first satisfies IOPS requirements, followed by latency, and then total bandwidth. To be successful, you must implement systems that satisfy all three variables. At no time can any one of the three (IOPS, Latency, Bandwidth) be eliminated, nor can they be degraded in their importance.

Our stress test of the DISOD proves this point. Our goal was to discover the limit of the system by causing database and/or operating system failure. We were pleasantly surprised when we could not cause complete failure, albeit individual web transactions began failing in large chunks when we surpassed 1.2 million web transactions per minute. Granted, 1.2 million web transactions per minute is extremely large. It simulates the load of large and active web-based companies.

Gigabytes Per Second



300 Megabytes per second not the whole story...

The average required throughput hovered around 300 megabytes per second, with peaks between 500 megabytes and 1100 Megabytes per second. Clearly, the 7 gigabyte/second limit of the DAAD was not reached. However, the size of the data-pipe is not the complete story.

The complete picture only comes into focus when additional system and IO statistics are analyzed. During the one-hour endurance test the following statistics stood out:

- CPU utilization on both DELL 920 RAC nodes hung around 90%
- Block gets/second maxed out at 750,000 per minute
- 374,552,538 select statements were issued
- 177,633,759 insert statements were issued
- 94,584,800 update statements were issued
- 104,826,049 commits were issued
- 15.256 Gigabytes were sent back to the SQL*Net Client
- IOPS were not maxed out but were heavily utilized

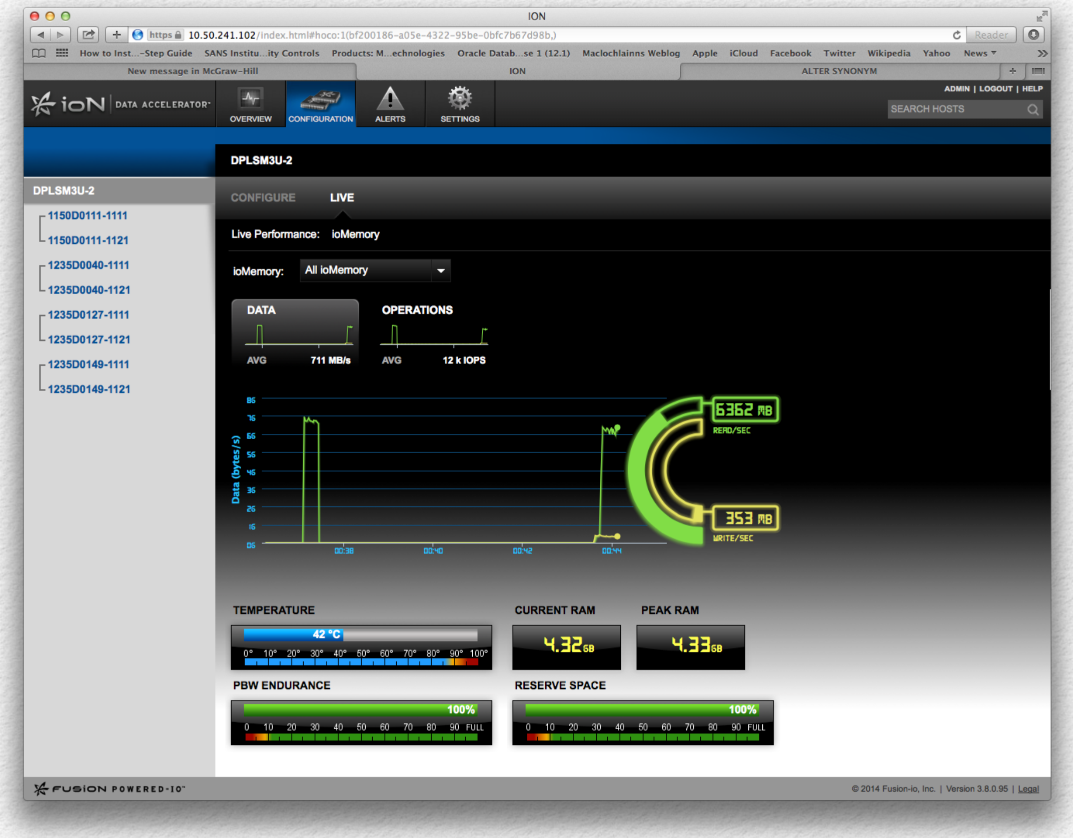
- Latency of each operation remained in the high micro-second or very low millisecond range

Notice that the CPU and network were heavily used. Moreover, the block gets per second were amazingly high. In fact, we had to add log files and increase them to 20GB in size (per log file) in order to achieve these results.

We determined that we had effectively discovered the DISOD CPU limit; however, we also concluded that the system had a very good balance of hardware as-it-stood. Our only suggestion to DELL was to split the compute nodes into smaller chunks, thereby providing flexibility. For example, if DELL engineers had used 2 processor compute nodes to create a 5 node RAC cluster, we would have been able to discover the IO limitations without pushing the physical core count beyond 100… A factor that is extremely important to customers who do not have enterprise license agreements with Oracle.

Our tests at the SanDisk Data Propulsion Lab showed that individual ION bandwidth limitations maxed out at or around 7 Gigabytes/second as shown in figure 3 (FusionIO/Infiniband Peak Performance). Their lab was much smaller and could not compete with the DELL DISOD. However, it was not designed to be an enterprise-level appliance. Instead, it consisted of only 2 compute nodes with (2) 10 core processors. Our maximum web transactions per minute topped out around 70,000 on these servers versus the 1.2 million that the DISOD produced.

We performed other tests in both the DELL and SanDisk labs that taxed the CPU, IOPS, and storage-subsystem. In one test, we created a 10 Billion row fact table and supporting dimension tables with indexes on all of their foreign and primary key columns. We wanted to show the difference between creating indexes in a serial manner to creating them all at once using the dbms_scheduler, Oracle supplied package. The results were dramatic.


https 10.50.241.102/index.html#hoco:1(bf200186-a05e-4322-95be-0bfc7b67d98b,)

DPLSM3U-2

Live Performance: ioMemory

ioMemory: All ioMemory

DATA AVG 711 MB/s

OPERATIONS AVG 12 k IOPS

6362 MB READ/SEC

353 MB WRITE/SEC

TEMPERATURE 42 °C

CURRENT RAM 4.32GB

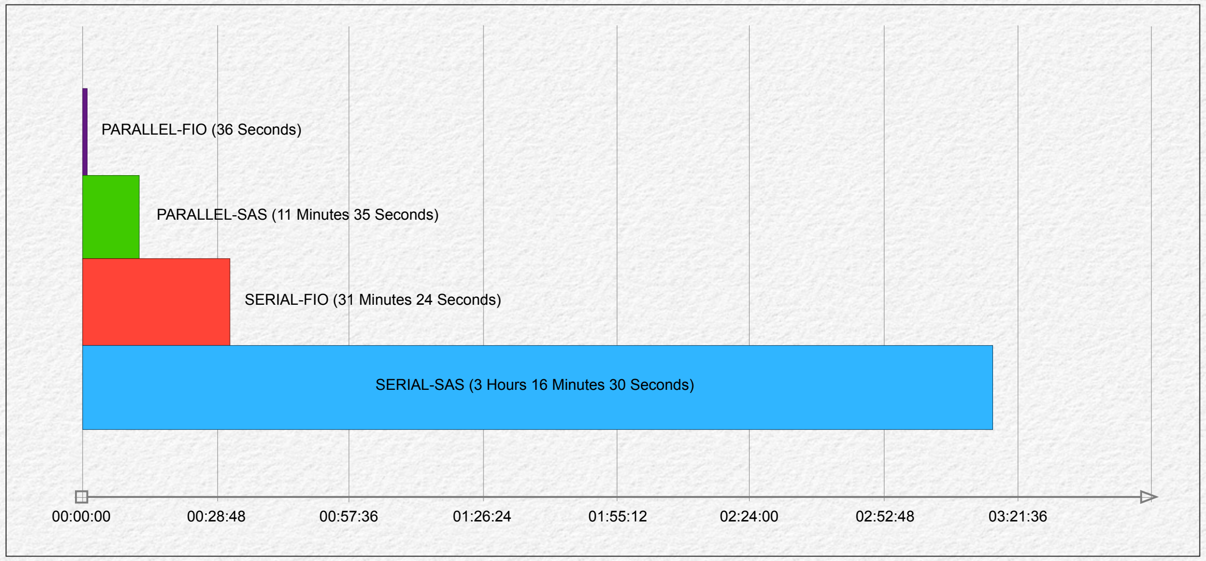
PEAK RAM 4.33GB

PBW ENDURANCE 100%

RESERVE SPACE 100%

© 2014 Fusion-io, Inc. | Version 3.8.0.95 | Legal

The parallel process completed all index rebuilds in only 36 seconds. That is because the longest running index build took only 36 seconds and all of the indexes were created at the same



time. Mind you, the CPU utilization was nearly 100% for those 36 seconds. You could complete this very common task within normal maintenance windows.

CPU, & Memory Requirements

Both transactional and warehousing systems require large amounts of CPU time. You will cripple your database if you neglect this factor. It is our sad experience that database professionals blame the storage professionals when asked why a database is not performing at expected levels. Storage professionals resist such notions, stating that their storage arrays are lightly used, not even approaching their advertised limits... and so it goes, one side blaming the other for less than satisfactory performance. Furthermore, both teams tend to gang up on the network team when they realize that database and storage systems are largely idle, waiting on the each other. In every case, we have seen that the real problem consists of poorly configured databases, networks, and storage arrays. No one team is completely responsible. They equally share the blame and must cooperate in order to reach resolution.

Aside from IO requirements, database professionals must consider parallel processing, system memory, and the network.

In one scenario, the authors dealt with an organization that purchased an expensive Exadata server. Then, without considering the overarching consequences, their engineers turned off all parallelism, throughout the database (All tables and indexes were set to NOPARALLEL). This organization essentially cut their Exadata system off at the legs. There was no way it could perform adequately because most of its processor cores sat unused. Our suggestion was to open the parallel processes up, allowing the system to function at its highest levels. After all, once purchased, CPUs DO NOT gain interest. On the contrary, they quickly depreciate. It is much better to over-use the system than to artificially limit it.

We have also seen systems that have ample processing power but are severely limited in total System Global Area (SGA) memory. For whatever reason, database professionals affix old limitations to new technology. They resist the notion of large SGAs, instead configuring instances with no more than 2 gigabytes of SGA by default. It is possible that the DBA is attempting to reduce overall licensure costs by putting as many instances on a single server as possible. However, this notion is like a two-edged sword. On one hand, the DBA takes advantage of license costs, maximizing total instances per server. On the other hand, the DBA figuratively neuters the database, preventing it from accomplishing its main tasks. For example, modern sorting and aggregation requirements require large amounts of SGA, sometimes ranging in the 100's of gigabytes.

Network Requirements

Your datacenter may have shared disk arrays capable of millions of IOPS, microsecond latency, and gigantic bandwidth. If you are lucky, you have a Tier-0 solid state storage device like the DAAD storage cells. Unlike spinning disks, they deliver consistent read and write performance in the 1 millisecond or less range. What's more, the right configuration can give you 99.999% uptime that typically outperforms slower spinners by a factor of 10x.

The DAAD device has a maximum capacity of 12TB of redundant storage space, per 2-unit pair, and consists of 8 SanDisk ioMemory devices. In our tests, each ioMemory was carefully mapped so that it provided maximum bandwidth and convenient mirroring, similar to RAID-1. The DAAD pair was interconnected by two InfiniBand interfaces, used by SanDisk's mirroring software. Each device had (4) 16 Gigabit Fibre Channel cards that created a network fabric capable of sustaining more than 5 Gigabytes per second to the database. Our actual speeds using Fibre Channel averaged 5.2 Gigabytes per second while our InfiniBand tests peaked at 7 Gigabytes per second, per device. We could not find any other vendor who could compete with SanDisk's speed; however, their architecture is only a few steps beyond simple-storage, block devices. You must use Oracle's Automatic Storage Management (ASM) to provide redundancy beyond their mirroring functionality.

DAAD devices require adequate networking and it does you no good to carefully plan database and storage servers with oodles of RAM, CPU, and high-speed disks if you skimp on network connectivity.

Transactional Test Setup

The *Oracle Database Solutions Group* created a tool called SwingBench, for generating realistic, web-based load tests. It's an extensible Java based testing tool that empowers web developers to define their own classes; however, it is a great database stress testing tool right out-of-the-chute.

The tool was easy to install. We simply downloaded the zip file provided by Dominique Giles at the following link. The only tricky part was that we were constrained to use the text only interface instead of its nifty GUI.

Install Java

Swingbench is a Java program, so you must install the Java JRE or SDK to use it. We downloaded it here. Our method of choice is to download the tape archive (tar) file and then to unzip it into a common location. You can see from the following command that our test server has Open JDK versions 1.3 through 1.7 installed.

```
[root@oel64 /home/oracle 1021]# cd /usr/lib
[root@oel64 /usr/lib 1022]# ll
total 108
drwxr-xr-x. 3 root root 4096 Jul 12  2013 anaconda-runtime
drwxr-xr-x. 3 root root 4096 Jul 12  2013 bonobo
drwxr-xr-x. 5 root root 4096 Jul 12  2013 ConsoleKit
drwxr-xr-x. 9 root root 4096 Jul 12  2013 cups
dr-xr-xr-x. 2 root root 4096 Nov  1  2011 games
drwxr-xr-x  3 root root 4096 Dec 20  2012 gcc
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.3.1
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.4.0
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.4.1
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.4.2
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.5.0
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.6.0
drwxr-xr-x. 2 root root 4096 Jul 23  2010 java-1.7.0
… truncated output.
```

We wanted to bring our installation up-to-date so, we removed the symbolic links created by the yum installer and placed the Oracle JDK in the same location as shown in the next commands.

```
[root@oel64 /usr/lib 1043]# yum remove java
[root@oel64 /usr/lib 1044]# tar xvf /home/oracle/jdk-8u31-linux-x64.tar
```

Then, we created symbolic links to /usr/bin like so.

```
[root@oel64 /usr/bin 1073]# cd /usr/bin
[root@oel64 /usr/bin 1074]# unlink java
[root@oel64 /usr/bin 1075]# ln -s /usr/lib/jdk1.8.0_31/bin/java java
```

The yum command removes java and gets rid of several symbolic links for us. It also removed all Open JDK versions in the same command. We could have done the same thing by removing directories and unlinking but it was easier for us to simply run the yum command. We can show what version of Java we are on by issuing the next command.

```
[root@oel64 / 1079]# java -version
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
```

If you use this method, you can have many versions of Java on your test server at the same time. Rolling forward or backward is as easy as changing the symbolic link.

Install Swingbench

Swingbench is easier to install than Java. It is a simple matter of unzipping the file downloaded from Dominique Giles web site. Once downloaded, issue the following command in your home directory as shown below.

```
[oracle@oel64 /usr/bin 676]$ cd /home/oracle
[oracle@oel64 ~ 677]$ tar xvf swingbench.tar
```

Install the Oracle Client

Swingbench provides a thin connection to Oracle so you do not have to install the Oracle OCI client; however, we installed the thick client because we needed to run additional testing against the DISOD and we wanted to use the OCI client instead of the thin client. We simplified the installation by running the yum install *oracle-rdbms-server-12cR1-preinstall* command. This setup all of the system configurations automatically. It is, by far, the fastest way to prepare a Linux host to receive either the Oracle client or database.

```
[root@oel64 ~ 1001]# yum install oracle-rdbms-server-
12cR1-preinstal -y
```

Next, we created the /u01/app/oracle directory and granted ownership to the Oracle user, like so.

```
[root@oel64 / 1003]# mkdir -p /u01/app/oracle
[root@oel64 / 1004]# chown -R oracle:oinstall /u01
[root@oel64 / 1005]# su - oracle
```

We like to keep all of our Oracle-related files together, so we created a directory under the /u01 folder named admin. Then, we placed all of our downloads, scripts, and installation files in this directory as follows.

```
[oracle@oel64 ~ 721]$ mkdir -p /u01/admin/script
[oracle@oel64 ~ 722]$ mkdir -p /u01/admin/download
[oracle@oel64 ~ 723]$ mkdir -p /u01/admin/install

[oracle@oel64 ~ 730]$ cd /u01/admin/install/
[oracle@oel64 /u01/admin/install 731]$ ll
total 8
drwxr-xr-x  8 oracle oracle 4096 Dec  7  2011 cpumonitor
drwx------ 12 oracle oracle 4096 Nov  1 13:41 swingbench

[oracle@oel64 /u01/admin/install 732]$ unzip
../download/linuxamd64_12102_client.zip
```

```
[oracle@oel64 /u01/admin/install 734]$ cd client
```

The Oracle client install was a snap because of these steps. We simply kicked off the runInstaller program and followed Oracle's installation steps. We were able to run all of our tests from this machine after installing Java, Swingbench, and the Oracle client. We made sure the server was powerful enough to start thousands of threads.

Run OEWizard

We had to create a swing bench schema and data to test with. There is a built-in GUI but our setup required command-line instead, so we issued the following.

```
[oracle@oel64 ~ 749]$ cd /u01/admin/install/swingbench/bin

[oracle@oel64 /u01/admin/install/swingbench/bin 750]$ ./oewizard -scale 100
-cs //disodFAI1.dbase.lab:1521/hamdb -dbap oracle -ts SOE -tc 80 -nopart -u
soe -p soe -cl -df "+DATA/soe.dbf" -create
```

Here is a list of all the parameters that the oewizard can receive. The list is extensive and we did not use all of them in the previous command.

parameters:

```
-allindexes          build all indexes for schema
-bigfile             use big file tablespaces
-c <filename>        wizard config file
-cl                  run in character mode
-compositepart       use a composite paritioning model if it exisits
```

```
-compress               use default compression model if it exists
-create                 create benchmarks schema
-cs <connectString>     connectring for database
-dba <username>         dba username for schema creation
-dbap <password>        password for schema creation
-debug                  turn on debugging output
-debugf                 turn on debugging output to file (debug.log)
-df <datafile>          datafile name used to create schema in
-drop                   drop benchmarks schema
-dt <driverType>        driver type (oci|thin)
-g                      run in graphical mode (default)
-generate               generate data for benchmark if available
-h,--help               print this message
-hashpart               use hash paritioning model if it exists
-hcccompress            use HCC compression if it exisits
-nocompress             don't use any database compression
-noindexes              don't build any indexes for schema
-nopart                 don't use any database partitioning
-normalfile             use normal file tablespaces
-oltpcompress           use OLTP compression if it exisits
-p <password>           password for benchmark schema
-part                   use default paritioning model if it exists
-pkindexes              only create primary keys for schema
-rangepart              use a range paritioning model if it exisits
-s                      run in silent mode
-scale <scale>          mulitiplier for default config
-sp <soft partitions>   the number of softparitions used. Defaults to cpu
count
-tc <thread count>      the number of threads(parallelism) used to generate
data.
-ts <tablespace>        tablespace to create schema in
-u <username>           username for benchmark schema
-v                      run in verbose mode when running from command line
-version <version>      version of the benchmark to run
```

The parameters we chose were:

-scale 100: creates a test schema that is 100 Gigabytes in size

-cs //disodFAI1.dbase.lab:1521/hamdb: tells the oewizard to create the test schema on the disodFAI1.dbase.lab computer, port 1521, with a SID of hamdb.

-dbap oracle: tells the oewizard that our DBA password is oracle

-ts SOE: tells the oewizard that our target tablespace name is SOE

-tc 80: tells the oewizard to create the sample data with 80 threads

-nopart: tells the oewizard not to partition sample data tables

-u soe: tells the oewizard the sample schema username

-p soe: tells the oewizard the sample schema password

-cl: tells the oewizard to execute in character mode

-df "+DATA/soe.dbf": tells the oewizard what datafile to use

-create: tells the oewizard to create a benchmark schema as well

That's it! All we had to do was wait for the data to be created. Once that was done, we edited only the first portion of the swingconfig.xml file, located in our */u01/admin/install/swingbench/bin* directory. We had to provide a connection string and driver type. The rest of the file remained unchanged.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<SwingBenchConfiguration
xmlns="http://www.dominicgiles.com/swingbench/config">
   <Name>"Order Entry (PLSQL) V2"</Name>
   <Comment>""</Comment>
   <Connection>
      <UserName>soe</UserName>
      <Password>soe</Password>
      <ConnectString>//disodFAI1.dbase.lab:1521/hamdb1</ConnectString>
      <DriverType>Oracle oci Driver</DriverType>
…truncated…
```

We started swing bench by issuing the charbench command. You can see from this command that many of the parameter switches are similar to the oewizard command. You can use it as is as long as you change the connection string and user count. Or, you can run the program in GUI mode.

```
$> ./charbench -c swingconfig_1.xml -cs //disodFAI1.dbase.lab:1521/hamdb1
-dt oci -cpuloc oraclelinux -u soe -p soe -uc 500 -min 0 -max 100 -a -v users,tpm,tps,cpu
```

If your network and security settings will allow it, we recommend the GUI mode over the command line because you can easily start, stop, and adjust settings. On the other hand, use the command line if you are resource constrained or want to get the most out of your test server.

Data Warehouses

Data Warehouse Key Points

1. **Data Warehouse systems can be successful if they create data that enables the business to make better decisions**
2. **Data Warehouse systems require a high amount of CPU, IO, and IOPS**
3. **Tuning must be designed in and include data organization, presorting, and correctly applied indexes**
4. **The DISOD Full table scanned: 6 billion rows in 44 seconds with flushed buffer cache and no tuning**

Data Warehouse Needs

The Data Warehouse Institute (DWI) defines business intelligence as a process that includes both the technology and tools required to turn data into business knowledge. Business knowledge drives profitable business actions. Moreover, Colin White, founder of BI Research, stated that the role of business intelligence is to provide, “the applications, tools, techniques, policies, and procedures for supporting analytical processes.” He argued that, “Information in and of itself is not useful. Even understanding information in and of itself is not useful. The goal of analysis is to enable better decisions.”

Because data warehousing efforts are so essential to business intelligence, warehouse engineers and business leaders responsible for that data must:

- Accept the leadership and responsibility to create systems that preserve business intention and need.
- Deliver data that is timely, valuable, of high quality, useful, and represents the enterprise common truth.
- Reduce total business intelligence costs by migrating toward managed or common systems of record instead of propagating data sources.

- Discover root causes of data discrepancy and provide proper training at the source, to remediate the need for frequent data scrubbing.
- Inspire trust within all employees, encouraging creative and innovative methods of ensuring data integrity.
- Break down barriers between business data silos and encourage data sharing across the organization.

Furthermore, they must not:

- Create data and data warehouses/services for the sake of having them.
- Focus on short-term goals while neglecting long-term vision.
- Rely only on complex technology to solve business problems.
- Discount one technology over another based on current popularity.
- Excuse itself from adoption of industry best practice because internal problems are different from the community.

Companies who do not heed this advice are bound to fail in their data warehousing efforts because their incorrect implementations will drive the cost of data warehouse projects too high and cause project delays.

The price associated to incorrect implementation come in two varieties, AKA hard and soft costs. We find that accountants love to scrutinize hard costs over soft costs because hard costs are easy to find and quantify; however, some soft costs can easily outgrow hard costs and exhaust multiple years of effort. For example, some companies might cringe at the hard costs associated to licensure of technology and opt for so-called free-ware. Then, they are required to hire many, highly specialized, engineers to hand-code solutions that may be trivial on the licensed platform. On the licensure side, the hard cost affixed includes maintenance and liability on the part of the software and or hardware vendor. On the free-ware side, companies may spend as much in employee salary as they will on the license, and, In some cases, they may spend much more. Even if they break even, those companies now have technical debt. Their homegrown product must now be maintained year-over-year. The end-result is that the free-ware method costs well beyond what the hard costs might have been.

On the other hand, companies that maintain correct focus on long-term vision while delivering on short-term gains tend to consistently deliver business value in their data warehousing efforts. When they are able to deliver data that is correct and timely, they help their executives make good business decisions that keep their company agile and competitive.

The ability to deliver data in a timely fashion becomes increasingly difficult as data volume grows into the Terabytes. For this reason many larger companies are turning to solutions provided via Oracle Exadata and Hadoop-Hive/Spark technologies. That's because both products provide for massive parallel execution of data processing.

It may be that a company cannot afford large, enterprise level systems that can cost millions of dollars for hardware alone. For that reason, middle-tier solutions can fill the void between small and enterprise systems. That's why we like the DELL DISOD so much. It's price point is nice and it delivers a good balance of CPU, Memory, and IO, all of which are mandatory for timely delivery of data.

To be successful, none of the CPU, Memory, and IO requirements can be neglected. For example, some might postulate that loading their servers with physical RAM will undoubtedly solve all of the warehouse needs. That's where they would be wrong. For one, the amount of money required to load their entire databases into memory is cost prohibitive. What's more, that data has to be loaded from physical storage to the memory at some time. If they have starved their systems of CPU and IO, that load will take a very-long-time. Likewise, if engineers load their servers with CPU and IO but neglect the amount of physical RAM the server has, they will undoubtedly cause their servers to swap memory to disk. It is far better to create systems that are well-balanced and tuned to take full advantage of CPU, Memory, and IO.

Data Warehouse Test Setup

We had to build a dataset of considerable size in order to test the extent of the DELL DISOD. We wanted to arrange the data into a dimensional model commonly known as the Star Schema. In the Star Schema, descriptive data is loaded into very wide but short tables called dimensions. Each row in a dimension represents unique attributes about the items a business wishes to measure.

For example, typical data warehouses contain descriptive dimensions that define customers, products, and promotions. Very deep but narrow tables, called facts are created that hold only measurable events or values.

In our Star Schema, we created two facts to organize events and earnings related to our video store model. The first fact was for sales and the second was for rentals. It is possible that these facts could have been merged but we wanted to tax the DISOD so we created two very deep facts. Each contained around 3 billion records. Our customer dimension held 5 million rows, and our product dimension held more than 800 thousand rows. Our date and promotion dimensions were relatively small.

Data Fabrication

Our data fabrication efforts were fairly simple but required a large set of seed data. Our first several scripts loaded the DISOD with large lists of names, and geographical information. We wanted to create a store the size of Blockbuster™ that contained a second arm of video rentals from kiosks like RedBox™.

We quickly realized that standard PL/SQL loops would be a very slow a process. This was the case even with bulk-collect methods. Instead, we turned to the trusty Create Table As (CTAS) method. It was the fastest way we used to create and move data. It is possible to reach the same speed if you significantly alter “`insert /*+ append */`” statements but the CTAS method was simple.

We also found that dynamic creation of iterators significantly effected our data creation efforts. We gained a speed increase of 4x when we created our iterator tables beforehand. They were simple to make but we were memory constrained (even with 1TB SGA). For example, we could not create iterator tables in the billions of rows without running out of memory. So we did the following:

```
CREATE TABLE iterator_1m
PARALLEL 80
NOLOGGING
COMPRESS FOR OLTP
PCTFREE 0
AS
    SELECT level i
      FROM dual
CONNECT BY level <= 1000000;
```

We created iterator tables like this until we hit 1 billion rows. Then, to overcome the memory limit and to speed the process up, our 5 and 10 Billion row iterators used the following technique:

```
CREATE TABLE iterator_5b
PARALLEL 80
NOLOGGING
COMPRESS FOR OLTP
PCTFREE 0
AS
WITH
  BIG_TABLE AS
  (
    SELECT * FROM iterator_1b UNION ALL
    SELECT * FROM iterator_1b UNION ALL
    SELECT * FROM iterator_1b UNION ALL
    SELECT * FROM iterator_1b UNION ALL
    SELECT * FROM iterator_1b
  )
SELECT rownum i
  FROM big_table;
```

Once we finished making our iterator tables, we used the Oracle built-in package dbms_random to create random joins between our iterator tables and the lookup tables we loaded earlier, like so (this block spans two pages):

```
CREATE TABLE CUSTOMER_DIM AS
WITH alpha AS (
SELECT
  ROWNUM row_num
, round(dbms_random.value(1, 8000)) address_number
, round(dbms_random.value(1, 38595)) street_name
, round(dbms_random.value(1, 38853)) zip_code
, round(dbms_random.value(1000000000000000, 9999999999999999)) credit_card
, round(dbms_random.value(1, 5)) credit_card_type
, round(dbms_random.value(1,3)) credit_term
, round(dbms_random.value(1920, 2000)) birth_year
, round(dbms_random.value(1,12)) birth_month
, round(dbms_random.value(1,30)) birth_day
, round(dbms_random.value(1,2)) gender
, round(dbms_random.value(15000, 125000)) income
, round(dbms_random.value(1,100)) race
, round(dbms_random.value(1,4000)) first_name
, round(dbms_random.value(1,4000)) middle_name
, round(dbms_random.value(1,997)) last_name
, round(dbms_random.value(1,9999999)) cell_phone
, round(dbms_random.value(1,9999999)) home_phone
, round(dbms_random.value(1,4000), 5) dt_created
FROM iterator_40m)
SELECT
  row_num customer_key
, 'USA' address_country
, A.address_number || ' ' || b.street_value || ' ' || c.city_name || ', '
|| c.state || ' ' || c.zip address_full
, A.address_number || ' ' || b.street_value address_primary
, NULL address_secondary
, NULL address_tertiary
, c.zip address_postcode
, c.state address_region
, cast((A.credit_card) AS varchar2(50)) credit_card
, CASE WHEN A.credit_card_type = 1 THEN 'AMEX'
       WHEN A.credit_card_type = 2 THEN 'VISA'
       WHEN A.credit_card_type = 3 THEN 'DISCOVER'
       WHEN A.credit_card_type = 4 THEN 'MASTER CARD'
       WHEN A.credit_card_type = 5 THEN 'OTHER'
  END credit_card_type
, CASE WHEN A.credit_term = 1 THEN '30 Days' WHEN A.credit_term = 2 THEN
'60 Days'       WHEN A.credit_term = 3 THEN '90 Days' END credit_term
, cast(A.birth_year AS varchar2(4)) || CASE WHEN A.birth_month < 10 THEN
'0' || cast(A.birth_month AS varchar2(4)) ELSE cast(A.birth_month AS var-
char2(4)) END ||   CASE WHEN A.birth_day < 10 THEN '0' || cast(A.birth_day
AS varchar2(4)) ELSE cast(A.birth_day AS varchar2(4)) END  DEMO-
GRAPHIC_BIRTHDATE
, CASE WHEN A.gender = 1 THEN 'M'
       WHEN A.gender = 2 THEN 'F'
  END DEMOGRAPHIC_GENDER
, A.income DEMOGRAPHIC_INCOME
, CASE WHEN A.race BETWEEN 1 AND 40 THEN 'WHITE'
       WHEN A.race BETWEEN 41 AND 59 THEN 'HISPANIC'
       WHEN A.race BETWEEN 60 AND 75 THEN 'BLACK'
       WHEN A.race BETWEEN 76 AND 100 THEN 'OTHER'
  END demographic_race
```

```
, CASE WHEN A.gender = 1 THEN fmn.gn_value
       WHEN A.gender = 2 THEN ffn.gn_value
  END || '.' ||
  CASE WHEN A.gender = 1 THEN mmn.gn_value
       WHEN A.gender = 2 THEN mfn.gn_value
  END || '.' ||
  ln.NAME || '@email.com' EMAIL
, NULL name_adopted
, ln.NAME name_family
, CASE WHEN A.gender = 1 THEN fmn.gn_value
       WHEN A.gender = 2 THEN ffn.gn_value
  END || ' ' ||
  CASE WHEN A.gender = 1 THEN mmn.gn_value
       WHEN A.gender = 2 THEN mfn.gn_value
  END || ' ' ||
  ln.NAME  name_full
, CASE WHEN A.gender = 1 THEN fmn.gn_value
       WHEN A.gender = 2 THEN ffn.gn_value
  END || ' ' ||
  CASE WHEN A.gender = 1 THEN mmn.gn_value
       WHEN A.gender = 2 THEN mfn.gn_value
  END name_given
, '(' || c.areacode || ')' || A.cell_phone  phone_cell
, NULL phone_fax
, '(' || c.areacode || ')' || A.home_phone phone_home
, NULL phone_pager
, sysdate - A.dt_created dt_created
, sysdate - A.dt_created DT_EFFECTIVE_START
, NULL DT_EFFECTIVE_END
, NULL dt_update
FROM  alpha A
JOIN   street_name b ON A.street_name = b.street_id
JOIN  zip_city c ON A.zip_code = c.zip_city_id
JOIN  list_male_name fmn ON fmn.gn_id = A.first_name
JOIN list_male_name mmn ON mmn.gn_id = A.middle_name
JOIN list_female_name ffn  ON ffn.gn_id = A.first_name
JOIN list_female_name mfn ON mfn.gn_id = A.middle_name
JOIN list_last_name ln ON ln.list_id = A.last_name;
```

Using methods like these allowed us to create an extremely large amount of realistic data in a very short period of time. The DELL DISOD had plenty of resources so we used as much as possible. If you attempt to do the same, you must remember that our DISOD was very well equipped. Our total Physical memory was 3TB, and our physical CPU cores totaled 80.

In addition, using prebuilt iterator tables and CTAS, we extensively used the Oracle built in package, dbms_scheduler, to create 80 simultaneous jobs that took our data creation efforts from 20 hours to around 8. We wish to impress on your mind the marvel that it was to create more than 200 Gigabytes of sample data in less than 8 hours. Normal systems would have difficulty producing that amount of data in 40 to 80 hours. Some systems may never be able to complete the task in a reasonable time.

Table Tuning

In order to see how fast the DISOD performed under different conditions, we created several copies of our video store schema. Those copies were used to test the way the DISOD handled real-world scenarios. Key variables included:

- Indexing (B-Tree & Bitmap)
- Partitioning - interval on date columns
- Compression - OLTP
- In-memory Column Storage and Compression
- Pre sorting data
- The PCTFREE parameter
- The Degree of Parallelism

Typical data warehouses are queried via AdHoc tools that shield business analysts from the writing of large and complicated SQL. These tools limit the way SQL can be written, opting out of performance in favor of database agnosticism. For this reason, we developed 4 queries that simulate the way that AdHoc tools perform queries against Star Schemas.

Our first query forces a full scan of the rental_item_fact. Our second query aggregates the sales of product by genre and state for the December 2012 time period. Our third query performs a pivot and then aggregates sales by region and year from 2004 to 2014. Finally, our fourth query aggregates sales by product genre for a single state over the life of the product.

We could have written many more queries but we wanted to focus on these four, simple ways to clearly measure how much work the DISOD could do, with different table and indexing configurations. We used the Linux Screen tool and PL/SQL to run all four queries while collecting system statistics. We were very pleased with the results.

Test Results

Our real time statistics package came from the *Oracle 11g PL/SQL Programming Workbook*, *Oracle 12c PL/SQL Programming Guide*, and *Oracle 12c PL/SQL Advanced Programming books* (particular attention to Chapter 10 of the Advanced Programing book). This package enabled us to view all of the system statistics, including the ASM subsystem.

The DELL DISOD performed beautifully in our data warehouse tests. We ran each test several times and then averaged the results to make sure our numbers were consistent. In all, our warehouse tests took a little more than 8 hours to complete. During this time the DISOD had:

- 5.7 Gigabytes per second sustained IO
- 95% CPU utilization (8 x 10c E7 processors)
- Full table scan: 6 billion rows in 44 seconds with flushed buffer cache
- 13 second full scan from cache after tuning
- 5,800 multiblock reads per second (db_file_multiblock_read_count=128)
- 742,400 blocks read per second
- Easily met the 7000 IOPs required for rapid query returns

We compared these results to a standard 2 Node RAC cluster, with traditional spinning disks. The best it could do was:

- 0.5 Gigabytes per second sustained IO
- Full Table Scan: 41 million rows in 8 seconds with flushed buffer cache

Furthermore, our tuning efforts showed that the DISOD completed the 6 billion row scan in as little as 13 seconds!

- 6 billion rows scanned in 44 seconds, buffer flushed, no tuning
- 6 billion rows scanned in 25 seconds tuned, buffer flushed
- 6 billion rows scanned in 13 seconds tuned, cached

We were incredibly impressed with the DELL DISOD and would recommend the solution to any shop wishing to span the gap between small/medium and large systems like Oracle Exadata. We have equal praise for Oracle's products and have been happy with their performance. Our experience with Exadata spans multiple years and the purchase and configuration of five, Exadata quarter-RAC systems. While we are happy with Oracle, we recognize the need for a mid-range solution. We feel that DELL nailed it and are happy to endorse their product. Our tests demonstrate clearly that DISOD is compelling for either transactional or data warehouse loads – or both! Hybrid deployments are feasible If intelligently managed using Resource Management Groups and Pluggable Databases.

***John Harper** currently works for workfront.com as a Data Architect and Sr. Database Administrator. He also worked for the Church of Jesus Christ of Latter-day Saints as a principal database engineer. He thoroughly enjoys working with data warehousing, business intelligence, and database engineers there. He has been working with databases for the past 14 years, specializing in Oracle administration, database architecture, database programming, database security, and information quality.*

John's mentors include Michael McLaughlin, Robert Freeman, Danette McGilvary, and many others who have spent considerable time becoming experts in their industry. John is both awed and inspired by their abilities and feels lucky to be associated with them.

Recently, John has had the opportunity to work closely with some of the top-notch minds in database security. He hopes to produce a series of publications focused on Oracle products such as Oracle Audit Vault and Database Firewall (AVDF) and Oracle Data Redaction.

John enjoys Japanese martial arts. During his teenage years and early adulthood, he took jujitsu, karate, judo, and aikido. He loves aikido and hopes to teach it one day. He would also love to learn kyudo if he can find any spare time. John lives with his wife of 24 years in Northern Utah County, Utah. They have two adopted daughters, whom they cherish and thoroughly spoil.

John's blog is at http://security.mclaughlinsoftware.com.

***Brandon Hawkes** is a database engineer for USANNA. He also worked for the Church of Jesus Christ of Latter-day Saints.*

He is responsible for maintaining the principal data warehouse. He enjoys developing business intelligence solutions that improve database performance.

Brandon graduated from BYU–Idaho with a degree in financial economics and a minor in computer information technology. He currently lives in Utah with his wife and three kids. During his free time, Brandon plays semi-pro football in the Rocky Mountain Football League, and he referees high-school football and basketball.

***Michael McLaughlin, D.C.S.**, is a professor at BYU–Idaho in the Computer Information Technology Department of the Business and Communication College. He is also the founder of McLaughlin Software, LLC, and is active in the Utah Oracle Users Group. He is the author of eight other Oracle Press books, such as Oracle Database 12c PL/SQL Programming, Oracle Database 11g PL/SQL Programming, and Oracle Database 11g PL/SQL Workbook.*

Michael has been writing PL/SQL since it was an add-on product for Oracle 6. He also writes C, C++, Java, Perl, PHP, and Python.

Michael worked at Oracle Corporation for over eight years in consulting, development, and support. While at Oracle, he led the release engineering efforts for the direct path CRM upgrade of Oracle Applications 11i (11.5.8 and 11.5.9) and led PL/SQL forward compatibility testing for Oracle Applications 11i with Oracle Database 9i. He is the inventor of the ATOMS transaction architecture (U.S. Patents #7,206,805 and #7,290,056). The patents are assigned to Oracle Corporation.

Prior to his tenure at Oracle Corporation, Michael worked as an Oracle developer, systems and business analyst, and DBA beginning with Oracle 6.

Michael lives in eastern Idaho within a two-hour drive to Caribou-Targhee National Forest, Grand Teton National Park, and Yellowstone National Park. He enjoys outdoor activities with his wife and children (six of nine of whom still live at home).

Michael's blog is at http://blog.mclaughlinsoftware.com. His twitter handle is @MacLochlainn.

B-Tree

In computer science, a B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children (Comer 1979, p. 123). Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. It is commonly used in databases and filesystems.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

Big Data

Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate. Challenges include analysis, capture, curation, search, sharing, storage, transfer, visualization, and information privacy. The term often refers simply to the use of predictive analytics or other certain advanced methods to extract value from data, and seldom to a particular size of data set.

Source: Wikipedia

Related Glossary Terms

RDBMS

Index Find Term

Business Intelligence (BI)

Business intelligence (BI) is the set of techniques and tools for the transformation of raw data into meaningful and useful information for business analysis purposes. BI technologies are capable of handling large amounts of unstructured data to help identify, develop and otherwise create new strategic business opportunities. The goal of BI is to allow for the easy interpretation of these large volumes of data. Identifying new opportunities and implementing an effective strategy based on insights can provide businesses with a competitive market advantage and long-term stability.

BI technologies provide historical, current and predictive views of business operations. Common functions of business intelligence technologies are reporting, online analytical processing, analytics,data mining, process mining, complex event processing, business performance management, benchmarking, text mining, predictive analytics and prescriptive analytics.

BI can be used to support a wide range of business decisions ranging from operational to strategic. Basic operating decisions include product positioning or pricing. Strategic business decisions include priorities, goals and directions at the broadest level. In all cases, BI is most effective when it combines data derived from the market in which a company operates (external data) with data from company sources internal to the business such as financial and operations data (internal data). When combined, external and internal data can provide a more complete picture which, in effect, creates an "intelligence" that cannot be derived by any singular set of data.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

DAAD

The Dell Acceleration Appliances for Databases (DAAD) storage array is a high performance, low-latency, backend, shared, storage device. Each unit contains 4 SanDisk ioDrive2 PCI-E NAND flash storage cards, 2 dual-port 16Gb Fiber Channel cards, and 1 dual-port Mellanox ConnectX-3 40Gb iSCSI card for the dedicated mirroring interconnect. The DAAD is an essential component in the Dell Integrated System for Oracle Databases (DISOD) and is capable of incredible speed, low-latency, and >1M IOPS.

Related Glossary Terms

DISOD, Terabyte

Index Find Term

DISOD

The **Dell Integrated System for Oracle Database (DISOD)** is a fully integrated hardware stack that is purpose built as a high performance Oracle Database rack solution. DISOD provides an out-of-box experience for customers where everything up to the point of the Oracle Database software installation is pre-configured and pre-installed. DISOD ships with pre-integrated racked, stacked and cabled hardware and with Oracle Linux with Unbreakable Enterprise Kernel (UEK) operating system software pre-installed and optimized. For the purpose of our testing, the DISOD was configured with Oracle Database 12c (12.1.0.1), although the DISOD supports Oracle Database versions back to 10g. The owner's guide below provides the details on the DISOD hardware components, rack and cabling information, and network and software configuration.

For detailed hardware and software configurations, please refer to:
Dell Integrated Systems for Oracle Databases: Owner's Guide 1.1

Related Glossary Terms

DAAD, Terabyte

Index Find Term

Exadata

Oracle Exadata is an architecture featuring scaleout industry-standard database servers, scale-out intelligent storage servers, and a high speed InfiniBand internal fabric that connects all servers and storage. Unique software algorithms in Exadata implement database intelligence in storage, PCI based flash, and InfiniBand networking to deliver higher performance and capacity. Four sizes of the Exadata Database Machine X4-2 are available, starting from the eighth rack system with 2 database servers and 3 Exadata Storage Servers, to the full rack system with 8 database servers and 14 Exadata Storage Servers.

Source: **Oracle Exadata Database Machine X4-2 Data Sheet**

Related Glossary Terms

Drag related terms here

Index Find Term

Gigabyte

One gigabyte is 1,000,000,000 bytes. This definition is used in all contexts of science, engineering, business, and many areas of computing. However, historically, the term has also been used in some fields of computer science and information technology to denote the gibibyte, or 1073741824 (10243 or 230) bytes. For instance, the memory standards of JEDEC, a semiconductor trade and engineering society, define memory sizes in this way.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

IOPS

IOPS (Input/Output Operations Per Second, pronounced eye-ops) is a common performance measurement used to benchmark computer storage devices like hard disk drives (HDD), solid state drives (SSD), and storage area networks (SAN). As with any benchmark, IOPS numbers published by storage device manufacturers do not guarantee real-world application performance.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

JSON

JSON (/ˈdʒeɪsən/ jay-sən), or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML.

Source: Wikipedia

Related Glossary Terms

RDBMS

Index Find Term

OLTP

Online transaction processing, or OLTP, is a class of information systems that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing. The term is somewhat ambiguous; some understand a "transaction" in the context of computer or database transactions, while others (such as the Transaction Processing Performance Council) define it in terms of business or commercial transactions. OLTP has also been used to refer to processing in which the system responds immediately to user requests. An automated teller machine (ATM) for a bank is an example of a commercial transaction processing application. Online transaction processing applications are high throughput and insert or update-intensive in database management. These applications are used concurrently by hundreds of users. The key goals of OLTP applications are availability, speed, concurrency and recoverability. Reduced paper trails and the faster, more accurate forecast for revenues and expenses are both examples of how OLTP makes things simpler for businesses. However, like many modern online information technology solutions, some systems require offline maintenance, which further affects the cost-benefit analysis of online transaction processing system.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

One to Many

In relational data modeling a one to many relationship exists when one parent is correlated with many children. For example a person can have more than one phone number. This is accomplished via unique keys called primary keys and child reference keys called foreign keys. The PERSON table owns a PERSON_ID unique identifier, which is referenced in the PHONE table like so:

```
PERSON                      TELEPHONE
=======                     ==========
person_id -||------------0< person_id
                            telephone_id
                            telephone_value

PERSON_ID GIVEN_NAME FAMILY_NAME
        1 FRED       FLINSTONE

PERSON_ID TELEPHONE_ID TELEPHONE_VALUE
        1            1 +1 123 555-1234
        1            2 +1 123 555-2345
        1            3 +1 123 555-3456
```

Related Glossary Terms

Drag related terms here

Index

Find Term

RAM

Random-access memory (RAM /ræm/) is a form of computer data storage. A random-access memory device allows data items to be read and written in roughly the same amount of time regardless of the order in which data items are accessed. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement delays.

Today, random-access memory takes the form of integrated circuits. RAM is normally associated with volatile types of memory (such as DRAM memory modules), where stored information is lost if power is removed, although many efforts have been made to develop non-volatile RAM chips. Other types of non-volatile memory exist that allow random access for read operations, but either do not allow write operations or have limitations on them. These include most types of ROM and a type of flash memory called NOR-Flash.

Integrated-circuit RAM chips came into the market in the late 1960s, with the first commercially available DRAM chip, the Intel 1103, introduced in October 1970.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

RDBMS

Relational Database Management Systems make up the bulk of transactional databases worldwide. They are based on the relational model, invented by Dr. Edgar F Codd, of IBM’s San Jose Research Laboratory.

E.F. Codd suggested that hierarchal and network databases contained crucial flaws that caused data to be out-of-date or orphaned due to transactional anomalies. What’s more, legacy database systems required the creation of complex programming to both put data in and retrieve it out of its databanks.

A modern-day example of hierarchal databases is the XML structure. Unlike databases in the past, XML can be easily queried and is less rigid; however, the XML structure remains hierarchal none-the-less.

Oracle’s current offering includes true object-oriented structures, XML, JSON, and relational database structures. This is why Oracle makes the distinction that they are an Object Relational Database Management System (ORDBMS).

Related Glossary Terms

Big Data, JSON, XML

Index Find Term

SGA

In the database management systems developed by the Oracle Corporation, the System Global Area (SGA) forms the part of the system memory (RAM) shared by all the processes belonging to a single Oracle database instance. The SGA contains all information necessary for the instance operation.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index

Find Term

SQL

SQL (i/ˈɛs kjuː ˈɛl/, or i/ˈsiːkwəl/; Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, though, most SQL code is not completely portable among different database systems without adjustments.

Source: Wikipedia

Related Glossary Terms

Drag related terms here

Index Find Term

Terabyte

The terabyte is a multiple of the unit byte for digital information. The prefix tera represents the fourth power of 1000, and means 1012 in the International System of Units (SI), and therefore one terabyte is one trillion (short scale) bytes. The unit symbol for the terabyte is TB.

1 TB = 1000000000000bytes = 1012bytes = 1000gigabytes.

Source: Wikipedia

Related Glossary Terms

DAAD, DISOD

Index Find Term

XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards.

The design goals of XML emphasize simplicity, generality and usability across the Internet.[5] It is a textual data format with strong support viaUnicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services.

Source: Wikipedia

Related Glossary Terms

RDBMS

Index Find Term

Zettabyte

A zettabyte is one sextillion bytes of contiguous information. Mark Liberman, an American linguist and professor in both phonetics and computer science at the University of Pennsylvania theorizes that all human speech ever spoken would fill 43 zettabytes if it were digitized with 16-bit audio.

Related Glossary Terms

Drag related terms here

Index Find Term