ORACLE
NOSQL DATABASE

Integrating Apache Spark with Oracle NoSQL Database

ORACLE WHITE PAPER | JANUARY 2016

Introduction

In recent times, big data analytics has become a key enabler for business success. Big data analytics requires performant data stores and high-level analytics frameworks providing the right abstractions. The low latency and high scalable storage provided by Oracle NoSQL Database[1] coupled with high performance analytics provided by Apache Spark[2] makes an attractive combination for performing big data analytics. In this paper we take distributed log analysis as a simple analytics use case and explain how to store log data in Oracle NoSQL Database and perform analytics on top of it using Apache Spark.

In the following sections we discuss:

- High-level architecture of Oracle NoSQL Database and Apache Spark and the integration between them
- Log analysis use case
- How to implement the use case using Oracle NoSQL Database and Apache Spark

Oracle NoSQL Database

Oracle NoSQL Database is a highly scalable, highly available, fault tolerant, “Always On” distributed key-value database which you can deploy on low cost commodity hardware in a scale-out manner. Use cases of Oracle NoSQL Database include Distributed Web-scale Applications, Real Time Event Processing, Mobile Data Management, Time Series and Sensor Data Management, Online Gaming etc. Oracle NoSQL Database offers all the features which are common to a typical NoSQL product like Elasticity, Eventually Consistent Transactions, Multiple Data Centers, Secondary Indexes, Security and Schema Flexibility. The key differentiators include ACID Transaction, Online Rolling Upgrade, Streaming Large Object Support, Engineered Systems, and Oracle Technology integrated.

Architecture diagram of Oracle NoSQL Database is as shown in Figure 1.

Apache Spark

Apache Spark is a powerful open source general-purpose cluster computing engine for performing high speed sophisticated analytics. Some of its key features include:

- Speed: Spark provides high speed analytics by harnessing its advanced Directed Acyclic Graph (DAG) execution engine which supports cyclic data flow and in-memory computing.
- Ease of use: Spark has easy to use APIs in Java, Scala, Python and R for working on large datasets. Spark offers over 80 high-level operators for performing actions and transformations.

[1] http://www.oracle.com/us/products/database/nosql/overview/index.html
[2] http://spark.apache.org/

- Generality: Spark supports a rich set of higher-level tools for SQL and structured data processing, machine learning, graph processing, and streaming. You can combine these libraries seamlessly in the same application to create complex workflows.
- Runs everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, S3, and any Hadoop data source.

Architecture diagram of Apache Spark is as shown in Figure 2.

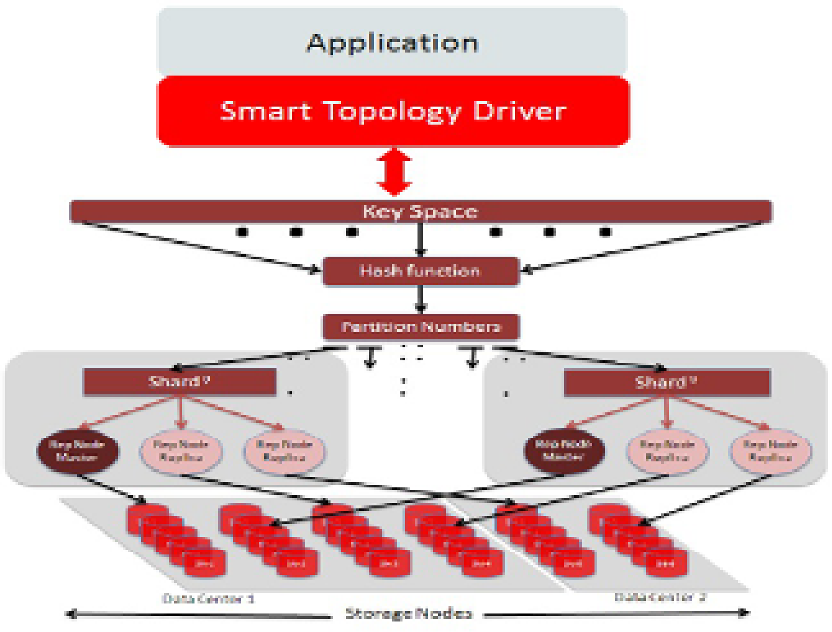

Figure 1. Architecture of Oracle NoSQL Database

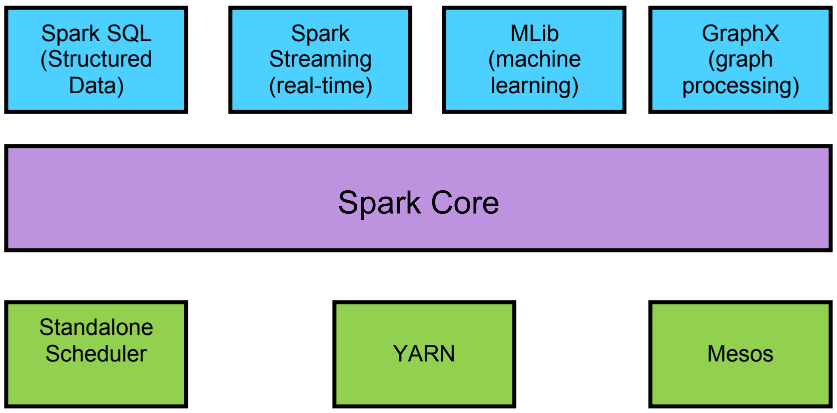
Spark SQL (Structured Data)
Spark Streaming (real-time)
MLib (machine learning)
GraphX (graph processing)
Spark Core
Standalone Scheduler
YARN
Mesos

Figure 2. High level architecture of Apache Spark

Resilient Distributed Datasets (RDD)

Spark is built around the concept of Resilient Distributed Datasets (RDD[3]). An RDD is a lazily evaluated read-only collection of records. It can be transformed into another RDD by operations such as map, filter, and join. Actions such as count are also supported to compute and output results. Most importantly Spark offers control on how the RDDs are persisted. They can be cached in-memory, or configured to be stored on disk, replicated across machines, etc.

Due to the caching abstractions provided, Spark is a compelling framework for applications which operate on a set of data repetitively. For example, iterative machine learning applications, and interactive data exploration and mining are compelling use case classes for Spark. Spark has been known to be used in a broad range of analytics applications such as in-memory analytics, traffic modeling, social network spam analysis, social network analysis, fraud detection, recommendations, log processing, predictive intelligence, customer segmentation, sentiment analysis, and IoT analytics.

At a high level, every Spark application consists of a driver program that launches various parallel operations on a cluster. Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster. Once you have a SparkContext, you can use it to build RDDs. Once an RDD is instantiated, you can apply a series of operations. To run these operations, driver programs typically manage a number of nodes called executors. Figure 3 shows a Spark application using two executors.

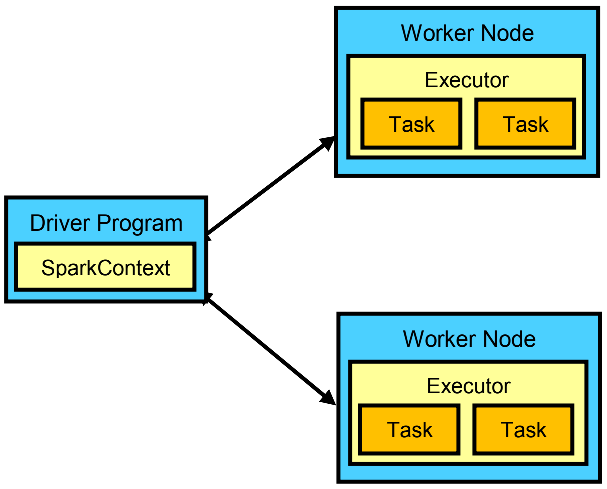

Figure 3. An example Spark application using two executors

Integrating Spark with Oracle NoSQL Database

Spark allows loading of data from a Hadoop file. This can be done by extending Hadoop's InputFormat class. So, any datastore that implements Hadoop's InputFormat specification can act as a data source to Spark. Oracle NoSQL Database provides TableInputFormat class which extends Hadoop's InputFormat class there by allowing data to be loaded from Oracle NoSQL Database.

[3] https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

Spark's newAPIHadoopRDD method allows to specify classes for InputFormat handling and classes for keys and values. In addition a configuration can be specified to be used by the InputFormat class. For more information, see section "Log Analysis Using Spark".

Log Analysis Use Case

The log analyzer program allows keyword-based search of the logs and categorizes the search results into classes for easier log comprehension. In addition it allows for filtering on the various fields of the log entries such as time period, host on which the logs entry had taken place, and the severity level of each entry.

For this use case we have taken logs output by Oracle NoSQL Database cluster while running a performance test. The entries get logged on all the nodes of the cluster and have the following format:

```
<timestamp> <log severity> <host> <log message>
```

However, there can be some entries, such as java exception traces, in the log file which do not conform to this format. All such log content is ignored.

Log Processing and Storage

Log files from all the nodes are collected into a single place before processing. Then for each file the log entries are parsed into fields and pushed into the Oracle NoSQL Database. The definition of the NoSQL table is pretty simple and is as shown below:

```
CREATE TABLE Log
    id INTEGER,
    tsMillis LONG,
    logLevel STRING,
    host STRING,
    message STRING,
    PRIMARY KEY (id))
```

The tsMillis field captures the timestamp in the form of the number of milliseconds since Epoch time. This allows for filtering of logs based on time periods such as since last hour, day, and week. All the rest of the field definitions are self explanatory.

Logs are pushed into Oracle NoSQL Database using the following command:

```
java loganalyzer.PushLogs <log_directory>
```

Log Analysis Using Spark

Once the log entries are in the database, Spark is used to analyze the logs as mentioned in the following steps:

1. The driver program creates a SparkContext object as shown below.

```
SparkConf sparkConf = new SparkConf().setAppName("SparkTableInput")
                                     .setMaster("local[2]");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
```

The above code specifies that two executors should be spawned on the local machine.

2. Read the log data from Oracle NoSQL Database into Spark as shown below:

```
conf.set("oracle.kv.kvstore", STORE_NAME);
hconf.set("oracle.kv.tableName", TABLE_NAME);
hconf.set("oracle.kv.hosts", KVHOST);
jrdd = sc.newAPIHadoopRDD(hconf, TableInputFormat.class,
                          PrimaryKey.class, Row.class);
```

The classes PrimaryKey, Row, and TableInputFormat are provided by Oracle NoSQL Database. PrimaryKey and Row are the classes for keys and values respectively. TableInputFormat is the class that handles the read access from the Oracle NoSQL Database. It reads the configuration parameters kvstore, tableName, and hosts to connect to the database. A successful execution of this method returns an RDD.

3. Specify filters to the tool from the command line. The actual usage looks as follows:

```
Usage: java loganalyzer.LogAnalyzer [options]
options:
  --for-the-past hour|day|week process logs for the past given time period
  --host <hostname>            process logs for this host
  --level INFO|SEVERE|WARNING   Log level to consider
  --search <keyword>           Search keyword
```

If filtering is requested, the filter method is used on this RDD by passing an appropriate filter function. For example, if the entries need to be filtered based on log level, the following code would do the needful:

```
levelData = logData.filter(new Function<Row, Boolean>(){
  public Boolean call(Row row){
    String level = row.get("logLevel").asString().get();
    return fLevel.equals(level);
  }
});
```

The call method takes a row as an argument and returns true if the logLevel matches the required level. This method gets executed on all the rows and finally the filter method returns a new RDD containing only the required rows.

4. Categorize the logs for easier comprehension, once the filtering is complete. See the following example log entries:

```
2015-12-23 12:03:04 SEVERE host1 Process 1234 crashed
2015-12-23 12:05:06 SEVERE host2 Process 4567 crashed
2015-12-23 12:07:08 SEVERE host3 Process 8910 crashed
2015-12-23 12:08:09 SEVERE host4 Process 1112 crashed
2015-12-23 12:09:10 SEVERE host1 Commit of transaction -1112 failed
2015-12-23 12:10:11 SEVERE host1 Commit of transaction -1314 failed
2015-12-23 12:11:12 SEVERE host2 Commit of transaction -1516 failed
```

While doing log analysis, the user may just want to know things such as how many process crashes or transaction failures occurred, without caring about the process ids etc. A simple categorization can be done by replacing the numerical values in the log messages.

Note: In a real application, it is possible to do a more complex analysis.

After performing this simple categorization, output looks as follows:

```
4: Process ? crashed
3: Commit of transaction -? failed
```

The process IDs and transaction numbers have been replaced with a '?' character. Also, the number of occurrences of each type of message is counted and printed in the first column. The above output says there were four process crashes and three transaction failures. This makes the log comprehension easier.

The code for message categorization is as follows:

```
// Pull the messages only from the rows and replace integers with '?'
JavaRDD<String> noIntMesgs = logData.map(new Function<Row, String>(){
  @Override
  public String call(Row row){
    return row.get("message").asString().get().replaceAll("\\d+", "?");
  }
});

// Compute distinct counts
Map<String, Long> counts = noIntMesgs.countByValue();
```

Using the map method the log messages are altered by replacing the numerical values in the message with a '?' and then countByValue is used to count the occurrences of each unique message.

Conclusion

In this paper, we have taken a log analysis use case to explain how to perform analytics in Spark on the data stored in Oracle NoSQL Database. Storing of data from Spark into Oracle NoSQL Database is a feature that we are working on and will be part of a future release.

The code for LogAnalyzer is available under the directory spark-nosql in the github repository https://github.com/ashutoshsnaik/oraclenosqldb.git

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Integrated Cloud Applications & Platform Services

Integrating Apache Spark with Oracle NoSQL Database
January 2016
Authors: Ravi Prakash Putchala, Ashutosh Naik, Ashok Joshi, Michael Schulman

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