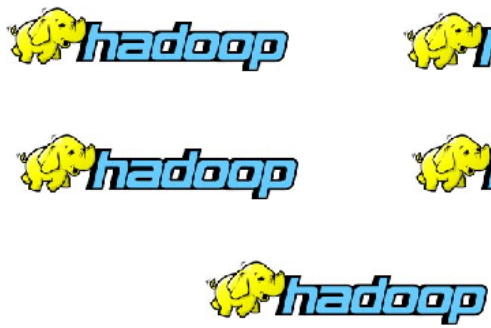
MapReduce for Parallel Computing

Amit Jain



Big Data, Big Disks, Cheap Computers

- *"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."* Rear Admiral Grace Hopper.

Big Data, Big Disks, Cheap Computers

- *"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."* Rear Admiral Grace Hopper.
- *"More data usually beats better algorithms."* Anand Rajaraman.

Big Data, Big Disks, Cheap Computers

- *"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."* Rear Admiral Grace Hopper.
- *"More data usually beats better algorithms."* Anand Rajaraman.
- *"The good news is that Big Data is here. The bad news is that we are struggling to store and analyze it."* Tom White.

Units and Units

Check out http://en.wikipedia.org/wiki/Petabyte

Introduction

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.
- Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.
- Allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Introduced by Google. Used internally for all major computations on around 1m servers! of data. Amazon is leasing servers to run map reduce computations (EC2 and S3 programs). Microsoft has developed Dryad (a super set of Map-Reduce). Balihoo used it for both production and R&D.

MapReduce Programming Model

- Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the Reduce function.

MapReduce Programming Model

- Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the Reduce function.
- The Reduce function, also written by the user, accepts an intermediate key K and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

MapReduce Programming Model

- Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the Reduce function.
- The Reduce function, also written by the user, accepts an intermediate key K and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.
- MapReduce Specification Object. Contains names of input/output files and optional tuning parameters. The user then invokes the MapReduce function, passing it the specification object. The user's code is linked together with the MapReduce library.

A MapReduce Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents.

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
     EmitIntermediate(w, "1");

reduce(String key, Iterable values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
     result += ParseInt(v);
  Emit(key, AsString(result));
```

MapReduce Exercise

- **Case Analysis or Capitalization Probability**: In a collection of text documents, find the percentage capitalization for each letter of the alphabet.

MapReduce Examples

- **Distributed Grep**: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.

MapReduce Examples

- **Distributed Grep**: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency**: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

MapReduce Examples

- **Distributed Grep**: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency**: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.
- **Reverse Web-Link Graph**: The map function outputs <target, source> pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>

More MapReduce Examples

- **Inverted Index**: The map function parses each document, and emits a sequence of <word, document ID> pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

More MapReduce Examples

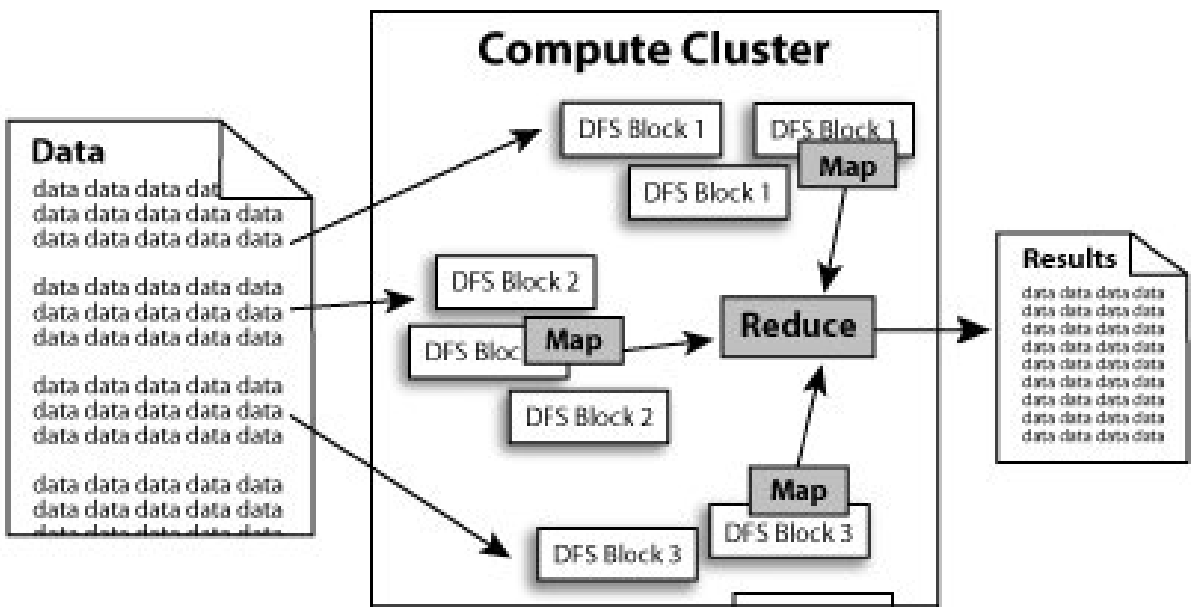
- **Inverted Index**: The map function parses each document, and emits a sequence of <word, document ID> pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.
- **Distributed Sort**: The map function extracts the key from each record, and emits a <key, record> pair. The reduce function emits all pairs unchanged. This computation depends on the partitioning and ordering facilities that are provided in a MapReduce implementation.

Hadoop is a software platform that lets one easily write and run applications that process vast amounts of data. 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 extremely efficient.
- Reliable: Hadoop automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

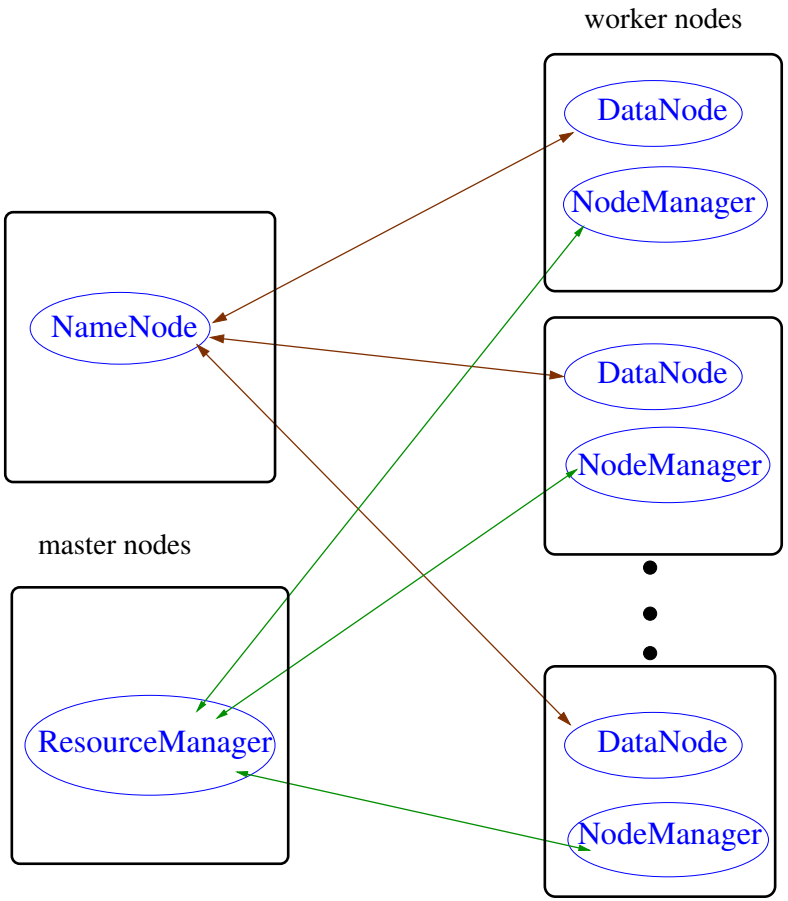
Hadoop Implementation

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



Hadoop Servers/Daemons

The Hadoop Distributed File Systems (HDFS) is implemented by a Name Node server on a master node and a Data Node server on each data node. The MapReduce framework is implemented by a Resource Manager tracker on a master node and a Node Manager on each worker node.



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 grew out of the Nutch Web Crawler project.

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 grew out of the Nutch Web Crawler project.
- 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 are recommended for development but not 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 grew out of the Nutch Web Crawler project.
- 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 are recommended for development but not as a full production Hadoop cluster.
- Used by Yahoo, Facebook, Amazon, RackSpace, Twitter, EBay, LinkedIn, New York Times, E-Harmony (!) and Microsoft (via acquisition of Powerset). Several cloud consulting companies like Cloudera.

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.)

```
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.)

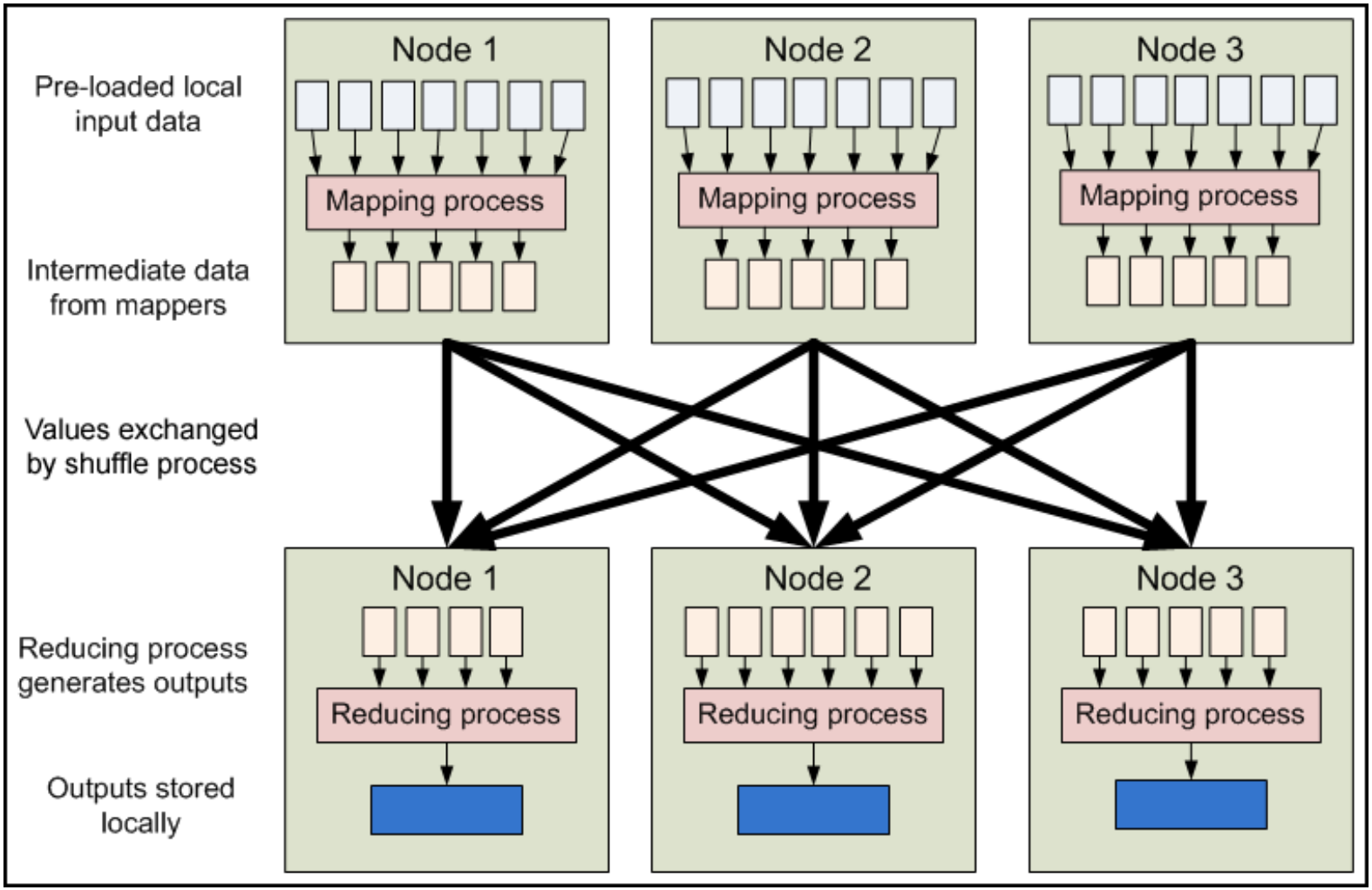
The reduce method is shown below.

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

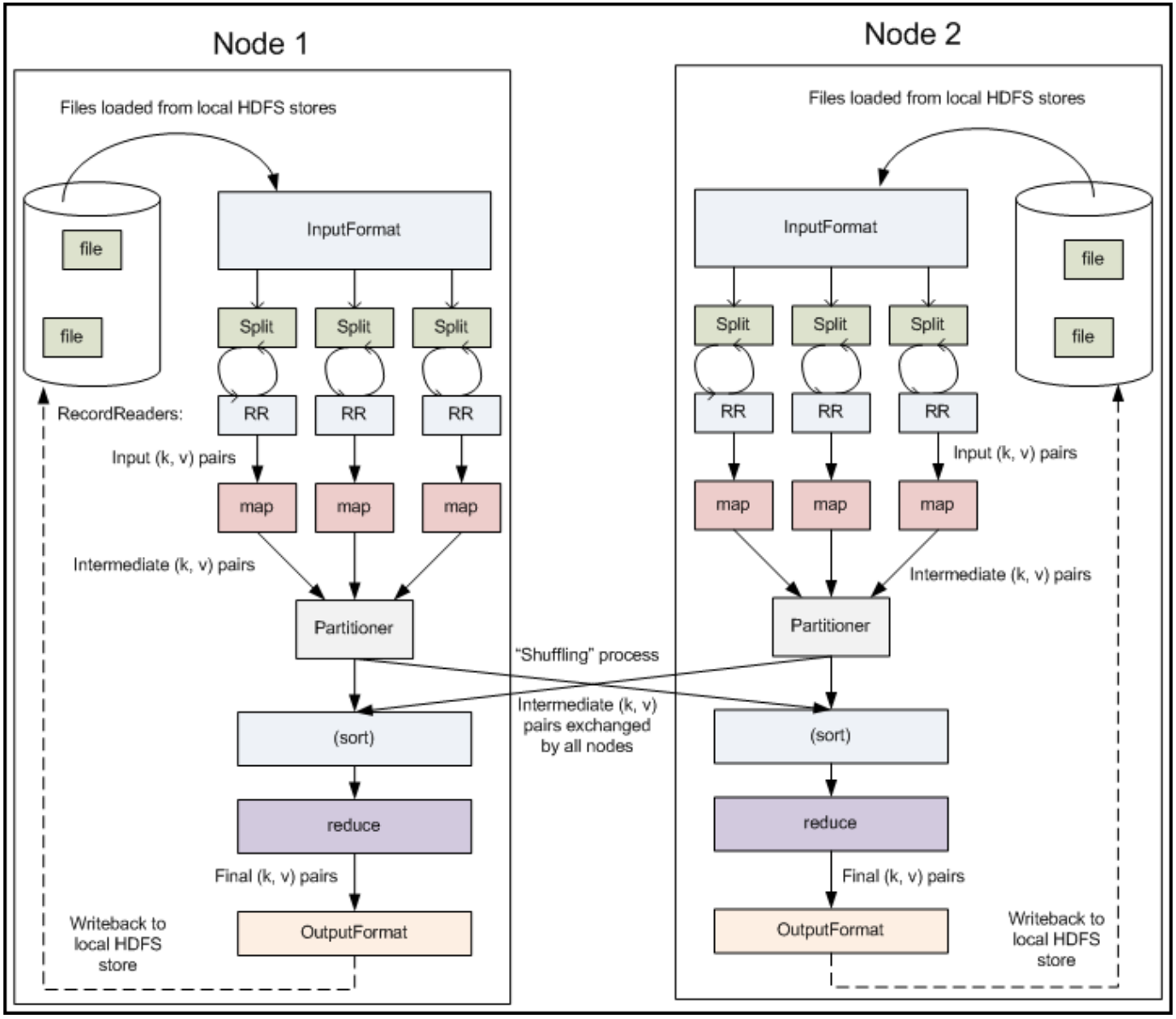
Inverted Index Example (contd)

```
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



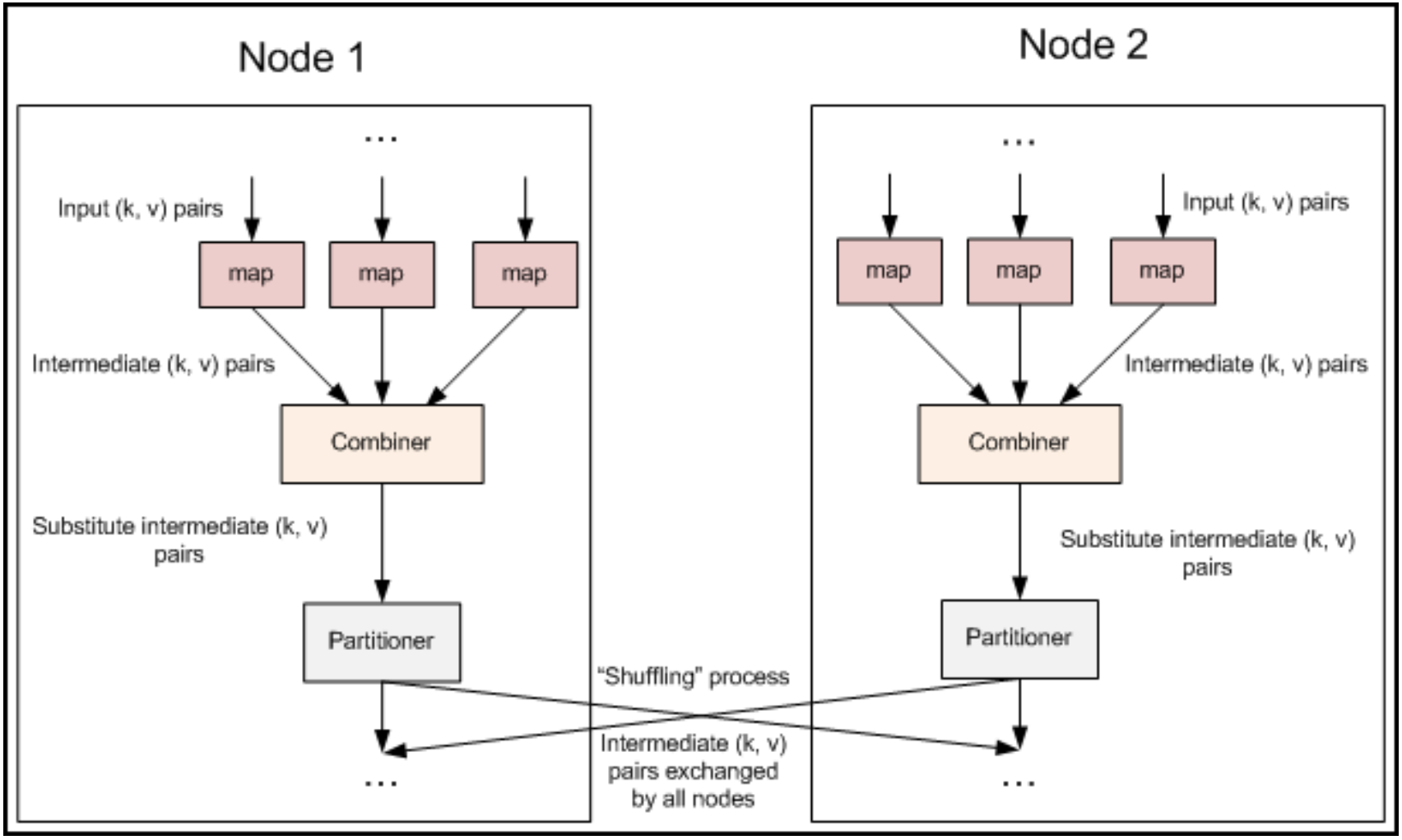
MapReduce: Detailed Data Flow



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



Setting up Hadoop

- Download a stable version (currently 2.7.3) of Hadoop from `http://hadoop.apache.org`.
- Unpack the tarball that you downloaded in previous step.

```
tar xzvf hadoop-2.7.3.tar.gz
```

Setting up Hadoop

- Download a stable version (currently 2.7.3) of Hadoop from `http://hadoop.apache.org`.
- Unpack the tarball that you downloaded in previous step.

  tar xzvf hadoop-2.7.3.tar.gz
- Edit conf/hadoop-env.sh and at least set JAVA_HOME to point to Java installation folder on your system. You will need Java version 1.7 or higher. In the lab, java is installed in `/usr/java/default`.

Setting up Hadoop

- Download a stable version (currently 2.7.3) of Hadoop from http://hadoop.apache.org.
- Unpack the tarball that you downloaded in previous step.

  tar xzvf hadoop-2.7.3.tar.gz
- Edit conf/hadoop-env.sh and at least set JAVA_HOME to point to Java installation folder on your system. You will need Java version 1.7 or higher. In the lab, java is installed in `/usr/java/default`.
- Now run bin/hadoop to test Java setup. You should get output similar to shown below.

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

Setting up Hadoop

- Download a stable version (currently 2.7.3) of Hadoop from http://hadoop.apache.org.
- Unpack the tarball that you downloaded in previous step.

  tar xzvf hadoop-2.7.3.tar.gz
- Edit conf/hadoop-env.sh and at least set JAVA_HOME to point to Java installation folder on your system. You will need Java version 1.7 or higher. In the lab, java is installed in `/usr/java/default`.
- Now run bin/hadoop to test Java setup. You should get output similar to shown below.

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

- Make sure that you can run ssh localhost 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
```

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.

```
mkdir input
cp etc/hadoop/*.xml input
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
    examples-2.7.3.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.

```
mkdir input
cp etc/hadoop/*.xml input
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
    examples-2.7.3.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@dslamit templates]$ cat core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
</configuration>

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

[amit@dslamit 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-2.7.3.jar
share/hadoop/mapreduce/hadoop-mapreduce-client-
    core-2.7.3.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.default.name</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

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh
- Create user directories for your login name
  bin/hdfs dfs -mkdir /user
  bin/hdfs dfs -mkdir /user/amit

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh
- Create user directories for your login name
  bin/hdfs dfs -mkdir /user
  bin/hdfs dfs -mkdir /user/amit
- Put input files into the Distributed file system.
  bin/hdfs dfs -put input input

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh
- Create user directories for your login name
  bin/hdfs dfs -mkdir /user
  bin/hdfs dfs -mkdir /user/amit
- Put input files into the Distributed file system.
  bin/hdfs dfs -put input input
- Start ResourceManager daemon and NodeManager daemon:
  sbin/start-yarn.sh
- Now run the distributed job and copy output back to local file system.
  bin/hadoop jar hadoop-*-examples.jar wordcount input output
  bin/hadoop dfs -get output output

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh
- Create user directories for your login name
  bin/hdfs dfs -mkdir /user
  bin/hdfs dfs -mkdir /user/amit
- Put input files into the Distributed file system.
  bin/hdfs dfs -put input input
- Start ResourceManager daemon and NodeManager daemon:
  sbin/start-yarn.sh
- Now run the distributed job and copy output back to local file system.
  bin/hadoop jar hadoop-*-examples.jar wordcount input output
  bin/hadoop dfs -get output output
- Point your web browser to localhost:8088 to watch the Hadoop Resource manager and to localhost:50070 to be able to browse the Hadoop DFS and get its status.

Pseudo-Distributed Mode

- Now create a new Hadoop distributed file system (HDFS) with the command:
  bin/hdfs namenode -format
- Start the Hadoop daemons.
  sbin/start-dfs.sh
- Create user directories for your login name
  bin/hdfs dfs -mkdir /user
  bin/hdfs dfs -mkdir /user/amit
- Put input files into the Distributed file system.
  bin/hdfs dfs -put input input
- Start ResourceManager daemon and NodeManager daemon:
  sbin/start-yarn.sh
- Now run the distributed job and copy output back to local file system.
  bin/hadoop jar hadoop-*-examples.jar wordcount input output
  bin/hadoop dfs -get output output
- Point your web browser to localhost:8088 to watch the Hadoop Resource manager and to localhost:50070 to be able to browse the Hadoop DFS and get its status.
- When you are done, stop the Hadoop daemons as follows.
  bin/stop-yarn.sh
  bin/stop-dfs.sh

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` as follows:

```
ssh -Y -L 50070:onyx:50070 onyx
```

Then point browser on your local system to `localhost:50030` 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.

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

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.
- *MapReduce tutorial at Yahoo*. http://developer.yahoo.com/hadoop/tutorial/
- *Data Clustering using MapReduce* by Makho Ngazimbi (supervised by Amit Jain). Masters in Computer Science project report, Boise State University, 2009.
- *Hadoop and Hive as Scalable alternatives to RDBMS: A Case Study* by Marissa Hollingsworth (supervised by Amit Jain). Masters in Computer Science project report, Boise State University, 2012