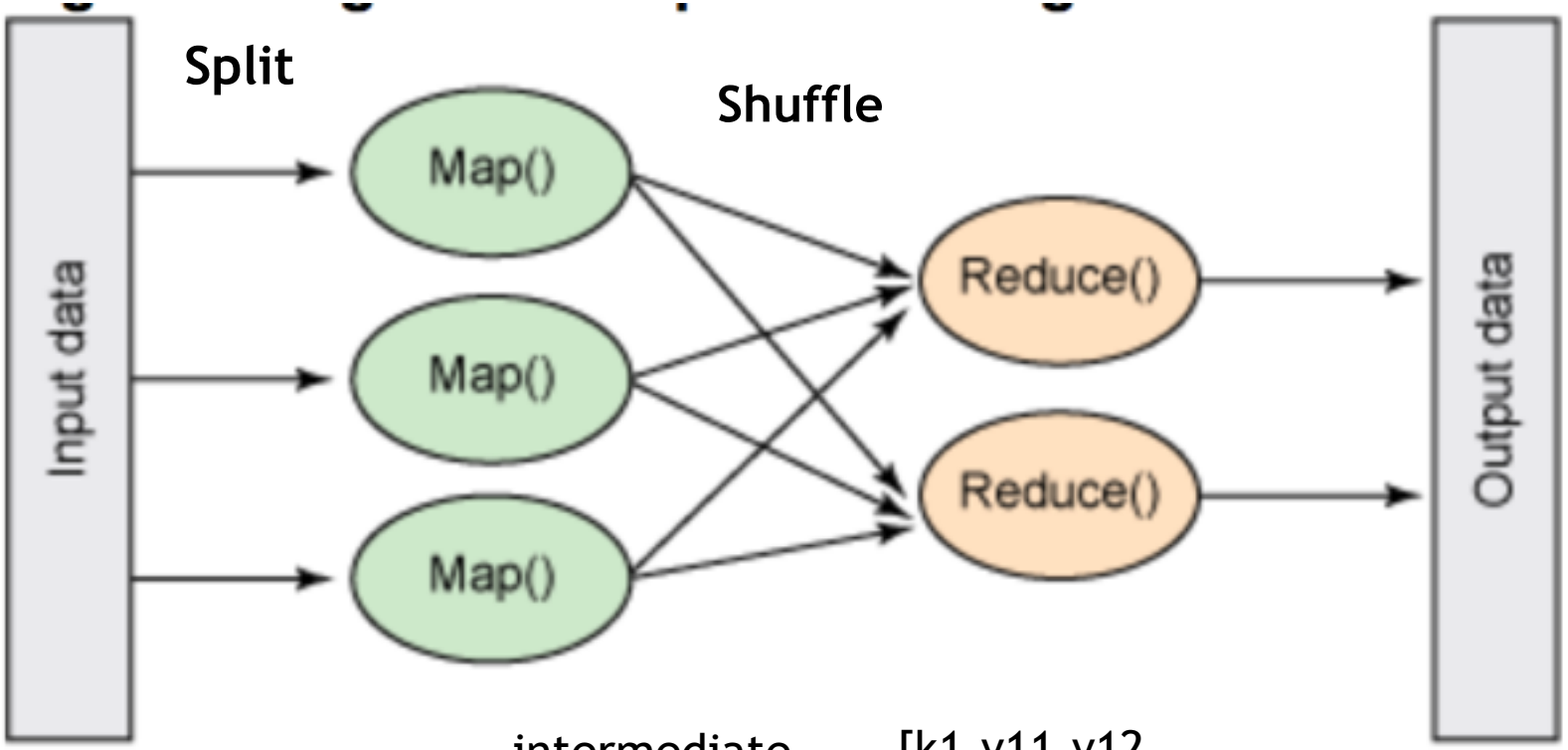
MapReduce: Programming

Spring 2015,
X. Zhang
Fordham Univ.

Outline

- Review and demo
  - Homework 1
  - MapReduce paradigm: hadoop streaming
  - Behind the scene: Hadoop daemons
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop configuration and hadoop command
- Towards developing a MapReduce Program
  - Tool: maven
  - Java MapReduce API
  - Unit testing

MapReduce Programming Model



Input: a set of [key,value] pairs

intermediate [key,value] pairs

[k1,v11,v12, …]
[k2,v21,v22, …]
…

Output: a set of [key,value] pairs

Recall: Homework 1

- In homework 1, both programs
  - Reads from standard input (usually keyboard), writes to standard output (usually terminal)
- Why not read from data file directly?
  - We can easily redirect input to a file
    - java Filter < ncdc/data
  - Or redirect output to a file
    - java Filter > ncdc/data > filtered_data

…

```
public class Filter {


     public static void main(String[] args) {
             BufferedReader reader = new BufferedReader(new
             InputStreamReader(System.in));
             String s = null;

             try{
                while ( (s = reader.readLine()) != null ){
                     String year = s.substring(15, 19);
                     String temperature = s.substring(87, 92);
                     int tempInt = Integer.parseInt(temperature);

                     System.out.println(year + " " + temperature);
               }
             }catch(IOException e ){
                        e.printStackTrace();
          }

     }

}
```

…

```
public class Max {

        public static void main(String[] args) {
                int year, temp;
                Scanner s = new Scanner (System.in);

                Map max = new HashMap();
                while  (s.hasNextInt()){
                      year = s.nextInt();
                      temp = s.nextInt();

                      if( max.containsKey(year)){
                             if( temp > (Integer)max.get(year)){
                                   max.put(year, temp);
                              }
                      }else{
                                 max.put(year, temp);
                      }
                }

               System.out.println(max);
        }

}
```

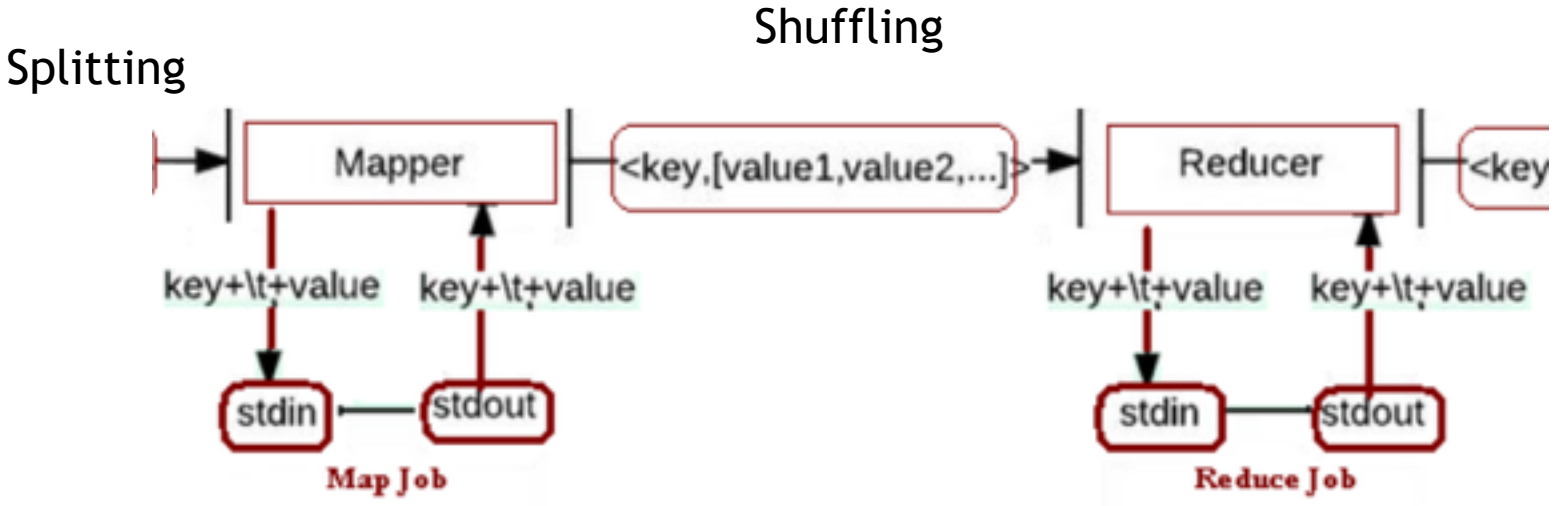
```
…

public class TempStat {

        public static void main(String[] args) {
                BufferedReader reader = new BufferedReader(
                  new InputStreamReader(System.in));
                String s = null;
                Map max = new HashMap();
                try{
                        while ( (s = reader.readLine()) != null ){
                                String year = s.substring(15, 19);
                                String temperature = s.substring(87, 92);
                                int tempInt = Integer.parseInt(temperature);

                                if( max.containsKey(year)){
                                        if( tempInt > (Integer)max.get(year)){
                                                max.put(year, tempInt);
                                        }
                                }else{
                                        max.put(year, tempInt);
                                }

                        }
                }catch(IOException e ){
                        e.printStackTrace();
                }

                System.out.println(max);
        }

}
```

Unix Command Pipeline

- So to find annual maximum temperature:
  - java Filter > ncdc/data > filtered_data
  - java Max < filtered_data > annual_max
- Better yet, we can avoid writing/reading filtered_data (a temporary file)
  - java Filter < ncdc/data | java Max > annual_max
- Similar to Map & Reduce phases!
  - Can we run these two programs in MapReduce Framework?
  - Answer: Hadoop Streaming API

Hadoop Streaming API

- A **generic API for MapReduce framework**
  - Mappers/Reducers can be written in any language, or some Unix commands
- **Mappers/Reducers act as "filters":** receive input and output on stdin and stdout
  - For text processing: each <key,value> pair takes a line,
  - Key/value separated by 'tab' character.
  - Mapper/Reducer reads each line (<key,value> pair) from stdin, processes it, and writes to stdout a line (<key,value> pair).



Hadoop Streaming API demo

- Using homework1's two programs
- Using unix commands and Max Program
- Can we simplify Max program, when used with MapReduce Streaming?

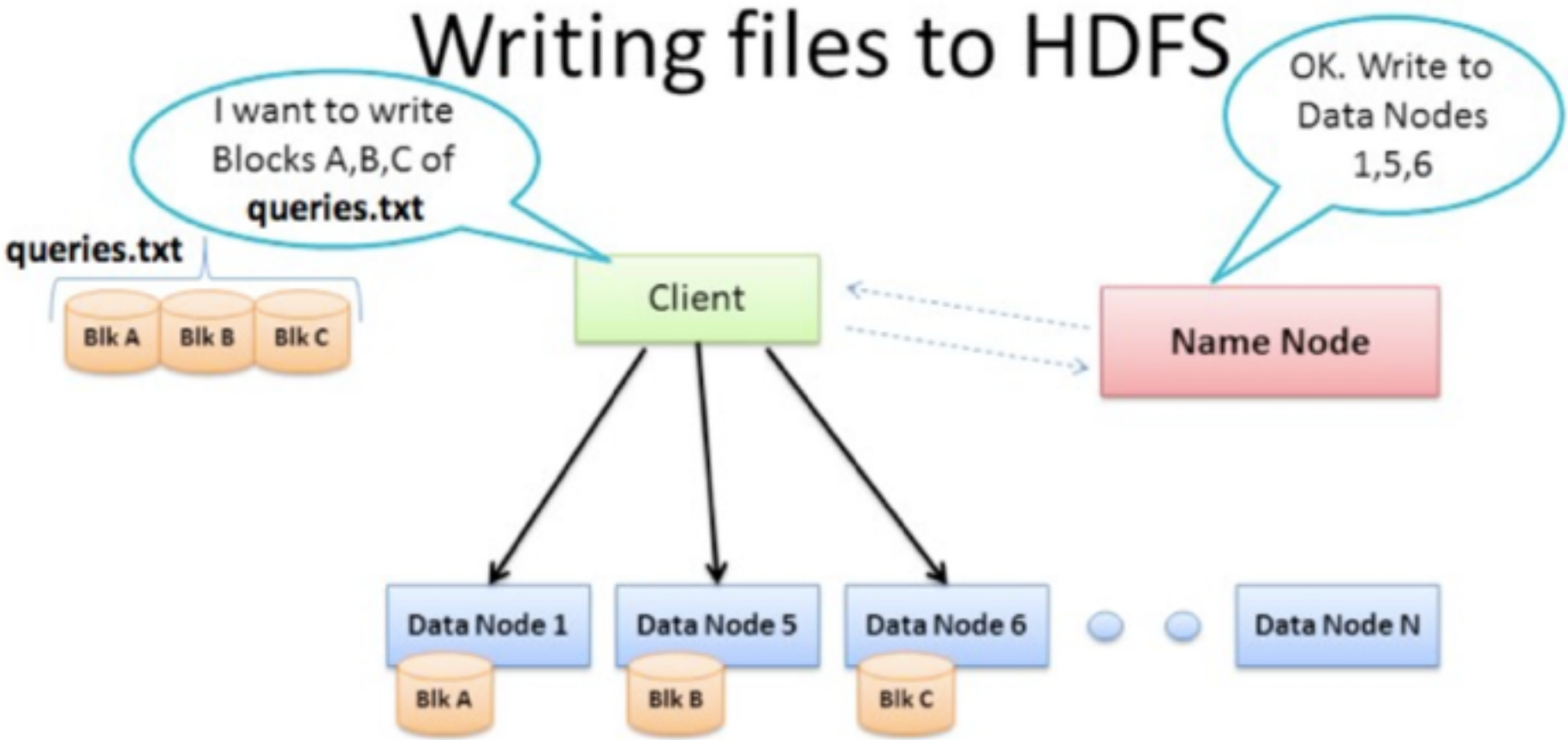
Outline

- Review and demo
  - Homework 1
  - MapReduce paradigm: hadoop streaming
  - Behind the scene: **Hadoop daemons**
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop configuration and hadoop command
- Towards developing a MapReduce Program
  - Tool: maven
  - Java MapReduce API
  - Unit testing

Hadoop Daemons

- Hadoop (HDFS and MapReduce) is a distributed system
  - Distributed file system
  - Support running MapReduce program in distributed and parallel fashion
    - Automatic input splitting, shuffling …
  - Provide fault tolerances, load balances, …
- To suppose these, several **Hadoop** Deamons (**processes** running in background)
  - HDFS: Namenode, datanode; MapReduce: jobtracker, resource manager, node manager …
  - These daemons communicates with each other via RPC (Remote Procedure Call) over SSH protocol.
  - Usually allow user to view their status via Web interface
  - Both above inter-process communication are via socket (network API)
  - Will learn more about this later.

HDFS: NameNode, DataNode



HDFS: NameNode & DataNode

- **namenode**: node which stores filesystem metadata i.e. which file maps to what block locations and which blocks are stored on which datanode.
- **Secondary namenode** regularly connects to primary namenode and keeps snapshotting filesystem metadata into local/remote storage.
- **data node**: where actual data resides
  - Datanode stores a file block and checksum for it.
  - update namenode with block information periodically, and before updating verify checksums.
  - If checksum is incorrect for a particular block i.e. there is a ***disk level corruption for that block,*** it skips that block while reporting block information to namenode. => namenode replicates the block somewhere else.
  - Send heartbeat message to namenode to say that they are alive => name node detects datanode failure, and initiates replication of blocks
  - Datanodes can talk to each other to rebalance data, move and copy data around and keep replication high.

Hadoop Daemons

	Daemon	*Default Port*	*Configuration Parameter*
HDFS	*namenode*	*50070*	*dfs.http.address*
	datanode	*50075*	*dfs.dataname.http.address*
	secondaryname node	*50090*	*dfs.secondary.http.address*

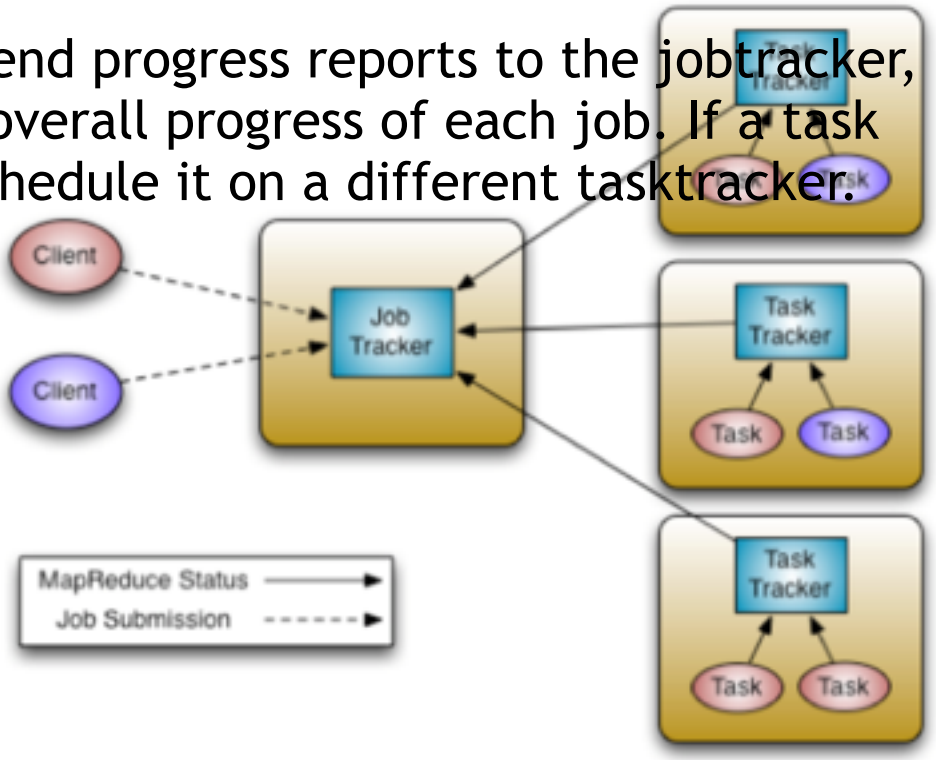
You could open a browser to http://<IP_address_of_namenode>:50070/to view various information about name node

Plan: install a text-based Web browser on puppet, so that we can use web based user-interface.

Hadoop 1.x

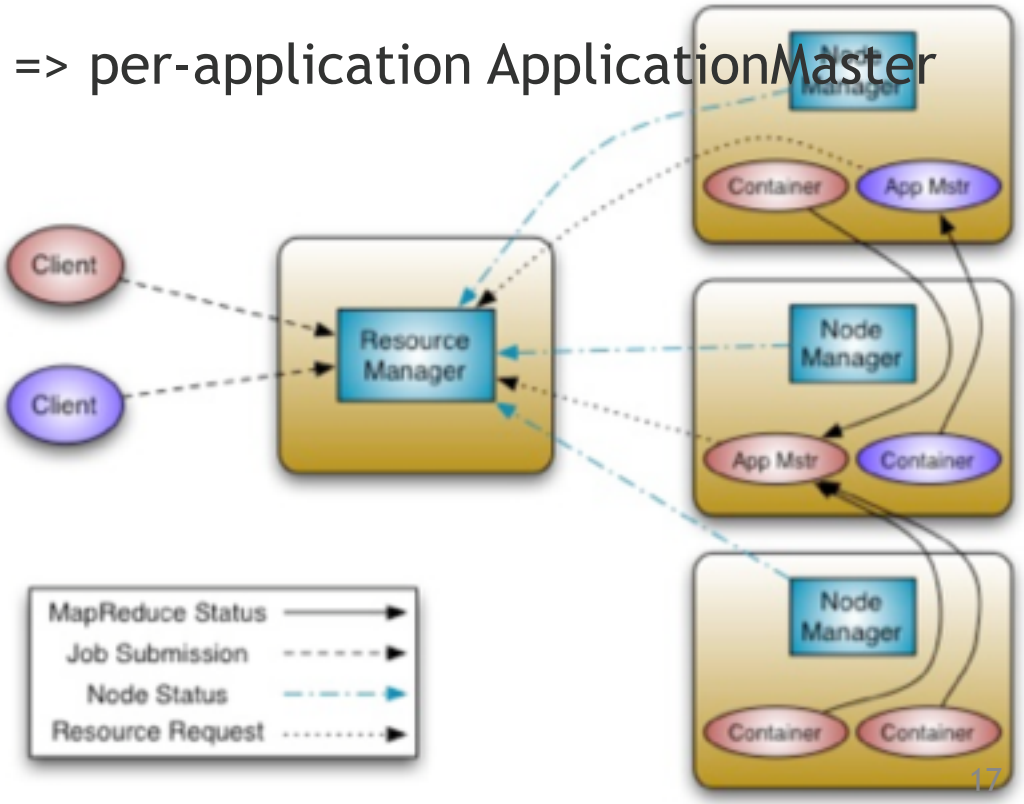
There are two types of nodes that control the job execution process: a *jobtracker* and a number of *tasktrackers*.

- jobtracker: coordinates all jobs run on the system by scheduling tasks to run on tasktrackers.
- Tasktrackers: run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker.



YARN: Yet Another Resource Negotiator

- Resource management => a global ResourceManager
- Per-node resource monitor => NodeManager
- Job scheduling/monitoring => per-application ApplicationMaster (AM).



YARN:

- **Master-slave System**: ResourceManager and per-node slave, NodeManager (NM), form the new, and generic, **system** for managing applications in a distributed manner.
- ResourceManager: ultimate authority that arbitrates resources among all applications in the system.
  - Pluggable **Scheduler**, allocate resources to various running applications
    - based on the *resource requirements* of the applications
    - based on abstract notion of a ***Resource Container*** which incorporates resource elements such as memory, cpu, disk, network etc.
- Per-application **ApplicationMaster**: negotiate resources from ResourceManager and working with NodeManager(s) to execute and monitor component tasks.

WebUI: for Yarn Daemons

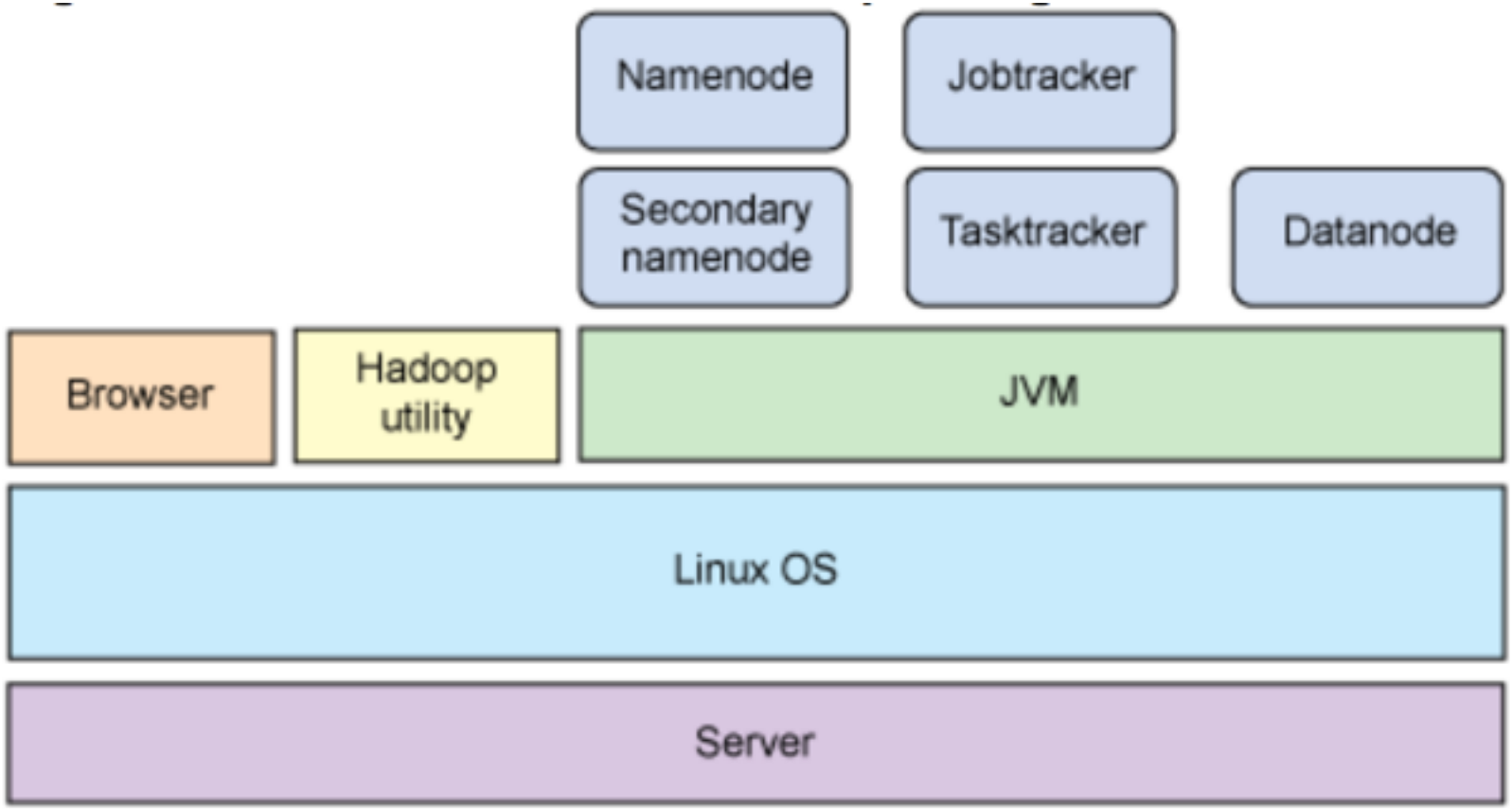
	DAEMON	*PORT*	*Configuration name*
YARN	*ResourceManager*	*8088*	yarn.resourceman-ager.webapp.address
	NodeManager	*50060*	yarn.nodeman-ager.webapp.address

URL to view status of ResouceManager: http://<IP address of RM>:8088

Outline

- Review and demo
  - MapReduce paradigm
  - Hadoop daemons
  - Hadoop configuration,
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop command
- Towards developing a MapReduce Program
  - Tool: maven
  - MapReduce framework: libraries
  - Unit testing

Pseudo-distributed mode



To check whether they are running:

ps -aef | grep namenode

Hadoop configuration

- Default setting: /etc/hadoop/conf
  - core-site.xml

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020</value>
</property>
```

  - hdfs-site.xml: configuration info. for HDFS

```
<   <name>dfs.replication</name>
    <value>1</value>
    …
    <name>dfs.safemode.extension</name>
    <value>0</value>
```

Hadoop configuration

- mapred-site.xml

```
 <name>mapred.job.tracker</name>
 <value>localhost:8021</value>

<name>mapreduce.framework.name</name>
 <value>yarn</value>


 <name>mapreduce.jobhistory.address</name>
 <value>localhost:10020</value>


 <name>mapreduce.jobhistory.webapp.address</name>
 <value>localhost:19888</value>
```

- yarn-site.xml: for YARN<configuration>

```
 <name>yarn.nodemanager.aux-services</name>
 <value>mapreduce.shuffle</value>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
 <value>org.apache.hadoop.mapred.ShuffleHandler</value>
```

Outline

- Review and demo
  - MapReduce paradigm
  - Hadoop daemons
  - Hadoop configuration,
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop command
- Towards developing a MapReduce Program
  - Tool: maven
  - MapReduce framework: libraries
  - Unit testing

Hadoop Command

```
[zhang@puppet ~]$ hadoop
Usage: hadoop [--config confdir] COMMAND
       where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
  checknative [-a|-h]  check native hadoop and compression libraries availability
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop
archive
  classpath            prints the class path needed to get the
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
```

hadoop fs

- run a generic filesystem user client
  - to specify file system: -fs <local|namenode:port>
    - local file system (local to the node): file:///
    - HDFS file system (distributed):
      - hdfs://localhost:8020
  - default file system is set in core-site.xml
- List of commands supported: use *$hadoop fs* to view all commands supported
  - ex: cat, chgrp, chmod, chown, copyFromLocal, copyToLocal, cp, rm, rmdir, du
  - Note: no cd command (which is really a shell internal command)…

Example usage of hadoop fs

- hadoop fs -fs local -ls
- // listing files/directories for current user's home directory in LOCAL filesystem
- hadoop fs -mkdir input_data
  - create a directory under /user/$user/ in HDFS
- hadoop fs -rm -r output_data

config option

- Copy system-wide configuration dir to your home:
  - cp -r /etc/hadoop/conf ~/pseudo-distr-conf
- Modify core-site.xml to change fs.defaultFS to local file system
  - <name>fs.defaultFS</name>
  - <value>file:///</value>
- run following command:
  - hadoop —config ~/pseudo-distr-conf fs -ls

Outline

- Review and demo
  - MapReduce paradigm
  - Behind the scene: Hadoop daemons
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop configuration and hadoop command
- Towards developing a MapReduce Program
  - MapReduce framework: libraries
  - Tool: maven
  - Unit testing

```java
// cc MaxTemperatureMapper Mapper for maximum temperature example
// vv MaxTemperatureMapper
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

Document for Hadoop 2.6.0

```
// cc MaxTemperatureReducer Reducer for maximum temperature example
// vv MaxTemperatureReducer
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
      Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

```java
// cc MaxTemperature Application to find the maximum temperature in the weather dataset
// vv MaxTemperature
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

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

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

Apache Hadoop Main 2.6.0 API

- For now, focus on Package org.apache.hadoop.mapreduce (replace org.apache.hadoop.mapred).
- Use Index to look up class/interface by name
  - Mapper, Reducer: a generic type (C++ template class) with **type parameters**
  - TextInputFormat, default InputFormat used by mapper, decides how input data is parsed into <key,value> pairs …

Running the program

```
#!/bin/bash
## This is a script that submit the MaxTemperature job to run on Hadoop

## prepare the input directory, make it available in the HDFS file system, which is
## used by default (configured in core-site.xml)
hadoop fs -mkdir hdfs://localhost:8020/user/zhang/ncdc
hadoop fs -copyFromLocal ~/SampleData/WeatherDataSet/* hdfs://localhost:8020/user/zhang/ncdc


#start the mapreduce job to analyze spending pattern
# The jar file which contains all three Java class files are provided, main class name,
#  and arguments to the main class is provided..
hadoop jar /home/zhang/MapReduceJava/maximum-temperature/target/maximumtemp-1.0.jar \
MaxTemperature ncdc/1901 out2

#Copy output file to local file system for easy access
mkdir new_output_maxtemp
hadoop fs -copyToLocal out2/* new_output_maxtemp
```

What is in the JAR?

- **JAR** (**J**ava **A**rchive): package file format used to aggregate Java class files and associated metadata and resources (text, images, etc.) into one file
  - built on ZIP file format and have `.jar` file extension.
  - order of entries in zip file headers: manifest needs to be first.
- Manifest file: contains the list of contents of a distribution
- To create or extract JAR files: `jar` command in JDK; or `zip` tools
- Example: view contents of a jar file
- $ jar tvf maximumtemp-1.0.jar

```
     0 Tue Jan 13 18:58:04 EST 2015 META-INF/
   130 Tue Jan 13 18:58:02 EST 2015 META-INF/MANIFEST.MF
  2278 Tue Jan 13 18:58:02 EST 2015 MaxTemperatureReducer.class
  1628 Tue Jan 13 18:58:02 EST 2015 MaxTemperatureWithCombiner.class
  2433 Tue Jan 13 18:58:02 EST 2015 MaxTemperatureMapper.class
  1541 Tue Jan 13 18:58:02 EST 2015 MaxTemperature.class
     0 Tue Jan 13 18:58:04 EST 2015 META-INF/maven/
     0 Tue Jan 13 18:58:04 EST 2015 META-INF/maven/com.zhang/
     0 Tue Jan 13 18:58:04 EST 2015 META-INF/maven/com.zhang/maximumtemp/
   791 Tue Jan 13 18:56:26 EST 2015 META-INF/maven/com.zhang/maximumtemp/pom.xml
   103 Tue Jan 13 18:58:02 EST 2015 META-INF/maven/com.zhang/maximumtemp/
pom.properties
```

Maven

- Maven: tool for building and managing Java-based project
- Making build process easy
    - Providing a uniform build system: **project object model (POM)** and a set of plugins
- Providing quality project information
    - Change log document created directly from source control
    - Cross referenced sources
    - Mailing lists
    - Dependency list
    - **Unit test reports including coverage**
- Providing guidelines for **best practices development**
    - **Testing**, project workflow (release management), layout of project's directory structure
- Allowing transparent migration to new features
- Integration with IDEs …

POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.zhang</groupId>
  <artifactId>maximumtemp</artifactId>
  <version>1.0</version>
  <name>maximumtemp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>2.6.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>2.6.0</version>
    </dependency>
  </dependencies>
</project>
~
```

For now, we will use this as template.

1. write POM.XML.

Directory Structure

```
[zhang@puppet maximum-temperature]$ tree
.
├── pom.xml
├── README.txt
├── src
│   └── main
│       └── java
│           ├── MaxTemperature.java
│           ├── MaxTemperatureMapper.java
│           ├── MaxTemperatureReducer.java
│           └── MaxTemperatureWithCombiner.java
└── target
    ├── classes
    │   ├── MaxTemperature.class
    │   ├── MaxTemperatureMapper.class
    │   ├── MaxTemperatureReducer.class
    │   └── MaxTemperatureWithCombiner.class
    ├── maven-archiver
    │   └── pom.properties
    └── maximumtemp-1.0.jar
```

For now, we will use this as template.

2. Create the java files under src/main/java directory

3. Run mvn package
to compile java files and package class files into .jar file

Configuring Maven

- A **repository** in Maven is used to hold build artifacts and dependencies of varying types
- Local Repository:
  - Location is specified in ~/.m2/settings.xml

```
. . .
<localRepository>/path/to/local/repo/</localRepository>
...
```

- Central Repository:
  - If a project declares a dependency that is not present in local repository, maven will automatically download it from a central repository, by default, it's located at:
  - Maven's central Repository: http://search.maven.org/#browse

MVN command line

```
[zhang@puppet maximum-temperature]$ mvn -help

usage: mvn [options] [<goal(s)>] [<phase(s)>]

Options:
 -am,--also-make                          If project list is specified, also
                                          build projects required by the
                                          list
 -amd,--also-make-dependents              If project list is specified, also
                                          build projects that depend on
                                          projects on the list
 -B,--batch-mode                          Run in non-interactive (batch)
                                          mode
 -b,--builder <arg>                       The id of the build strategy to
                                          use.
 -C,--strict-checksums                    Fail the build if checksums don't
                                          match
 -c,--lax-checksums                       Warn if checksums don't match
 -cpu,--check-plugin-updates              Ineffective, only kept for
                                          backward compatibility
 -D,--define <arg>                        Define a system property
 -e,--errors                              Produce execution error messages
 -emp,--encrypt-master-password <arg>     Encrypt master security password
```

Maven Phases

Software Development Lifecycle and phases

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test compiled source code using a suitable **unit testing framework**. These tests should not require code be packaged or deployed
- **package**: take compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

There are two other Maven lifecycles of note beyond the *default* list above. They are

- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

Outline

- Review and demo
  - MapReduce paradigm
  - Behind the scene: Hadoop daemons
    - Standalone mode, **pseudo-distributed mode**, distributed mode
  - Hadoop configuration and hadoop command
- Towards developing a MapReduce Program
  - Tool: maven
  - MapReduce framework: libraries
  - Unit testing: instruction will be provided in homework/ lab3