An Oracle White Paper
February 2011

Hadoop and NoSQL Technologies and the Oracle Database

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction

Innovation in the database software industry continues to be extremely vibrant. In just the last couple of years, new scalability capabilities have revolutionized the performance that database systems can deliver. These include the use of very large, multi-terabyte flash memories; compression which can dramatically increase the amount of data cached in memory and the speed of disk scans for large queries; and the movement of select database logic into storage to speed functions such as row selection, compression, encryption, and more. For Oracle, this has culminated in Exadata which is an integrated software and hardware product incorporating all of these new innovations in a scale-out architecture that effectively provides unlimited scalability for all types of applications.

The world of open source database technology has also been evolving rapidly. In this paper we will focus on two relatively new developments: Hadoop (MapReduce) and NoSQL. The common goal for both is massive scalability and support for what is called "Big Data", i.e., environments where data can range from tens or hundreds of terabytes to petabytes or more in size. The purpose of this paper is to explain the use cases for these technologies and how they relate to and coexist with the Oracle Database.

Open Source Database Technologies

Open source Database technologies have been used to build and successfully run some of the most innovative new applications that have emerged in recent years. Developers with new ideas have been able to quickly develop and then deploy these applications at a low cost with open source databases. They didn't need to spend time and energy convincing others to invest in the purchase of commercial database products or hire the expertise of professional DBAs.

Today, Oracle Corporation is the leading open source database vendor. Oracle acquired InnoDB, the leading transactional storage engine for MySQL, in 2005. Berkeley DB, a high-performance embedded database for key/value data, was acquired in 2006. MySQL, the leading open source SQL-based relational database, was acquired in 2010. These products have thrived since their acquisition and Oracle has continued to invest significantly in their development.

Many Oracle customers have substantial deployments of both the Oracle Database and open source databases. Many of the new, innovative applications that use open source databases today require a relatively narrow range of functionality when compared to more mainstream business applications for which the Oracle Database is by far the leading platform. Over time some of these applications will need to provide more complex

functionality to remain competitive in their market and will need to move to or integrate with the database platform that can best support these additional requirements. Oracle has been working to support integration between these environments where that can add value. Oracle has also been working to support low-cost and low-risk migrations from open source databases to the Oracle Database when this is the next step in the application lifecycle.

Hadoop

Hadoop is a generic processing framework designed to execute queries and other batch read operations against massive datasets that can be tens or hundreds of terabytes and even petabytes in size. The data is loaded into or appended to the Hadoop Distributed File System (HDFS). Hadoop then performs brute force scans through the data to produce results that are output into other files. It probably does not qualify as a database since it does not perform updates or any transactional processing. Hadoop also does not support such basic functions as indexing or a SQL interface, although there are additional open source projects underway to add these capabilities.

Hadoop operates on massive datasets by horizontally scaling (aka scaling out) the processing across very large numbers of servers through an approach called MapReduce. Vertical scaling (aka scaling up), i.e., running on the most powerful single server available, is both very expensive and limiting. There is no single server available today or in the foreseeable future that has the necessary power to process so much data in a timely manner.

Hundreds or thousands of small, inexpensive, commodity servers do have the power if the processing can be horizontally scaled and executed in parallel. Using the MapReduce approach, Hadoop splits up a problem, sends the sub-problems to different servers, and lets each server solve its sub-problem in parallel. It then merges all the sub-problem solutions together and writes out the solution into files which may in turn be used as inputs into additional MapReduce steps.

Hadoop has been particularly useful in environments where massive server farms are being used to collect the data. Hadoop is able to process parallel queries as big, background batch jobs on the same server farm. This saves the user from having to acquire additional hardware for a database system to process the data. Most importantly, it also saves the user from having to load the data into another system. The huge amount of data that needs to be loaded can make this impractical.

Hadoop Use Cases

Many of the ideas behind the open source Hadoop project originated from the Internet search community, most notably Google and Yahoo. Search engines employ massive farms of inexpensive servers that crawl the web retrieving web pages into local files. They then process this data using massive parallel queries to build indexes to enable search.

The actual algorithms used by search engines to determine the relevance to search terms and quality of web pages are very specialized, sophisticated, and highly proprietary. These algorithms are the secret sauce of search. The application of the algorithms to each webpage and the aggregation of the results into indexes to enable search is done through MapReduce processes and is more straightforward although done on a massive scale. The map function identifies each use of a potential search parameter in a webpage. The reduce function aggregates this data, e.g., determines the number of times a search parameter is used in a page.

There are use cases for Hadoop in many other industries. Some large websites use Hadoop to analyze usage patterns from log files or click-stream data that is generated by hundreds or thousands of their webservers. The scientific community can use Hadoop on huge server farms that monitor natural phenomena and/or the results of experiments. The intelligence community needs to analyze vast amounts of data gathered by server farms monitoring phone, email, instant messaging, travel, shipping, etc. to identify potential terrorist threats. The publishing industry may use Hadoop to index and/or reformat huge document stores. It is important to note, however, that not all highly parallel tasks are amenable to efficient MapReduce solutions.

Hadoop and the Oracle Database

Sometimes the processing needed is just a one-off. For example, a search company needs to use the search indexes generated from today's web contents. Tomorrow those indexes will be replaced by processing tomorrow's data. The processing might also be self contained. Publishing data might never need to be combined with other data the publisher may possess or acquire. In some cases, though, more complex processing of historical data with current data or with entirely different types of data will be needed that require a different type of system.

Many open source and commercial database products do not support operations such as parallel query through horizontal scaling but there are a number that do. Some of the commercial database products in particular are very advanced and offer many, many capabilities that Hadoop does not. These include many very powerful performance optimizations, sophisticated analytic functions, and ease of use features, including rich declarative features that allow even very complex analysis to be done by non-programmers. These also include enterprise class features for security, auditing, maximum availability, disaster recovery, etc. These capabilities have been produced and refined over decades of development in an extremely competitive marketplace. The Oracle Database is both the technology leader and the market share leader in this area. Again, Exadata is

Oracle's most recent, major leap in new capability for these and other types of applications.

When richer capabilities are needed, commercial databases like Oracle may not replace Hadoop but rather coexist with it. Hadoop and the Oracle Database complement each other. The sheer magnitude of data involved makes it very sensible to use the "cheap cycles" of server farms to transform masses of unstructured data with low information density into smaller amounts of information dense structured data that is then loaded into Oracle. Advanced mechanisms for loading the data that make maximum use of both Hadoop and Oracle capabilities would also be very useful.

NoSQL

There are an extremely large number of NoSQL databases being developed. Some of the more prominent include Apache Cassandra, MongoDB, Voldemort, Apache HBase, SimpleDB, and BigTable. New ones seem to pop up regularly, gain prominence, and then fade from favor. All of these technologies are in the early stages of development and most of the early adopters employ programming staffs that participate in their development. The Oracle open source database Berkeley DB is used as a storage engine for Dynamo, Voldemort, GenieDB and others. Oracle InnoDB is used as a storage engine for Voldemort, Riak, and others.

Some advocates of the term "NoSQL" emphasize that this is a distinctly different approach from SQL-based databases and is fundamentally non-relational. Others soften the term by saying "NoSQL" stands for "Not Only SQL". Indeed, there are projects underway to build SQL interfaces to NoSQL databases.

The focus of this paper is on NoSQL databases that are designed to execute very large volumes of simple updates and reads against a single, very large dataset. They are designed to handle the processing volume needed to support millions and potentially even hundreds of millions of online users. The datasets involved, although typically not as large as Hadoop is targeted at, may reach tens or hundreds of terabytes in size or larger.

Like Hadoop, these NoSQL databases do this by horizontally scaling across very large numbers of servers, e.g., tens or hundreds or even thousands of servers. The technique used by these NoSQL databases for horizontal scaling is essentially the same as has been deployed for many years to support very high volume systems using conventional relational databases. The technique is called sharding. It requires that a separate database run on each server and that the data be physically partitioned so that each database has its own subset of the data stored on its own local disks.

A problem with sharding, however, is that you lose key relational database capabilities when you shard. Specifically, you lose much of your ability to do joins, transactions, and schema changes. You also need to compromise ACID (atomicity, consistency, isolation, and durability) principles, usually consistency must be relaxed.

The core idea behind the NoSQL databases is that if you are going to lose all this capability when you shard why complicate the database by supporting these features. Instead, support other features that provide some partial functionality in some of these areas and, most of all, try to simplify and automate what is the biggest challenge - the sharding mechanism itself.

Lost Capabilities

To join any data with any other data in a sharded environment you need to be able to do distributed queries that span potentially very large numbers of separate databases. Distributed queries are more complicated than queries that join data within a single database. These complications can create overhead and be prohibitively slow. NoSQL databases, therefore, typically don't support joins.

Similarly, transactions that allow updates to multiple rows to be committed together (or all rolled back should a failure occur) require distributed transactions in a sharded environment. Distributed transactions require some form of two phase commit protocol which is more complicated and expensive than what is needed if all the data is contained within a single database Two phase commit can be slow, and worse, can compromise availability because failures can cause data to become locked and inaccessible (called 'indoubt transactions") until the failure is repaired. NoSQL databases, therefore, typically don't support transactions that involve updates to data in multiple tables or even multiple rows within a table.

It is also prohibitively difficult to make schema changes in a sharded environment. Some databases enable a schema change, such as adding a column to a table, to be done while the system is online within one database, but not across multiple databases. If you attempt to make a schema change across each database individually then the application will sometimes see the column in the table and sometimes not. If the application attempts to access the column but it is not there it will fail. NoSQL databases, therefore, typically don't support online schema changes. Many are even "schema-less". The application becomes entirely responsible for keeping track of the format of the data in the database and dealing with the different formats that it may see.

Partial Functionality

To provide partial functionality is some of these areas, NoSQL databases typically support some mechanism for adding new data on the fly. There is a great deal of variation in the exact functionality that the many different NoSQL databases provide (including document, columnar, and key/value approaches) but the core idea is that there be some way to add new data elements or columns to tables and the values contained in those

columns in a manner that applications can readily detect when the new columns are there or not.

Key/value support is one flexible and straightforward way of doing this. Key/values make it possible to add additional logic into applications to deal with online schema changes. Every data value is accompanied by a key which labels the data much like a column name. Applications can be programmed to test and appropriately deal with whether a new column and its value exist yet or not on a row by row basis.

Key/values also provide partial functionality to deal with the lack of joins by allowing de-normalized or pre-joined data to be stored in a single data set (or table). For example, say you need to store data for a blogging system where you have blog entries and comments on blog entries. With a relational system you could store them in a normalized format with blog entries in one table and comments in another with the appropriate foreign key links and join them when you need to. With key/value pairs you would have a dataset (table) containing the blog entries and then add an arbitrary list of key/value pairs to each row to store the comments in each blog entry.

Sharding

Simplifying and automating sharding is probably the most difficult challenge facing the NoSQL databases. A way needs to be found to distribute or partition the data across potentially huge numbers of servers in a manner that allows each data item to be readily located when needed and, very importantly, to balance the processing load evenly across all the servers. The biggest challenge is handling changes that force you to rebalance the distribution of the data, or repartition. This can be very difficult on a huge system with vast amounts of data and vast numbers of users always online. It is not clear whether any of the current NoSQL products have solved all these problems.

Manual sharding schemes are typically highly application dependant and require customized solutions with repeated revisions to make them work. For example, a social network may choose to partition based on user name. All users whose last name starts with "A" on one server, "B" on another, and so on. It would need to be more complicated than this since there are probably far more people whose name starts with an "A" in the world than "Z" but this can be worked out at least initially using various techniques such as lookup tables and hashing. What happens, though, when your service becomes suddenly popular in a country where people's last names do frequently start with "Z"? The data will need to be repartitioned.

Producing a generic auto-sharding solution that really works in production is difficult. There have been a number of attempts at using various forms of consistent hashing as the basis for auto-sharding. Some keys, though, have inherent problems with even distributions such as states or countries, e.g., a social network will probably have many more members from California than Rhode Island. Unique keys, like user account id,

often have balanced load distributions when hashed but have other challenges such as dealing with changes in overall processing load.

For example, what happens if you are running on a thousand servers and discover that the overall system needs 10% more processing power, or 100 additional servers, to handle the load? Cassandra, one of the newer NoSQL technologies, would add this additional capacity by identifying the 100 most heavily used servers and splitting the data on each server with one of the new servers. This may not work well:

- If the load was unbalanced so that there were 100 servers that were seriously overloaded then splitting those 100 servers may be the last thing you want to do. Splitting a server requires that you copy half of its data to the new server. That is a big operation to perform when you are already overloaded. Access to data on these servers may become unacceptably slow. Still worse, very heavy load conditions can cause systems to fail in many unexpected ways that could cascade to other systems, and so on.

- If the load was evenly balanced across the 1000 servers adding more servers probably won't do what you want either. 100 of these servers would now have double the capacity to handle their load and, in this case, about 90% of the additional capacity would go unused, while the other 900 servers would continue to be overloaded.

NoSQL and the Oracle Database

NoSQL products do have the advantage that because they are open source they may be more accessible to developers building entirely new, highly innovative applications. The applications, though, must be simple enough to fit within the constraints and limitations that NoSQL databases inherently possess.

When applications become more complex the Oracle Database provides a very natural next step in the application lifecyle because the Oracle Database supports the data and processing volumes being targeted by the NoSQL database. It does this by horizontally scaling across both servers and storage with Oracle technologies such as Real Application Clusters (RAC) and Automatic Storage Management (ASM) without the restrictions of the NoSQL approach.

With Real Application Clusters running on the database servers and Automatic Storage Management managing the storage you can horizontally scale across very large numbers of servers and storage devices but it is all one database. These technologies do not need or use distributed queries or two-phase commit to support joins or transactions. Also, schema changes can be done online which greatly simplifies application development.

Sometimes it does make sense for performance reasons to store data in de-normalized ways if the application is always going to access child table data through the parent table.

For example, you may know that comments on a blog entry are only going to be accessed when you are reading the blog entry. Oracle, though, has mechanisms for this, such as clustered indexes, that allow you to store the data in a pre-joined format for maximum performance.

Oracle Database technology is also extremely effective at dynamically balancing processing load across database servers and storage. Its forte is allowing for change. Servers and storage can be added or removed online. All data is accessible to all servers all the time. The processing involved in adding servers and storage is evenly distributed across the whole system so that no one component is over loaded.

Most of all, this technology is proven. Thousands of the largest and most mission critical systems in the world today have been running on the Oracle Database with RAC and ASM technologies with great success for years.

Conclusion

Open source Database technologies have been used to build and successfully run some of the most innovative new applications that have emerged in recent years. This is primarily because the developers of these applications have found open source databases to be very accessible. Developers with radical new ideas have been able to develop and deploy these applications quickly without the need to spend time and energy convincing others to invest in the purchase of commercial database products or the expertise of professional DBAs. Oracle today is the largest open source database vendor, with InnoDB, Berkeley DB, and MySQL to address this market need.

Many Oracle Database customers also have substantial deployments of open source databases. Oracle, therefore, has been working to support integration between these environments where that can add value. Hadoop and the Oracle Database complement each other in environments where massive server farms collect data that needs sophisticated analysis.

Oracle has also been working to support low-cost and low-risk migrations from open source databases to the Oracle Database. Many of the new, innovative applications that use open source databases today will need to provide more complex functionality to remain competitive in their market in the future and will need to move to the database platform that can best support that. The best, and really only solution, for complex applications that require extreme scalability and support for "Big Data" is the Oracle Database.

Hadoop and NoSQL Technologies and the Oracle Database
January 2011
Author: Gordon Smith

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together