Apache Hadoop

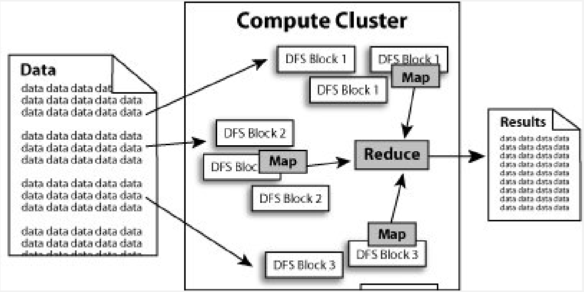
Amit Jain

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.

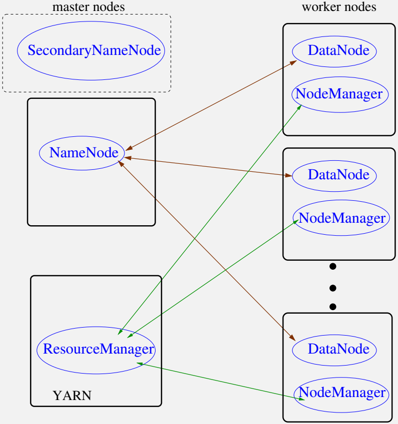
Hadoop Implementation

- 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.
- Hadoop programs can be developed using Eclipse (or other Java IDEs) on MS Windows or Linux platforms. To use MS Windows requires Cygwin package. MS Windows/Mac OSX can be used for development but not supported as a full production Hadoop cluster.
- Used by Facebook, Amazon, RackSpace, Twitter, EBay, LinkedIn, New York Times, E-Harmony (!) and Microsoft (via acquisition of Powerset). Most of Fortune 50 are using Hadoop.

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`. 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
```

- Navigate to the unpacked folder and then edit etc/hadoop/hadoop-env.sh and set JAVA_HOME to point to Java installation folder on your system. In the department labs, it is already set to `/usr/lib/jvm/java-11`
- To use Java 11 (Java 11 or Java 8 are required), add the following two lines to the end of your

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

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

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

Setting up Hadoop (2)

- 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 (3)

- 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
    ```

  - 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

Developing Hadoop MapReduce in Eclipse

- Create a Java project in Eclipse (or your IDE). Add at least the following three jar files as external jar files (found in the hadoop download) for the project:

```
share/hadoop/common/hadoop-common-3.3.6.jar
share/hadoop/mapreduce/hadoop-mapreduce-client-
    core-3.3.6.jar
share/hadoop/hdfs/lib/commons-cli-1.2.jar
```

- Develop the MapReduce application. Then generate a jar file using the *Export* menu. Make sure to specify the main class when exporting the jar file.

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. Create the jar file to run either using an IDE or manually (see the `README.md` file in the folder for instructions).
- 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.

Port Forwarding (aka Tunneling) to Access Hadoop Web Interface

- Use ssh port forwarding to enable access to Hadoop ports from a browser at home. We will need to forward ports 9870 and 8088. Let us see how to forward one port first.
- To forward one port, log in to `cscluster00.boisestate.edu` as follows:

```
ssh -Y -L 9870:localhost:9870 cscluster00.boisestate.edu
```

- Then point browser on your local system to `localhost:9870` and you will have access to the Hadoop web interface without physical presence in the lab or the slow speed of running a browser remotely.
- To forward multiple ports, use multiple `-L` options as follows:

```
ssh -Y -L 9870:localhost:9870 -L 8088:localhost:8088
    cscluster00.boisestate.edu
```

- To tunnel via onyx to cscluster00 from your machine (at home)!

```
ssh -L 9870:localhost:9870 onyx.boisestate.edu  ssh -L
    9870:localhost:9870 cscluster00
```

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.

Hadoop Map-Reduce Inputs and Outputs

- The Map/Reduce framework operates exclusively on $<key, value>$ pairs, that is, the framework views the input to the job as a set of $<key, value>$ pairs and produces a set of $<key, value>$ pairs as the output of the job, conceivably of different types.
- The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.
- The user needs to implement a Mapper class as well as a Reducer class. Optionally, the user can also write a Combiner class.

$$\text{(input)} < k1, v1 > \rightarrow \text{map} \rightarrow < k2, v2 > \rightarrow \text{combine} \rightarrow < k2, v2 >$$

$$\rightarrow \text{reduce} \rightarrow < k3, v3 > \text{(output)}$$

MapReduce API

```
public class MyMapper extends
    Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> { ... }

protected void map(KEYIN key,
                   VALUEIN value,
                   Mapper.Context context)
             throws IOException,
                   InterruptedException

public class MyReducer extends
    Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> { ... }

protected void reduce(KEYIN key,
                      Iterable<VALUEIN> values,
                      Reducer.Context context)
                throws IOException,
                      InterruptedException
```

WordCount example with Hadoop API

Problem: To count the number of occurrences of each word in a large collection of documents.

```
/**
 * Counts the words in each line.
 * For each line of input, break the line into words
 * and emit them as (word, 1).
 */
public static class TokenizerMapper
  extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context)
  throws IOException, InterruptedException {
    StringTokenizer itr =
        new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

See full code here: WordCount.java

WordCount Example with Hadoop API (contd.)

```
/**
 * A reducer class that just emits the sum of the input
   values.
 */
public static class IntSumReducer
  extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context)
    throws IOException, InterruptedException
    {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
 }
```

WordCount Example with Hadoop API (contd.)

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");

    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

Case Analysis Example

See full code here: CaseAnalysis.java

```
public class CaseAnalysis {
public static class Map extends Mapper<LongWritable, Text, Text,
    IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private final static IntWritable zero = new IntWritable(0);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  Context context)
  throws IOException, InterruptedException
  {
    String line = value.toString();

    for (int i = 0; i < line.length(); i++) {
      if (Character.isLowerCase(line.charAt(i))) {
        word.set(String.valueOf(line.charAt(i)).toUpperCase());
        context.write(word, zero);
      } else if (Character.isUpperCase(line.charAt(i))) {
        word.set(String.valueOf(line.charAt(i)));
        context.write(word, one);
      } else {
          word.set("other");
          context.write(word, one);
      }
    }
   }
}
```

Case Analysis Example (contd.)

```
public static class Reduce
  extends Reducer<Text, IntWritable, Text, Text> {
  private Text result = new Text();

  public void reduce(Text key, Iterable<IntWritable> values,
      Context context) throws IOException, InterruptedException
  {
    long total = 0;
    int upper = 0;

    for (IntWritable val: values) {
        upper += val.get();
        total++;
    }
    result.set(String.format("%16d %16d %16d %16.2f", total, upper,
            (total - upper), ((double) upper / total)));
    context.write(key, result);
  }
}
```

Case Analysis Example (contd.)

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).
    getRemainingArgs();

  if (otherArgs.length != 2) {
    System.err.println("Usage: hadoop jar caseanalysis.jar <in>
     <out>");
    System.exit(2);
  }

  Job job = new Job(conf, "case analysis");
  job.setJarByClass(CaseAnalysis.class);
  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Inverted Index Example

Given an input text, an inverted index program uses Mapreduce to produce an index of all the words in the text. For each word, the index has a list of all the files where the word appears. See full code here: InvertedIndex.java

```
public static class InvertedIndexMapper extends
    Mapper<LongWritable, Text, Text, Text>
{
  private final static Text word = new Text();
  private final static Text location = new Text();

  public void map(LongWritable key, Text val, Context context)
    throws IOException, InterruptedException
  {
    FileSplit fileSplit = (FileSplit) context.getInputSplit();
    String fileName = fileSplit.getPath().getName();
    location.set(fileName);

    String line = val.toString();
    StringTokenizer itr = new StringTokenizer(line.toLowerCase());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, location);
    }
  }
}
```

Inverted Index Example (contd.)

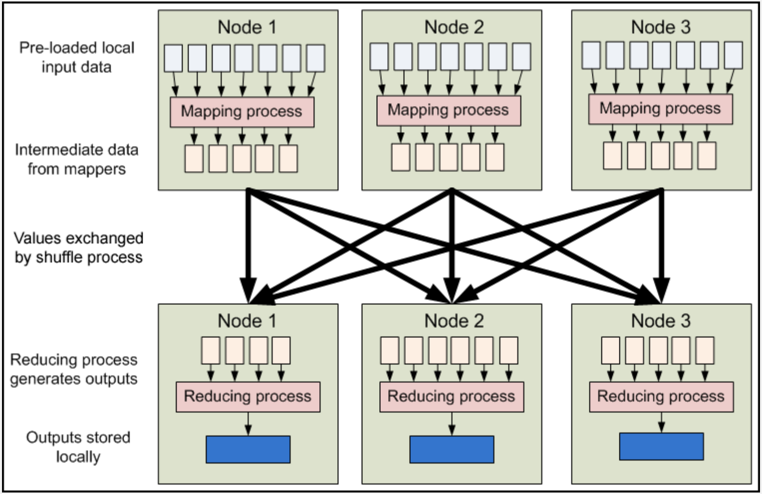
The reduce method is shown below.

```
public static class InvertedIndexReducer extends
                    Reducer<Text, Text, Text, Text>
{
    public void reduce(Text key, Iterable<Text> values, Context
    context)
    throws IOException, InterruptedException
    {
        boolean first = true;
        StringBuilder toReturn = new StringBuilder();
        while (values.hasNext()) {
            if (!first)
                toReturn.append(", ");
            first = false;
            toReturn.append(values.next().toString());
        }
        context.write(key, new Text(toReturn.toString()));
    }
}
```

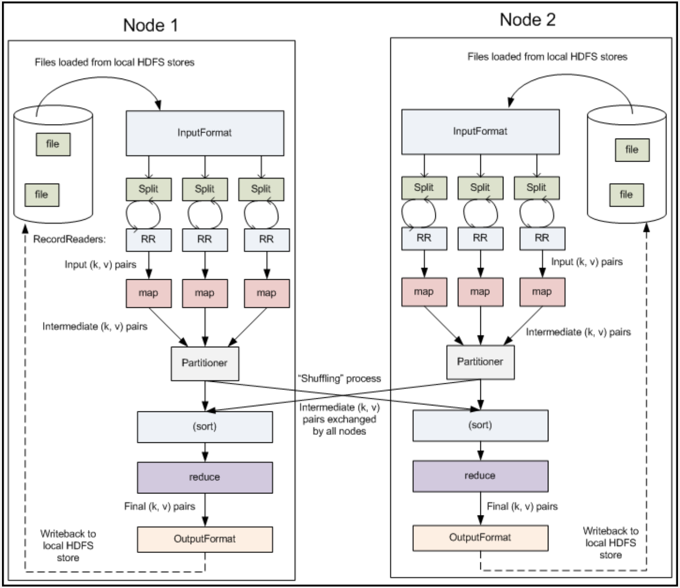
Inverted Index Example (contd)

```
public static void main(String[] args) throws IOException
{
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf,
                          args).getRemainingArgs();

  if (args.length < 2) {
    System.out
    println("Usage: InvertedIndex <input path> <output path>");
    System.exit(1);
  }
  Job job = new Job(conf, "InvertedIndex");
  job.setJarByClass(InvertedIndex.class);
  job.setMapperClass(InvertedIndexMapper.class);
  job.setReducerClass(InvertedIndexReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

MapReduce: High-Level Data Flow



MapReduce: Detailed Data Flow



Top-N Example

- Given a list of movies with the number of views, find the top 10 movies by number of views (assume that the number of views is unique)
- See example code: Hadoop/myExamples/top-n-movies-v1
- Illustrates the setup/cleanup technique where we can take only one action for a map and reduce.
- What if we have multiple movies with the same number of views?
- See example code: Hadoop/myExamples/top-n-movies-v2

Fully Distributed Hadoop

Normally, Hadoop runs on a dedicated cluster. In that case, the setup is a bit more complex than for the pseudo-distributed case.

- Specify hostname or IP address of the master server in the values for fs.defaultFS in core-site.xml and and mapred.job.tracker in mapred-site.xml file. These are specified as host:port pairs. The default ports are 9000 and 9001.
- Specify directories for dfs.name.dir and dfs.data.dir and dfs.replication in conf/hdfs-site.xml. These are used to hold distributed file system data on the master node and worker nodes respectively. Note that `dfs.data.dir` may contain a space- or comma-separated list of directory names, so that data may be stored on multiple devices.
- Specify mapred.system.dir and mapred.local.dir in conf/hadoop-site.xml. The system directory must be accessible by server and clients. The local directory determines where temporary MapReduce data is written. It also may be a list of directories.

Fully Distributed Hadoop (contd.)

- Specify mapred.map.tasks (default value: 2) and mapred.reduce.tasks (default value: 1) in conf/mapred-site.xml.This is suitable for local or pseudo-distributed mode only. Choosing the right number of map and reduce tasks has significant effects on performance.
- Default Java memory size is 1000MB. This can be changed in `conf/hadoop-env.sh`. This is related to the parameters discussed above.
- List all worker host names or IP addresses in your conf/workers file, one per line. List name of master nodes (can be more than one) in conf/masters.

https://github.com/BoiseState/CS535-resources/blob/master/examples/hadoop/local-scripts/templates/core-site.xml.distributed

Sample Config Files

- Sample core-site.xml file.
- Sample hdfs-site.xml file.
- Sample https://github.com/BoiseState/CS535-resources/blob/master/examples/hadoop/local-scripts/templates/core-site.xml.distributedmapred-site.xml file.
- Sample `hadoop-env.sh` file. Only need two things to be defined here.

```
# Set Hadoop-specific environment variables here.
# The java implementation to use.  Required.
export JAVA_HOME=/usr/java/default
...
# The maximum amount of heap to use, in MB. Default is 1000.
# export HADOOP_HEAPSIZE=2000

...
```

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/