Using Amazon's Elastic MapReduce

Use Elastic MapReduce to run a MapReduce job on the cloud

Write your own Mapper and Reducer programs to use in a Streaming MapReduce Job.

Further Analysis of the Data

Processing a single file sequentially like we did in the previous page does not really answer the question of "What was the most popular page for the month of June 2013" or how many hits did any particular page get on a particular day. To answer this we must:

- ***aggregate*** the view counts and
- generate a ***daily timeline*** of page views for each article we are interested in.

In order to process such a large dataset (~65 GB compressed), we will setup an Elastic MapReduce job Flow. You will have to write simple Map and Reduce programs in the language of your choice

In this part of the project you will understand some of the key aspects of Elastic MapReduce and run an Elastic MapReduce job flow.

Note: For this checkpoint, assign the tag with Key: `Project` and Value: `1.2` for all resources

Introduction to MapReduce

The MapReduce programming model, pioneered by Google is designed to process big data using a large number of machines. In a MapReduce program, data, stored as **Key/Value pairs**, is processed by a **Map** function. Mappers output a transformed set of Key/Value pairs, which are subsequently processed by a **Reduce** function (Figure 1.1).

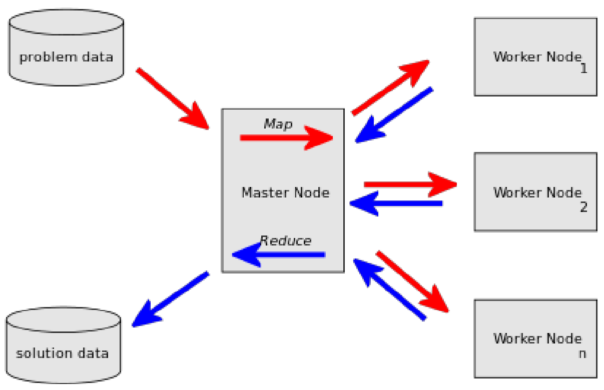

Figure 1.1: MapReduce Overview (Image from Wikipedia)

MapReduce was designed to run in large server farms similar to machines that are deployed at Google's data centers and is proprietary. Hadoop is an open-source implementation of Google's MapReduce, which will be covered in detail in the Unit, "Programming Models". Amazon's Elastic MapReduce is an implementation of Hadoop, and is designed for quick provisioning of Hadoop clusters and fast transfers to/from S3.

In this part of the project you will understand some of the key aspects of Elastic MapReduce and run an Elastic MapReduce job flow.

An Example Elastic MapReduce Job flow : Wordcount

This example is a python script implementation of a classic Hadoop example. In the map phase Word Count takes a directory of text input files, scans through all of them, outputting a single word (key) and the number "1" (value) per each line. The aggregate reducer takes input, line-by-line, from all the mappers and sums each instance of that Key/Value pair. The final output is a list with each word, and the number of times that word occurred in all the input files.

You can see the code here.

If you want to try running this job flow, perform the following steps:

1. In the AWS Management Console, click **Services** and click on **Elastic MapReduce**
2. In the Elastic MapReduce view, click **Create Cluster**.
3. At the top of the page, click on **Configure Sample Application** and select Wordcount.
4. Enter an S3 location for your program output folder. Please **do not** use the bucket you created for project hand-ins. Create another bucket on S3 for this as well as other requirements.
5. Make sure logging and debugging is enabled , and click OK.
6. In the **Cluster Configuration** section:
   - Enter a name for the cluster
   - Enter an S3 location for your log folder, if you have not already. Please **do not** use the bucket you created for project hand-ins. Create another bucket on S3 for this as well as other requirements.
   - Make sure logging and debugging is enabled.
7. In the **Tag** section:
   - Tag your cluster as discussed in the project primer and in the recitation demos.
8. In the **Software Configuration** section:

- Use the AMI 2.4.2 (Hadoop 1.0.3) distribution from Amazon.
- There is no need to install any additional applications for this example.

9. In the **Hardware Configuration** section:
   - Use the **m1.small** instance for both the master and core nodes
   - Specify 2 core nodes.
   - There is no need to use any task nodes for this example.

10. In the **Security and Access** section:
    - Specify an EC2 keypair that you've already created in the AWS Account Setup page.
    - There is no need to add any IAM user access or IAM roles in this example.

11. In the **Bootstrap Options** section, proceed without any bootstrap actions
12. Finally, in the **Steps** section, you can see a step named ""wordcount" has already been configured for you. Click on the edit icon to see the detail of this step. For the purpose of this example, you may use the default input location, or specify your own input to test.
13. Set **Auto-terminate** to **No**, so that you can log on to your cluster even after the job is finished if you need to debug the job. Make sure you manually terminate your cluster after you are done from the EMR console!
14. Review all of your options and click the **Create Cluster** button after everything looks OK. Remember that after this point, you will be charged for the cluster.

Amazon will then provision a Hadoop cluster with 1 master node and 2 data nodes, and submit the job to it. This takes about 5 minutes to instantiate the virtual machines, configure them, etc. You can monitor your job progress from the console and inspect the output and log S3 locations for information regarding your job.

Writing your own Mappers and Reducers

Now lets get back to processing the wikipedia dataset. In this part you will write your own mappers and reducers to perform the following tasks on the entire 1-month input dataset. Create an EMR Job Flow to:

1. Filter out elements based on the rules discussed in the previous page
2. Aggregate the pageviews from hourly views to daily views
3. Calculate the total pageviews for each article
4. For every article that has page-views over 100,000, print the following line as output:

```
<total month views>\t<article name>\t
                  <date1:page views for date1>\t <date2:page views for date2>
...
```

5. **Note**: Print out the page views for all dates in a single line
6. **Getting the input filename from within a Mapper**: As the date/time information is encoded in the filename, Hadoop streaming makes the filename available to every map task through the environment variable named `map.input.file`. For example, the filename can be accessed in python using the statement `os.environ["map.input.file"]`, or in Java using the statement `System.getenv(“map.input.file”)`.

When running your EMR job flow, use spot instances if their price is less than on-demand pricing. For the checkpoint, We will require you to arrive at a job flow configuration that will finish in less than **1 hour, but not using more than 5 USD** (calculated using on-demand pricing). We suggest using the EMR Ruby Client to setup multiple job flows to test various type and number of instances to arrive at an appropriate configuration.

When you are ready, proceed to complete the checkpoint quiz below:

checkpoint

Elastic MapReduce