MapReduce and AWS(Java)

1. Introduction

Mapreduce

MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers.

- Map step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. The worker node processes the smaller problem, and passes the answer back to its master node.
- Reduce step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output the answer to the problem it was originally trying to solve.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel. Similarly, a set of reducers can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. Another way to look at MapReduce is as a 3-step parallel and distributed computation:

< K1,V 1 > −map− < K2,V 2 > −shuffle− < K2,alistof V 2 > −reduce− < K3,V 3 >

1. Prepare the Map() input the MapReduce system designates Map processors, assigns the K1 input key value each processor would work on, and provides that processor with all the input data associated with that key value. Run the user-provided Map() code Map() is run exactly once for each K1 key value, generating output organized by key values K2.
2. Shuffle the Map output to the Reduce processors the MapReduce system designates Reduce processors, assigns the K2 key value each processor would work on, and provides that processor with all the Map-generated data associated with that key value.
3. Run the user-provided Reduce() code Reduce() is run exactly once for each K2 key value produced by the Map step. Produce the final output the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

AWS

Amazon Elastic MapReduce (Amazon EMR) is a web service that makes it easy to quickly and cost-effectively process vast amounts of data. Amazon EMR uses Hadoop, an open source framework, to distribute your data and processing across a resizable cluster of Amazon EC2 instances.

You will need to create your own educational AWS account in order to run your jobs for this lab, and will get 30-40ish dollars of credit. You will need to provide your credit/debit card information upon account activation.

2. A Simple WordCount Example

Setup

Java

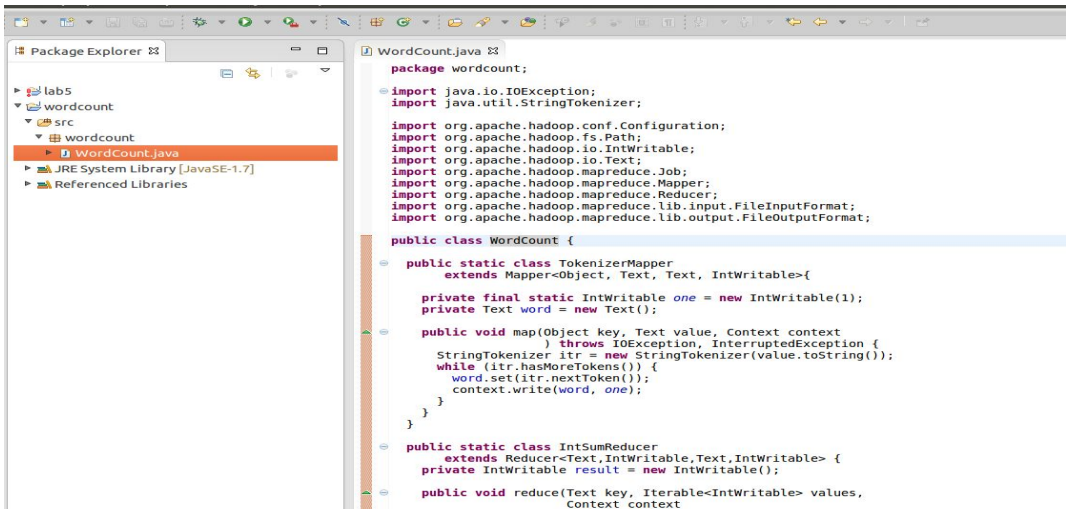
For this lab we are using Java. You need to install Java JDK 1.7 + on you OS, whether it's Windows, Mac OS or Ubuntu. You don't need VM for this lab.
We recommend using Eclipse for development, which is also available for OSes mentioned above. This example tutorial will be using Eclipse.

Hadoop

We use Hadoop 2.7.2 for this lab, please download Hadoop here. You DO NOT need to setup Hadoop on your own machine. Instead, you'll just need to use the dependency included with the Hadoop binary distribution in 'share/hadoop' folder.

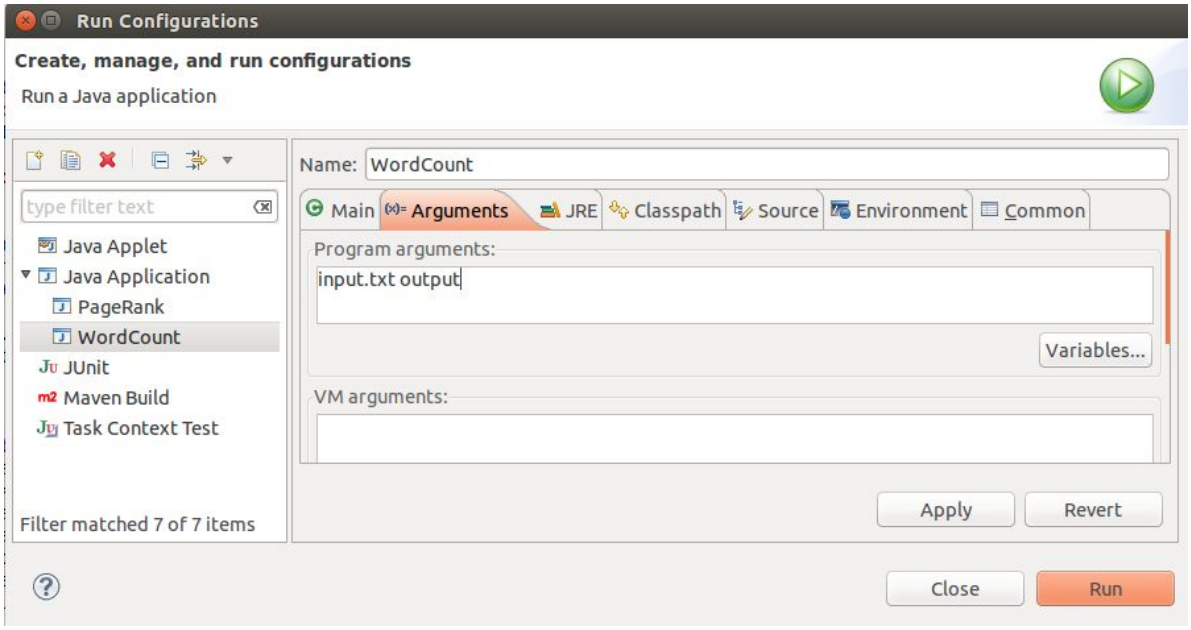
Run WordCount Example Locally.

Now start Eclipse and create your 'wordcount' project with the wordcount example code.
The example code is adopted from Apache Hadoop website WordCount V1. You can just copy the code and paste to your 'WordCount.java' source file.



There are quite a bunch of deps (.jar files) in the Hadoop binary distribution to add to your wordcount project to make it compilable, refer to the last section of this document for a list of deps to include.

After resolving the deps problem, we need to supply the input: in your wordcount project folder, create a text file ‘input.txt’ and paste some random text to it.
Right click the ‘WordCount.java’ file, in ‘Run As’ select ‘Run Configurations’, and supply the arguments for the ‘WordCount’ Java application:

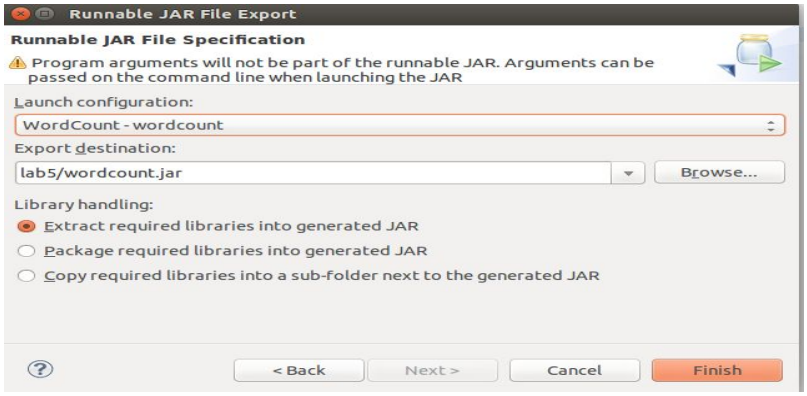


Click ‘run’ and your program should be running. You will find a folder ‘output’ in your project folder and there is a ‘part-r-00000’ file that contains the count of each word -- you have got the WordCount example successfully running!

Run WordCount Example on AWS.

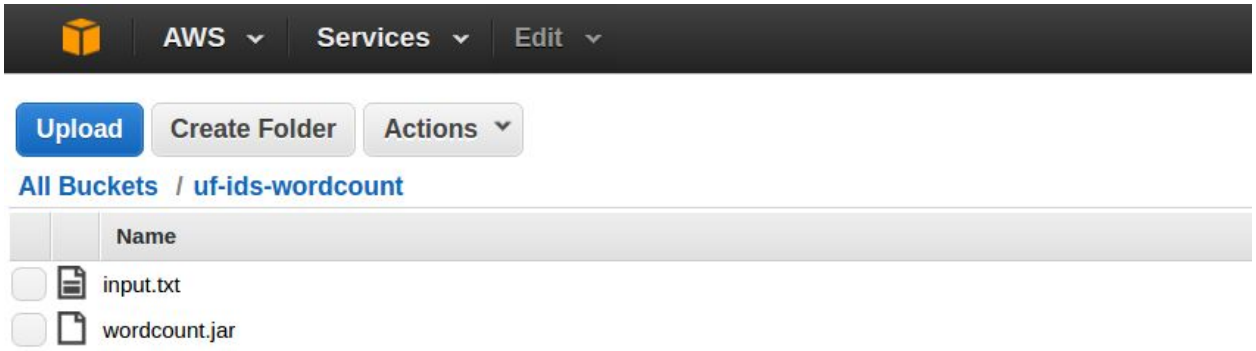
Export your locally-tested code to a runnable jar file

In order to run our program on AWS, first we need to create a runnable jar: right click your Eclipse wordcount project -> export -> java -> Runnable JAR file, click next, in ‘Launch configuration’, select ‘WordCount - wordcount’ as the main entry point of your program in JAR file, select a proper export destination, and click finish.



Upload input.txt file and runnable Jar file to S3.

In your web browser sign in into your AWS account web console, ‘Services’ -> ‘S3’, create bucket ‘uf-ids-wordcount’ -- you’ll need to use a different bucket name, but here we use uf-ids-wordcount for illustration purpose.

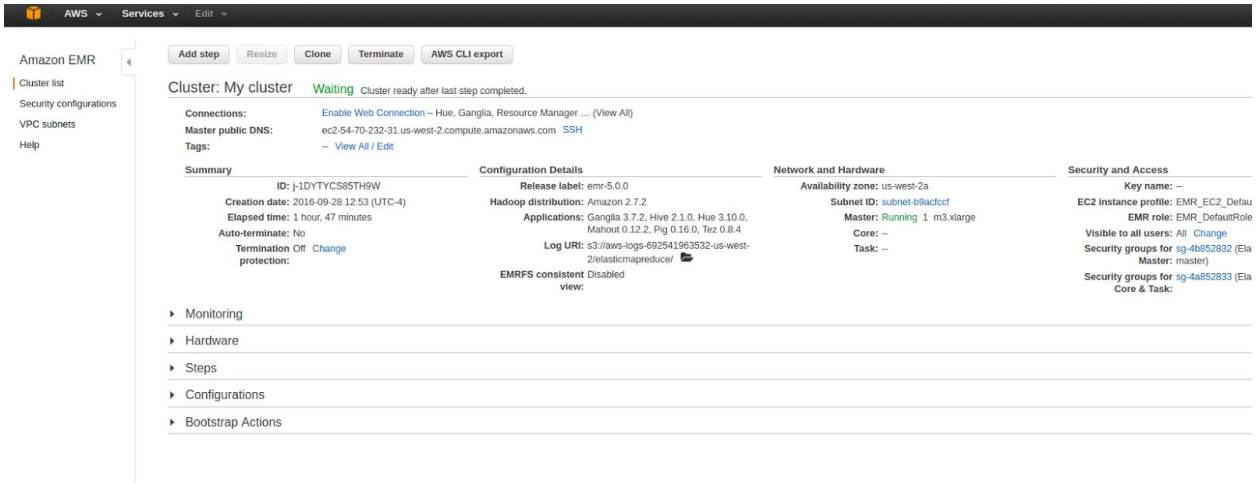


Start a single-instance cluster

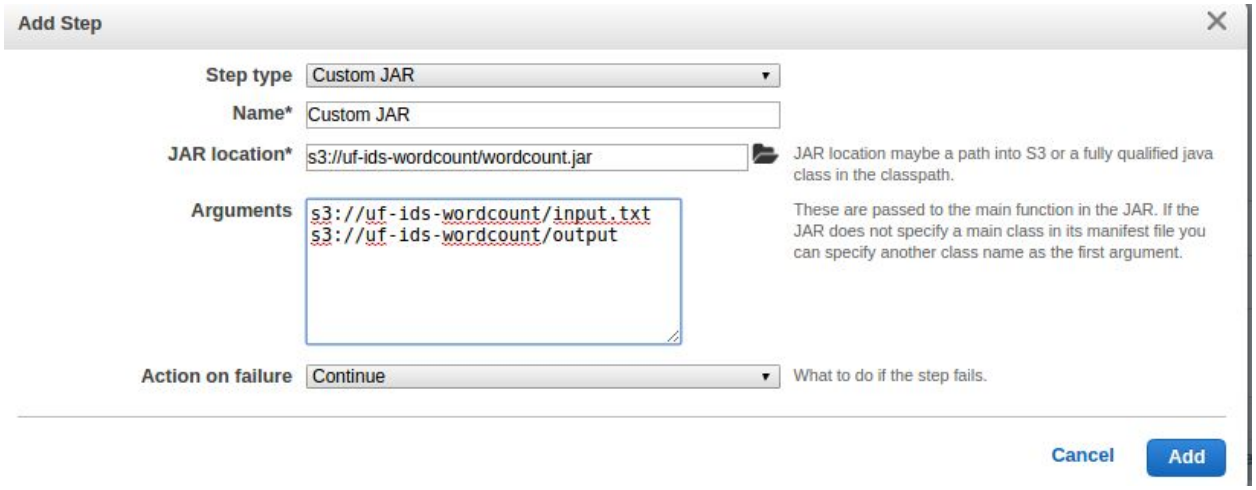
In your AWS web console, ‘Services -> EMR -> Create cluster’, in ‘Software configuration’ make sure you choose the ‘Hadoop 2.7.2’ version (it should be default).
And in ‘Hardware configuration’, choose m3.large and input number of instance to be 1, since we are just running a very simple testing job.
Click ‘Create cluster’ on the bottom to start, it’ll take a few minutes.

Run the uploaded Jar.

When the cluster is created, click the cluster and you can view the details of the cluster and manage it:



To run our job using the uploaded Jar file, Click ‘Steps’ -> ‘Add step’, and specify the location of our jar file and arguments for input and output.



Click ‘Add’ and wait for the step’s completion. You’ll find an ‘output’ folder containing the output files ‘part-r-xxxxxx’ in your S3 bucket. You have successfully run your wordcount job on AWS!

Important:

Don’t forget to terminate your cluster after using it, otherwise it will cost your extra credits/money!

3. Extract Valid wikilinks and Generate Adjacency Graph

In this lab we will use the wikilinks data set downloadable here:
https://s3.amazonaws.com/uf-ids-data/enwiki-latest-pages-articles.xml
You can truncate this file to smaller size for debugging.
In this lab, your job is to generate an adjacency graph from the dataset provided.

1. Write a MapReduce job that extracts wikilinks and also remove all the red links. The dataset contains, for each article, a list of links going out of that article, as well as the articles title. The input for this job should be the Wikipedia dataset above, and the job should look at the XML text for each article, extract title and wikilinks. A single page would be represented as follows (with some fields and attributes omitted):
   <page> <title>Title</title> ...
   <text ...>
   Body of page
   [[link text]]
   </text>
   </page>
   Your task is to extract the Title and the wikilinks from these articles. Within the text of the page, a wikilink is indicated by [[link text]]. In the simplest kind of link, the link text is the name of another page (with any spaces replaced with underscores). A more complicated kind gives different text to appear as well as the name of the page: [[page name|text to appear]]. How to differentiate wikilinks to other types of links? Please refer to the http://en.Wikipedia.org/wiki/Help:Wikilinks#Wikilinks.
   There two formats of wikilinks as described in the above Wikipedia page. And repeating here:
   A. [[abc]] is seen as “abc” in text and links to page “abc”. We need to extract “abc” out.
   B. [[a|b]] is labelled “b” on this page but links to page “a”. We need to extract “a” out.
   But “abc” in (A) and “a” in (B) may be a red link which basically means there is no page with title “abc” and “a” in the wiki dataset. You need to exclude all red links in this job.
   Note: we need to replace all the empty space ‘ ’ in the Title and links(title of other pages) with an underline ‘_’. No other processing is needed in this project.
2. Write another MapReduce job to generate the Adjacency graph, i.e a graph with the format:
   page_title1 link1 link2 ...
   page_titile2 link2 link3 ...
   With each line representing a page, and each item separated by tabs..

Input and output

Your Java Jar file should take two arguments: a path to input and a path to an output folder, and the output folder should have the layout:

output/ --- output folder, the second argument to your Jar file.
graph/ -- containing the part-r-xxxxx files of the actual output, will be examined by TA.
temp/ -- containing intermediate results, will be ignored by TA.

Please use HDFS API instead of Java File API for files operations, as Java File API would not be able to handle paths like ‘s3://bucket_name’ on AWS or paths on HDFS.

What to submit

Your submission should include your source code, and a runnable Jar file that handles the input and output mentioned above. Also please include a report on the method you use to remove the redlinks, and what you find difficult while working on this lab.

Your submission layout should be like the following to facilitate the TA's grading:

Firstname_Lastname/ *-- Your firstname and lastname as shown on Canvas*
src/ *-- source code only, no deps or project folder.*
extract.jar *-- your jar file should be named exactly as shown here.*
report.pdf *-- your report.*

Compress your folder as a zip file and submit it.

Please note: failure to comply with the input & output requirement or submission format will result in zero point for lab 5. You will need to come to TA's office hour for a regrade at minimum 30% penalty.

How to handle XML file in Hadoop.

To handle large XML input file efficiently in Hadoop, please use 'XMLInputFormat' to split the the large XML file into pages, you can include the Java file in your project (change the package name though, and set the start and end delimiter respectively) and by using code:

```
job.setInputFormatClass(XmlInputFormat.class);
```

In your mapper function, you can use dom4j package to handle each page to get the title and text element efficiently.

How to Extract wikilinks in Text

You'll need to extract links using regular expressions, to avoid the excruciating details on how to construct the regular expression and extract wikilinks, we provide you the way to to it:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

Pattern pattern = Pattern.compile("\\[\\[(.*?)(\\||\\]\\])");
Matcher matcher = pattern.matcher(text_in_the_page);
while (matcher.find()) {
        //do the processing
}
```

Please use the code above for extracting wikilinks in the text.

Appendix

Deps for wordcount example

This is excerpted from the '.classpath' file in the Eclipse project. You can just add it to your '.classpath' file and replace the 'path_to' with the actual path to where your downloaded Hadoop 2.7.2, or you can add the deps manually. This set of deps might not be the minimal set, but it should work.

```
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/hadoop-common-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-logging-1.1.3.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/guava-11.0.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/log4j-1.2.17.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-lang-2.6.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-collections-3.2.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-io-2.4.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-configuration-1.6.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/hadoop-auth-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/slf4j-api-1.7.10.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/yarn/hadoop-yarn-common-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/jackson-core-asl-1.9.13.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/jackson-xc-1.9.13.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/avro-1.7.4.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/yarn/hadoop-yarn-api-2.7.2.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/commons-httpclient-3.1.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/httpcore-4.2.5.jar"/>
<classpathentry kind="lib" path="/path_to/hadoop-2.7.2/share/hadoop/common/lib/httpclient-4.2.5.jar"/>
```

Extra Tutorials

https://aws.amazon.com/training/intro_series/
http://en.Wikipedia.org/wiki/MapReduce
http://aws.amazon.com/elasticmapreduce/
http://en.Wikipedia.org/wiki/Help:Wikilinks#Wikilinks