Mu Sigma

Getting started with Hadoop

Do The Math

Chicago, IL
Bangalore, India
www.mu-sigma.com

August 6, 2012

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"

Agenda

- The Story of Hadoop
- Building Blocks of Hadoop
- Working with Hadoop

Why Hadoop?

- The amount of digital information produced in 2011 was around 1800 exabytes, that was around 10 times the information produced in 2006
- The majority of this data is “unstructured” – complex data poorly suited for management by structured storage systems like relational databases
- Unstructured data comes from many sources and takes many forms – web logs, text files, sensor readings, user-generated content like product reviews or text messages, audio, video, photos and the like
- Large volumes of complex unstructured data can hide important insights
  - Do user logs from a web site contain information about relationships among individual customers?
  - Can a collection of nucleotide sequences be assembled into a single gene?
- Companies that can extract facts like these from huge volume of data can better control the processes and costs; can better predict demand and can build better products

Where did Hadoop come from?

- The underlying technology was invented by Google in their early days so that they could usefully index all the rich textual and structural information they were collecting, and then present meaningful and actionable results to users

- In 2004 a paper named **MapReduce: Simplified Data Processing on Large Clusters** by Jeffrey Dean and Sanjay Ghemawat from Google Lab was published

- This paper inspired Doug Cutting to develop an open-source implementation of the Map-Reduce framework. He named it Hadoop, after his son's toy elephant

What is Hadoop?

- Dealing with big data requires two things:
  - Inexpensive, reliable storage
  - New tools for analyzing unstructured and structured data
- Apache Hadoop is a powerful open source software platform that addresses both of these problems. The platform was designed to solve problems where you have a lot of data, perhaps a mixture of complex and structured data, and it doesn't fit nicely into tables
- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model
- It is designed to scale up from a single server to thousands of machines, each offering local computation and storage
- Rather than rely on hardware to deliver high-avaiability, the library itself is designed to detect and handle failures at the application layer

How is Hadoop architected?

- Hadoop is designed to run on a large number of machines that don’t share any memory or disks
- That means that you can buy a whole bunch of commodity servers, put them together in a rack and run hadoop on each one
- When you load data into Hadoop, the hadoop framework chops the data into pieces that are then spread across the different servers
- There is no one place where you can go to and talk with your data. Hadoop keeps track of where the data resides on the servers
- Data Replication: The pieces of your data is also saved at multiple servers. And because there are multiple copies, data stored on a server that goes offline or dies can be automatically replicated from a known good copy

Building Blocks of Hadoop

- Hadoop employs master/slave architecture for both distributed storage and distributed computation. Running Hadoop means running a set of daemons or resident programs in your network. Some exist on one server, while some exists on multiple servers. The daemons include:
- NameNode
- DataNode
- Secondary NameNode
- JobTracker
- TaskTracker

Namenode – master of the HDFS

- Hadoop's distributed storage system is called Hadoop File System (HDFS). The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks.

- The NameNode is the bookkeeper of HDFS. It keeps track of how your files are broken down into file blocks, which nodes store those blocks and the overall health of the distributed filesystem.

- The function of the NameNode is memory and I/O intensive**.** For one hadoop cluster, we will be having only one NameNode running. So if a NameNode fails your whole cluster goes down. NameNode is a single point of failure for the Hadoop cluster. For other daemons, if their host nodes fail because of software or hardware reasons, the Hadoop cluster continue to function smoothly or you can quickly restart the daemons that went down.

- Note: Since Hadoop 0.23.2, high availability Namenode was introduced which eliminates the SPOF

Datanode – the data handling daemon in every slave node

- There will be one datanode per slave machine in your hadoop cluster. Each slave machine will host DataNode daemon to perform the grunt work of the distributed filesystem - reading and writing HDFS blocks to actual files on the local filesystem.

- Whenever we want to write a HDFS file, the NameNode will tell your client which DataNode each block resides in or where it should write the block. The client communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Also DataNode can communicate with other DataNodes to replicate its data blocks for redundancy.

- DataNodes constantly reports to the NameNode. Upon initialization, each of the DataNodes informs the NameNode of the blocks it is currently storing. After this mapping is complete, the DataNodes continually poll the NameNode to provide information regarding local changes as well as receive instructions to create, move or delete blocks from the local disk.

Secondary NameNode – keeps a copy of NameNode's HDFS metadata

- Secondary NameNode is an assistant daemon for monitoring the state of HDFS cluster. Like NameNode, each cluster has one Secondary NameNode. It does not receive or record any real-time changes to HDFS, instead it communicates with the NameNode to take the snapshots of the HDFS metadata at intervals defined by the cluster configuration.

- Since NameNode is single point of failure for a Hadoop-cluster, Secondary NameNode's snapshots help minimize the downtime and loss of data. However, NameNode failure requires human intervention to reconfigure the cluster to use the Secondary NameNode as the primary NameNode.

JobTracker – Assigns jobs to the tasktrackers and gets the work done

- Once we submit the code to the cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they are running. If any task fails, the JobTracker will automatically relaunch the task, possibly on a different node, up to the predefined limit of retries.

- There is only one JobTracker daemon per cluster and it typically runs on a server which is the master node of the cluster.

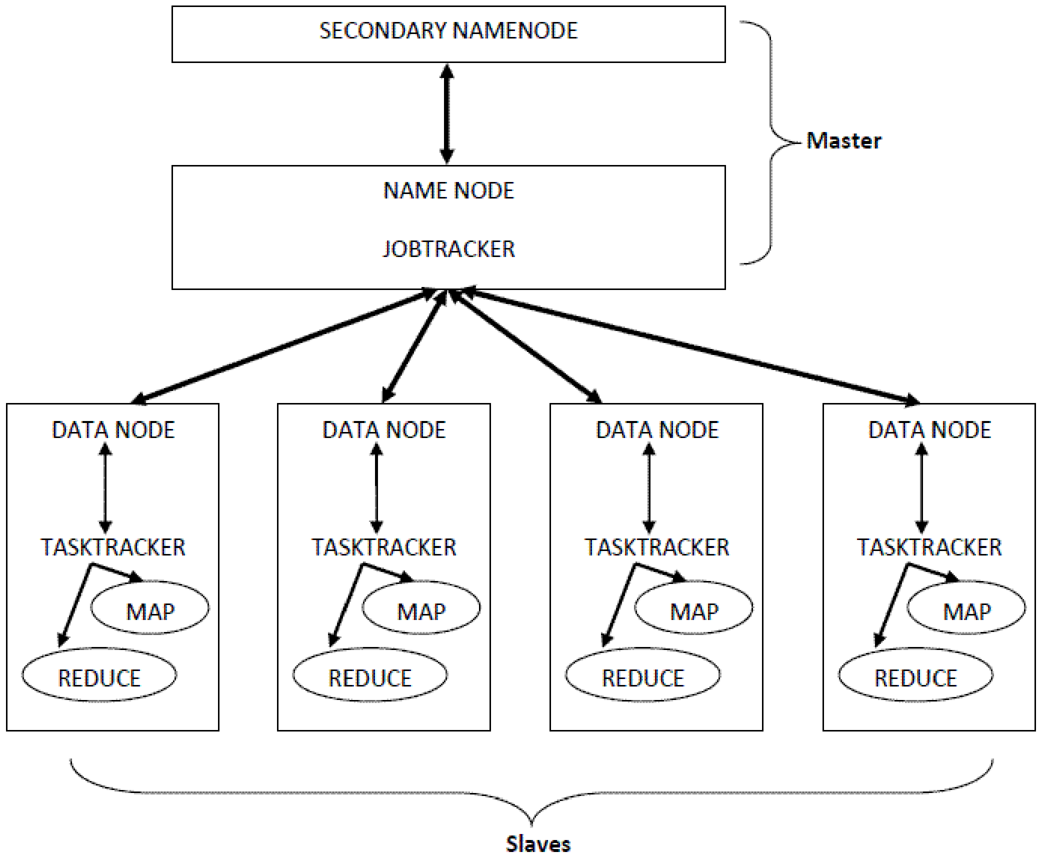
TaskTracker

- Computing daemons also follow master/slave architecture: JobTracker is the master overseeing the overall execution of the MapReduce job and the TaskTrackers manage the execution of individual tasks on each slave node.

- Each TaskTracker is responsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many maps or reduce tasks in parallel.

- One responsibility of the TaskTracker is to constantly communicate with the JobTracker. If JobTracker fails to receive any heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes to the cluster.

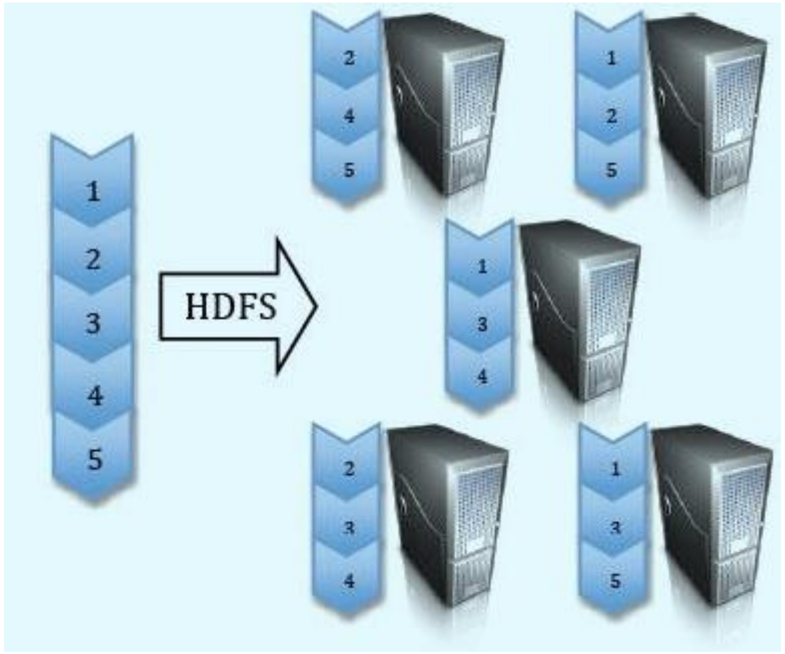
The Daemons


SECONDARY NAMENODE
NAME NODE
JOBTRACKER
Master
DATA NODE
TASKTRACKER
MAP
REDUCE
DATA NODE
TASKTRACKER
MAP
REDUCE
DATA NODE
TASKTRACKER
MAP
REDUCE
DATA NODE
TASKTRACKER
MAP
REDUCE
Slaves

How is data stored in the HDFS

- Here a very basic idea of how files are stored in HDFS is show
- First the file is split into blocks of 64 MB or 128 MB depending upon the default block size
- These blocks are then distributed across the cluster depending upon the replication factor
- By default the replication factor is 3
- Thus even if a few datanodes crash or become

  unavailable the data is not lost



Lets get started with Hadoop

- The $HADOOP_HOME/bin directory contains some scripts used to launch Hadoop DFS and Hadoop Map/Reduce daemons

1. **start-dfs.sh** - Starts the Hadoop DFS daemons, the namenode and datanodes. Use this before start-mapred.sh

2. **stop-dfs.sh** - Stops the Hadoop DFS daemons

3. **start-mapred.sh** - Starts the Hadoop Map/Reduce daemons, the jobtracker and tasktrackers

4. **stop-mapred.sh** - Stops the Hadoop Map/Reduce daemons

5. **start-all.sh** - Starts all Hadoop daemons, the namenode, datanodes, the jobtracker and tasktrackers.

6. **stop-all.sh** - Stops all Hadoop daemons.

Test if the daemons have started

- Use the command **jps** to find out if the processes have started.
- jps output should look something like this

```
$ jps
1375 SecondaryNameNode
1560 TaskTracker
12835 Jps
1246 DataNode
1453 JobTracker
1138 NameNode
```

- Check the corresponding daemon logs if any of the daemons don't start as expected.

Hadoop Web Interface

- http://localhost:50030/- web UI for MapReduce job tracker(s)
- http://localhost:50060/ - web UI for task tracker(s)
- http://localhost:50070/ - web UI for HDFS name node(s)

Understanding the difference between HDFS and the local linux file system

- HDFS is distributed, the files that you upload into the HDFS will be split into blocks and it will be distributed across the cluster
- Hadoop jobs can only access data in the HDFS, so if you have data in the local file system, say in your home directory, you can’t give its location as an input parameter for your job
- You need to copy the file into the HDFS before running the job. You will get a better understanding as we run some examples

Running some examples

- To keep thing simple we will run the “hello world” of the hadoop world, the wordcount
- Wordcount is a simple mapreduce program that reads a file and counts the number of times a word occurs in that file
- Hadoop comes with a set of example programs that are bundled in the hadoop examples jar file. You can use these programs for testing out hadoop, wordcount is one of the examples.
- Wordcount needs 2 arguments;
  - The input folder containing files for processing and
  - The output-folder for writing the results
- As it was told before, hadoop does not understand local file system, so we need to copy the files to the HDFS before we can run our wordcount program on them

Running WordCount on hadoop

- First make sure your hadoop is running.
  - $ start-all.sh

- Copy files that needs to be processed from local filesystem to HDFS so that the hadoop job can access the data
  - $ hadoop fs -copyFromLocal /home/hadoop/Documents/Hadoop/GettingStartedWithHadoop/Datasets/WordcountInput

- Check whether your data is copied properly on HDFS filesystem or not.
  - $ hadoop fs -ls
  - $ hadoop fs -ls input

- Now we have the data to be processed, run wordcount example on it.
  - $ cd $HADOOP_HOME
  - $ hadoop jar hadoop-examples-0.20.2-cdh3u4.jar wordcount TrainingDatasets/Hadoop/WordcountInput TrainingDatasets/Hadoop/WordcountOutput

Viewing the output from the HDFS

- The output will be stored in a HDFS location as specified in the output argument
  - TrainingDatasets/Hadoop/WordcountOutput
- This is a HDFS location and you can view the output by the following command
  - hadoop fs –ls TrainingDatasets/Hadoop/WordcountOutput/
  - hadoop fs –cat TrainingDatasets/Hadoop/WordcountOutput/*

Configuring Hadoop to our needs

- Hadoop configuration files are stored in $HADOOP_HOME/conf location. Make sure while making any changes to these files, login as hadoop user that can run hadoop jobs.

- Configuring & tweaking hadoop is done by manipulating the following files

- Open these files to understand what they do

1. **hadoop-env.sh** --- deals with hadoop specific environment variables.

2. **core-site.xml** --- specifying HDFS storing directory as well as HDFS URI

3. **hdfs-site.xml** --- specifying how many copies our cluster have to maintain

4. **mapred-site.xml** --- specifying details about jobtracker

5. **masters** --- specifying hostname of master

6. **slaves** --- specifying hostname of slaves

Let’s wrap it up!

Hadoop is Accessible, Robust, Scalable and Simple

- Accessible: It runs on large clusters of commodity machines. It also supports cloud computing services such as Amazon’s EC2
- Robust: Hadoop is architected with the assumption of frequent hardware failures. So we can say it is designed to support fault tolerance
- Scalable: Hadoop scales linearly to handle larger data by adding more nodes to the cluster
- Simple: Allows user to quickly write efficient parallel code

  You can also use Hive, Pig or hadoop streaming if you not a big fan of java programming

- Please visit these links
  - http://wiki.apache.org/hadoop/FAQ
  - http://hadoop.apache.org/

Thank You

Chicago, IL
Bangalore, India
August 6, 2012
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly prohibited"