Oracle NoSQL Database – Scalable, Transactional Key-value Store

Ashok Joshi, Sam Haradhvala, Charles Lamb
Oracle: Database Development
Burlington, MA 01803
ashok.joshi@oracle.com, sam.haradhvala@oracle.com, charles.lamb@oracle.com

***Abstract* - Oracle NoSQL Database is a highly scalable, highly available key-value database, designed to address the low latency data access needs of the "big data" space. Among its unique features, Oracle NoSQL Database provides major and minor keys, flexible durability and consistency policies, and integration with both MapReduce infrastructure and conventional relational data. Major and minor keys enable transactional guarantees across multiple data records. Flexible durability and consistency polices allow applications to trade off durability, consistency, availability, and performance on a per-operation basis. Integration with alternate data processing and management systems provides a cohesive environment for accommodating today's big data management needs.**

Keywords-NoSQL; Big Data; Database systems.

I. INTRODUCTION

Simply put, big data is an informal term that encompasses all sorts of data such as web logs, sensor data, tweets, blogs, user reviews, SMS messages etc. It is characterized by high volume (hundreds of terabytes or more), high variety (e.g., no inherent structure, one row “looks” very different from another) and high velocity (hundreds of thousands of operations per second). It is possible to derive valuable information (e.g., sentiment analysis) by aggregating big data in some way, though, quite often, an individual data row may or may not provide much value or insight. The conventional wisdom is that it is cost-prohibitive to manage and process big data using only traditional relational database technologies.

Companies such as Google [1], [2], Amazon [3], LinkedIn [4] and Facebook [5] have demonstrated that there is significant business benefit in harnessing big data. They have also made major contributions to the database community in terms of algorithms and frameworks for massive-scale distributed processing using commodity processors.

This has resulted in a huge surge of interest in big data in the information technology industry and commercial community. The allure of increasing profitability, reducing costs, being better “connected” with customers and improving the business is too hard to ignore. However, for a variety of reasons described below, only a small number of organizations have been able to successfully leverage the business value of big data.

The pioneers of big data processing employ armies of super-smart developers to work on big data problems. Commercial organizations have neither the budget nor the ability to invest significantly in “deep” software development projects. Big data are inherently unstructured or semi-structured; this makes it difficult to process big data. Finding the “nuggets of gold” in the vast amounts of unstructured data requires specialized technologies and expertise. Finally, in order to obtain the maximum benefit, big data need to be combined with traditional structured data (SQL data repositories).

In October 2011, Oracle announced a suite of products and technologies aimed at providing a complete and comprehensive solution to address the big data needs of the market. A key piece of this technology stack is Oracle's entrant into the NoSQL space, the Oracle NoSQL Database.

We begin with an overview of Oracle NoSQL Database. This is followed by a description of important features including major and minor keys, sharding and replication, and consistency and durability policies that can be selected on a per-operation basis. The next section discusses the role and benefits of interactive big data processing. We present some performance results that clearly demonstrate the excellent throughput, response time and scalability of Oracle NoSQL Database. Finally, we conclude the paper with our view of the big data processing landscape and a short description of the comprehensive hardware and software technologies and solutions from Oracle, designed and optimized to address the big data processing needs of the market.

II. ORACLE NOSQL DATABASE

Big data processing falls into two major categories – interactive data management and batch processing. The Oracle NoSQL Database [6] is a scalable, highly available, key-value store that can be used to acquire and manage vast amounts of interactive information.

Oracle NoSQL Database uses Oracle Berkeley DB Java Edition [7] as the underlying data storage engine, thus leveraging all the performance and availability benefits that it has to offer. Berkeley DB Java Edition is a mature product that also provides many of the features and characteristics that are necessary for a building a distributed key-value store such as Oracle NoSQL Database.

Figure 1 illustrates the architecture of an Oracle NoSQL Database deployment; in this example, there are two client nodes and multiple server nodes for managing key-value data. The system is designed to handle large numbers of client nodes as well as servers.

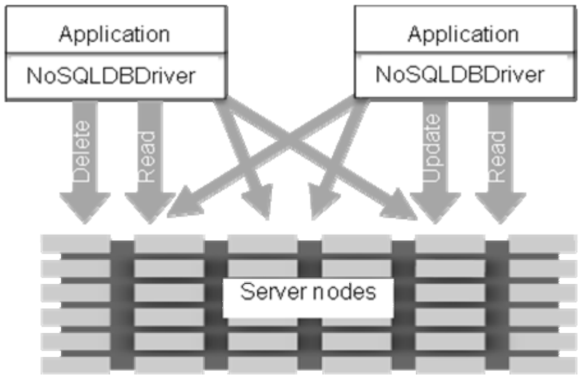

Figure 1: Oracle NoSQL Database architecture.

A. *Major Keys, Minor keys and values*

Oracle NoSQL Database provides a key-value paradigm to the application developer. Every entity (record) is a set of key-value pairs. A key has multiple components, specified as an ordered list. The major key identifies the entity and consists of the leading components of the key. The subsequent components are called minor keys. This organization is similar to a directory path specification in a file system (e.g., /Major/minor1/minor2/). The "value" part of the key-value pair is simply an uninterpreted string of bytes of arbitrary length.

This concept is best explained using an example. Consider storing information about a person, John Smith, who works at Oracle Headquarters, start date Jan 1, 2012 and has a telephone number +1650-555-9999. The user's Id might be a logical choice for major key for the person entity (for example, 123456789). Further, the 'person' entity might contain personal information (such as the person's telephone number) and employment information (such as work location and hire date). The application designer can associate a minor key (e.g. personal_info) with the personal information (+1-650-555-9999) and another minor key (e.g. employment_info) with the employment information (Oracle Headquarters, start date Jan 1, 2012).

Specifying the major key "123456789" would return "John Smith". Specifying "/123456789/personal_info" as the key would access John Smith's personal information; similarly, "/123456789/employment_info" would be the key to access the employment information. Leading components of the key are always required. NoSQL Database internally stores these as three separate key-value pairs; one for the user_id, a second for user_id/personal_info and the third for user_id/employment_info.

The API for manipulating key-value pairs is simple. The user can insert a single key-value pair into the database using a put operation. Given a key, the user can retrieve the key-value pair using a get operation or delete it using a delete operation. The get, put, and delete operations operate on only a single (multi-component) key. NoSQL Database provides additional APIs that allow the application to operate on multiple key-value pairs within an entity (same major key) in a single transaction.

The major key determines which shard the record will belong to. All key-value pairs associated with the same entity (same major key) are always stored on the same shard. This implementation enables efficient, "single shard" access to logically related subsets of the record. Figure 2 illustrates the concept of major and minor keys.

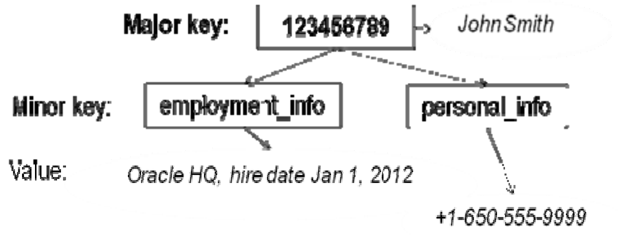

Figure 2: Major and minor keys.

Oracle NoSQL Database also provides an unordered scan API that can be used to iterate through all the records in the database; unordered scans do not have transaction semantics, though only committed data will be returned to the application.

B. *Shards and Replicas*

Oracle NoSQL Database is a client-server, sharded, shared-nothing system. The data in each shard are replicated on each of the nodes which comprise the shard. As discussed earlier, Oracle NoSQL Database provides a simple key-value paradigm to the application developer. The major key for a record is hashed to identify the shard that the record belongs to. Each key-value pair is always stored and managed within a single shard. Oracle NoSQL Database is designed to support changing the number of shards dynamically in response to availability of additional hardware. If the number of shards changes, key-value pairs are redistributed across the new set of shards dynamically, without requiring a system shutdown and restart.

Each shard is highly available. A shard is made up of a single master node which can serve read and write requests, and several replicas (usually two or more) which can serve read requests. Replicas are kept up to date using streaming replication. Each change on the master node is committed locally to disk and also propagated to the replicas. If the master node should fail, one of the surviving replicas is automatically elected as a master and processing continues uninterrupted. As soon as the failed node is repaired, it rejoins the shard, is brought up to date and then becomes available for processing read requests. Thus, the Oracle NoSQL Database server can tolerate failures of nodes within a shard and also multiple failures of nodes in distinct shards. By proper placement of masters and replicas on server hardware (racks and interconnect switches), Oracle NoSQL Database achieves very high levels of availability on commodity servers.

C. *Consistency and Durability*

Distributed systems need to address the notion of consistency, since there is a lag between making a change on one node and propagating the same change to another replica. On the other hand, distributed systems can take advantage of the multiple copies of data to alleviate the "commit to disk" bottleneck. In particular, these systems can consider a transaction as being committed after receiving acknowledgements for the changed record from the replicas, without waiting for the disk I/O to complete. NoSQL Database supports the notions of variable consistency for read operations and varying degrees of durability for update operations. Further, NoSQL Database exposes these options at the API level so that the application designer can make the appropriate tradeoffs between performance and consistency/durability on a per operation basis. Many other systems with a similar architecture provide only a coarse level of control over consistency and durability (e.g., system-wide choice configured at system start-up); per-operation consistency and durability enables a broad class of applications by giving developers more control over the data.

There are several choices for read consistency. The application can specify absolute consistency if it needs the most recent version or can also specify time or LSN-based (log sequence number) consistency for read operations. Note that LSNs are not visible to the application directly; rather, they manifest themselves as point-in-time version handles. For example, an application might be willing to tolerate reading data that is no more than one second out-of-date. LSN-based consistency is useful in scenarios where the application modifies a record at a certain LSN x and wants to ensure that a subsequent read operation will read a version of that same record that is at least as current as the change identified by LSN x (it is okay to read a more recent version). Finally, the application can also specify that it doesn't care how consistent the data are, for a particular read request. Oracle NoSQL Database routes the request to the appropriate node (master or one of the replicas) in the shard based on the desired consistency.

In a shared everything architecture [8], making a transaction durable requires that the data management system commit the changes to disk (log) before acknowledging the completion of the transaction. In a distributed, master-replica architecture such as Oracle NoSQL Database, the transaction must be made durable on the master and also propagated to the replicas. This presents some interesting opportunities and tradeoffs. For example, the master node may choose to issue a lazy log write and concurrently send commit messages to the replicas. This strategy makes the transaction durable by "committing to the network", which is desirable if network latency is lower than disk latency. The transaction can be considered as committed if one or more replicas have received the changes associated with a transaction. The lowest latency choice is to issue a lazy log write at the master, concurrently send non-blocking commit messages to the replicas, and acknowledge the completion of the transaction. For additional assurance of durability, the system may choose to wait for acknowledgement messages from the replicas. Several other variants of this strategy are possible.

Transaction durability is thus determined by a combination of log write at the master node, log writes at the replicas, sending transaction commit messages to the replicas and receiving commit message acknowledgements from the replicas. Further, the system can decide whether to wait for acknowledgements from a majority of the replicas or all replicas. Of course, each legitimate combination of these options also influences performance and availability. For example, "write to local disk, write to replica disk, wait for acknowledgements from every replica" provides the highest level of durability but is also expensive in terms of latency and throughput.

Figure 3 illustrates the durability and consistency options that are available in Oracle NoSQL Database. NoSQL Database allows the user to choose the durability policy on a per operation basis. NoSQL Database uses this information during commit processing in order to achieve the best performance while honoring the durability requirements of the operation.

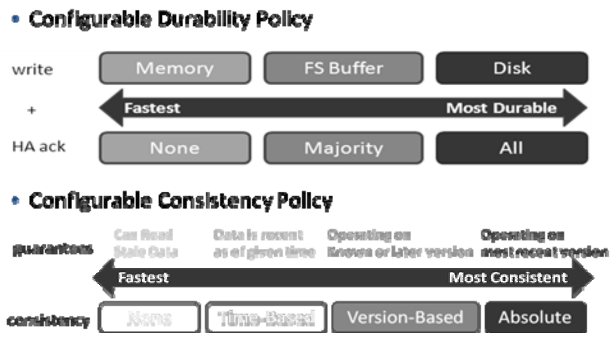

Figure 3: Durability and Consistency.

D. *Interactive Big Data Processing*

Oracle NoSQL Database has been designed for applications that need fast, predictable, low latency access to vast amounts of data. Let us examine how Oracle NoSQL Database benefits such applications by considering a typical E-commerce environment. Such systems manage vast numbers of user profiles and have stringent response time requirements. Whenever a user visits the site, the retailer provides a personalized experience based on the user's profile. If no such profile exists, the site must create one. These user profiles will change over time as the retailer learns more about the users through interactions. Different user profiles may contain radically different information and the retailer may decide to collect new information at any time. Oracle NoSQL Database addresses this use case by virtue of its flexible key-value paradigm and scales to meet increasing customer demand. Figure 4 illustrates some performance data [9] for Oracle NoSQL Database using Yahoo Cloud Serving Benchmark [10]. The graph on the left illustrates scalable write performance while the graph on the right illustrates scalable performance for a mixed read and update workload.

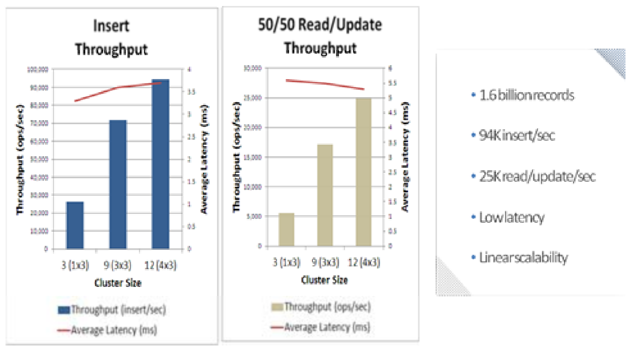

Figure 4: Performance on YCSB.

E. *Big Data: The Big Picture*

Oracle recognized early on that in order to derive maximum benefit from big data, it is necessary to combine and process unstructured and semi-structured content together with structured data. Business-critical information is primarily stored in relational database repositories. This information can be augmented with information from unstructured content in order to obtain business insights that can be gained only by combining structured and unstructured data. For example, sales forecast information is typically stored in relational repositories. By combining sales forecast information with big data content such as political trends, weather predictions, etc., it is possible to improve the accuracy of the sales forecasts.

Relational database systems have sophisticated algorithms for data warehousing, analysis and reporting. These algorithms typically work well with structured (row and column) data. There is also a wide array of products, tools and services available to analyze relational data, generate business intelligence and drive decisions. These tools and processes form the cornerstone of business data processing and analysis today.

In comparison, the tools available for processing big data (unstructured data) are relatively scarce and immature. However, it is possible to transform unstructured content into structured (row and column) format so that it becomes available for processing with data warehousing and business intelligence technologies. During this transformation process, one can also aggregate and cleanse the data to reduce the volume and increase information density of the content.

Oracle recently introduced a suite of complementary technologies to manage and process big data and also combine big data with traditional data warehousing and data analysis technologies for maximum business benefit [11]. Oracle NoSQL Database provides the interactive big data management component of this integrated solution. Oracle has adopted Cloudera's distribution of Apache Hadoop [12] in order to provide MapReduce capabilities and the open source distribution of R [13] for advanced analytics. Oracle Big Data Connectors, a separately licensed product, provides a high-performance Hadoop to Oracle Database integration solution. Oracle database and Oracle Business Intelligence tools provide the data warehousing, mining and analysis capabilities.

Oracle's approach to big data is unique for three reasons: it leverages the existing investments in data management and processing, seamlessly brings the benefits of big data to the enterprise, and finally, provides a commercial-grade, comprehensive solution to process and leverage all the data in the enterprise.

Oracle has gone a step further and also delivered the Big Data Appliance [14] that delivers software and hardware packaged together into an optimized platform that simplifies the management, analysis and mining of all the data in an enterprise.

ACKNOWLEDGMENT

Margo Seltzer, Alan Bram, Dave Segleau and Marie-Anne Neimat provided valuable feedback on earlier drafts of this paper.

We are very grateful to the Oracle Labs team and Jeffrey Alexander, in particular, for their help and advice on the architecture and design of Oracle NoSQL Database.

We are very grateful to Cisco and Raghu Nambiar in particular, for partnering with us to run the benchmarks on Cisco UCS.

REFERENCES

[1] http://research.google.com/archive/bigtable.html [retrieved: Oct, 2012]
[2] http://research.google.com/archive/mapreduce.html [retrieved: Oct, 2012]
[3] http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf [retrieved: Oct, 2012]
[4] http://project-voldemort.com/ [retrieved: Oct, 2012]
[5] http://en.wikipedia.org/wiki/Apache_Cassandra [retrieved: Oct, 2012]
[6]Oracle NoSQL Database Documentation: http://www.oracle.com/technetwork/products/nosqldb/overview/index.html [retrieved: Oct, 2012]
[7] Oracle Berkeley DB Java Edition documentation: http://www.oracle.com/technetwork/products/berkeleydb/documentation/index.html [retrieved: Oct, 2012]
[8] Anupam Bhide: An Analysis of Three Transaction Processing Architectures. VLDB 1988: 339-350
[9] Ashok Joshi, Raghunath Nambiar: Cisco UCS Ecosystem for Oracle: Extend Support to Big Data and Oracle NoSQL Database: http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns944/le_34301_wp.PDF [retrieved: Oct, 2012]
[10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan and Russell Sears. http://research.yahoo.com/node/3202 [retrieved: Oct, 2012]
[11] http://www.oracle.com/us/technologies/big-data/index.html [retrieved: Oct, 2012]
[12] http://www.cloudera.com/ [retrieved: Oct, 2012]
[13] http://www.r-project.org/ [retrieved: Oct, 2012]
[14] http://www.oracle.com/us/products/database/big-data-appliance/overview/index.htm [retrieved: Oct, 2012