Apache Hadoop

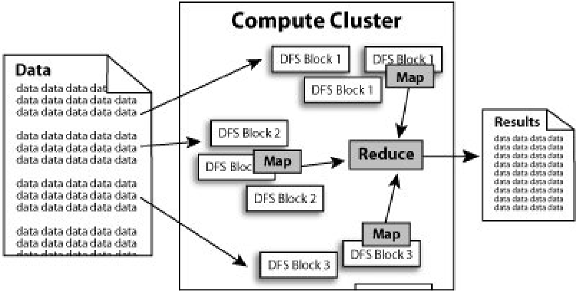
Amit Jain

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

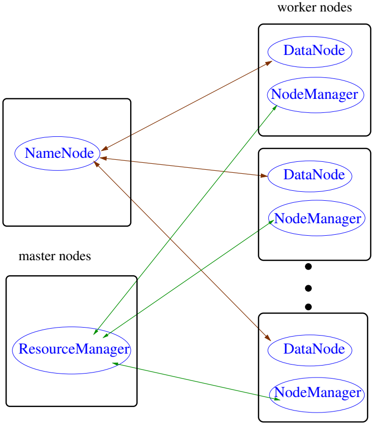
- Hadoop implements a MapReduce framework, using the Hadoop Distributed File System (HDFS).
- MapReduce divides applications into many small blocks of work. MapReduce can then process the data where it is located.
- HDFS creates multiple replicas of data blocks for reliability, placing them on compute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

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/NetBeans 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.

Setting up Hadoop (1)

- Create a folder named hadoop-install in your home directory.
- Download a stable version (currently 2.9.2) of Hadoop from http://hadoop.apache.org. The downloaded file should be hadoop-2.9.2.tar.gz. Move it to the hadoop-install folder from your downloads folder.

```
mv tar xzvf hadoop-2.9.2.tar.gz ~/hadoop-install/
cd ~/hadoop-install/
```

Setting up Hadoop (1)

- Create a folder named hadoop-install in your home directory.
- Download a stable version (currently 2.9.2) of Hadoop from http://hadoop.apache.org. The downloaded file should be hadoop-2.9.2.tar.gz. Move it to the hadoop-install folder from your downloads folder.

```
mv tar xzvf hadoop-2.9.2.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-2.9.2 in the hadoop-install folder.

```
tar xzvf hadoop-2.9.2.tar.gz
```

Setting up Hadoop (1)

- Create a folder named hadoop-install in your home directory.
- Download a stable version (currently 2.9.2) of Hadoop from http://hadoop.apache.org. The downloaded file should be hadoop-2.9.2.tar.gz. Move it to the hadoop-install folder from your downloads folder.

```
mv tar xzvf hadoop-2.9.2.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-2.9.2 in the hadoop-install folder.

```
tar xzvf hadoop-2.9.2.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 onyx lab, it is already set to /usr/lib/jvm/java-11

Setting up Hadoop (1)

- Create a folder named hadoop-install in your home directory.
- Download a stable version (currently 2.9.2) of Hadoop from http://hadoop.apache.org. The downloaded file should be hadoop-2.9.2.tar.gz. Move it to the hadoop-install folder from your downloads folder.

```
mv tar xzvf hadoop-2.9.2.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-2.9.2 in the hadoop-install folder.

```
tar xzvf hadoop-2.9.2.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 onyx lab, it is already set to /usr/lib/jvm/java-11
- Make a symbolic link (shortcut) to point to it as shown below.

```
ln -s hadoop-2.9.2 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
```

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
```

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 sbin]$ which start-dfs.sh
~/hadoop-install/hadoop/sbin/start-dfs.sh
```

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 sbin]$ which start-dfs.sh
~/hadoop-install/hadoop/sbin/start-dfs.sh
```

- Make sure that you can run ssh localhost date command without a password. If you cannot, then set it up as follows:

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

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 sbin]$ which start-dfs.sh
~/hadoop-install/hadoop/sbin/start-dfs.sh
```

- Make sure that you can run ssh localhost date command without a password. If you cannot, then set it up as follows:

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.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-2.9.2` folder).

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

bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
    examples-2.9.2.jar grep input output 'dfs[a-z.]+'

cat output/*
```

Standalone Mode

- Hadoop comes ready to run in standalone mode out of the box. Try the following to test it (from the `hadoop-2.9.2` folder).

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

bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
    examples-2.9.2.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.

```
[amit@kohinoor templates]$ cat core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
</configuration>

[amit@kohinoor templates]$ cat  hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
</configuration>

[amit@kohinoor templates]$ cat mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
</configuration>
```

Developing Hadoop MapReduce in Eclipse

- Create a Java project in Eclipse. 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-2.9.2.jar
share/hadoop/mapreduce/hadoop-mapreduce-client-
    core-2.9.2.jar
share/hadoop/hdfs/lib/commons-cli-1.2.jar
```

- Develop the MapReduce application. Then generate a jar file using the *Export* menu.

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 slaves file that lists the names of all the hosts in the cluster. The default slaves file is conf/slaves it should contain just one hostname: `localhost`.

Pseudo-Distributed Mode Config Files

- To run everything in one node, edit three config files so that they contain the configuration shown below:

```
--> etc/hadoop/core-site.xml
<configuration>
<property> <name>fs.defaultFS</name>
           <value>hdfs://localhost:9000</value> </property>
</configuration>

--> etc/hadoop/hdfs-site.xml
<configuration>
<property> <name>dfs.replication</name>
           <value>1</value> </property>
</configuration>

--> etc/hadoop/mapred-site.xml
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>

--> etc/hadoop/yarn-site.xml
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

Pseudo-Distributed Mode (1)

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

  ```
  hdfs namenode -format
  ```

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
  ```

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

  ```
  hdfs dfsadmin -report
  ```

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 elephant 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.

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
```

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
```

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
  ```

- Point your web browser to localhost:50070 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. You would normally leave this window open to see your jobs running.

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).

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
```

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
  ```

- Now run the pseudo-distributed mapreduce job.

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

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
```

- Now run the pseudo-distributed mapreduce job.

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

- 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, I recommend stopping them so if the machine gets turned off, you don't run into issues.

  ```
  stop-yarn.sh
  ```

  ```
  stop-dfs.sh
  ```

- 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 to Access Hadoop Web Interface

Use ssh port forwarding to enable access to Hadoop ports from a browser at home. Log in to `onyx.boisestate.edu` as follows:

```
ssh -Y -L 50070:onyx.boisestate.edu:50070
onyx.boisestate.edu
```

Then point browser on your local system to `localhost:50070` 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.

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.

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.

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.

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

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

Map/Reduce API

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

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

Map/Reduce 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 new 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 new 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 new 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.)

```java
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.)

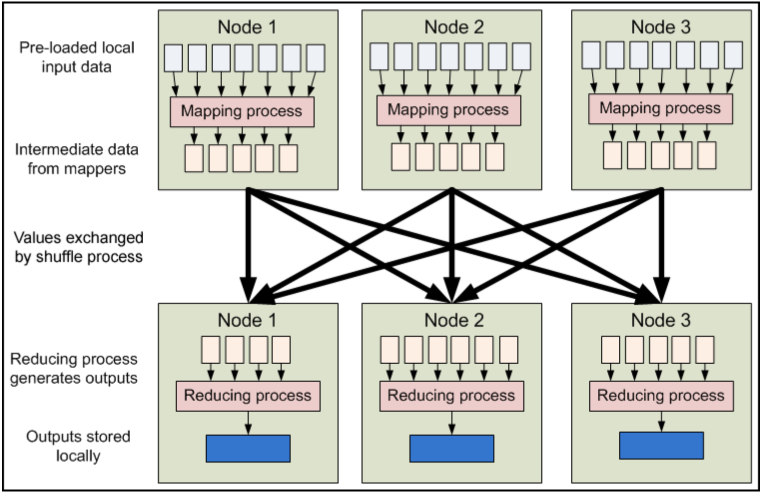
The reduce method is shown below.

```
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()));
    }
}
```

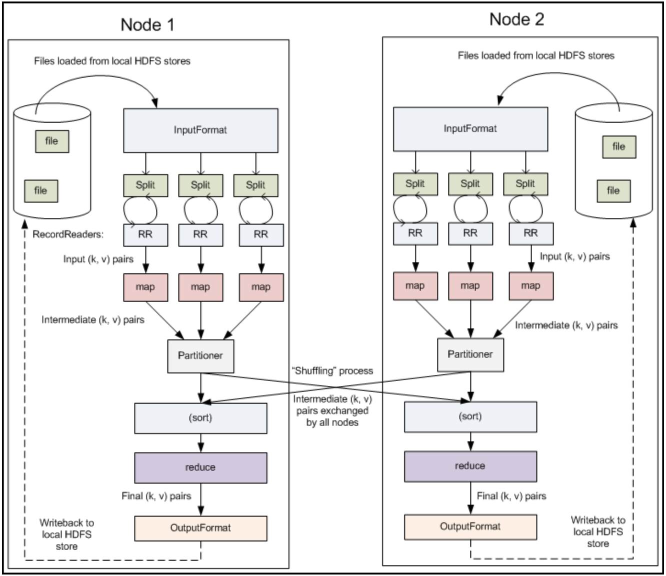
Inverted Index Example (contd)

```java
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



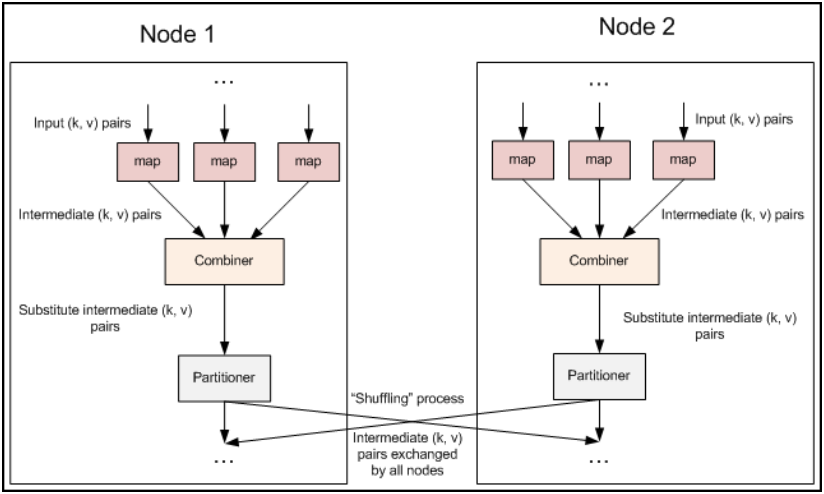
MapReduce: Detailed Data Flow



MapReduce Optimizations

- Overlap of maps, shuffle, and sort
- Mapper locality
  - Schedule mappers close to the data.
- Combiner
  - Mappers may generate duplicate keys
  - Side-effect free reducer can be run on mapper node
  - Minimizes data size before transfer
  - Reducer is still run
- Speculative execution to help with load-balancing
  - Some nodes may be slower
  - Run duplicate task on another node, take first answer as correct and abandon other duplicate tasks
  - Only done as we start getting toward the end of the tasks

Using a Combiner to Reduce Network Traffic



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.default.name 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.

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.default.name 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 slave 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.

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.default.name 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 slave 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.

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. See Chapter 9 in the Hadoop book by Tom White for further discussion.

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. See Chapter 9 in the Hadoop book by Tom White for further discussion.
- List all slave host names or IP addresses in your conf/slaves file, one per line. List name of master node in conf/master.

Sample Config Files

- Sample core-site.xml file.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>fs.default.name</name> <value>hdfs://node00:9000</value>
  <description>The name of the default filesystem.</description>
</property>
</configuration>
```

- Sample hdfs-site.xml file.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property> <name>dfs.replication</name> <value>1</value> </property>
<property>
  <name>dfs.name.dir</name><value>/tmp/hadoop-amit/hdfs/name</value>
</property>
<property>
  <name>dfs.data.dir</name> <value>/tmp/hadoop-amit/hdfs/data</value>
</property>
</configuration>
```

Sample Config Files (contd.)

- Sample mapred-site.xml file.

```
<?xml version="1.0"?> <?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>

<configuration>
<property>
  <name>mapred.job.tracker</name> <value>hdfs://node00:9001</value>
</property>
<property>
  <name>mapred.system.dir</name> <value>/tmp/hadoop-amit/mapred/system</value>
</property>
<property>
  <name>mapred.local.dir</name> <value>/tmp/hadoop-amit/mapred/local</value>
</property>
<property>
   <name>mapred.map.tasks.maximum</name> <value>17</value>
   <description>
      The maximum number of map tasks which are run simultaneously
      on a given TaskTracker individually. This should be a prime
      number larger than multiple of the number of slave hosts.
   </description>
</property>
<property>
   <name>mapred.reduce.tasks.maximum</name> <value>16</value>
</property>
<property>
   <name>mapred.reduce.tasks</name> <value>7</value>
</property>
</configuration>
```

Sample Config Files (contd.)

- 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

...
```

Running multiple copies of Hadoop on Onyx Cluster

- Normally only one copy of Hadoop runs on a cluster. In our lab setup, we want to be able to run multiple copies of Hadoop where the namenode (`node00`) is overloaded but each user has unique datanodes that they schedule via PBS.
- In order to do this, each user needs unique ports for the following:

property	default value	config file
fs.default.name	`hdfs://node00:9000`	core-site.xml
mapred.job.tracker	`hdfs://node00:9001`	mapred-site.xml
mapred.job.tracker.http.address	`0.0.0.0:50030`	mapred-site.xml
dfs.datanode.address	`0.0.0.0:50010`	hdfs-site.xml
dfs.datanode.ipc.address	`0.0.0.0:50020`	hdfs-site.xml
dfs.http.address	`0.0.0.0:50070`	hdfs-site.xml
dfs.datanode.http..address	`0.0.0.0:50075`	hdfs-site.xml
dfs.secondary.http.address	`0.0.0.0:50090`	hdfs-site.xml

- We have provided a script cluster-pickports.sh that lets the user pick a base port 60000 and then it sets the three config files with numbers. So with base port of 60000, the eight ports get set to 60000, 60001, 60010, 60020, 60030, 60070, 60075 and 60090. The script is available via the class examples code repository at: lab/Hadoop/local-scripts

Reference: How to use Hadoop in the lab (part 1)

- Setup Hadoop as shown earlier in the notes.

Reference: How to use Hadoop in the lab (part 1)

- Setup Hadoop as shown earlier in the notes.
- Then setup the configuration files for distributed mode of operation and modify configuration file based on your group's assigned port range (check the teams page on the class website) using the following script.

```
cd ~/hadoop-install/local-scripts
setmod.sh distributed
cluster-pickports.sh <base-port>
```

Reference: How to use Hadoop in the lab (part 1)

- Setup Hadoop as shown earlier in the notes.
- Then setup the configuration files for distributed mode of operation and modify configuration file based on your group's assigned port range (check the teams page on the class website) using the following script.

```
cd ~/hadoop-install/local-scripts
setmod.sh distributed
cluster-pickports.sh <base-port>
```

- Running Hadoop in pseudo-distributed mode in the lab isn't recommended because of potential port conflicts with the real distributed Hadoop.

Reference: How to use Hadoop in the lab (part 2)

- Login to onyx via `ssh` to create the first console.
- Open a second window/console on onyx to grab some nodes from PBS.
- In the original window on onyx, create the cluster using the provided script. Then check on the status of the cluster.

```
cd ~/hadoop-install/local-scripts
create-hadoop-cluster.sh
cd ../hadoop

...wait for a few seconds...

bin/hdfs dfsadmin -report
```

- After you are done running jobs and you have copied the relevant output from HDFS to your local file system, destroy the Hadoop distributed file system as follows:

```
cd ~/hadoop-install/local-scripts
cluster-remove.sh
```

- Finally, exit from the PBS shell in the second console to release the nodes.

References

- *MapReduce: Simplified Data Processing on Large Clusters* by Jeffrey Dean and Sanjay Ghemawat, Google Inc. Appeared in: OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
- *Hadoop: An open source implementation of MapReduce*. The main website: http://hadoop.apache.org/.
- *Hadoop: The Definitive Guide*. Tom White, June 2012, O'Reilly.