Real-Time Healthcare Analytics on Apache Hadoop* using Spark* and Shark*

Executive Summary

Healthcare providers can get more valuable insights, manage costs, and provide better care options to patients by using data analytics and solutions. Big data technologies are enabling providers to store, analyze, and correlate various data sources to extrapolate knowledge. Benefits include efficient clinical decision support, lower administrative costs, faster fraud detection, and streamlined data exchange formats. It is projected that adoption of health data analytics will increase to almost 50 percent by 2016 from 10 percent in 2011, representing a 37.9 percent compound annual growth rate[1].

Apache Hadoop* and MapReduce* (MR*) technologies have been in the forefront of big data development and adoption. However, this architecture was always designed for data storage, data management, statistical analysis, and statistical association between various data sources using distributed computing and batch processing. Today's environment demands all of the above, with the addition of real-time analytics. The result has been systems like Cloudera Impala*[2] and Apache Spark*[3], which allow in-memory processing for fast response times, bypassing MapReduce operations.

To compare MapReduce with real-time processing, consider use cases like full text indexing, recommendation systems (e.g., Netflix* movie recommendations), log analysis, computing Web indexes, and data mining. These are processes that can be allowed to run for extended periods of time.

Spark use cases include stream processing (e.g., credit card fraud detection), sensor data processing (e.g., data from various sources joined together), and real-time querying of data for analytics (e.g., a business intelligence Web dashboard for 24/7 operations).

Organizations can now store large datasets in Hadoop Distributed File Systems (HDFS) and use real-time analytics software built on top of architecture like Spark to access data directly from HDFS, bypassing any data migration headaches.

The ease of access to data has also taken precedence over the need to write programs in Java*, Scala*, or Python* languages. Apache Shark[4] is a large-scale data warehouse system that runs on a Spark cluster and allows SQL* queries to run up to 100 times faster than Hive* (which uses MapReduce).

Intel is interested in collaborating with companies in the healthcare industry to accelerate this work. Intel wants to provide businesses with an open-enterprise Hadoop platform alternative for next-generation analytics and life sciences.

Author

Abhi Basu,
Big Data Solutions,
Intel Corporation
abhi.k.basu@intel.com

Contributor

Terry Toy,
Big Data Solutions,
Intel Corporation
terry.toy@intel.com

Contents

Called Intel® Distribution for Apache Hadoop software, this solution enhances both manageability and performance and is optimized for Intel® Xeon® processors[5].

This paper explains how to install and configure Apache Spark and Shark on Intel Distribution for Apache Hadoop software 3.0.2 and perform simple analytical queries using SQL (Hive QL*) in real time.

Objectives

Specifically, the key objectives of this paper are to:

- Define the operating system, software, stack and tools.
- Define the setup and configuration of Apache Spark and Shark.
- Run some tests from Shark to validate that the installation of all components was successful.

Audience

Software developers and technologists can use this document to install the above software as a proof point.

System Setup and Configuration

Components

Table 1 shows the components of the solution.

Table 1. Solution Components

Component	Details
Hardware (One-Node Cluster Used)	• Intel® Xeon® processor E5-2680 (FC-LGA10 2.7GHz 8.0GT/s 20MB 130W 8 cores CM8062107184424 [2-CPU]) • 128 GB RAM (1333 Reg ECC 1.5V DDR3 Romley) • KDK – Grizzly Pass 2U 12x3.5 SATA 2x750W 2xHS Rails Intel R2312GZ4GC4 • 300GB SSD 2.5in SATA 3Gb/s 25nm Intel Lyndonville SSDSA2BZ300G301 710 Series • 2TB HDD 3.5in SATA 6Gb/s 7200RPM 64MB Seagate Constellation* ES ST2000NM0011 • NIC - Niantic* X520-SR2 10GBase-SR PCI-e Dual Port E10G42BFSR or E10G42BFSRG1P5 Duplex Fiber Optic • LSI HBA LSI00194 (9211-8i) 8 port 6Gb/s SATA +SAS PCIe 2.0 Raid LP
Operating System	Cent OS 6.4 (Developers Workstation version)
Application Software	1. Intel® Distribution for Apache Hadoop* software version 3.0.2 (Hadoop 2.0.4) 2. Scala 2.9.3 3. Apache Spark 0.8.0 4. Apache Shark 0.8.0

Software Setup

These instructions assume a fully-functional version of Intel® Distribution of Apache Hadoop software 3.0.2 (refer to https://hadoop.intel.com/ pdfs/IDH-InstallGuide_R3-0_EN.pdf) has already been installed on Cent OS 6.4 (Developers Workstation Version) and is ready for use as a single-node Hadoop cluster. All installations were performed logged in as root user to the system. We installed a single-node Spark cluster on the same node, running parallel to Hadoop and with access to HDFS.

Figure 1[6] demonstrates the Shark and Spark architecture we are setting up. MapReduce is bypassed by accessing HDFS directly from the Spark cluster.

Scala

If you do not have Scala 2.9.3 installed on your system, you can download it:

- $ wget http://www.scala-lang.org/files/archive/scala-2.9.3.tgz
- $ tar xvfz scala-2.9.3.tgz

Update /root/.bashrc here:

- export SCALA_HOME=/path/to/scala-2.9.3
- export PATH=$SCALA_HOME/bin:$PATH

Spark

Download source from http://spark.incubator.apache.org/downloads.html. Look for version 0.8.0.

- Build using your current Hadoop version to be able to access HDFS from Spark: SPARK_HADOOP_VERSION=2.0.4 sbt/sbt assembly [IDH 3.0.2 version is 2.0,.4]
- $ mv spark-*-bin-*/ spark-0.8.0
- Edit spark-0.8.0/conf/slaves to add the hostname of each slave, one per line. For our setup, there should only be “localhost”.
- Edit spark-0.8.0/conf/spark-env.sh to set SCALA_HOME and SPARK_WORKER_MEMORY.
- export SCALA_HOME=/path/to/scala-2.9.3
- export SPARK_WORKER_MEMORY=16g

Note that SPARK_WORKER_MEMORY is the maximum amount of memory that Spark can use on each node. Increasing this allows more data to be cached; however, be sure to leave memory (e.g., 1 GB) for the operating system and any other services the node may be running.

Shark

First, download the binary distribution of Shark 0.8.0. The package contains two folders, shark-0.8.0 and hive-0.9.0-shark-0.8.0-bin.

- $ wget https://github.com/amplab/shark/releases/download/v0.8.0/shark-0.8.0-bin-cdh4.tgz
- $ tar xvfz shark-*-bin-*.tgz
- $ cd shark-*-bin-*

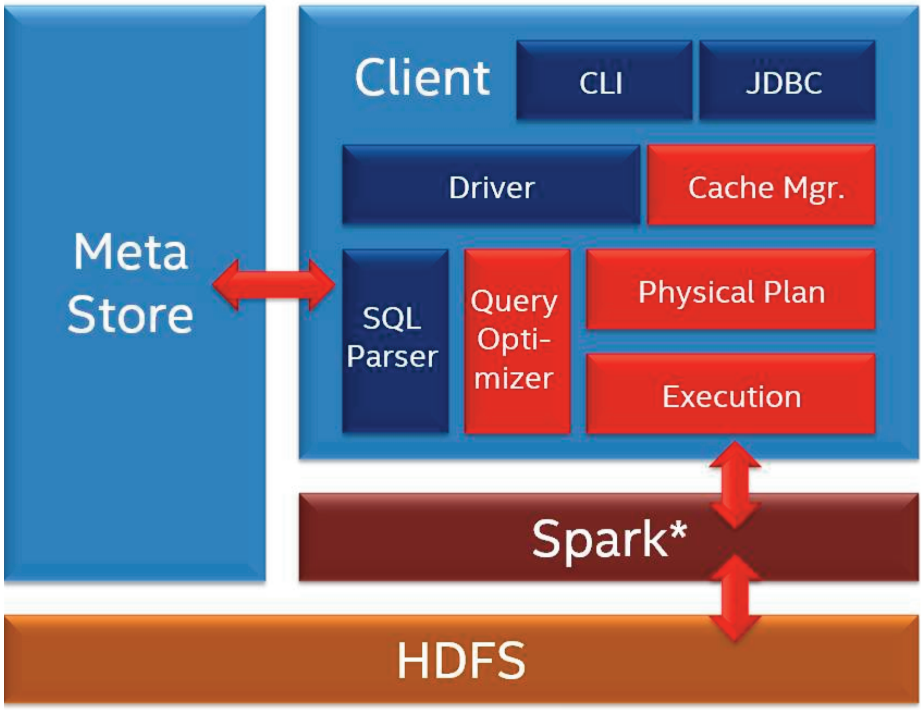

Figure 1. The architecture of the Shark and Spark solution[6]

Next, edit shark-0.8.0/conf/shark-env.sh to set the HIVE_HOME, SCALA_HOME and MASTER environmental variables. The master URI must exactly match the spark:// URI shown at port 8080 of the standalone master. Go to http://<host-name>:8080 to obtain that value.

Add to shark-env.sh:

- export HADOOP_HOME=/usr/lib/hadoop
- export HIVE_HOME=/path/to/hive-0.9.0-shark-0.8.0-bin
- export MASTER=<Master URI>
- export SPARK_HOME=/path/to/spark
- export SPARK_MEM=16g
- source $SPARK_HOME/conf/spark-env.sh
- The last line is there to avoid setting SCALA_HOME in two places. Make sure SPARK_MEM is not larger than SPARK_WORKER_MEMORY set in the previous section. SPARK_MEM is memory allocated to each Spark application (every Shark CLI is a Spark app).
- If you are using Shark on an existing Hive installation, be sure to set HIVE_CONF_DIR (in shark-env.sh) to a folder containing your configuration files. Alternatively, copy your Hive XML configuration files into Shark's hive-0.9.0-bin/conf.

Finally copy all Hadoop*.jar (and Hadoop-hdfs.jar) to $SHARK_HOME/lib_managed/jars/org.apache.hadoop/ respective sub-folders.

Note that if there is more than one node, you need to copy the Spark and Shark directories to slaves. The master should be able to SSH to the slaves. We ignore this for our single node setup.

Testing

Start the Spark cluster: /path/to/spark-0.8.0/bin/start-all.sh (provide passwords to root for all slaves)

Start Shark CLI

- /path/to/shark-0.8.0/bin/shark (Starts Shark CLI)
- Run some SQL commands - show tables, select count(*) from tables, etc.

To verify that Shark can access HDFS, you can try the following example, which creates a table with sample data:

- Move file in /path/to/shark/hive/examples/files/kv1.txt and move it to a location on hdfs like /tmp/.

Start Shark CLI (/path/to/shark-0.8.0/bin/shark) and run the following commands:

- CREATE TABLE src(key INT, value STRING);
- LOAD DATA LOCAL INPATH '/tmp/kv1.txt' INTO TABLE src;
- SELECT COUNT (*) FROM src; ; [This query runs over Shark on disk]
- CREATE TABLE src_cached AS SELECT * FROM SRC; [This creates the table in memory and access should be faster than Shark on disk]
- SELECT COUNT (*) FROM src_cached

Run the SQL commands a few times and take average of runs on Shark over disk versus Shark over RAM to see the performance differences. Larger datasets would be a better alternative for some impromptu benchmarks.

To learn more about Intel Distribution for Apache Hadoop software, visit http://www.intel.com/content/www/us/en/software/intel-hpc-distribution-for-apache-hadoop-software.html.

[1]" Data Analytics Poised for Big Growth," http://www.healthcareitnews.com/news/data-analytics-assume-power-health-it. Accessed February 2, 2014.
[2]"Cloudera Impala," http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html. Accessed February 2, 2014.
[3]"Apache Spark," http://spark.incubator.apache.org/. Accessed February 2, 2014.
[4]"Apache Shark," http://shark.cs.berkeley.edu/. Accessed February 1, 2014.
[5]"Intel® Distribution for Apache Hadoop Software," http://hadoop.intel.com/products/distribution. Accessed February 2, 2014.
[6]Zaharia, Matei, "Spark and Shark," http://www.slideshare.net/Hadoop_Summit/spark-and-shark. Accessed January 21, 2014.
[7]"Shark User Guide," https://github.com/amplab/shark/wiki/Shark-User-Guide. Accessed January 15, 2014.