Queen Mary University of London

WEEK 3 TUTORIAL

HOW TO RUN HADOOP PROGRAMS

CLOUD COMPUTING

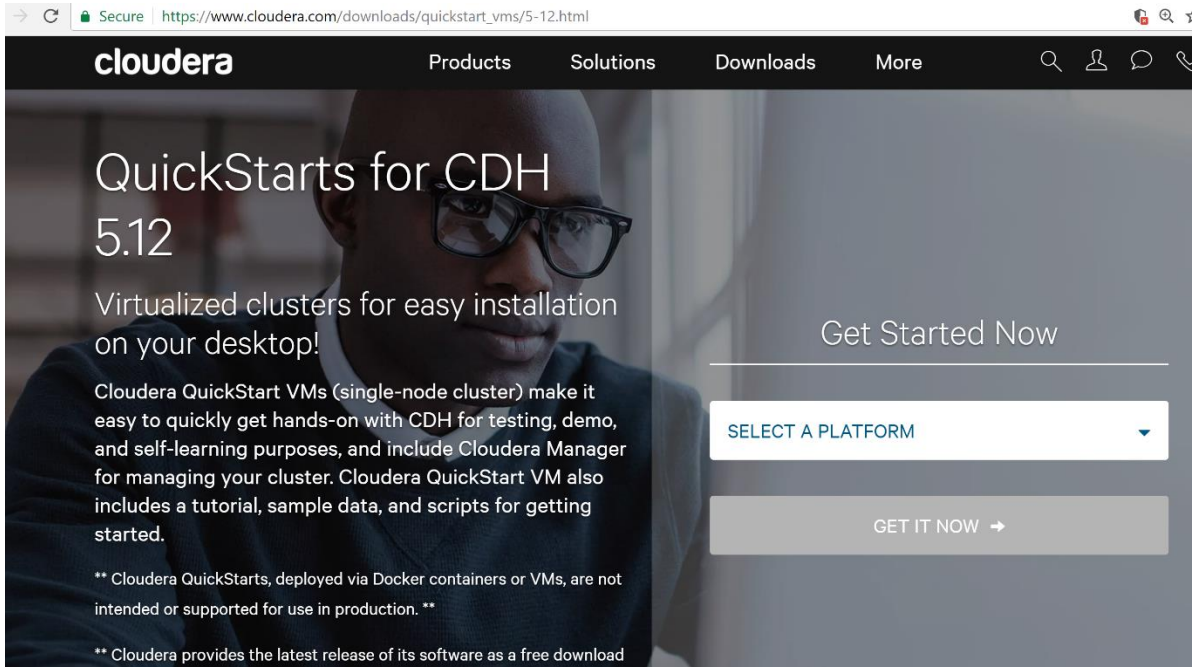
Dr. Atm Shafiul Alam

Queen Mary University of London

School of Electronic Engineering and Computer Science

Cloudera Quickstart VM

- A Virtual Machine image you can download
- Contains Linux distribution with Hadoop pre-installed



Cloudera Quickstart VM

- To download VMware Workstation Player:

https://www.vmware.com/uk/products/workstation-player/workstation-player-evaluation.html

- To download the Cloudera Quickstart VM:

https://downloads.cloudera.com/demo_vm/vmware/cloudera-quickstart-vm-5.13.0-0-vmware.zip

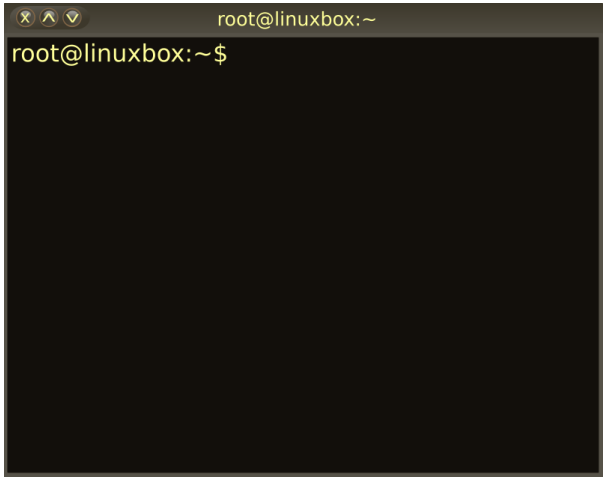
Cloudera Quickstart VM

- Download VMware Player
- Download Cloudera Quickstart VM
- Load Cloudera in VMware
- …you now have a working Hadoop installation

Using HDFS

- All data in Hadoop is stored in HDFS
  - Both inputs (to mappers) and outputs (from reducers)
- When you want to use Hadoop you **must** first add your data into the cluster's HDFS
- When you want to access data generated by a reducer, you should download it into your local filesystem

HDFS commands



- `hadoop fs -help`
  - Lists the file system commands
- `hadoop fs -ls`
  - Lists the contents of root folder
- `hadoop fs -ls data_dir`
  - Lists the contents of the folder called `data_dir`

HDFS commands

- `hadoop fs –mkdir data_dir`
  - Creates a directory called data_dir
- `hadoop fs -copyFromLocal my_file.txt  data_dir`
  - Uploads data into HDFS
- `hadoop fs -copyToLocal my_output.txt local_dir`
  - Downloads data from HDFS to local file system

Writing the code

- The code we've done so far is pseudocode
  - **Not Hadoop!**

Let's learn the real code today…

Writing the code

- Create a new folder called `src`
- This is where you will place your `.java` files
- You must now create three java files:
  - Mapper class
  - Reducer class
  - Job Description (contains a main method)
- Collectively: these are your **Job**

Types in Hadoop

- Hadoop uses "special" types, e.g.
  - `IntWritable`: replaces `int`
  - `LongWritable`: replaces `long`
  - `Text`: replaces `String`
- `set()` method changes value
  - E.g. `text.set("hello")`
- `get()` or `toString()` retrieves value
  - E.g. `myIntWritable.get()`
  - E.g. `myText.toString()`

`Context` in Hadoop

- The `map` method and `reduce` method need access to the Hadoop framework
  - E.g. to emit key-value pairs
- The `map` method takes a `Context` object
- The `reduce` method takes a `Context` object
- This allows the methods to "talk" to the rest of Hadoop

emit() in Hadoop

- `emit()` is not the real method name
- We actually use `write(key, value)`
- `write()` method is in the context object
  - `context.write(key, value)`

Writing you mapper

```
public class MyMapper
    extends Mapper<LongWritable, Text, Text, IntWritable>
{

    public void map(LongWritable key, Text value,
                                      Context context)
      {
      context.write(new Text("A"),    w IntWritable(1));

      }
}
```

Reference to the Context object

Writing your mapper

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text,
                                                        IntWritable> {

    private final IntWritable one = new IntWritable(1);
    private Text data = new Text();

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

        StringTokenizer itr = new StringTokenizer(value.toString(), " ");

         while (itr.hasMoreTokens()) {
           data.set(itr.nextToken()); //next word
            context.write(data, one); //same as emit
            }
    }
}
```

Writing your reducer

```
public class MyReducer
        extends Reducer<LongWritable, Text, Text, IntWritable>
{

    public void reduce(LongWritable key, Iterable<Text>
                                        values, Context context)
      {
      context.write(new Text("A"),      IntWritable(1));
      }

}
```

`Iterable` – a list of values.
Remember – each key can be associated with multiple values…

Writing your reducer…

```
public 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 value : values) {
        //CODE GOES HERE - should sum up the counts
        }

        result.set(sum); //sets result to value of sum

        //CODE GOES HERE - should emit the result for each word
    }
```

Writing your job (main class)…

```
public class WordCount {

  public static void runJob(String[] input, String output) throws Exception {

    Configuration conf = new Configuration();

    Job job = new Job(conf);

    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class); //Sets the mapper class
    job.setReducerClass(IntSumReducer.class); //Sets the reducer class
    job.setMapOutputKeyClass(Text.class); //Tells Hadoop the type of output key
    job.setMapOutputValueClass(IntWritable.class); //Tells Hadoop the type of output value

    FileInputFormat.setInputPaths(job, StringUtils.join(input, ","));

    Path outputPath = new Path(output);
    FileOutputFormat.setOutputPath(job, outputPath);
    outputPath.getFileSystem(conf).delete(outputPath, true);
    job.waitForCompletion(true);
  }
```

……

Writing your job…

…continued

```
  public static void main(String[] args) throws Exception {
        runJob(Arrays.copyOfRange(args, 0, args.length-1),
                                                  args[args.length-1]);
        //gets folders for input/output
  }

}
```

List of input folders

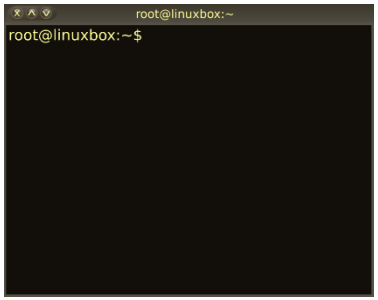
The output folder

Building your code (i.e. job)

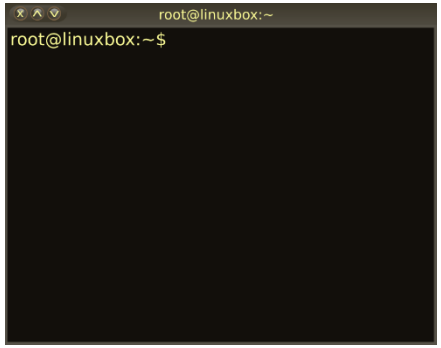
- Your job is contained within a jar file
  - Contains your Mapper, Reducer, and Job description
  - Plus other Hadoop-specific code
- This gets "injected" into Hadoop
- We use `ant` to compile our jar file
- From command line:
  - `ant clean dist`
- By default `ant` uses **build.xml** as the name for a build file

Launching the job

- Next we must launch the job in Hadoop
- We use the Hadoop command again
- From command line:
  - `hadoop jar jarfile job_class [params…]`
- For example:
  - `hadoop jar dist/WordCount.jar WordCount input out`



Checking the status



- `hadoop fs -ls out`
  - part-r-00000 _SUCCESS

- `hadoop fs -copyToLocal my_output.txt local_dir`
  - Downloads data from HDFS to local file system
  - **E.g.:** `hadoop fs -copyToLocal out/part-r-00000`

You're now ready to code!