In this lab you will be creating a MapReduce application and running it in AWS using Elastic MapReduce.  EMR supports several different execution platforms, but we will be using Apache Hadoop, which is an open-source implementation of MapReduce (and probably the most popular one).

Learning to Write Applications for Hadoop

Start by reading this Hadoop tutorial, which shows you how to write MapReduce applications for Hadoop.  It's probably a good idea to follow along by trying to compile and run the sample application.

[https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (Links to an external site.)Links to an external site.](https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)

(If you need a Linux machine to run Hadoop on, you can use the cs1 server or launch a *micro* sized EC2 instance to work on. Just remember that EC2 instances won't save their data if you terminate them, but you can always create an AMI if you want.)

Also, if you download/install Hadoop, keep in mind that Elastic MapReduce uses version 2.7.3 of Hadoop. I've found so far that any 2.7.x version works, but stay away from 2.8, 3.0, etc.

**Note:** If you are trying to run Hadoop on the ***cs1 server***, there are a couple of configuration settings you will need to make. In your home directory, use your text editor of choice (emacs, vim, nano, etc.) to edit the **.bashrc** file.  (Make sure you don't miss the 'dot' at the front of the file name!)  Then add these lines:

export JAVA\_HOME=/usr/lib/jvm/java  
export HADOOP\_CLASSPATH=$JAVA\_HOME/lib/tools.jar  
  
alias jar='jar -J-Xms512m -J-Xmx512m'

Finally, you will need to log out and then back in again for the changes to take effect.

Other Languages

If you prefer not to use Java for your program, you can allegedly write Python code for Hadoop.  This may work for other languages too...  Here is a page with information about this.  ***I have not tested this****and cannot personally guarantee it will work.*  This is very much a "try at your own risk" situation.  If you decide to try this, I will give my best effort to support it, but I cannot guarantee to solve problems that you may run into.  If you want to try this, I strongly suggest that you *start very early* so there's still time to finish if it doesn't work an you need to fall back to using Java anyway.

[http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/#reduce-reducer-py (Links to an external site.)Links to an external site.](http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/#reduce-reducer-py)

You are allowed to do this assignment in any "reasonably popular" language that you can get to work with Hadoop and run on Amazon EMR.  (If you have a doubt about whether a particular language is "reasonably popular" then ask the professor.)

Running Applications on Elastic MapReduce

The next step is to learn how to use Amazon EMR.  I recommend you first try running the tutorial application (from the previous section) since the code is provided and you already tried it out locally so you know that that much at least works correctly (rather than trying to run your own application that is still being developed and debugged).

Uploading the Application

The first thing you need to do is to upload the application and input data.  We'll use the AWS Simple Storage Service (S3) for this.  The S3 documentation can be found here:

[https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html)

First, create a bucket for you application.  (See the "Create a Bucket" section of the documentation for instructions.)  I named mine "wordcount-lillethd", but you will have to use a slightly different name since S3 bucket names must be globally unique.  (You also won't be able to name it just "wordcount" - I checked.)  ;)

Now upload ("Add an Object") your .jar file to the S3 bucket.

Create a folders in the bucket named "input".

Now go into the "input" folder and upload all of you application's input data files here.  (For the WordCount example, these are just plain text file that contain words. They do not require any particular format.)

Running the Application

Documentation for submitting a "Custom JAR" to EMR is here:

[https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-launch-custom-jar-cli.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-launch-custom-jar-cli.html)

For the "Arguments" part, assuming you wrote your program as shown in the Hadoop tutorial, you will need three arguments. The first is the main class (WordCount, in the tutorial example), the second is the folder containing the inputs, and the third is the folder where you want the outputs to be saved. (The folder for the outputs should *not* exist - EMR will create it for you.)

So assuming you named your S3 bucket "wordcount" and created the folder "input" (containing your input files) and want EMR to create the folder "output" to write you output to, your arguments would look like this:

WordCount s3://wordcount/input s3://wordcount/output

Of course you will need to adjust this to use your actual bucket name instead of "wordcount"...  (and watch out for typos in these names!)

Other Languages (continued)

If you are using the instructions above to implement your program in a language other than Java, you will be using a "Streaming program" instead of a Custom JAR.  Documentation for submitting those to EMR is here"

[https://docs.aws.amazon.com/emr/latest/ReleaseGuide/CLI\_CreateStreaming.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/CLI_CreateStreaming.html)

Monitoring Your EMR Job

Once it's running, you can go to the EMR console and click on "Clusters".  All your EMR compute clusters are listed there. To view the details of a cluster, click on it's name.

In the details, the "Hardware" tab will show you the different groups of server instances that are running as part of your EMR cluster.  (You can also go to the EC2 console, and you'll see the instances listed there as well... but it's probably best to control them through the EMR console rather than controlling them directly through the EC2 console, unless something has gone awry and you need to be a little more "forceful" with your commands.)

In the EMR cluster details, the "Steps" tab will show the different steps you have instructed the cluster to perform and their current status.  (At the moment, we have only given it a single step, which is either the "Custom JAR" step or "Streaming program" step.)  You can also look at the logs for each step here.  The 'stderr' log contains the error output from Hadoop itself.

Processing Web Data with Elastic MapReduce

Now let's get to writing our own application. "[Common Crawl (Links to an external site.)Links to an external site.](https://commoncrawl.org/)" is a public data set of web crawl data that is made available on S3 (which means it is easy to use as an input to EMR).  The S3 URI for their data set is here:

s3://commoncrawl/crawl-data/CC-MAIN-2018-05/segments/1516084886237.6/warc

However, the data set is so large, it may not be feasible to run your code on all of it for this assignment.  I have created a subset of 50 input files (out of their total 800) that you can use instead for the assignment.  You can now use this URI for your application's input to point it at the curated subset of the data set:

s3://seattleu-cloud-computing/common-crawl

The input files are in the WARC format.  More info about the file format is here:

[https://commoncrawl.org/2014/04/navigating-the-warc-file-format/ (Links to an external site.)Links to an external site.](https://commoncrawl.org/2014/04/navigating-the-warc-file-format/)

If you want to download an example WARC file, you could download this one (the first file in the data set):

[https://commoncrawl.s3.amazonaws.com/crawl-data/CC-MAIN-2018-05/segments/1516084886237.6/warc/CC-MAIN-20180116070444-20180116090444-00000.warc.gz (Links to an external site.)Links to an external site.](https://commoncrawl.s3.amazonaws.com/crawl-data/CC-MAIN-2018-05/segments/1516084886237.6/warc/CC-MAIN-20180116070444-20180116090444-00000.warc.gz)

***Warning:****This download is about 1 GB, and it is a compressed file. When you uncompress it, it will be a little under 4 GB! This may create some issues if you try to open the whole file in a regular text editor...  Using Linux/MacOS commands to view only part of it at a time seem to be fine, though, e.g.,*head, tail, grep*, etc.*

**Note:** You don't need to worry about the fact that it's compressed when you take it as input to EMR.  Hadoop should detect this and uncompress them automatically, passing uncompressed data to you map function.

The following tools are optional for this assignment, but you may find them helpful in dealing with the WARC file format.  (If you find other tools for using WARC with EMR / Hadoop that you prefer to you instead, you may do so. This is just the one that Common Crawl themselves link to.)

[https://github.com/Smerity/cc-warc-examples (Links to an external site.)Links to an external site.](https://github.com/Smerity/cc-warc-examples)

What to Implement

For every email address that appears on any web pages, output that email address and the number of *domains* (not individual pages) on which the email address may be found.

For example, if my email address (lillethd@seattleu.edu) was found on 3 different pages of the Seattle U and CS department websites (i.e., seattleu.edu domain), once on Linked-In, and in two different places on my personal website, then you would want to output:

lillethd@seattleu.edu, 3

(that's once for seattleu.edu, once for Linked-In, and once for my personal website's domain)

This means that you will need to search through each record (i.e., pages retrieved) in the WARC data, get the URL of the page, extract the domain name from the URL, then look through the page request for any patterns that are valid email addresses.

Creating a Regular Expression that matches valid email addresses would be a good way to do that (although this is just a recommendation, and you're free to go about this any way you can make work correctly).

**Note:** You may assume that valid email address patterns will only appear in the response bodies (i.e., the text of the actual web page), so you do *not* need to be careful about search only response bodies and not requests, response headers, and WARC headers/metadata.

For the domain names, you have two options that will be acceptable for this assignment:  Either you can just use the complete domains for everything, or you can use second-level domains (i.e., the ".com" and the part immediately before it - for example, instructure.com is the second-level domain corresponding to the full domain seattleu.instructure.com).

Tips

If you're not very experienced with Java (or even if you are but have not seen these concepts before), it will help to know a bit about ***packages***, ***classpath***, and ***jar files***.  The following links explain these concepts.

*Packages (be sure to click through all the pages in the "lesson")*:

[https://docs.oracle.com/javase/tutorial/java/package/index.html (Links to an external site.)Links to an external site.](https://docs.oracle.com/javase/tutorial/java/package/index.html)

*Classpath*:

[https://stackoverflow.com/questions/2396493/what-is-a-classpath (Links to an external site.)Links to an external site.](https://stackoverflow.com/questions/2396493/what-is-a-classpath)

[http://www.kevinboone.net/classpath.html (Links to an external site.)Links to an external site.](http://www.kevinboone.net/classpath.html)

**Note:** When working with Hadoop, you need to use the HADOOP\_CLASSPATH environment variable instead of the regular CLASSPATH one (but it otherwise works the same way), and using the -cp or -classpath compile flag should not be done.

*JAR files*:

[https://docs.oracle.com/javase/tutorial/deployment/jar/build.html (Links to an external site.)Links to an external site.](https://docs.oracle.com/javase/tutorial/deployment/jar/build.html)

[http://blog.cloudera.com/blog/2011/01/how-to-include-third-party-libraries-in-your-map-reduce-job/ (Links to an external site.)Links to an external site.](http://blog.cloudera.com/blog/2011/01/how-to-include-third-party-libraries-in-your-map-reduce-job/)

**Note:** I find method #2 for including libraries on EMR to work best.  (I don't think method #3 will work at all in our environment, since we're running a cluster on an ongoing basis.)

The structure I've found to work for my JAR file with Hadoop and EMR is the following.  Note how certain files are in certain directories - this is important.  (GetEmails is the name of *my* class, but you may have named your differently.)

GetEmails.class  
GetEmails$EmailFinderMapper.class  
GetEmails$ValueListReducer.class  
org/commoncrawl/warc/WARCFileInputFormat.class  
org/commoncrawl/warc/WARCFileRecordReader.class  
lib/webarchive-commons-jar-with-dependencies.jar

If you want to check the contents of you JAR file after you've created it, you can do that with this command (just replace "getemails.jar" with the name of your JAR file).

jar tf getemails.jar

For your testing and debugging, it is recommended that you use a single WARC file and the default cluster size of 3 nodes (2 core compute nodes + 1 master). However, when your code is working and you run it on the 50 file curated data set, you should use a larger cluster!  I recommend about **10-12 nodes**.

What to Submit

Submit your assignment by attaching files in Canvas.  (You may compress the files if that's convenient - .zip, .tar, .gz, and .tgz formats are all acceptable.)  You should attach the following:

* All your source code files for you application
  + only include what you wrote for the "What to Implement" section above; e.g., do not include the WordCount example
  + only include source files, not compiled files (such as .class or .jar file)
  + only include source you wrote yourself; do not include Hadoop itself or any other support tools you used (but do see the README file below)
* All your output from running your application on the web crawl data
* A README file that you write
  + this should be a plain text file (*not*, e.g., a MS Word or PDF file) and should be named either README, README.md, or README.txt
  + it should include the following information
    - an explanation of each source code file you included (i.e., a one or two line description of what that file contains)
    - a list of supporting software tools that are required to compile and/or run your application (other than Hadoop itself), and web links to where they can be downloaded
    - optional: descriptions of the output files you included (this isn't necessary if they're all just the same, but if you have anything special/unusual going on, you should clarify that)
    - optional: any other information that would help me understand what in all the files you submitted or that would help me compile and run your project on Elastic MapReduce

Interested in Public Data Sets?

Going beyond this course, if you have interest in public data sets, a list of large data sets that are available conveniently in AWS can be found here:

[https://aws.amazon.com/public-datasets/ (Links to an external site.)Links to an external site.](https://aws.amazon.com/public-datasets/)