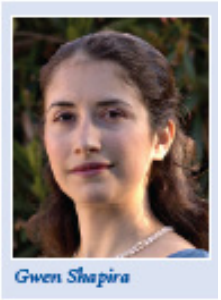
Oracle Big Data Appliance

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.

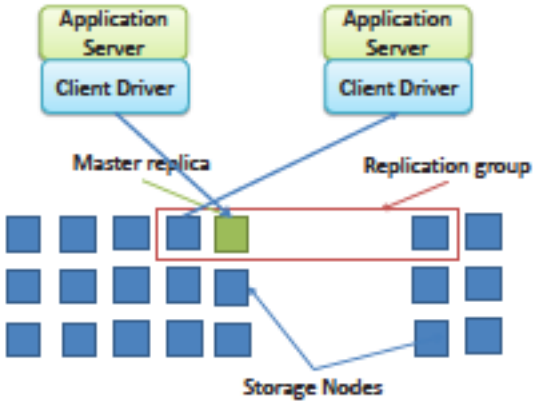
The main software components of Oracle Big Data Appliance are Cloudera Hadoop and Oracle NoSQL. Both are non-relational databases that were designed for high scalability. But as we'll soon see, they are very different in their architecture, design goals, and use cases. The most striking thing about Cloudera 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.



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

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 assigned 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 leastloaded 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 ex-

ample, 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.

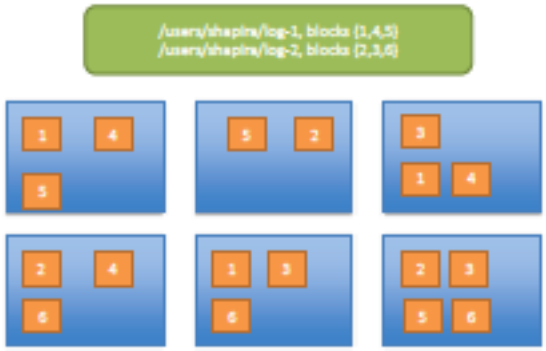
Hadoop is an architecture designed to use inexpensive and unreliable hardware to build a massively parallel and highly scalable data-processing cluster. It was designed so that adding servers will result in a proportional increase in load capacity and that server failure will result in performance decline 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 enterprise 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.

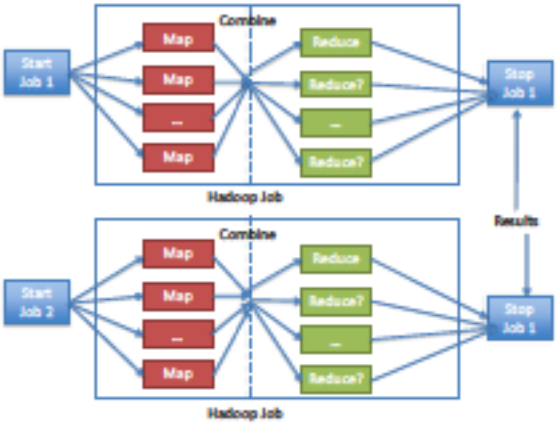
Similar to Oracle NoSQL, HDFS also improves reliability by copying each block to at least three different servers in the cluster. The replication doesn't just provide failover in case a node crashes; it also allows multiple jobs to process the same data in parallel on different servers. (http://hadoop.apache.org/common/docs/current/hdfs_design.html)



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 preconfigured."

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.