Notes about the Coursera course

Hadoop Platform and Application Framework

Claudia Gutiérrez Escribano
Diego Duque Zumajo

November 13, 2016

Hadoop Basics

0.1 Introduction

The Apache Hadoop software library is a **framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models**. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.
The key idea is to move computation to data, not data to computation.
The project includes these modules:

- Hadoop Common: The common utilities that **support the other Hadoop modules**. It contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput **access to application data**. See the HDFS comic (http://www.mail-archive.com/common-user@hadoop.apache.org/
for more information about how it works.
- Hadoop YARN: A framework for job scheduling and cluster resource **management**.
- Hadoop MapReduce: A YARN-based system for **parallel processing of large data sets**.

Other Hadoop-related projects at Apache, about them we will talk later, are: Hive, Pig, Spark, Zookeeper, HBase, etc.

0.2 Apache Sqoop

Tool designed for efficiently transferring bulk data between Apache Hadoop and structure datastores such as relational databases.

0.3 Hbase

Hbase is akey component of the Hadoop stack, as its design caters to applications that requieres really fast random access to significant data set.

- Column-oriented database management system.
- Key-value store.
- Based on Google big Table
- Can hold extremely large data
- Dynamic data model
- Not a Relational DBMS.

0.4 Pig

It's a scripting language for creating MapReduce [1] programms using Hadoop.

- Hihg level programming on top of Hadoop MaReduce.
- These language is called Pig Latin.
- Data analysis problems as data flows.

You can execute Pig scripts in other languages.

0.5 Hive

Data warehouse software facilities queryng and managing large datasets residing in distribute storage.

0.6 Oozie

Workflow scherudler system to manage Apache Hadoops jobs. Supports: MApReduce, Pig, Apache Hive, and Sqoop, etc.

0.7 Zookeeper

- Provides operational services for a hadoop cluster.
- Centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing.

0.8 Flume

Distributed, reliable, and avaible service for efficiently collecting, aggregating, and moving large amount of data.

0.9 Spark

A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation. Spark Benefits:

- Multi-stage in-memory primitives provides performance up to 100 times faster for certain applications.
- Allows user programs to load data into a cluster's memory and querit repeatedly.
- Well.suited to machine learning.

Appendix

[1] **MapReduce** is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
A MapReduce program is composed of a Map() procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).
An example to understand how MapReduce works:

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)
```

```
function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
     sum += pc
  emit (word, sum)
```

With this piece of code a document is split into words, and each word is counted by the *ma*p function, using the word as a key. The framework put together all the pairs with the same key and feeds them to the same call to *reduce*. so, this function only has to sum all its input to find the total appearances of that word.