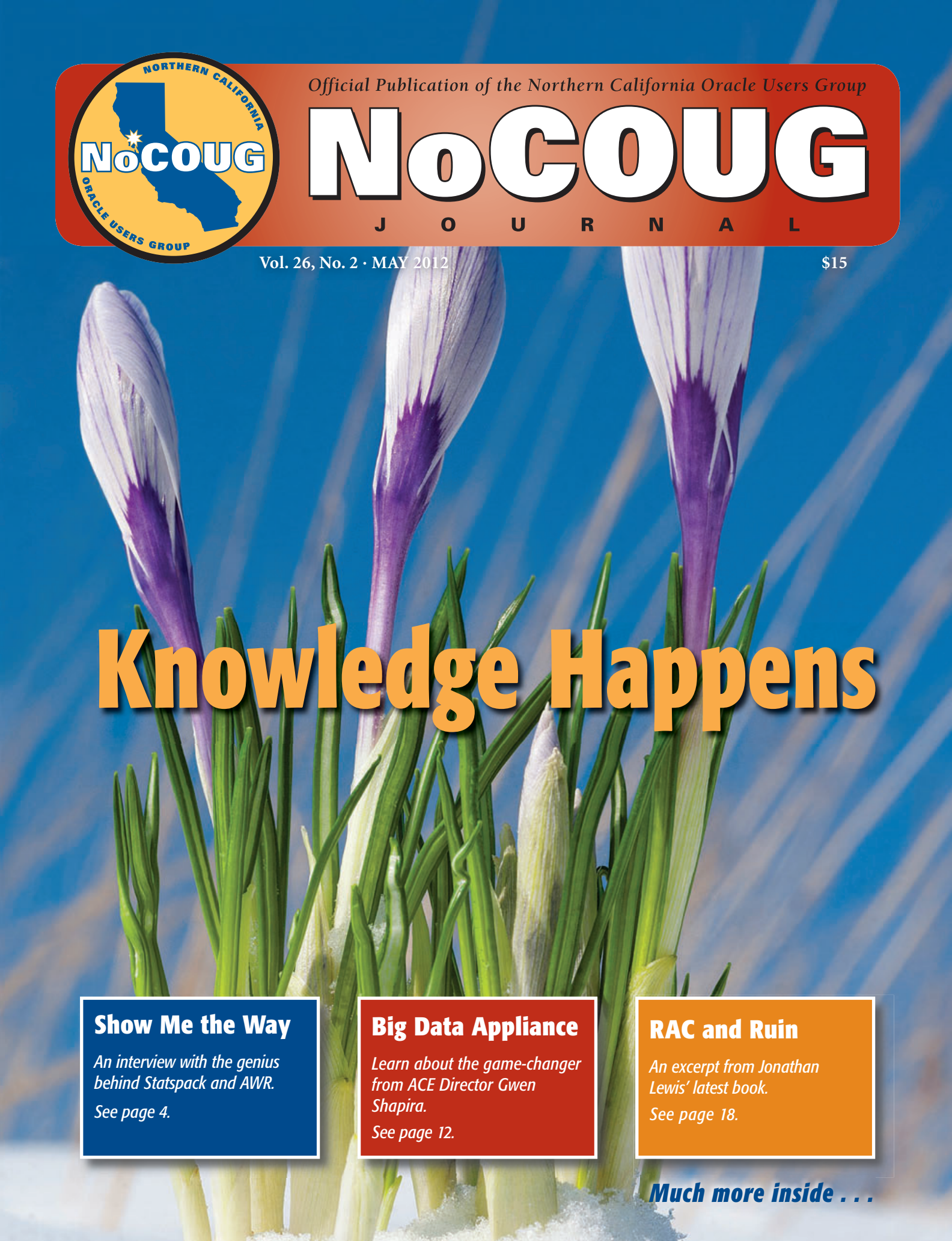
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 26, No. 2 · MAY 2012 $15



Knowledge Happens

Show Me the Way

An interview with the genius behind Statspack and AWR.

See page 4.

Big Data Appliance

Learn about the game-changer from ACE Director Gwen Shapira.

See page 12.

RAC and Ruin

An excerpt from Jonathan Lewis' latest book.

See page 18.

Much more inside . . .

Thanking the Team

Take a moment to think about the huge amount of effort that goes into this publication. Your first thought might be about the care and attention required of the authors. Yes, writing is hard work. Now consider each author's years of hard-won experience; then add it up. The cumulative amount of time spent to acquire the knowledge printed in each issue is decades—maybe even centuries.

But let's take a moment to thank the people who make it possible for us to share this knowledge with you. Without the dedication and skill of our production team, all that we'd have is a jumble of Word files and a bunch of JPEGs. Copyeditor Karen Mead of Creative Solutions transforms our technobabble into readable English. Layout artist Kenneth Lockerbie and graphics guru Richard Repas give the *Journal* its professional layout.

Finally, what really distinguishes this *Journal* is that it is actually printed! Special thanks go to Jo Dziubek and Allen Hom of Andover Printing Services for making us more than just a magnetically recorded byte stream. ▲

—*NoCOUG Journal* Editor

Table of Contents

ADVERTISERS

Show Me The Way

with Graham Wood

Graham Wood

Graham Wood has been working with Oracle Database for 25 years. He is currently a product manager for the Oracle RDBMS based in Redwood Shores, Calif. He has architected and tuned some of the largest Oracle databases, and has presented around the world on Oracle performance–related topics.

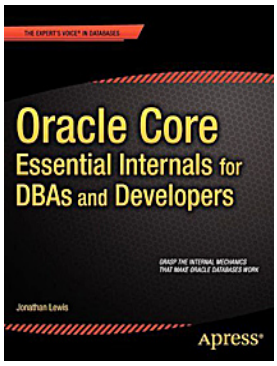
I have it on very good authority (Tom Kyte in the current issue of* Oracle Magazine*) that you are the genius and innovator behind Statspack and Automatic Workload Repository. I am in awe. Tell me the story behind that.

Wow, starting with a memory test! When Oracle V6 was introduced it contained the first V$ views, such as V$SYSSTAT and V$FILESTAT. These structures were created to allow database development to understand which parts of the code were being executed, and how often, during the running of the OLTP benchmarks that had started to appear at that time. The database shipped with two scripts that were used to produce a report from the V$ views during a benchmark run. These were bstat.sql, which captured the current contents of the V$ views at the start of the benchmark into a set of tables, and estat.sql, which captured the contents at the end of the benchmark into another set of tables, produced a report from the two sets of tables, and then dropped them. I was working in a small specialist performance group in Oracle UK at the time and it occurred to us, being database guys, that it might be useful for production systems to do regular captures of the V$ views and to keep this data around for rather longer as a repository of performance data. We wrote some scripts and started to distribute them inside Oracle, and they also found their way out to several customers. This was the original "Stats Package," as we called it. As new releases of the database came out, I upgraded the scripts, probably most notably with the inclusion of V$SQL in Oracle V7 in the Stats7 package. In 1996 I moved to Oracle HQ in Redwood Shores to work in the Server Technologies Performance Group, and one of the goals that I set myself was to get the scripts shipped with the product so that all customers could use them. They finally made it into the database distribution in Oracle 8*i* as Statspack after being updated and enhanced by Connie Green. And the rest, as they say, is history, with almost all big Oracle sites using Statspack to keep a history of performance data.

When we started development of Oracle 10*g*, one of the main focus areas for the release was to be manageability, and a key part of that was to simplify performance analysis and to make recommendations for performance improvement. The most important part of this for me was to be able to automate performance analysis for a database instance and to identify the key areas where improvements could be made. Basically, unless the analysis is correct, there is no point in trying to make recommendations. In order to do this we needed to have a common currency for reporting across the components of the database and for quantifying the performance of a system. This led to the introduction of the concept of DB Time, the time spent in the database by user sessions, which allowed us to do quantitative comparisons between different components and also to quantify the impact on the system of an issue—for example that a single SQL statement represents 27% of all of the user time spent in the database. One of the main objectives of this was to make DBAs more effective by directing them to areas where they were likely to be able to make the greatest improvements in performance, rather than them spending time and effort on making changes that produced little benefit. To do all of this needed much more infrastructure than there was in Statspack and in Oracle 10*g*, and a lot of effort went into ensuring that we had adequate data available to do analysis of a performance problem the first time that it occurred. This resulted in an automatically managed repository of data (AWR), which contained not only data from normal V$ views containing cumulative statistics but also metric data and sampled activity data in the Active Session History. The combination of all of these data sources has taken performance analysis to a different level.

Tom Kyte's favorite performance story is about a database that was always slow on rainy Mondays. What's your favorite performance story from your own experiences?

"The most common problem that I see is that of flawed analysis: fixating on a particular statistic or event, which means that you never get to the root cause of the problem."



Beating the Oracle Optimizer!

with Jonathan Lewis

August 17

This is a follow-on from last year's course "Designing Optimal SQL." It focuses on writing SQL to emulate transformations that are currently not available to the Oracle Optimizer. Last year's course was essentially a guide to writing "normal" SQL in the best possible way; this course is about writing "abnormal" SQL, because that's the only efficient thing to do sometimes!

We start with a brief reminder of what we need to know to write efficient and scalable SQL. Then we examine the B-tree and bitmap access paths that are automatically available to Oracle as it accesses a single table, before extending Oracle's strategies with a few manually constructed strategies that become available if we can rewrite the SQL.

We next examine a very simple join and note a fundamental limitation in the optimizer's ability to find the best strategy for joining two tables. We see how we can overcome this limitation—at a cost of more complex SQL—and look at the way we need to think about joins to minimize the work we do, noting that the possible benefit isn't always as great as we might first think. We also review a method for keeping the code simpler at a risk of variable performance, and then find that we can eliminate the variability by again increasing the complexity.

After setting the groundwork with single-table access paths and two-table joins, we go on to more complex examples, showing how the principle can be used to emulate data warehouse query patterns in a structure designed for OLTP data access—even to the extent of emulating a Star Transformation in Standard Edition Oracle where bitmap indexes are not implemented.

Falling back to slightly more standard SQL, we take a look at the way in which we can use features like function-based indexes, virtual columns, and deterministic functions in the newer versions of Oracle to reduce work. We also look at the ways in which structures such as sorted hash clusters and partitioning allow us to rethink the way we write SQL to minimize the work done.

Register at http://www.speak-tech.com

One company that I worked with early on in my Oracle career asked me to help them improve the performance of a large batch report which was produced every night and went out to six people around the organization. It was causing problems for all of the rest of their batch operations by consuming a large amount of resources. The first improvement was to run the report once and print six copies rather than run the same report six times! Then I spoke to the folks who received the report and found out that three of them immediately tossed it in the trash (this was before the days of recycling), and the other three never looked beyond the first four summary pages as they now had an online system that allowed them to look at the details. We ended up changing the report to just produce the summary, and the overnight batch load on the system dropped by about 95% from the start point. It was definitely a case of it always being faster to *not* do something than to do it.

> ***"The title of software professional comes with a requirement to deliver quality product, not just hope that hardware will bail you out."***

What are the typical issues you see when you are asked to look at a performance problem? Indexes? Statistics?

Well by the time I get called in to look at a performance problem these days there have probably already been quite a few people looking at it before, so all of the obvious things have already been tried. So, to be honest, the most common problem that I see is that of flawed analysis: fixating on a particular statistic or event, which means that you never get to the root cause of the problem and you end up trying to deal with a long list of symptoms. Much better to take a top-down approach and make sure you have the real cause before trying to fix things. If you have a really bad headache you could try and find a better aspirin or lie down in a darkened room, but you might be better to just stop banging your head against the wall. Having said that, I do still see a lot of problematic SQL, and drilling down to the root cause has become so much easier with the introduction of SQL Monitor. It is one of my top features of Oracle 11*g*, both for DBAs and developers, as it makes it so easy to find out exactly where in the plan the high resource usage and bad cardinality estimates are coming from, without even having to look at the details of the SQL itself. And, of course, I still see applications that have poor connection management and perform unnecessary parsing, even though we have been telling folks how to do it right for a couple of decades now.

I've heard a rumor that attendees of the Real World Performance events are being told that "tune" is a four-letter word. Is that some sort of insider joke? What does it mean?

I think that you have me confused with Cary Millsap! Cary differentiates between "tuning" and "optimizing." The four-letter word that we talk about in the Real World Performance Day is "hack." We define hacking as making changes without having diagnosed the root cause of the problem, without having scoped the problem or solution, and without being able to detail the expectation, in terms of what changes can be expected in the database performance statistics, of applying the "fix." Most commonly these days the supporting argument for applying a hack is "well, I found a website that said if I set _go_faster in the init.ora I will run at least three times faster." While Google can obviously be a good source of information, you have to remember that not everything that you read on the Internet is true. There really is no good alternative to doing proper performance analysis (although the availability of DB Time and ADDM make it easier) and proper testing, in your environment and with your data.

In* Oracle on VMware*, Dr. Bert Scalzo makes a case for "solving" performance problems with hardware upgrades. What's your opinion about this approach?[1]

Ah, the "hardware is the new software" approach, as my colleague Andrew Holdsworth calls it. Software was called software because it was the part of the system that was "soft' and could easily be changed. These days we often see customers who will do anything they can to avoid changing the application, no matter how bad it is. Hardware upgrades can only ever "ameliorate" a subset of performance problems. If the system is CPU bound, then adding more CPU cycles may make things better, but the benefits that you get will be, at best, the 2x every 18 months of Moore's Law. But most systems with performance problems these days are not CPU bound, and even when they are, there is also a real possibility that adding more CPU will actually further reduce the performance of the system by increasing contention on shared structures. The performance benefits of fixing the software can be orders of magnitude greater and, if done well, make it so that the system is better able to scale with hardware upgrades. The cheap hardware theory primarily applies to CPU, although larger, cheaper memory can also help but often requires that the box is changed anyway. Storage system upgrades are rarely cheap. Although $/GB has been falling rapidly, $/GB/s and $/IOP/s have not, and reducing I/O performance problems will always involve increasing either one or the other of these throughput

[1] Here's the full quote from Dr. Scalzo's book: *"Person hours cost so much more now than computer hardware even with inexpensive offshore outsourcing. It is now considered a sound business decision these days to throw cheap hardware at problems. It is at least, if not more, cost effective than having the staff [sic] tuned and optimized for the same net effect. Besides, a failed tuning and optimization effort leaves you exactly where you started. At least the hardware upgrade approach results in a faster/better server experiencing the same problem that may still have future value to the business once the fundamental problem is eventually corrected. And, if nothing else, the hardware can be depreciated, whereas the time spent tuning is always just a cost taken off the bottom line. So, with such cheap hardware, it might be a wiser business bet to throw hardware at some solutions sooner than was done in the past. One might go so far as to make an economic principle claim that the opportunity cost of tuning is foregoing cheap upgrades that might fix the issue and also possess intrinsic value. Stated this way, it is a safe bet that is where the business people would vote to spend."*

metrics. I would guess that most of the readers of your magazine would think of themselves as software professionals. To me that title comes with a requirement to deliver quality product, not just hope that hardware will bail you out.

Saying No to NoSQL

Just when I thought I'd finished learning SQL, the NoSQL guys come along and tell me that SQL databases cannot deliver the levels of performance, reliability, and scalability that I will need in the future. Say it isn't so, Graham.

Well we hear much pontificating about the benefits of NoSQL, but so far I haven't seen any audited industry-standard benchmark results as proof points. I have seen many claims from NoSQL evangelists that traditional RDBMSs cannot meet their requirements, only to find on further analysis that they tried a single open-source RDBMS, ran into some problems, and generalized from there. It is also interesting in the light of your previous question about using cheap hardware to try and resolve performance problems, that NoSQL solutions are developer intensive, as much of the functionality that would be provided by a SQL RDBMS has to be handcrafted for each solution. But I'm sure over time we will see winners appear from the current plethora of NoSQL products.

What about Big Data? Can't SQL databases handle big data then?

To me the case for Big Data comes down to two key areas: unstructured data and high-volume, low-value data such as web logs. This data could be stored in an RDBMS, but more typically what we are seeing customers doing is using Big Data techniques to extract information from these types of data sources and then storing this data in their RDBMS. This is the type of environment that Oracle's recently announced Big Data Appliance is designed to help with.

The NoSQL salesmen insist that I need "sharding" instead of partitioning. Did they get that right?

Partitioning in the database has the huge benefit of being transparent to your application and your application developer. Using sharding requires that you move the management of the shards into your own application code. Do you want to develop your own code to perform queries across all of your shards and to do two-phase commits when you need to do a transaction that would affect multiple shards? And is such custom code development really cheap?

Professor Michael Stonebraker claimed in the 100th issue of the* NoCOUG Journal *that traditional SQL databases should be* "sent to the home for tired software." *Has innovation really stopped at 400 Oracle Parkway? Has Larry sailed off into the sunset?

There have been many technologies that have claimed that they will replace SQL RDBMS over the last 30 years, including object databases and XML. SQL databases are still alive and well and contain the mission-critical data that is the lifeblood of businesses. Having a standard language, SQL, and a sound basis on relational theory means that SQL databases have stood the test of time in an industry where hype and fashion are rampant. In terms of 400 Oracle Parkway (where most of database development is housed) there are still many new features being built into the Oracle database that will increase the benefit that customers get from using the product. But you will have to wait for the product announcements to hear about those. And, of course, as the next America's Cup is in San Francisco. Larry is still very much around and involved.

The Whole Truth About Exadata

Is Exadata a case of solving performance problems with hardware upgrades? Put another way: is the performance improvement from Exadata exactly what one might expect from the bigger sticker price, no more and no less?

Well the stock answer is that it is an engineered system that is designed to be capable of very high throughput. The software allows us to utilize the hardware much more effectively. There are customers who have upgraded to Exadata and seen the hardware upgrade benefits, typically 5–10x performance improvement, which is enough to get them into ads in *The Economist* and airports around the world. But the customers who have fully exploited the capabilities of Exadata have seen orders of magnitude more benefit. In our Day of Real World Performance presentations we load, validate, transform, collect optimizer statistics, and run queries on 1TB of raw data in less than 20 minutes. That sort of performance can transform what IT can deliver to the business and has far greater value than the sticker price.

Is Exadata as good for OLTP workloads as it is for OLAP? (You can be frank with me because what's said in these pages stays on these pages!)

Well Exadata is certainly a very capable OLTP box. It has fast CPUs and can perform huge numbers of very fast I/Os with large numbers of IOPS by utilizing the flash cache in the storage cells. And OLTP performance is all about CPU horsepower and large numbers of IOPS. But I think it is fair to say that there is less "secret sauce" in Exadata as an OLTP platform than there is for data warehousing.

Show Me the Way

Thank you for answering my cheeky questions today. Someday, I hope to know as much about Oracle Database performance as you. Can you show me the way? Your book, perhaps?

Well I think that the key to being a good performance analyst is making sure that you spend time upfront correctly scoping the problem and then avoid jumping to conclusions while doing a top-down analysis of the data. When you are looking for solutions, make sure that the solution that you are implementing matches the scope of the problem that you started with, as opposed to a mismatched scope. The classic example of scope mismatch is making a database-level change, like changing an init.ora parameter, to solve a problem that is scoped to a single SQL statement. Much better to use techniques like SQL Profiles or SQL Baselines that will only affect the single SQL. Using that approach will get you a long way. As far as my book, I guess I still need to write it; it will be a cheeky book! ▲

Oracle WebLogic Server 11g Administration Handbook

A Book Review by Brian Hitchcock

Details

Authors: Sam Alapati
ISBN: 978-0-07-177425-3
Pages: 528
Year of Publication: 2011
Edition: 1
List Price: $60
Publisher: Oracle Press
Overall Review: Excellent resource for anyone new to WebLogic Server.
Target Audience: Anyone that will be managing WebLogic Server.
Would you recommend this book to others: Yes.
Who will get the most from this book? Administrators.
Is this book platform specific: No.
Why did I obtain this book? See overall review below.

Overall Review

I need to be prepared to support Fusion Applications. I found many resources, and among them were three books from Oracle Press. This is the second of them. The first was *Managing Oracle Fusion Applications* and the third is Handbook and *Oracle Fusion Middleware 11g Architecture and Management*. Since I have reviewed other Oracle Press books, they sent me copies of each to read and review.

Fusion Applications is built on top of Fusion Middleware, and WebLogic Server is a central piece of Fusion Middleware. WebLogic Server (WLS) provides the "container" where Java EE applications are deployed. The business functionality of Fusion Applications is provided by Java EE applications, which explains why WebLogic Server is critical to supporting Fusion Applications.

Even if you won't be supporting Fusion Applications, WebLogic Server is replacing Oracle Application Server in various Oracle products. As an example, the latest version of Oracle Enterprise Business Suite (EBS), version 12.2, will have WebLogic Server.

I've been working with WebLogic Server and Fusion Applications for about a year, and I wanted to read this book because I had several specific issues I wanted to learn about. I saw many WebLogic Server instances go into the state FAILED_NOT_RESTARTABLE, and I needed to understand what causes this. While this book never mentions this issue by name, I did learn how the WebLogic Server instance goes through various states while starting up. When this process encounters issues, this is when the instance fails.

This book is very dense with specific technical detail. I learned many things about managing WebLogic Server. The focus is on how things work so that you can implement and manage WebLogic Server in your environment. Many issues that had been confusing to me about WebLogic Server were cleared up by reading this book.

Introduction

The introduction explains that WebLogic Server is mainly used to deploy web applications. It provides the environment specified by the Java Enterprise Edition (Java EE) standards to support enterprise Java applications. Further, WebLogic Server is itself a Java program that supports various services needed by Java EE applications. I agree with the author's view that WebLogic Server presents an overwhelming number of new concepts that a beginner must deal with. I've been there very recently myself. The target audience is explained to be WebLogic Server administrators, middleware administrators, and DBAs that need to support WebLogic Server. This is important in that many DBAs will get tasked with WebLogic Server support, if only because it is another Oracle product. The difference between web server and web application server (such as WebLogic Server) is explained clearly. I was reassured to read that no prior knowledge of WebLogic Server or Java programming is assumed or needed.

Chapter 1—Installing WebLogic Server and Using the Management Tools

The three major administrative tools for managing WebLogic Server are identified here: the Administration Console, the Node Manager utility, and the WebLogic Scripting Tool (WLST). By the way, it turns out that you must like acronyms to work with WebLogic Server. The available versions of WebLogic Server are explained (SE, EE, and Suite). The author points out that there is a lot of new terminology to learn: for example, the concept of a "machine" in the WebLogic Server world, which can be confusing. This caught my attention because I was confused about this term myself. The list of terms that is explained, each in detail, is lengthy and welcome. I learned a lot from this. Following this is a section on WebLogic Server concepts. Again, I learned a lot. Specific examples include Configuration and Runtime MBeans, Listen Ports and Listen Threads, and why there are two versions of the Java

Virtual Machine (JVM) included with WebLogic Server.

The chapter then covers the WebLogic Server installation process. The author covers this in detail, and I installed WebLogic Server and the sample applications on my laptop. I highly recommend this. Using the Administration Console, logging in, and navigating all make a lot more sense when you are using your own installation. WebLogic Scripting Tool is also covered and many examples are explained.

Chapter 2—Administering WebLogic Server Instances

Managing WebLogic Server instances is the focus of this chapter. The Admin Server and Managed Servers are explained as well as the difference between development and production modes. The many options available for starting and stopping WebLogic Server instances are covered. Node Manager capabilities, as well as how to start, stop, monitor and configure Node Manager, are covered. When WebLogic Server starts up it goes through various "states," ending up in a RUNNING state ready to accept and service client requests. This process, referred to as the "life cycle," was something I had not seen before. It helped me understand what causes WebLogic Server instances to fail.

This chapter also provides an introduction to the Ant tool. Ant is a Java-based tool that uses XML build files to start and stop WebLogic Server instances as well as many other tasks. Again, this helped me because I had seen the message "BUILD SUCCESSFUL" many times but didn't know this was generated from Ant.

Chapter 3—Creating and Configuring WebLogic Server Domains

The concept of a domain is central to understanding WebLogic Server. The structure of a domain is covered, including server instances, server clusters, and the directory structure that is set up when you create a domain. The XML configuration file for the domain is explained. MBeans come up again as these are the mechanism Java uses to monitor and change the configuration of servers within the domain.

When you make changes to the configuration of any of the servers in a domain, you do so through the Administration Console. This process makes changes to the editable MBeans, and the Admin Server uses a Lock and Edit Mechanism to make sure only one person is making configuration changes at a time. This process is covered as well as creating domain templates. These templates are useful when you need to create many WebLogic Server domains.

The steps needed to create a domain are presented, which extends the installation process begun in Chapter 2. Advanced domain configuration options and the default network channel as well as the administration channel are explained.

Chapter 4—Configuring Naming, Connections, Transactions, and Messaging

The goal of this chapter is to explain how WebLogic Server provides services needed by enterprise applications. The services covered are naming, database connectivity, transactions, and messaging. Each of these services is supported by a different part of Java, and each is covered. JNDI, which is Java Naming and Directory Interface, provides a way to connect to naming and directory services such as DNS and LDAP. Java applications use JNDI to find resources by name. JDBC, of course, is used for database connectivity. JTA, Java Transaction API, is used to control transactions. The Java Messaging Services, JMS, allow Java applications to create, send, and receive messages.

How each of these services works, as well as how to configure each, is covered in detail. I found the section on JDBC to be very valuable. It explains data sources, which WebLogic Server uses to set up pools of database connections. Creating and configuring data sources is discussed. The JMS section is very thorough.

Chapter 5—Configuring the WebLogic Server Environment

Here we learn about thread pools, which WebLogic Server uses to service requests. The management of thread pools is how WebLogic Server manages its run-time performance. Each request is assigned to an execute thread. WebLogic Server automatically adjusts the number of threads available as the workload varies. Work managers are covered in detail. They allow you to set up different priorities for different requests.

When there are too many requests, WebLogic Server can become overloaded. The options for dealing with overload and failure conditions are covered. Especially interesting to me was the coverage of dealing with stuck threads. This is often the cause of managed servers going to FAILED_NOT_RESTARTABLE state.

The steps needed to set up WebLogic Server self-health monitoring are presented, followed by a more detailed look at how to optimize the network configuration. This involves creating custom network channels.

Chapter 6—Monitoring and Troubleshooting WebLogic Server

WebLogic Server provides many ways to monitor the server instances. Overall, these monitoring features are part of the WebLogic Diagnostic Framework (WLDF). The monitoring dashboard is part of the Administration Console. Examples are shown for generating a diagnostic image capture, which provides a snapshot of what is happening inside the server and other components like JDBC and JMS. These images can be saved in a diagnostic archive.

We have learned in earlier chapters that MBeans are used in Java to store server configuration. The information in MBeans can be gathered by configuring metric collection. The Harvester component of WLDF is then used to collect the specified metrics on the schedule you set up.

WLDF also provides Instrumentation, which means code that can be inserted into server instances and deployed applications to generate extra instrumentation, which is useful when diagnosing server and application issues. The Data Accessor component of WLDF is provided to retrieve the diagnostic information that has been captured. The navigation needed to access the monitoring screens within the Administration Console is shown, and examples of WLST script for monitoring are provided.

WebLogic Server provides logs for both server instance runtime information and application events. The different logs for the domain; each managed server; and the various subsystems, such as JDBC and JTA, are described. The structure of the logs,

how to view them, and how to maintain them are all explained.

This chapter concludes with a discussion of WebLogic Server troubleshooting. Java thread dumps are covered as well as the core dumps generated when a JVM crashes.

Chapter 7—Working with WebLogic Server Clusters

WebLogic Server clusters provide scalability and reliability by providing load-balancing and failover capabilities. This chapter covers how to set up and manage WebLogic Server clusters. WebLogic Server clusters always belong to only one WebLogic Server domain, and the Administration Server for the domain does not run on any of the cluster nodes. This means that for a two-node cluster, for example, you need three nodes all in the same domain. Two of these nodes are the WebLogic Server cluster where applications are deployed, and the third node is where the Administration Server runs.

Three cluster architectures are discussed: basic, multitier, and proxy. The main difference among these is where the web tier, presentation tier, and object tier (the business logic) are running in the cluster.

You can configure a cluster using a WebLogic Configuration Wizard, the Administration Console, or WLST commands. This process involves creating the managed servers that will become the cluster members, creating the cluster itself, and then adding nodes to the cluster.

The options for starting and stopping are covered as well as how to monitor a cluster from within the Administration Console. WebLogic Server offers various algorithms for load-balancing servlets, JSPs, EJBs, and RMI objects on the cluster.

When a server failure occurs, the cluster can migrate an entire server or specific services to another server in the cluster, depending on the configuration chosen.

Chapter 8—Understanding WebLogic Server Application Deployment

Finally, the whole point of using WebLogic Server comes into focus. We set up WebLogic Server to deploy enterprise applications, and now we learn how deployments are done. The various types of applications that can be deployed in WebLogic Server are discussed. Applications are deployed to targets such as managed servers and clusters. The tools supplied with WebLogic Server for deploying applications are described.

WebLogic Server has been described earlier as a container for Java applications. When a Java application is deployed to a managed server, that sever needs to know the environment and product-specific configuration details. These are handled by specifying deployment descriptors and annotations. Deployment plans support moving an application from one server to another. A common example is moving an application from test to production. The deployment descriptors are exported from the test server to a deployment plan. This is then edited for the production environment.

Applications are packaged into an archived file or an exploded archive directory. The details of both and the reasons to use one or the other are covered. Using the Administration Console to deploy an application is presented, and many screenshots help explain the process. Examples of deployment using WLST scripts are also shown.

Chapter 9—Managing WebLogic Server Security

This chapter covers many topics relating to WebLogic security. We begin with an overview of how WebLogic Server security relates to Java EE security. WebLogic Server uses Oracle Platform Security Services, which was designed to secure Fusion Middleware. OPSS provides many security services, including authentication, SSO, authorization, security providers, and security stores.

WebLogic Server security focuses on securing server resources. These include the Administration Console, WLST, and the various server logs. The logical grouping of the various security resources is called a "security realm." The configuration of all security providers and users, etc., is contained in the security realm, and all of this is stored in an LDAP server or an RDBMS.

Security providers handle specific security functions, such as authentication, identity assertion, and role mapping. The default security realm (myrealm) is created when you create a domain and contains all the required providers. Configuration options for providers are discussed, and screenshots from the Administration Console are included.

Users, groups, roles, and security policies are defined, and the process to create and configure each is covered. The configuration of the embedded LDAP server is explained next, as well how to migrate to a RDBMS security store. This chapter finishes with SSL configuration and overall Best Practices for WebLogic Server security.

Chapter 10—WebLogic Server Performance Tuning

The author tells us that performance tuning covers a broad area, including OS, JVM, server instances, databases, transactions, and more. This means that this chapter can only touch on some highlights of this subject. For each component, the critical issues for performance tuning are identified.

For the WebLogic Server, thread management and network I/O are the hot issues. For WebLogic Server, configuring Work Managers can help, but too much of this can actually hurt. For network I/O, tuning muxers, which read incoming requests on a server, using network channels, and tuning various network layer parameters can improve performance.

Tuning the JVM focuses on memory management. The section explaining how the JVM uses memory was great. I didn't know how this worked. The issues around garbage collection are interesting, and I learned a lot reading about the configuration options available. Overall best practices for performance tuning are presented.

Conclusion

Oracle is moving to Java EE applications, and this means WebLogic Server is replacing Oracle Application Server. Anyone that supports Oracle systems will likely run into WebLogic Server in the not-too-distant future. This book is densely packed with technical detail for those that will be WebLogic Server administrators. I learned a great deal from reading this book, and it has helped me on the job. ▲

The statements and opinions expressed here are the author's and do not necessarily represent those of Oracle Corporation.

Two Million Database Professionals Count on One Solution.

Simply the best for Oracle database professionals - Toad 11. Supported by over a decade of proven excellence, only Toad combines the deepest functionality available, extensive automation, and a workflow that enables database professionals of all skill and experience levels to work efficiently and accurately. Countless organizations empower their database professionals with Toad. The best just got better.

Watch the video at **www.quest.com/Toad11SimplyBetter**.

QUEST SOFTWARE®
Simplicity At Work™

Oracle Big Data Appliance

Gwen Shapira

by Gwen Shapira

This article is the second of a series by Pythian experts that will regularly be published as the "Performance Corner" column in the NoCOUG Journal.

The main software components of Oracle Big Data Appliance are Cloudera Hadoop and Oracle NoSQL. Both are non-relational databases that were designed for high scalability. But as we'll soon see, they are very different in their architecture, design goals, and use cases. The most striking thing about Cloudera Hadoop and Oracle NoSQL is that they are open source and available for download from Cloudera and Oracle websites. You can experiment with the software, develop prototypes, and explore possible architectures before you commit to purchase a new device. Of course, you can also deploy the software on your own hardware without ever purchasing a device from Oracle.

Oracle NoSQL

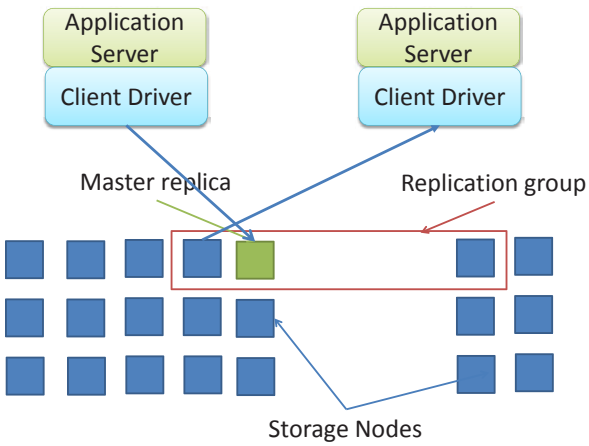
NoSQL is a recent development in the data storage architecture landscape. Popular websites such as Amazon, Google, and Facebook encountered a growing need to scale their databases across large clusters and between multiple data centers while keeping latency to only few milliseconds for 99% of the transactions.

Existing database solutions proved either too unreliable, had too high latency, were not scalable enough, or were too expensive. These organizations realized that different data sets impose different requirements, and a user's shopping history does not have the same consistency, durability, and isolation requirements that the transaction history of a bank account will require. They are willing to relax consistency in return for increased scalability, large-scale distribution, high availability, and low latency. In addition to not being fully ACID, NoSQL stores do not implement the relational model. They typically support a simpler data model such as key-value pairs, supporting data retrieval by the key but with limited support for join operations or secondary indexes.

Oracle NoSQL is a key-value store, based on Berkeley DB Java Edition. It is distributed and designed for low latency, high volume, and high availability.

As a distributed data store, Oracle NoSQL is installed on multiple servers. Those servers, typically physical devices, are referred to as "storage nodes." Each one of the storage nodes contains one or more "replication nodes," which are grouped into "replication groups." The replication groups are the way Oracle NoSQL minimizes the possibility of data being lost or unavailable as the result of a server crash. Oracle recommends that each storage node will contain only one replication node.

Each replication node in a replication group contains identical data and is placed on a separate storage node, perhaps in different data centers. In the event of a server crash, only one partition will be lost, and the data will still be accessible on other storage nodes. In each replication group, one of the replication nodes is designated the "master node," and this is the only node that can service data modification. The other nodes in the group are read-only but can become the master node if the master node fails. The number of nodes in a replication group determines how many servers can fail while the system is still available.



The data model of Oracle NoSQL is a variation of a key-value map. The key is a string, and it has "major key path" and "minor key path" components. The value can be of any data type. Records are allocated to specific data partitions according to their keys and are stored in the replication group as-

"A user's shopping history does not have the same ACID requirements that the transaction history of a bank account will require."

signed to the partition. Records with the same major key are assigned to the same partition and are therefore stored on the same physical devices. This means that all records with the same major key can be updated in a single transaction, and it also means that if there are only a small number of major keys, the load on the physical servers will not be balanced.

Oracle NoSQL allows the application developers to choose the desired level of consistency and durability for each record and each operation. This choice has a significant impact on the performance of the system and its reliability. Most NoSQL databases offer this level of flexibility, and benchmarks of those databases often show amazing performance simply because the developers reduced consistency and durability to levels that may not be acceptable in practical applications. It is always recommended to read the small print when encountering impressive benchmark results.

With Oracle NoSQL, developers control the durability of an operation with two decisions: how many nodes must acknowledge a write operation before it is considered successful and whether the new data is actually written to disk before the node acknowledges the operation.

Write operations can be acknowledged by all replication nodes in the group, a majority of the replication nodes, or none of the replication nodes. Requiring all nodes to acknowledge each write operation means that all nodes will return the same consistent information on subsequent reads, but it also means that write operations will take longer, and if a single node crashes, all write operations to the group will fail.

In the other extreme, if only the master has to acknowledge, write operations will continue to happen even if only one node is left in the group. But reads from the slave nodes may return data that is older than the data in the master node, because newly written data will not be sent immediately from the master to the slave nodes.

When a node acknowledges a write operation, it will either write the data to disk before acknowledging a successful operation (the way a redo buffer is written immediately on commit) or it can acknowledge the operation immediately and write to disk later (the way DBWR writes dirty buffers to disk)—it can send the write request to the operating system immediately but not force the OS to write the data to disk before returning control to the NoSQL process.

The other major choice that Oracle NoSQL leaves to developers is the consistency level. Developers can decide for each read operation whether they need the most recent data written to the system or whether slightly older data will do. For example, when Facebook displays a list of notifications sent to a specific user, it is fine if the list of messages is actually few minutes old and the most recent messages will show up with a small delay. When you check out from an online store, you do need the shopping basket to list your most recent purchases.

Application developers can choose between:

- Absolute consistency, where data is read from the master and guaranteed to be the most recent.
- No consistency, where data is served from the least-loaded slave regardless of how new it is.
- Time-based consistency, where the developer specifies how recent the data should be and the client searches for a node that will satisfy this condition.
- Version-based consistency, where the developer specifies a version number and requires the data to be of that version or newer. This is normally done to maintain consistency between multiple read-modify-write operations.

"In usual database architectures, data is brought from the SAN to the processors. Hadoop brings the processing to the data."

Note that unlike many other NoSQL databases, Oracle NoSQL does not support eventual consistency, where the server stores multiple conflicting versions of the data and returns all versions to the client during read operations, and the client resolves the conflict and updates the database with the result.

Cloudera Hadoop

Oracle Big Data Appliance contains Cloudera's version of Apache Hadoop. Hadoop is a platform for storing and processing large amounts of unstructured data—for example, logs from web servers of online retailers. These logs contain valuable data: what each customer looked at, how long he stayed in the website, what he purchased, etc. But these logs are just text files; like Oracle's alert log, they contain repetitious data, useless messages from developers, and several different text formats. In addition, log files have no indexes; if you look for specific piece of information, you are required to read the whole file. These attributes mean that unstructured data will typically require more disk space, disk bandwidth, and processing resources than equivalent data loaded into a relational database.

Hadoop is an architecture designed to use inexpensive and unreliable hardware to build a massively parallel and highly scalable data-processing cluster. It was designed so that adding servers will result in a proportional increase in load capacity and that server failure will result in performance decline but never in system failure or wrong results.

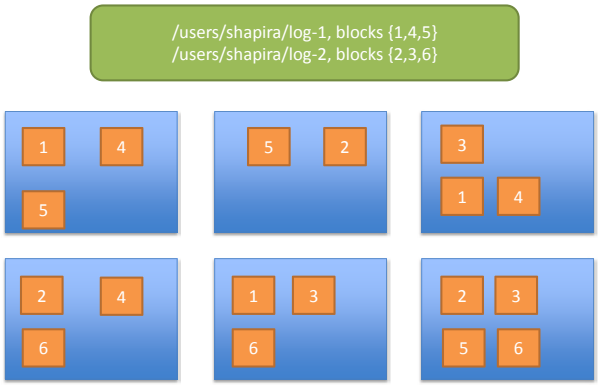
To support these design goals, the architecture is shared nothing: Nodes rarely talk to each other, so there is little overhead involved in synchronizing the processing servers, and each node uses its own local disks. Data is spread across the cluster when it is loaded, and computation usually runs on the server where the data is located. This allows spreading the load across the cluster without running into network bottlenecks. In usual database architectures, data is brought from the SAN to the processors. Hadoop brings the processing to the data.

Hadoop is made of two components: HDFS, a distributed and replicated file system, and Map-Reduce, an API that simplifies distributed data processing.

HDFS provides redundant storage for massive amounts of data. It is designed for use cases similar to those of an enter-

prise data warehouse: Data is loaded once and scanned completely by each processing job. File sizes are typically very large, and to reflect that, Hadoop's default block size is 64MB (compare with Oracle's 8KB!). Sustained high throughput is given priority over low latency, and there is no random access to files in HDFS. In addition, the files are read only: They are created; data is loaded into them; and when loading is finished, the file is closed and no additional changes can be made to the file.

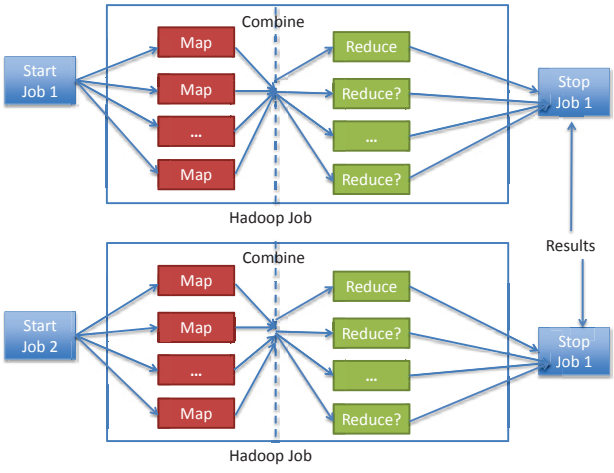
Similar to Oracle NoSQL, HDFS also improves reliability by copying each block to at least three different servers in the cluster. The replication doesn't just provide failover in case a node crashes; it also allows multiple jobs to process the same data in parallel on different servers. (**http://hadoop.apache.org/common/docs/current/hdfs_design.html**)



Map-reduce is a method to distribute processing jobs across the servers. Jobs are split into small, mostly independent tasks. Each task is responsible for processing data in one block, and whenever possible it will run on a server that stores that block locally.

As the name suggests, map-reduce has two phases: map and reduce. Map tasks filter and modify the data. This is analogous to the "where" portion of a query and to non-aggregating functions applied to the data. The reduce phase applies the data aggregation: group by and aggregating functions such as sum and average.

Since map-reduce jobs are limited to filtering and aggregating, most complex analytical queries do not translate well to map-reduce and are therefore difficult to implement in Hadoop.



Hadoop is a far more basic system than a relational or even a NoSQL database. It provides services similar to the operating system while leaving the vast majority of the work to developers. As with any platform, you will not be surprised to discover than software was written to run on top of Hadoop and provide better tools for data loading and processing.

> ***"The idea is to spread the I/O and processing load among many cheap machines instead of investing in a few expensive servers."***

Notable examples are as follows:

- **Pig and Hive:** Both are query languages. Instead of writing map-reduce jobs in Java from scratch, Pig and Hive are high-level languages that make this querying far easier and even accessible to non-programmers. Hive is very similar to SQL and even requires schema definitions for the data in HDFS files. Pig looks far more like explain plans, giving developers more control over the way data is accessed and processed.
- **Sqoop:** Connects to relational databases such as MySQL and Oracle and allows transferring data between the database and Hadoop.
- **Flume:** Aggregates log files from multiple sources and loads them into Hadoop.
- **Oracle Loader for Hadoop:** Allows users to build map-reduce jobs that load data into Oracle. Essentially the last step in the reduce process, it generates Oracle data blocks that are loaded directly into the database. It is the fastest way to load Hadoop data into Oracle.

There are many more tools designed to make life with Hadoop easier. Hadoop developers and administrators are encouraged to search for them, as there is no need to reinvent the wheel.

Oracle NoSQL—Design Considerations

A mobile application like Draw Something™ is a classic use case for a NoSQL database. The use case is very simple:

"*Two players alternate turns between drawing a picture for the other to guess. The person drawing chooses one of three guess words to draw. After the drawer has finished drawing, the guesser will view an instant replay of the drawing, minus hesitation and delays. The guesser is given a number of blank spaces and scrambled letters to type the guess word.*" (**http://en.wikipedia.org/wiki/Draw_something**)

This game is easy to model with key-value pairs. (Please note that I am describing my idea of how I'd implement a similar application, and all of the numbers are based on my imagination. They do not represent Draw Something's actual data or architecture. It is highly unlikely that Draw Something actually uses Oracle NoSQL.)

We'll note that each pair of players is allowed to have only one drawing between them at any given time: Either I'm send-

ing you a drawing or you're sending me a drawing. I am not allowed to send you a second drawing while you are guessing.

Because there is one and only one drawing for each two players, the key can be the name pairing: name1-name2 for example. We can create name2-name1 as a dummy key at the same time to avoid duplicates, or we can always sort the names alphabetically. The value will be the drawing, which sounds like a small video. We'll also want to store a bit that says whose turn is it now.

> ***"The big question is, do we want to buy Oracle Big Data Appliance, or just run the software on our own cluster?"***

Lets guess that Draw Something has about 100M users; if each has 10 friends on average, we are looking at 1 billion keys, each at a size of 100 bytes. Let's say each value takes 20KB and we are looking at 20TB of data. To be safe, we'll want each record replicated three times. Why? A replication factor of 3 is recommended by Oracle (**http://docs.oracle.com/cd/NOSQL/html/AdminGuide/store-config.html#rep-factor**) and typically used by NoSQL stores. If you spread your servers across multiple data centers, you will want to consider a larger replication factor to allow local reads from each data center. With the replication factor, we are looking at around 60TB of data.

How would we configure our NoSQL database for this data? Let's assume each of our servers has 2TB of storage available for data. We will be looking at 30 nodes to satisfy our storage requirements.

Now let's look at the workload. Most of the operations will be updates—replacing an existing image with a new one; a few create operations from users who have new friends; and there are almost no deletes. We expect exactly one read for every write: I draw something and you look at it and draw back. With this read-write mix, we'll want more replication groups, since only one node in the group can service writes, and a lower replication factor, since we don't need many slave nodes to handle a large read load.

With 30 storage nodes, we'll define 10 replication groups of three replication nodes each. More replication groups will allow higher write throughput but will cause the nodes to become unbalanced. For example, if we went with 30 replication groups to make sure we have a master node for each storage node, we will end up with three replication nodes on each storage node. In the current version of Oracle NoSQL, there is no way to make sure all master nodes end up on the separate storage nodes and prevent a single node from potentially becoming a bottleneck. To be on the safe side, we will stay with a balanced configuration of one replication node per storage node.

Each replication group requires at least one data partition. However, it is recommended to have many more, since future versions of Oracle NoSQL will allow adding replication groups and nodes, and the data will be balanced between the groups by moving partitions between the nodes. Too few partitions and there will be no room for growth, or the nodes will become unbalanced. While there is some overhead involved in a large number of partitions, we still recommend a very large number of partitions to avoid the risk of running into this limit: let's say, 30,000 partitions for our example.

At this point we have a topology for our Oracle NoSQL cluster, and we are ready to install. It should go without saying that this configuration should be well tested before it goes live—especially for an unbalanced load that can cause a node to become a bottleneck as the demands from the database increase. At this release of Oracle NoSQL, once the cluster is defined, nodes cannot be added, so plan on enough space to allow for a year of growth.

Once the cluster is installed, we need to define the size of the memory. The main data structure of Oracle NoSQL is a b-tree, and the database uses an internal cache called "JE cache" to store the blocks in this structure. With 1TB of data per node, there is no way we can fit all our data into memory, but we can improve performance significantly if we can fit most of the internal blocks of the b-tree into memory. In addition to the JE cache, Oracle NoSQL also uses the file system (FS) cache. FS cache can be used more effectively than JE cache, since records in FS cache don't have Java object overhead.

The Oracle NoSQL administration guide gives the following formula on how to size disk I/O based on the expected cache hit ratios and required number of transactions per second: **http://docs.oracle.com/cd/NOSQL/html/AdminGuide/select-cache-strategy.html#cache-size-advice**

```
((read + create + update + delete)ops/sec * (1 - cache hit fraction)) /
Number of replication nodes) = required disk IOPs/sec
```

In our system, let's assume 100,000 transactions per second and a 50% cache hit ratio:

```
(100,000*0.5)/30 = 1666.67 IOPs/sec = around 10 disks.
```

So either 10 disks per server are required or a larger number of storage nodes and replication groups.

Oracle NoSQL arrives with the DBCacheSize utility, allowing you to estimate the cache size per storage node, and the Oracle NoSQL Administrator Guide has a spreadsheet to help calculate the Java heap size.

To get an idea of the IOPs and latencies that are supported by Oracle NoSQL, I suggest taking a look at Oracle's white paper. (**https://blogs.oracle.com/charlesLamb/entry/oracle_nosql_database_performance_tests**)

On a relatively small 12-node cluster, an insert throughput of 100,000 operations per second was achieved with a 95% latency of 7ms. This performance is also achievable on Oracle database, but it will require a very fast, well-tuned storage system.

Hadoop Design Considerations

The classic use case for Hadoop is processing web server logs to gain insight about website visitors and customer activities. Another favorite use case is analyzing other text files such as blog posts, Twitter streams, and job posts to gain insights on trendy topics, customer complaints, and the job market. As an Oracle consultant, I typically see Hadoop used to run ETL processes: Data is extracted from the OLTP database, processed, aggregated, and sometimes mixed with results from the processing of unstructured data. The results are

loaded into the enterprise data warehouse, typically running on Oracle database, where the business analysts can use their BI tools to process the data.

As an example, we'll take a very simple use case where we go through website log files and, based on IPs, determine how many users from each state made a purchase at our online store.

The map stage is simple: we go through the website logs, select the log lines that indicate a successful purchase, match the IP address in the line with a U.S. state, and if there is a match, write the state to the output file. Each reduce task will receive a list of occurrences of a specific state and will only have to count how many times the state appears in the list.

To maximize performance, we'll want to make sure there is sufficient processing and disk bandwidth for the map and reduce tasks, and enough network bandwidth to send the data from mappers to reducers and for replication between nodes.

Hadoop clusters are usually sized by their storage requirements, which are typically high. Suppose our website generates 100GB of log files per day. With a replication factor of 3, we have 300GB every day and around 6TB each month. This means that to satisfy the storage requirements of the next year, we'll need around 20 servers with 2TB storage in each.

The processing servers will require either one 2TB disk or two 1TB disks. In any case, do not use RAID—since Hadoop handles replication, RAID is neither required nor recommended. A ratio of 1HD per 2 cores per 6–8GB RAM is considered a good fit for most Hadoop applications, which tend to be I/O bounded. If the workload is particularly heavy on processing, more cores will be required. The idea is to spread the I/O and processing load among many cheap machines instead of investing in few expensive servers. We typically assume that each map or reduce job will require 1.5 cores and 1–2GB RAM. Like any database server, Hadoop should never swap.

In addition to disk requirements, Hadoop can consume vast quantities of bandwidth. For each TB loaded into HDFS, 3TB will be sent to different Hadoop nodes for replication. During processing, the map tasks send the output to the reducers for processing over the network, if we are processing 1TB data and not filtering, that's an additional 1TB of data sent over the network. Of course, the results of the reduce phase are written to HDFS too and are also replicated three times over the network. Nodes should be connected at 1Gb/s at least, and if your reduce jobs generate large amounts of output, a faster network is recommended.

Each reduce tasks only analyzes a specific portion of the data. To aggregate IPs by state, 50 reduce jobs are necessary (one for each state). The data is sorted and partitioned between the map and reduce job, so each reduce task can look at its own part of the sorted data. However, it is very likely that the reduce task for California will need to process more data than the task for Montana. Data skew is known to cause difficult-to-solve performance problems in Oracle Database, and it is still a problem with Hadoop. Designing jobs to avoid this problem, aggregating by hash keys whenever possible, is a big part of the job of Hadoop developers. As administrator of a Hadoop system, the best you can do is use Hadoop's fair-share scheduler rather than the default scheduler. The fair-share scheduler will make sure that smaller tasks will not have to wait until the larger tasks finish processing but will get access to processors.

Oracle Big Data Appliance—Hardware

Now that we have some idea of the hardware and configuration requirements from our NoSQL and Hadoop use cases, the big question is, do we want to buy Oracle Big Data Appliance, or just run the software on our own cluster?

The Big Data Appliance (BDA) has 18 Sun x4270 M2 servers per rack. Each node has 48GB RAM, 12 (Intel Xeon 5675) cores, and 12 disks of 3TB each. Notably, there are no SSD and 36TB storage per node is far above what we planned for.

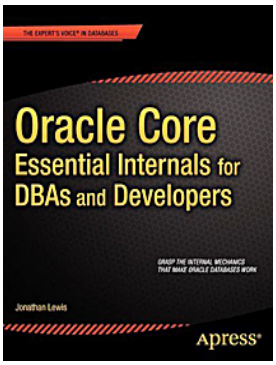
For our NoSQL applications, we need to re-plan our cluster. Our 60TB disk space requirement can now be satisfied from just two servers, but our IOP requirements will still demand 30. Additional appliances can be purchased and connected to grow the cluster, but perhaps a smarter move will be to purchase the memory upgrade, get the servers with 144GB RAM, and reduce the latency by having a better cache hit ratio rather than more disks.

> ***"If Oracle Big Data Appliance matches your hardware requirements, it is not a bad way to get the entire cluster pre-configured."***

For our Hadoop cluster, we will notice that we get 1 HD and at least 4GB RAM per core. This gives more memory and processing per disk space that most Hadoop workloads would require. To maximize the utilization on a machine, the memory can be expended to 144GB RAM, and memory-hungry Oracle NoSQL can be co-located with disk-hungry Hadoop. As far as I know, this configuration was not tested by Oracle, so testing will be needed to make sure it doesn't overload the servers.

The biggest benefit Oracle Big Data has to offer for Hadoop clusters is the Infiniband (IB) network. As we discussed, HDFS replication and communication between map/reduce tasks requires significant amounts of bandwidth. With Infiniband, the problem is solved. If your Hadoop use case requires loading the results into your Oracle data warehouse, and it happened to be running on Exadata, IB can be used to connect Big Data Appliance to Exadata and speed up the data-loading process.

Oracle Big Data Appliance is sold for around $500,000. Dell sells servers with six cores, 16GB RAM, and 12TB HD for around $6,000. Fifty-four of those will cost $324,000 and have more cores and the same amounts of memory and storage as Oracle's offering. Of course, if my data processing is using a lot of network capacity, we'll need to add Infiniband to the mix, which will bring the total cost up. Either way, a cluster of this size will cost close to a half-million dollars, so if Oracle Big Data Appliance matches your hardware requirements, it is not a bad way to get the entire cluster pre-configured in one big box. ▲



RAC and Ruin

An excerpt from *Oracle Core: Essential Internals for DBAs and Developers* by Jonathan Lewis

Jonathan Lewis

This is an excerpt from the book Oracle Core: Essential Internals for DBAs and Developers *published by Apress, Nov. 2011, ISBN 1430239549; Copyright 2011. For a complete table of contents, please visit the publisher site:*
http://www.apress.com/9781430239543.

Up to this point, everything I've said has been about a single instance addressing a single database; but Oracle allows multiple instances to share access to a single database, which means multiple independent data caches, multiple independent log buffers, and multiple independent SGAs—all accessing the same set of physical files, all using the same data dictionary. Without a constant stream of negotiation going on between instances, it would be impossible to ensure that the multiple instances behaved consistently, and we would see cases where data changes made by one instance were lost due to interference from another instance. This need for cross-instance traffic is the only significant new concept that we need to focus on when we think about RAC—the Real Application Cluster.

The ideas we need to consider are as follows: How can latch activity work if the thing you want to protect is in the memory of another instance? How can you modify your copy of a data block when another instance may have a copy that is a newer version? If you have an execution plan that includes a particular indexed access path, how do you become aware of the fact that another instance has dropped that index? If you are trying to create a read-consistent copy of a data block, how do you ensure that the SCNs of all instances that might have changed that block are kept in synch?

There are only two pieces of functionality that we need to address all these questions—global enqueues, and cache coherency—and these are the main topics we will examine in this chapter. This will give us a chance to revisit and refine our understanding of locks and latches while focusing on the critical changes that can cause problems with RAC.

Before looking at the mechanics, however, we will spend a little time looking at an overview of what RAC is, why you might want it, and the threats that the Oracle developers addressed when creating the product. We'll also take a look at the way the recovery mechanisms we've reviewed fold neatly into RAC to allow one instance to take over when another fails. Since most of the discussion relating to RAC is fairly abstract, I've also picked a commonly-used programming feature to make concrete points about creating an application that performs well on RAC.

Note: I felt the need for a little alliteration in the chapter title, and it is very easy to ruin things if you don't understand a little bit about how RAC works; but RAC doesn't necessarily lead to ruin.

The Big Picture

To my mind, understanding how RAC works when it's up and running isn't the really difficult bit—it's getting it installed and running in the first place that's complicated. There are many layers that have to be configured correctly before the installation is complete. No matter what Oracle does to make things easier, there is an inherent volume of complexity involved in making sure that all the individual pieces are working correctly and cooperating; and each time you do it or, at least, each time I do it, there are some new bits to worry about.

Luckily, I don't feel compelled to tell you how to install RAC because there are several blog items (such as the series by Tim Hall at **www.oracle-base.com**), massive installation notes on the Oracle Technet (OTN), and entire books (such as Pro Oracle Database 11*g* RAC on Linux by Martin Bach and Steve Shaw (Apress, 2010)) written to explain it. I'm only going to give you the highlights of the run-time mechanics that show you why you need to be careful in using it.

I will, however, give you a picture of what RAC means (see Figure 8-1).

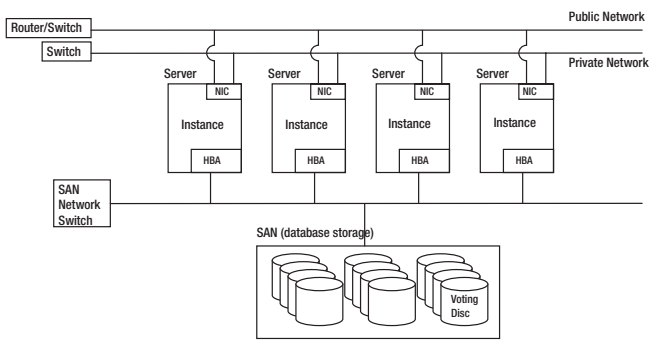


Figure 8-1. Schematic of a RAC system

The following are key points to pick up from this schematic:

- Each machine (generally referred to as a node) that runs an instance must be attached to two networks—one is a public network used to accept connections from end-user programs, the other is a private network used only by the instances to talk to each other and maintain a coherent picture of the combined SGAs. The private network needs to have the lowest latency that you can afford to buy.
- All instances must be able to access all the storage.
- There is a special disc that acts as a control mechanism in the event of a breakdown in communications between the machines. (If you want to get very sophisticated with RAC—especially if you decide to stretch your RAC across long distances—you will need a minimum of three such disks, ideally in three physically separate locations).

The following are things that cannot be seen in the diagram:

- There are couple of layers of software (the cluster services) running at a level between the operating system and the Oracle instance that make sure that the machines can communicate with each other.
- There is another network involved that is invisible because it's a "virtual network," driven by Oracle's cluster software, sitting on the public network. It's actually the virtual network that client programs use to communicate with the Oracle instance. This arrangement makes it possible for Oracle to inform the client code very quickly that one of the database server machines has failed, so that the client can be redirected to a surviving machine with a minimum time-lag (see sidebar).
- If you are using Oracle Corporation's preferred installation approach there will be two instances running on each machine, the standard "database" instance, identified by parameter instance_type as being an instance of type RDBMS, and the "storage management" instance, identified by parameter instance_type as being an instance of type ASM. (If you want to run instances for several different databases on the each machine you would only need one ASM instance per machine—effectively ASM is functioning rather like a logical volume manager, as a layer between the Operating System and the normal database instances.)

Virtual IP Addresses and SCAN

When you configure a RAC system, one of the tasks you have to do at the operating system level is assign each machine a "virtual" IP address (VIP). Immediately after startup the cluster service software inserts this code layer above the real IP address, and the machines switch to communicating through the VIPs.

The big difference between real and virtual IP addresses is that the real IP address is logically bound to a specific piece of hardware (technically the MAC address on the network card); whereas the VIP address is controlled by software and its association with a particular piece of hardware can be changed dynamically.

If a database machine were to fail when a client program was attached to it through the real IP, the client program would wait a long time (tens to hundreds of seconds) for a response from the server before deciding that the server machine had failed. Because Oracle is using virtual IPs, when a machine fails, one of the other instances will detect the failure very rapidly and take on the virtual IP of the failed machine to handle current requests. This reconfigures the system to stop future connections going to a failed VIP (until the machine and instance are live again). This means that clients will not experience a long wait if the instance they are addressing fails.

As another aid to minimize waits caused by network problems, Oracle 11.2 has introduced the SCAN (Single Client Access Name) as part of its Grid Infrastructure. Before you can use SCAN, you have to get your system administrator to set up some new DNS (Domain Name Service) entries, or assist you in setting up a subsection of the DNS to use Oracle's GNS (Grid Naming Service). Once this has been set up, any client programs will be able to reference the system by SCAN and you won't have to reconfigure any client settings if you need to move the system to different hardware, or add or remove nodes.

Staying Alive

Looking at Figure 8-1, you might start to wonder what happens if the network fails and the different instances can't talk to each other. If the network breaks, you could end up with two sets of instances that are convinced that they have exclusive access to the database—with the result that they keep overwriting each others' changes.

There are two layers at which this threat could occur: an Oracle instance may seem to lose contact with the rest of the instances, or a machine may lose contact with the rest of the machines. We'll work from the bottom of the stack upwards and worry about the machine level first.

The Machine Defenses

The cluster services on each machine can keep in touch with each other through the network, but they also keep in touch through the voting disc shown in Figure 8-1, and this leads to several options for dealing with communication problems.

Every second, every machine writes to the "voting" disc—there is a file on the disc holding one block per machine, and each machine simply increments a counter in its block. In the simplest case, a machine may discover that it can't write to the file—in which case it takes itself out of the cluster and may reboot itself to protect the cluster. (At this point, of course, the instance will also terminate abruptly, and one of the other instances will have to go through crash recovery for it.)

As each machine writes its block, however, it also checks the blocks written by every other machine in the cluster, so it can detect any cases of a machine that hasn't updated its own block in the recent past. (It's possible that a machine in this state could have cluster software that is misbehaving but still be running, and have a live instance on it—and that poses a threat to the database.) In this case, the discovering machine is allowed to terminate the guilty machine with extreme prejudice.

It's possible that every machine can write to the file, but suffer from a network problem that stops some machines from hearing each other across the network. In this case one machine will attempt to ring-fence the voting disc and start counting how many other machines it can talk to. If it finds that it belongs to a networked group that contains more than half the machines or, if the total number of machines is even, half the machines including the machine with the lowest known cluster id—it will force the remaining machines out of the cluster and make them reboot. If it can't get the necessary quorum (hence the alternative name of "quorum disk" for the voting disk), it will take itself out of the cluster.

Inevitably things are more complex than the outline I've given, but you can appreciate that the cluster software has to be quite clever (and chooses to be very pessimistic) when it comes to ensuring that a machine doesn't jeopardize the health of the database by running with links to the rest of the cluster that are suspect.

The Oracle Defenses

There is a similar type of strategy at the Oracle level. As each instance starts up it broadcasts its presence to the other instances, so every instance always knows how many other instances should be alive. Similarly, when an instance shuts down (properly) it says goodbye. There is a brief period during both startup and shutdown when the latest state of the instance group is passed around and instances "rebalance" themselves to cater for the change in number. From this point onward, every instance is in constant communication over the network with every other instance (with LMON in particular acting as the network heartbeat process).

There's also a file-based heartbeat going on; as we saw in Chapter 6, the checkpoint process (CKPT) updates the control file roughly every three seconds. In RAC every instance has its own CKPT process, so the control file can act as the heartbeat file for Oracle. If you dump the control file at level 3 (see Appendix) you will see the Checkpoint Progress Records—that's the critical section of the control file that contains one record for each instance that the instance has to keep up to date.

In principle, therefore, it's possible for an instance to detect that it can no longer write to the control file, and terminate itself for the good of the system. It's also possible, in principle, for another instance to notice that an instance has hung (and, as we shall see shortly, if one instance hangs then eventually every other instance is liable to end up waiting for it to do something). We might hope, therefore, that Oracle could include code to allow one instance (if it's in the quorum, of course) to take down another instance. Unfortunately, I don't think that this can happen until 11.2, where yet another new process called LMHB (lock manager heart beat monitor) has been introduced to determine very rapidly that some lock processes are getting hung up waiting for lock requests to be serviced.

So we have a way of putting together several pieces of equipment to create something that behaves like a single, large system; and we have ways to protect the entire system from the problems caused by its parts. But it's complicated, and why would you want to deal with something complicated?

What's the Point?

Whenever you look at any technology, or any feature within a technology, it's worth asking why you might want to use it—what's the benefit? If you can't describe what you hope to achieve, then you can't decide how best to use the technology and how to measure your degree of success.

There are two arguments put forward for RAC—scalability and high availability, and the scalability argument comes in two flavors: throughput and response time. So, before you start chasing RAC, you need to decide which of these two arguments matters to you, and how best to implement RAC on your site to ensure that you turn the theory into reality.

High Availability

The theory of high availability says that if one of the nodes in your RAC system fails, it will only be moments before one of the other nodes recovers the redo from the failed node, the nodes rebalance, and everything then keeps on running. The failover should be virtually transparent to the front-end (although transparency isn't something that applies to transactions that are in flight unless you recode the application). There are four obvious questions to ask in response to this argument, as follows:

- How important is it to restart, say, within 10 seconds? Can we live with an alternative that might be just a little slower?
- How often does a machine fail compared to a disk or a network? (Assuming that we're not going to cover every possible single point of failure (SPOF)). How much benefit do we get from RAC?
- Do we have the human resources to deal with the added complexity of the RAC software stack and the level of skill that is still needed to handle patches and upgrades?
- How many nodes do we need to run before the workload for N nodes can run on N-1 nodes without having a performance impact that would make us uncomfortable?

Alternative technologies include things like basic cluster failover as supplied by the operating system vendor, or a simple physical standby database if we want to stick to pure Oracle technology—options where another machine can be available to take on the role of the database server. The drawbacks are that the time to failover will be longer and the other machine is an idle machine, or used for something else but with spare capacity, or it's going to give you degraded response time while it's acting as the failover machine. It's quite easy to set up the database server to fail over fairly quickly—but don't forget that every little piece of your application has to find the new machine as well.

Looking at SPOFs—how many options are you going to cover? Doubling up network cards is easy, the same goes for switches, but doubling up the storage and the actual network cabling is harder—and what's the strategy for a power outage in the data centre? You might also look at the complexity of the whole RAC stack and ask how much time you're going to lose

(continued on page 26)

Don't Throw Out the Baby with the Bathwater

by Iggy Fernandez

Iggy Fernandez

I have on good authority from a NoSQL aficionado that *"the only requirement from NoSQL is to be nonrelational, and therefore without joins."* Obviously if relational joins are computationally expensive and NoSQL is a viable alternative, then NoSQL wins. But are relational joins expensive *by definition*?

Suppose that we have the following tables; the example is from the 1970 paper *A Relational Model of Data for Large Shared Data Banks* by the founder of the relational movement, Dr. Edgar (Ted) Codd, which was reprinted in the 100th issue of the *NoCOUG Journal*.

```
CREATE TABLE employees (
  employee# INTEGER NOT NULL,
  name VARCHAR2(16),
  birth_date DATE
);

CREATE TABLE job_history (
  employee# INTEGER NOT NULL,
  job_date DATE NOT NULL,
  title VARCHAR2(16)
);

CREATE TABLE salary_history (
  employee# INTEGER NOT NULL,
  job_date DATE NOT NULL,
  salary_date DATE NOT NULL,
  salary NUMBER
);

CREATE TABLE children (
  employee# INTEGER NOT NULL,
  child_name VARCHAR2(16) NOT NULL,
  birth_date DATE
);
```

Suppose that we have the following data: Employee 1 (Ignatius) has two children (Iniga and Inigo) and has held two jobs (Programmer and Database Admin). The salary history for each job is provided.

```
INSERT INTO employees
  VALUES (1, 'IGNATIUS', '01-JAN-1970');
INSERT INTO children

  VALUES (1, 'INIGA', '01-JAN-2001');
INSERT INTO children
  VALUES (1, 'INIGO', '01-JAN-2001');

INSERT INTO job_history
  VALUES (1, '01-JAN-1991', 'PROGRAMMER');
INSERT INTO job_history
  VALUES (1, '01-JAN-1992', 'DATABASE ADMIN');

INSERT INTO salary_history
  VALUES (1, '01-JAN-1991', '1-FEB-1991', 1000);
INSERT INTO salary_history
  VALUES (1, '01-JAN-1991', '1-MAR-1991', 1000);
INSERT INTO salary_history
  VALUES (1, '01-JAN-1992', '1-FEB-1992', 2000);
INSERT INTO salary_history
  VALUES (1, '01-JAN-1992', '1-MAR-1992', 2000);
```

Suppose that we need to create a "shopping cart" with all the information about employee 1. Here is an XML version of the shopping cart.

```
<?xml version="1.0" encoding="US-ASCII"?>
<employee>
  <name>IGNATIUS</name>
  <birth_date>1970-01-01</birth_date>
  <children>
    <child>
      <child_name>INIGA</child_name>
      <birth_date>2001-01-01</birth_date>
    </child>
    <child>
      <child_name>INIGO</child_name>
      <birth_date>2001-01-01</birth_date>
    </child>
  </children>
  <job_history>
    <job>
      <job_date>1991-01-01</job_date>
      <title>PROGRAMMER</title>
      <salary_history>
        <salary>
          <salary_date>1991-02-01</salary_date>
          <salary>1000</salary>
        </salary>
        <salary>
          <salary_date>1991-03-01</salary_date>
          <salary>1000</salary>
        </salary>
      </salary_history>
    </job>
    <job>
      <job_date>1992-01-01</job_date>
      <title>DATABASE ADMIN</title>
      <salary_history>
```

```
      <salary>
        <salary_date>1992-02-01</salary_date>
        <salary>2000</salary>
      </salary>
      <salary>
        <salary_date>1992-03-01</salary_date>
        <salary>2000</salary>
      </salary>
    </salary_history>
  </job>
 </job_history>
</employee>
```

The above XML document was produced by the following query. Tim Hall has an excellent tutorial on SQL/XML functions on his website (**http://www.oracle-base.com**).

```
SELECT
  XMLROOT (
    XMLELEMENT ("employee",
      XMLFOREST (
        e.name AS "name",
        e.birth_date AS "birth_date",
        (
          SELECT
            XMLAGG (
              XMLELEMENT ("child",
                XMLFOREST (
                  c.child_name AS "child_name",
                  c.birth_date AS "birth_date"
                )
              )
            )
          FROM children c
          WHERE c.employee# = e.employee#
        ) AS "children",
        (
          SELECT
            XMLAGG (
              XMLELEMENT ("job",
                XMLFOREST (
                  j.job_date AS "job_date",
                  j.title AS "title",
                  (
                    SELECT
                      XMLAGG (
                        XMLELEMENT ("salary",
                          XMLFOREST (
                            s.salary_date AS "salary_date",
                            s.salary AS "salary"
                          )
                        )
                      )
                    FROM salary_history s
                    WHERE s.employee# = j.employee#
                    AND s.job_date = j.job_date
                  ) AS "salary_history"
                )
              )
            )
          FROM job_history j
          WHERE j.employee# = e.employee#
        ) AS "job_history"
      )
    )
  ) AS details
FROM employees e
WHERE employee# = 1;
```

Eight separate physical storage containers are (obviously) required in the above example—one for each of the four tables and one for each of the four indexes that are (obviously) necessary for efficient retrieval of a single employee's information. Therefore, a minimum of eight data blocks will have to be retrieved in order to construct the shopping cart—many more if the tables contain real volumes of data. If it takes 5 milliseconds on average to retrieve a block from disk, the above query may need a minimum of 40 milliseconds, which may be unacceptable in some demanding scenarios. The natural temptation therefore is to abandon the relational model as Amazon did in the case of Dynamo, the NoSQL product that started it all.

However, the conclusion that eight storage containers are needed for our example is a misinterpretation of Dr. Codd's vision. Dr. Codd did *not* dictate how data should be stored.

- He did *not* dictate that each stored table should occupy one physical file.
- He did *not* dictate that data should be stored in row-major order.
- He did *not* dictate that stored tables should have only one storage representation each.
- He did *not* dictate that data should be stored in normalized form only.
- He did *not* dictate that a single data block should only contain data from a single table.
- He did *not* dictate that data cannot be stored in compact forms (such as shopping carts.)

In other words, shopping carts would be an acceptable storage representation for stored tables. It was *not* necessary for Amazon to abandon the relational model when creating Dynamo. Amazon mistook implementation deficiencies (deficiencies of existing implementations) for intrinsic deficiencies and threw out the baby with the bathwater.

P.S. Oracle Database has always allowed us to store data from multiple tables in a form that closely resembles a shopping cart. Each block in a multi-table cluster can store data from multiple tables. Every single Oracle Database on the planet contains *multi-table clusters*, because they are used internally by Oracle Database to turbocharge the performance of the data dictionary. On page 379 of *Effective Oracle by Design*, Tom Kyte quotes Steve Adams as saying: "*If a schema has no IOTs or clusters, that is a good indication that no thought has been given to the matter of optimizing data access.*" Let me show how multi-table clusters can be used to reduce the number of blocks needed to construct a shopping cart to just one by using a *hash cluster*; all data for employee 1 will be stored in a single data block and no indexes will be necessary. First we create a "hash cluster" using the command "CREATE CLUSTER employees (employee# INTEGER) hashkeys 1000". Then we add the clause "CLUSTER employees (employee#)" to the table creation statements and insert the data as usual. We can then confirm that all the data for employee 1 is stored in the same block.

(continued on page 26)

Many Thanks to Our Sponsors

NoCOUG would like to acknowledge and thank our generous sponsors for their contributions. Without this sponsorship, it would not be possible to present regular events while offering low-cost memberships. If your company is able to offer sponsorship at any level, please contact NoCOUG's president, Iggy Fernandez, at **iggy_fernandez@hotmail.com**. ▲

Long-term event sponsorship:

CHEVRON

ORACLE CORP.

Thank you! Year 2012 Gold Vendors:

- Amazon Web Services
- Confio Software
- Database Specialists
- Delphix
- GridIron Systems
- Quest Software
- Quilogy Services

For information about our Gold Vendor Program, contact the NoCOUG vendor coordinator via email at: **vendor_coordinator@nocoug.org**.

TREASURER'S REPORT

Naren Nagtode, *Treasurer*

Beginning Balance		
January 1, 2012		**$ 49,034.64**
Revenue		
Membership Dues	20,345.00	
Meeting Fees	100.00	
Vendor Receipts	9,200.00	
Advertising Fee	–	
Training Day	–	
Conference Sponsorship	–	
Interest	3.84	
Paypal balance	–	
Total Revenue		**$ 29,648.84**
Expenses		
Regional Meeting	5,893.25	
Journal	1,510.04	
Membership	698.57	
Administration	418.56	
Website	–	
Board Meeting	391.59	
Marketing	–	
Insurance	–	
Vendor expenses	14.80	
Membership S/W subscription	50.00	
Training Day	–	
IOUG-Rep	–	
Miscellaneous	–	
Total Expenses		**$ 8,976.81**
Ending Balance		
March 31, 2012		**$ 69,706.67**

Third International

NoCOUG SQL & NoSQL Challenge

Sponsored by Pythian—*Love Your Data*™

BE IT KNOWN BY THESE PRESENTS that the Wicked Witch of the West needs your help to create a magic spell to ensure that the Third Annual Witching & Wizarding Ball is a grand success. A great tournament has been organized for practitioners of the arts of Es-Cue-El & No-Es-Cue-El to demonstrate their magic powers.

Mind-Boggling Puzzle

The Wicked Witch of the West has invited six friends to the Third Annual Witching & Wizarding Ball at Pythian Academy of Es-Cue-El & No-Es-Cue-El. Burdock Muldoon and Carlotta Pinkstone both said they would come if Albus Dumbledore came. Albus Dumbledore and Daisy Dodderidge both said they would come if Carlotta Pinkstone came. Albus Dumbledore, Burdock Muldoon, and Carlotta Pinkstone all said they would come if Elfrida Clagg came. Carlotta Pinkstone and Daisy Dodderidge both said they would come if Falco Aesalon came. Burdock Muldoon, Elfrida Clagg, and Falco Aesalon all said they would come if Carlotta Pinkstone and Daisy Dodderidge both came. Daisy Dodderidge said she would come if Albus Dumbledore and Burdock Muldoon both came.

The Wicked Witch of the West needs an Es-Cue-El or No-Es-Cue-El spell to determine whom she needs to persuade to attend the wizarding ball in order to ensure that all her invitees attend.

Awesome Prizes

The *August Order of the Wooden Pretzel* will be conferred on the winner, in keeping with the celebrated pronouncement of the supreme wizard of Pee-El/Es-Cue-El that *"some people can perform seeming miracles with straight Es-Cue-El, but the statements end up looking like pretzels created by somebody who is experimenting with hallucinogens."* As if that singular honor were not enough, a fiery wizarding device that magically displays Oracular tomes will be bestowed upon the champion.

May the best witch or wizard win!

RULES: Pythian will award a Kindle Fire or an Amazon gift card of equal value to the winner. Prizes may be awarded to runners-up at the discretion of NoCOUG. Submissions should be emailed to **SQLchallenge@nocoug.org**. Contestants may use any SQL or NoSQL technology to create the magic spell. The competition will be judged by Oracle ACE Director Gwen Shapira and the editor of the *NoCOUG Journal*, Iggy Fernandez. Judging criteria include correctness, originality, efficiency, portability, and readability. The judges' decisions are final. The competition will end at a time determined by the organizers. The judges and organizers reserve the right to publish and comment on any of the submissions, with due credit to the originators. More information about the problem and additional rules can be found at **http://www.nocoug.org/**.

Pythian
love your data

(continued from page 22)

```
SQL> SELECT DISTINCT DBMS_ROWID.ROWID_BLOCK_
NUMBER(rowid)
  2  FROM employees where employee# = 1;
                       27745

SQL> SELECT DISTINCT DBMS_ROWID.ROWID_BLOCK_
NUMBER(rowid)
  2  FROM children where employee# = 1;
                       27745

SQL> SELECT DISTINCT DBMS_ROWID.ROWID_BLOCK_
NUMBER(rowid)
  2  FROM job_history where employee# = 1;
                       27745

SQL> SELECT DISTINCT DBMS_ROWID.ROWID_BLOCK_
NUMBER(rowid)
  2  FROM salary_history where employee# = 1;
                       27745
```

Therefore, no matter how many queries are needed or how complex they are, only one block of data needs to be retrieved into memory in order to answer any and all questions about employee 1. Because we are using hash clusters, no indexes are necessary if we are retrieving information about only one employee. However we are not prevented from creating indexes as usual, so it would still be possible to answer questions efficiently that involve more than one employee, such as: "Find all employees who have held the post of database administrator." In other words, we get to eat our cake and have it too.

P.S. When I interviewed Baron Schwartz, chief performance architect of Percona, for the 101st issue of the *NoCOUG Journal*, I asked him, "What do you think of the NoSQL movement? Is it a threat to MySQL?" Here was his reply:

I think we have all learned a lot from it in the last few years, but I think that some of the strongest proponents have actually misunderstood what the real problems are. There was a lot of discussion that went something like "SQL is not scalable," when in fact that's not true at all. Current implementations of SQL databases indeed have some scalability limitations. That is an implementation problem, not a problem with the model. The baby got thrown out with the bathwater. I believe that some of the emerging products, such as Clustrix, are going to open a lot of people's eyes to what is possible with relational technology. That said, many of the approaches taken by nonrelational databases are worth careful consideration, too. In the end, though, I don't think that we need 50 or 100 alternative ways to access a database. We might need more than one, but I would like some of the non-SQL databases to standardize a bit. ▲

(continued from page 20)

on patching and upgrading. Oracle Corp. is constantly working towards "rolling upgrades"—but it still has a way to go.

Scalability

There are two different concepts that people tend to think of when thinking about scalability. These are as follows:

- Get the same job done more quickly—improved response time
- Get more copies of the same job done at the same time—improved throughput

It's quite helpful to think of the first option in terms of individual big jobs, and the second in terms of a large number of small jobs. If you have a batch process or report that takes 40 minutes to complete, then sharing it across two instances may allow it to complete in 20 minutes, sharing it across four nodes may allow it to complete in 10 minutes. This image probably carries faint echoes of parallel execution—and the association is quite good; if you hope to get a shorter completion time without rewriting the job you're probably going to have to take advantage of the extra nodes through parallel execution. If parallel execution comes to your aid, the threat is the extra cost of messaging. There are overheads in passing messages between layers of parallel execution slaves, and the overheads are even greater if the slaves are running in different instances. If you want to make big jobs faster (and can't improve the code), maybe all you need is more memory, or more CPUs or faster CPUs before you move to greater complexity.

If your aim is to allow more jobs to run in the same time—let's say you're growing a business and simply have more employees doing the same sort of thing on the system—then adding more instances allows more employees to work concurrently. If you can run 50 employees at a time on one instance, then maybe you can run 100 at a time on two, and 200 at a time on four. You simply add instances as you add employees.

In favor of this strategy is the fact that each instance has its own log writer (lgwr) and set of redo log files—and the rate at which an instance can handle redo generation is the ultimate bottleneck in an Oracle system. On the downside, if you have more processes (spread across more instances) doing the same sort of work, you are more likely to have hot spots in the data. In RAC, a hot spot means more traffic moving between instances—and that's the specific performance problem you always have to be aware of.

Again you might ask why you don't simply increase the size of a single machine as your number of users grows. In this case, there's an obvious response: it's not that easy to "grow" a machine, especially when compared to buying another "commodity" machine and hanging it off the network. Indeed, one of the marketing points for RAC was that it's easier to plan for growth—you don't have to buy a big machine on day one and have it running at very low capacity (but high cost) for a couple of years. You can start cheap and grow the cluster with the user base.

Note: *One of the unfortunate side effects of the "start big" strategy that I've seen a couple of times is that a big machine with a small user base can have so much spare capacity that it hides the worst performance issues for a long time—until the user base grows large enough to make fixing the performance issues an emergency.* ▲

Database Specialists: DBA Pro Service

DBA PRO BENEFITS

- *Cost-effective and flexible extension of your IT team*
- *Proactive database maintenance and quick resolution of problems by Oracle experts*
- *Increased database uptime*
- *Improved database performance*
- *Constant database monitoring with Database Rx*
- *Onsite and offsite flexibility*
- *Reliable support from a stable team of DBAs familiar with your databases*

CUSTOMIZABLE SERVICE PLANS FOR ORACLE SYSTEMS

Keeping your Oracle database systems highly available takes knowledge, skill, and experience. It also takes knowing that each environment is different. From large companies that need additional DBA support and specialized expertise to small companies that don't require a full-time onsite DBA, flexibility is the key. That's why Database Specialists offers a flexible service called DBA Pro. With DBA Pro, we work with you to configure a program that best suits your needs and helps you deal with any Oracle issues that arise. You receive cost-effective basic services for development systems and more comprehensive plans for production and mission-critical Oracle systems.

DBA Pro's mix and match service components

Access to experienced senior Oracle expertise when you need it
We work as an extension of your team to set up and manage your Oracle databases to maintain reliability, scalability, and peak performance. When you become a DBA Pro client, you are assigned a primary and secondary Database Specialists DBA. They'll become intimately familiar with your systems. When you need us, just call our toll-free number or send email for assistance from an experienced DBA during regular business hours. If you need a fuller range of coverage with guaranteed response times, you may choose our 24 x 7 option.

24 x 7 availability with guaranteed response time
For managing mission-critical systems, no service is more valuable than being able to call on a team of experts to solve a database problem quickly and efficiently. You may call in an emergency request for help at any time, knowing your call will be answered by a Database Specialists DBA within a guaranteed response time.

Daily review and recommendations for database care
A Database Specialists DBA will perform a daily review of activity and alerts on your Oracle database. This aids in a proactive approach to managing your database systems. After each review, you receive personalized recommendations, comments, and action items via email. This information is stored in the Database Rx Performance Portal for future reference.

Monthly review and report
Looking at trends and focusing on performance, availability, and stability are critical over time. Each month, a Database Specialists DBA will review activity and alerts on your Oracle database and prepare a comprehensive report for you.

Proactive maintenance
When you want Database Specialists to handle ongoing proactive maintenance, we can automatically access your database remotely and address issues directly — if the maintenance procedure is one you have pre-authorized us to perform. You can rest assured knowing your Oracle systems are in good hands.

Onsite and offsite flexibility
You may choose to have Database Specialists consultants work onsite so they can work closely with your own DBA staff, or you may bring us onsite only for specific projects. Or you may choose to save money on travel time and infrastructure setup by having work done remotely. With DBA Pro we provide the most appropriate service program for you.

DatabaseSpecialists

CALL 1-888-648-0500 TO DISCUSS A SERVICE PLAN

NoCOUG
P.O. Box 3282
Danville, CA 94526

NoCOUG Spring Conference Schedule

Thursday, May 31, 2012—CarrAmerica Conference Center, Pleasanton, CA

Please visit **http://www.nocoug.org** for updates and directions, and to submit your RSVP.
Cost: $50 admission fee for non-members. Members free. Includes lunch voucher.

8:00 a.m.–9:00	Registration and Continental Breakfast—Refreshments served
9:00–9:30	**Welcome:** Iggy Fernandez, NoCOUG president
9:30–10:30	**Keynote:** *Oracle Database Appliance Unplugged!*—Sohan DeMel, Oracle Corporation
10:30–11:00	**Break**
11:00–12:00	**Parallel Sessions #1**
	Auditorium: *Acquiring Big Data*—Dave Rubin, Oracle Corporation
	Tassajara: *Measuring and Forecasting Scalability and Performance from Network Traffic* —Baron Schwartz, Percona
	Diablo: *Running Oracle in the Public Cloud*—Shakil Langha, Amazon Web Services
12:00–1:00 p.m.	**Lunch**
1:00–2:00	**Parallel Sessions #2**
	Auditorium: *Modern Platform Topics for Modern Oracle Database Professionals—Part I* —Kevin Closson, EMC
	Tassajara: *AWR/ASH—Understanding an Application's Database Usage*—Jerry Brenner, Guidewire
	Diablo: *Introducing Oracle Exalytics In-Memory Machine*—Manan Goel, Oracle Corporation
2:00–2:30	**Break and Refreshments**
2:30–3:30	**Parallel Sessions #3**
	Auditorium: *Modern Platform Topics for Modern Oracle Database Professionals—Part II* —Kevin Closson, EMC
	Tassajara: *A Look at MySQL for Oracle DBAs*—Abhaid Gaffoor
	Diablo: *Information Discovery—Introducing Oracle Endeca*—Manan Goel, Oracle Corporation
3:30–4:00	**Raffle**
4:00–5:00	**Parallel Sessions #4**
	Auditorium: *Introduction to Dtrace*—Kyle Hailey, Delphix
	Tassajara: *Big Data Analysis Using Oracle R Enterprise*—Vaishnavi Sashikanth, Oracle Corporation
	Diablo: *To Be Announced*
5:00–	**NoCOUG Networking and No-Host Happy Hour**

RSVP *required* at http://www.nocoug.org

The *NoCOUG Journal* design and production: Giraffex, Inc., S.F.
Front cover photo: Pickwick Crocuses Emerging From Under Snow by Serge Vero/Photos.com