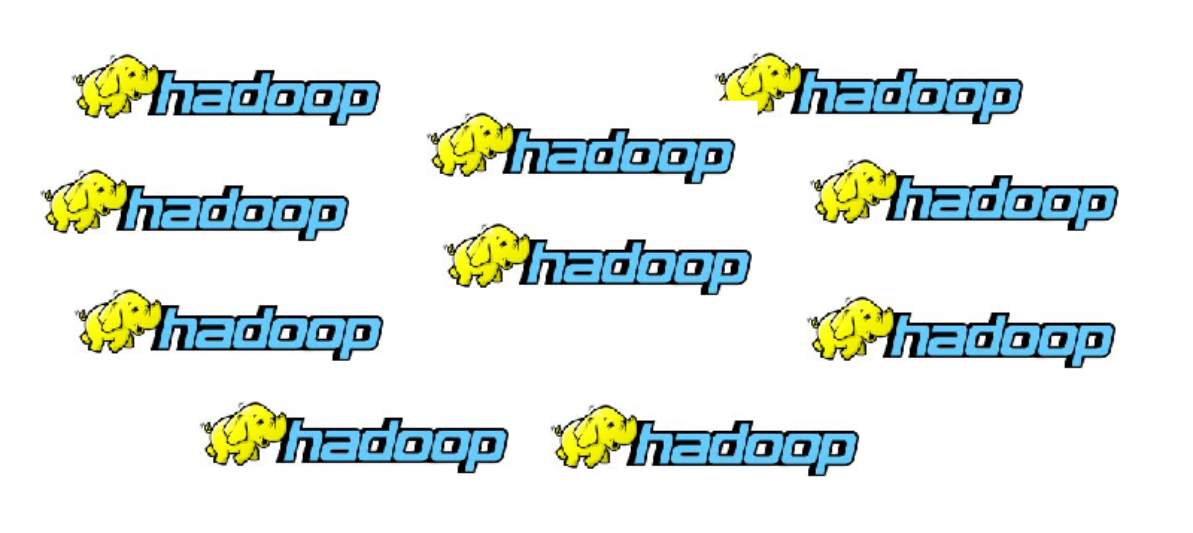
Apache Hadoop

Amit Jain

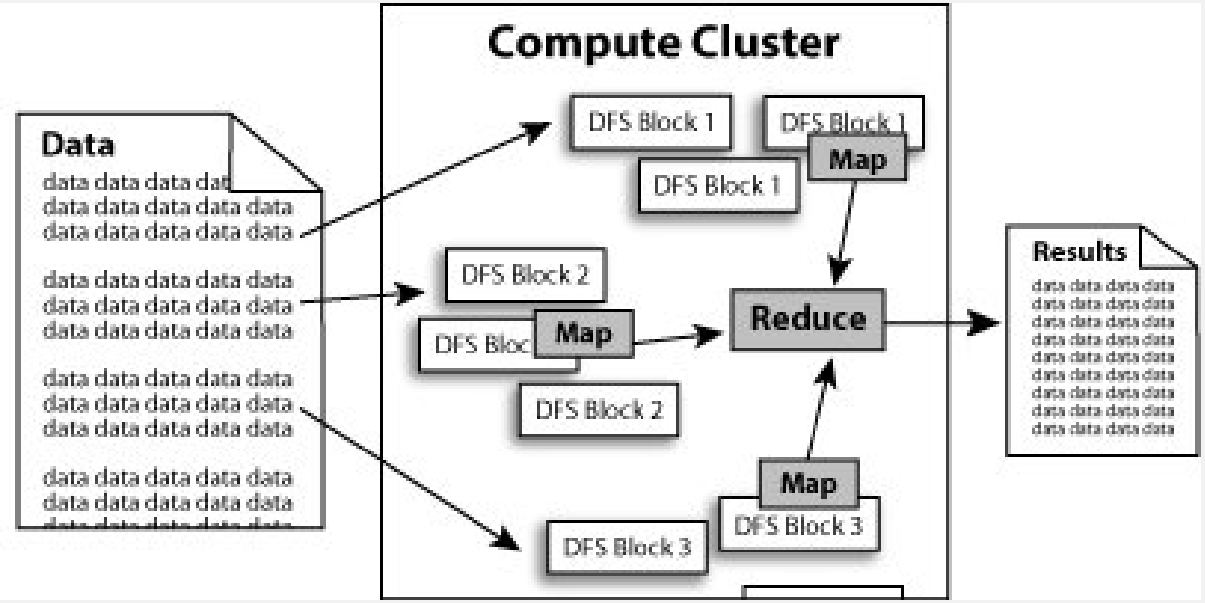


The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Features of Hadoop:

- Scalable: Hadoop can reliably store and process Petabytes.
- Economical: It distributes the data and processing across clusters of commonly available computers. These clusters can number into the thousands of nodes.
- Efficient: By distributing the data, Hadoop can process it in parallel on the nodes where the data is located. This makes it efficient.
- Reliable: Hadoop automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

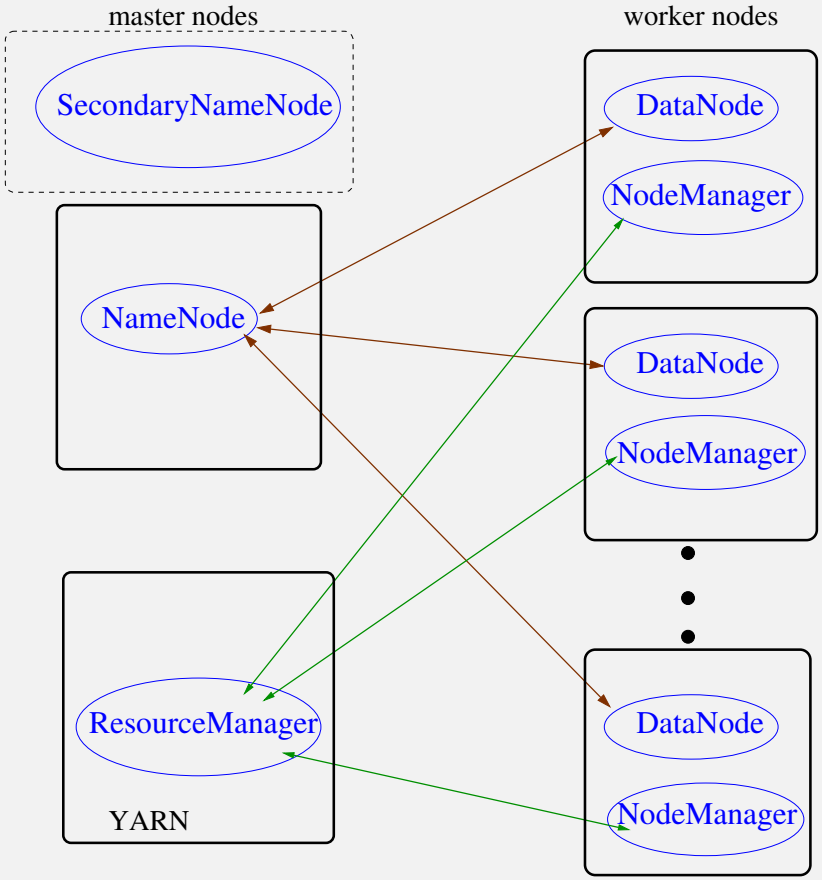
Hadoop Implementation

- Hadoop implements a MapReduce framework, using the Hadoop Distributed File System (HDFS).
- MapReduce divides applications into many small blocks of work. MapReduce can then process the data where it is located.
- HDFS creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster.



Hadoop Servers/Daemons

The Hadoop Distributed File Systems (HDFS) is implemented by a `NameNode` server on a master node and a `DataNode` server on each data node.
The MapReduce framework is implemented by a `ResourceManager` on a master node and a `NodeManager` on each worker node. There are additional servers depending upon the setup.



Hadoop History

- Hadoop is sub-project of the Apache foundation. Receives sponsorship from Google, Yahoo, Microsoft, HP and others.
- Hadoop is written in Java. Hadoop MapReduce programs can be written in Java as well as several other languages.
- Used by Facebook, Amazon, RackSpace, Twitter, EBay, LinkedIn, New York Times, E-Harmony (!) and Microsoft (via acquisition of Powerset). Most of Fortune 50 and beyond are using Hadoop regularly!

Hadoop Development Environments

- Hadoop clusters use Linux as the production environment so you will need to be somewhat familiar with using Linux to be able to use a hadoop cluster effectively.
  - See the class resources for a handy Linux guide and a short course.
- MS Windows and Mac OSX can be used for development but not supported as a full production Hadoop cluster.
- Hadoop MapReduce Java programs can be developed using Eclipse (or other Java IDEs) on MS Windows or MacOsX or Linux platforms. Hadoop has a full API for Java programs.
- For Python programs, we can use Hadoop in the simple streaming node. Or we can use the Pydoop package that provides a full interface with Hadoop API. For developing the code, we can use Jupyter notebook/lab or any IDE that supports Python.

Setting up Hadoop (1)

- Create a folder named `hadoop-install` in your home directory.
- Download a stable binary version (currently 3.3.6) of Hadoop from `http://hadoop.apache.org`. The downloaded file should be `hadoop-3.3.6.tar.gz` (`hadoop-3.3.6-aarch64.tar.gz` in Linux running on VM on a newer Mac with M1/M2 chips). Move it to the `hadoop-install` folder from your downloads folder.

```
mv hadoop-3.3.6.tar.gz ~/hadoop-install/
cd ~/hadoop-install/
```

- Unpack the tarball that you downloaded in previous step using the command shown below. It should create a folder named `hadoop-3.3.6` in the `hadoop-install` folder.

```
tar xzvf hadoop-3.3.6.tar.gz
```

- Make a symbolic link (shortcut) to point to it as shown below.

```
ln -s hadoop-3.3.6 hadoop
```

Setting up Hadoop (2)

- Make sure you have Java version 11 installed on your system. Either have it as your default or find out its installation folder.
- Navigate to the unpacked Hadoop folder and then edit etc/hadoop/hadoop-env.sh and set JAVA_HOME to point to Java installation folder on your system. You can use a text editor such as `vim` a graphical text editor such as `gedit` to accomplish this step. In the department lab and cluster, it is already set to `/usr/lib/jvm/java-11`

```
cd hadoop
cd etc/hadoop
gedit hadoop-env.sh
--> search for JAVA_HOME in the file and change it to the following (or
    similar)
export JAVA_HOME=/usr/lib/javm/java-11
```

- To use Java 11 (Java 11 is required) for your login account, add the following two lines to the end of your `.bashrc` startup file. Again. no need to do this on the department lab or cluster.

```
echo "export JAVA_HOME=/usr/lib/jvm/java-11" >> ~/.bashrc
echo "export PATH=$JAVA_HOME/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
```

Setting up Hadoop (3)

- Setup your path to include hadoop commands as follows:

```
echo "export HADOOP_HOME=~/hadoop-install/hadoop/" >> ~/.bashrc
echo "export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH" >> ~/.
    bashrc
source ~/.bashrc
```

- Test it to see if it finds the `hadoop` command

```
[amit@kohinoor ~]$ which hadoop
~/hadoop-install/hadoop/bin/hadoop
```

- Test it to see if it finds the `start-dfs.sh` command

```
[amit@kohinoor ~]$ which start-dfs.sh
~/hadoop-install/hadoop/sbin/start-dfs.sh
```

- Make sure that SSH server software is installed and running (it is already setup in the lab).

```
sudo yum install openssh-server
sudo systemctl enable sshd
sudo systemctl start sshd
```

Setting up Hadoop (4)

- Make sure that you can run ssh localhost date command without a password. If it asks for a password, then you have two options.
  - If you already have a SSH key (for example, from the backpack command for this or other classes), then you simply do the following step:

    ```
    cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
    ```

    or

    ```
    cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys
    ```

    depending on the type of key you find in your `/.ssh` folder.
  - Otherwise, you need to generate a SSH key and setup the authorization as follows:

    ```
    ssh-keygen -t ed25519
    cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys
    ```

- Now run the hadoop command from anywhere on your system to test the Java setup. You should get output similar to shown below.

```
[amit@kohinoor hadoop]$ hadoop
Usage: hadoop [--config confdir] COMMAND
where COMMAND is one of:
...
```

Hadoop Running Modes

We can use hadoop in three modes:

- *Standalone mode*: Everything runs in a single process. Useful for debugging.
- *Pseudo-distributed mode*: Multiple processes as in distributed mode but they all run on one host. Again useful for debugging distributed mode of operation before unleashing it on a real cluster.
- *Distributed mode*: "The real thing!" Multiple processes running on multiple machines.

Standalone Mode

- Hadoop comes ready to run in standalone mode out of the box. Try the following to test it (from the `hadoop` install folder).

```
mkdir input
cp etc/hadoop/*.xml input

bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
    examples-3.3.6.jar grep input output 'dfs[a-z.]+'

cat output/*
```

- To revert back to standalone mode, you need to edit three config files in the etc/hadoop folder: core-site.xml, hdfs-site.xml and mapred-site.xml and make them be basically empty. See below for links to template examples of these files.
  - core-site.xml
  - hdfs-site.xml
  - mapred-site.xml
- The provided setmode.sh script in the `CS535-resources` repo does that for us automatically. Run it as follows:
  `./setmode.sh standalone`

Pseudo-Distributed Mode

To run in pseudo-distributed mode, we need to specify the following:

- The NameNode (Distributed Filesystem master) host and port. This is specified with the configuration property fs.default.name.
- The Replication Factor should be set to 1 with the property dfs.replication. We would set this to 2 or 3 or higher on a real cluster.
- A workers file that lists the names of all the hosts in the cluster. The default workers file is etc/hadoop/workers it should contain just one hostname: `localhost`

Pseudo-Distributed Mode Config Files

- To run in pseudo-distributed mode on your machine, we need to edit three config files: `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`.
- The provided setmode.sh script in the `CS535-resources` repo does that for us automatically.
- Run the `setmode.sh` script found in `examples/Hadoop/local-scripts` folder of the class examples git repo to configure the setup to run in pseudo-distributed mode.
  `./setmode.sh pseudo-distributed`

Pseudo-Distributed Mode (1)

- Now create a new Hadoop Distributed File System (HDFS) with the command:

  `hdfs namenode -format`

- Start the Hadoop daemons.

  `start-dfs.sh`

- Give the elephants a few seconds to get up and stretch! Then check status of HDFS. You should see one live datanode.

  `hdfs dfsadmin -report`

- For pseudo-distributed mode, Hadoop DFS stores all the data metadata in the folder `/tmp/hadoop-amit`, where you would replace `amit` by your login name on your system. Go poke around it carefully!
- If you are having trouble getting the DFS started, a simple hack is to remove that folder.

  `/bin/rm -fr /tmp/hadoop-amit`

  WARNING! This command will destroy all the data in the DFS!! In a production cluster, we don't give this type of access to normal users.

Common issues with starting Hadoop DFS

- Check if another process is using port 9000 or 9870.

```
netstat -nap | grep 9000
```

```
netstat -nap | grep 9870
```

- If it is your own process, then try to kill the process

```
killall -9 <process-id>
```

- Make sure you have adjusted the config files using the provided `setmode.sh` script!

Pseudo-Distributed Mode (2)

- Create user directories for your login name on that system.
  Note that Hadoop converts your username to all lowercase.
  We would only do this one time, when we create the DFS

  ```
  hdfs dfs -mkdir /user
  ```

  ```
  hdfs dfs -mkdir /user/amit
  ```

- Now start the *ResourceManager* and *NodeManager* daemons:

  ```
  start-yarn.sh
  ```

- **Optional**: Start the *Job History* daemon:

  ```
  mapred --daemon start historyserver
  ```

Pseudo-Distributed Mode (3)

- Point your web browser to localhost:9870 to check the status of Hadoop DFS and browse it on the web.
- Point your web browser to localhost:8088 to watch the Hadoop ResourceManager. We would normally leave this window open to see your jobs running.
- Point your web browser to localhost:19888 to see the history of your jobs.

Pseudo-Distributed Mode (3)

- Navigate to the wordcount example in the class resources repository. A jar file containg a compiled java MapReduce program has been provided as a test. (see the `README.md` file in the folder for instructions on creating a new one).
- Put input files into the Distributed file system. Check if they got copied to the DFS.

  ```
  hdfs dfs -put input input
  ```

  ```
  hdfs dfs -ls input
  ```

  Note that if we skip the last argument, the `hdfs` command names the `input` folder the same on the DFS.
- Now we can run the pseudo-distributed mapreduce job.

  ```
  hadoop jar wc.jar input output
  ```

  Note that if we did not specify the main class when creating the jar (or if there are multiple classes that are main classes), we can specify the name of the class to run after the `wc.jar` argument.
- Copy the output back to local file system. Check the output.

  ```
  hdfs dfs -get output output
  ```

Pseudo-Distributed Mode (3)

- When you are done, stop the Hadoop daemons as follows. In a production cluster, they would be running all the time but in the lab or on your personal machine, we recommend stopping them so if the machine gets turned off, we don't run into issues when we come back.

```
stop-yarn.sh
```

```
stop-dfs.sh
```

```
mapred --daemon stop historyserver
```

  (The command above is needed only if we started the history server)
- To find out more about a command such as `hadoop dfs`, just type it without arguments and it will print a help summary.

Streaming Hadoop Map-Reduce with Python

- Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. For example:

```
hadoop jar ~/hadoop-install/hadoop/share/hadoop/tools/
    lib/hadoop-streaming-3.3.6.jar \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /usr/bin/wc
```

- We can use Python scripts in the same way, where we use stdin and stdout to stream the data to and from the HDFS.

```
hadoop jar ~/hadoop-install/hadoop/share/hadoop/tools/
    lib/hadoop-streaming-*.jar \
-mapper mapper.py -reducer reducer.py \
-input input -output output \
-file ./mapper.py -file ./reducer.py
```

- We need to use the file option to specify to hadoop to bundle our Python scripts so they are available inside Hadoop.
- See the word count example in Python in the folder in the class repository: Hadoop/myExamples/word-count/python.
- See Apache Hadoop streaming guide for more information.

References

- *Hadoop: An open source implementation of MapReduce*. The main website: http://hadoop.apache.org/.
- Documentation for Hadoop 3.3.6: https://hadoop.apache.org/docs/r3.3.6/
- *Hadoop: The Definitive Guide*. Tom White, June 2015, O'Reilly.
- Documentation for Hadoop 3.3.6 Streaming: https://hadoop.apache.org/docs/r3.3.6/hadoop-streaming/HadoopStreaming.html
- Using Streaming Python in Hadoop. See this tutorial https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/. However, be aware that it is using an older version of Hadoop so see the example in our repo for the updated version.
- Pydoop: a Python interface to Hadoop that allows you to write MapReduce applications in pure Python https://crs4.github.io/pydoop/