Parallel Programming using EMR

Show Submission Password

Module	Open	Deadline
Parallel Programming using EMR	01/25/2016 00:01 -0500	01/31/2016 23:59 -0500

- ✔ Big Data Analytics
- ✔ Elastic Map Reduce
- ✔ Analysis of Monthly Pageviews
- ✔ Success

Introduction

Learning Objectives

This weekly module will encompass the following learning objectives:

1. Explore a large-scale dataset
2. Process a large-scale dataset using MapReduce
3. Use Elastic MapReduce to run a MapReduce job on the cloud
4. Understand the benefits of frameworks such as MapReduce to analyze large volumes of data

Warnings

EMR jobs are extremely expensive. Make sure you start small. There are two components to the cost: a per-instance EMR fee and the actual EC2 cost for an instance. Use spot instances to reduce the second component.

Do the WordCount example first so you use existing code and are familiar with setting up a cluster.

The AssessMes are not for clicking around and hacking through. Make sure you understand the cost calculations.

General Details

The following table contains some general information about this project module (P1.2):

Applicable Languages

- TAs can help you if you use Java/Python
- Apart from those, you may use any language that EMR streaming supports. However, the TAs may not be able to help you resolve problems in other languages.
- We require Java 7 or Python 2.7, as these are the versions installed on the grader/your instance.

Tasks	Total Budget	Cloud Platforms
2 (MapReduce code and Data Analysis)	$15	AWS only

The 21st century has been marked by an explosion in the volume and capacity of wireless communications, the World Wide Web, and the Internet. We are now part of a global marketplace of billions of users who produce, share and access petabytes of data. For consumers, for example, this simplified access to producing, disseminating and accessing digitized information has led to an overload of choices. While this may or may not be good for consumers, it also means that sellers and producers of products and services must compete for an increasingly smaller share of the market. With most competitors selling similar (or even identical) products, a key differentiator that determines the popularity of a product is advertising.

Before the "Big Data" age, businesses would market their products using a shotgun approach- selling randomly and indiscriminately to a broad audience. However, the effectiveness of this approach is generally quite low, as the broad audience is generally uninterested in the product being sold. The rise in the availability of data, which could reflect consumer habits and choices, could be valuable to businesses in order to improve the effectiveness of their advertising. Data creates value in several ways, including by enabling experimentation, segmenting populations and supporting decision-making with automation.

Consider the specific use case of trending topics that we are working with. Instead of mass marketing their product, a company can analyze, in near real-time, the viral topics of the day, and design contextual advertising based on a popular topic that is relevant to their product. The Network Advertising Initiative found in 2009 that targeted advertising produced almost three times as much revenue per ad as non-targeted advertising, and was two times more effective at converting ad-clickers into buyers. For even a medium-sized company, this can mean a difference of millions of dollars in sales.

In this module, we will perform some simple analytics to find trending topics from last month's Wikipedia page views (i.e. trends from December 2015). We will start by filtering out English Wikipedia articles exactly like we did in Project 1.1, then aggregate all the pagecounts data for all of December 2015, condense the most important results into a small output file and perform some post-processing to gain real insight into the events of the month.

Big Data Analytics

Processing a single file sequentially like we did in the last module does not really answer the question of "What was the most popular page for the month of December 2015?" or "How many hits did any particular page get on a particular day?"

To be able 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 functions or programs in the language of your choice.

In this module you will understand some of the key aspects of Elastic MapReduce (EMR) and run an Elastic MapReduce job flow. You will need to clearly understand the pricing and tagging model of EMR (https://aws.amazon.com/elasticmapreduce/pricing/) before we start.

Resource Tagging

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

Project Grading Penalties

The following table outlines the violations of the project rules and their corresponding grade penalties for this project.

These rules apply for the week starting January 25 and ending on January 31.

Violation	Penalty of the project grade
Using more than $15 of AWS resources	10%
Using more than $30 of AWS resources	100%
Using any "Project" tags apart from "Project":"1.2"	10%

Failing to tag all your instances (including all of instances in the EMR cluster)	10%
Attempting to hack/tamper the autograder in any way	100%

Introduction to MapReduce

The MapReduce (http://en.wikipedia.org/wiki/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.

MapReduce was designed to run in large server farms similar to machines that are deployed at Google's data centers and is proprietary. Hadoop (http://hadoop.apache.org) is an open-source implementation of Google's MapReduce, which will be covered in detail in the OLI Unit on Programming Models. Hadoop presents MapReduce as an analytics engine and, "under the hood," employs the Hadoop Distributed File System (HDFS). HDFS partitions input datasets into fixed-size chunks (blocks), distributing them on participating cluster nodes. Jobs can subsequently process HDFS blocks in parallel at distributed machines, thus exploiting the parallelism enabled by partitioning datasets. MapReduce breaks jobs into multiple tasks denoted as map and reduce tasks. All map tasks are encapsulated in the "map phase", and reduce tasks are encompassed in the "reduce phase". The map phase can have one or many map tasks, and the reduce phase can have zero or many reduce tasks.

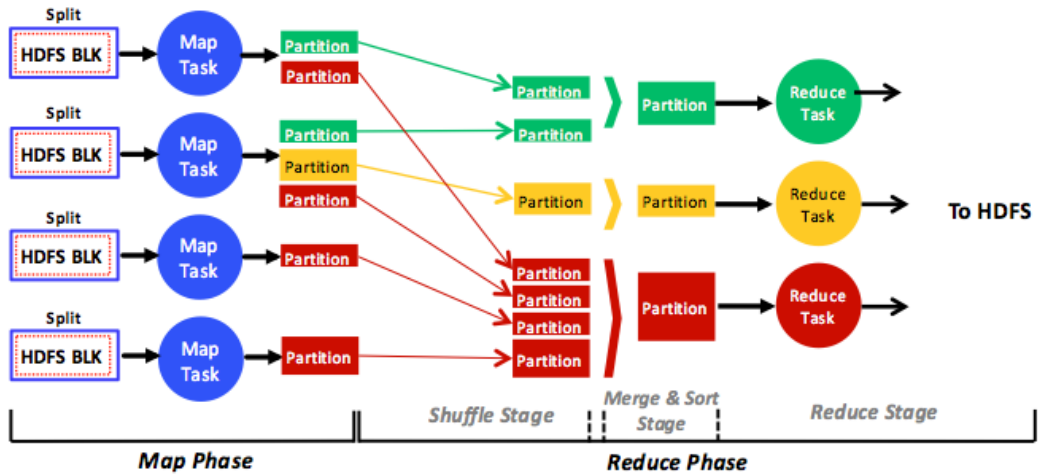

Figure 1: MapReduce Overview

Figure 1 demonstrates a full, although simplified, view of the MapReduce analytics engine. Map tasks operate on distributed HDFS blocks, and reduce tasks operate on map tasks' output, denoted as intermediate output, or partitions. Each map task processes one or many distinct HDFS blocks (more on this shortly), and each reduce task processes one or many partitions. In a typical MapReduce program, the map tasks that are running across all the input HDFS blocks are the same and the reduce tasks running across all the partitions are also the same.

Video: Introduction to MapReduce

Amazon's Elastic MapReduce is a PaaS implementation of Hadoop, 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.

> **Warnings**
> Please read the write-up and AWS resources about EMR carefully before starting an EMR cluster. EMR clusters can be very expensive so make sure you understand fully the requirements of the project before provisioning any resources.

Example EMR Job Flow: Wordcount

In this section, we cover a simple EMR job that performs wordcount.

Java | Python

The following video will walk through the process of writing a Streaming Elastic MapReduce program in Java:

Video: Writing a Streaming Program in Java for Elastic MapReduce

Once you have your code ready the following video contains the instructions on how to run the EMR job flow

Video: Running a Java Streaming Program on EMR

In addition, you can find the mapper (https://s3.amazonaws.com/15619public/examples/wordcount_mapper.java) and reducer (https://s3.amazonaws.com/15619public/examples/wordcount_reducer.java) code for the Java implementation.

Running the Wordcount MapReduce Job

In the example explained in the video, the map phase of 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 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 occurres in all input files.

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 **Go to advanced options**.
4. In the **Advanced Cluster Configuration** section:
    - Enter a name for the cluster
    - Enter an S3 location for your log folder, if you have not already. You can specify a location on a bucket you have already created eg: s3://mybucket/logs
    - Make sure logging and debugging is enabled.
    - For bucket names, we strongly recommend using only lowercase English letters [a-z] and/or digits [0-9].
    - Please avoid periods (.), dashes (-), uppercase English letters [A-Z].
5. In the **Tag** section:
    - Tag your cluster as discussed in the project primer and in the recitation demos.
    - Remember that spot request tags **may not** propagate to the underlying instances. You will also need to tag your cluster once it is up and running.
6. In the **Software Configuration** section:
    - Use the EMR-4.0.0 (Hadoop 2.6.0) distribution from Amazon.
    - There is no need to install any additional applications for this example.
7. In the **Hardware Configuration** section:
    - Use the **m3.xlarge** instance for both the master and core nodes
    - Specify 2 core nodes.
    - There is no need to use any task nodes for this example.
    - You may use different number and type of instance only if you have enough budget.
8. 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.

9. In the **Bootstrap Options** section, proceed without any bootstrap actions
10. Finally, in the **Steps** section, add a new **Streaming program** step.
    - In the **Mapper** column, specify the **name (not the s3 path)** of the mapper file (python) or **command** to execute the mapper file (Java). In the video this was **wordcount_mapper.py**. Do the same for the **Reducer** column.
    - Specify the s3 location of the **input** file.
    - Specify the s3 location of the **output**, this will be created by EMR. This folder **should not** exist, or your step will fail.
    - In the **Arguments** column, use "-files" option to specify the location of the mapper file on s3. For the video example(python), enter " `-files s3://15319videodemo/python/wordcount_mapper.py,s3://15319videodemo/python/wordcount_reducer.py` ". For java, you need to give the path to your jar file(s) or class file(s). To add a jar file, enter " `-files s3://your_jar_file_path_from_S3/jar_filename1.jar,s3://your_jar_file_path_from_S3/jar_filename2.jar,...` ". Adding a class file is similar.
11. 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 with the EMR console!
12. 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 core nodes, and submit the job to it. This takes about 5 minutes to instantiate the virtual machines and configure them. 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 let's 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.

Tasks to Complete

To complete Project 1.2, you need to finish the following tasks:

1. Design a MapReduce Job Flow to:
    i. Filter out elements based on the rules discussed in Project 1.1 (./85).
    ii. Getting the input filename from 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 variables `mapreduce_map_input_file` . For example, the filename can be accessed in python using the statement os.environ["mapreduce_map_input_file"], or in Java using the statement System.getenv("mapreduce_map_input_file")
    iii. Aggregate the pageviews from hourly views to daily views.
    iv. Calculate the total pageviews for each article.
    v. For every article that has page-views over 100,000, print the following line as output (`\t` is the tab character):

    ```
    [total month views]\t[article name]\t[date1:page views for date1]\t[date2:page views for date
    ```

2. Once you have designed and tested your MapReduce job flow on a small portion of the dataset, please run it on the entire dataset of December 2015 using Elastic MapReduce.
3. The dataset can be found at `s3://cmucc-datasets/wikipediatraf/201512/` .
4. Please note the cluster configuration and runtime in minutes of your solution.
5. Please **do not** use "." (periods) or, in general, any other non alphanumeric characters in your bucket name (the bucket to which your mapper and reducer code is uploaded), otherwise the EMR job might fail.
6. You may want to preserve your cluster by unchecking the "Terminate on failure" option and adding steps manually in the EMR web console.
7. Once the results are ready, launch a t1.micro AMI `ami-bcd8f8d6` to make the submission.
8. Download your results file from S3 to your AMI.
9. Merge your results into one output file.
10. Test your Mapper and Reducer using `submitter` .
11. Complete the questions asked in `runner.sh` .
12. Submit your final answers using `submitter` .

Notes and Suggestions

Controlling Costs
Given that you have a limited budget to do this part of the project, make sure you have thoroughly developed and tested your MapReduce program before launching an expensive EMR cluster. Using Spot instances will allow you to save significantly on EMR clusters.

Platform Version
Please use Python 2.7 or Java 1.7. EMR streaming jobs do not support Java 1.8+ or Python 3+.

Python code should have the `#!/usr/bin/env python` or similar shebang on line 1.

Output Rules
The output should conform to the following specifications:

1. Date should be in `yyyymmdd` format
2. Dates should be sorted in chronological order. This means that `20151201` appears before `20151202`
3. Print out the page views for all dates in a single line

Here is an example of the output line expected from the reducer:

```
10   Dopamine    20151201:1    20151202:2    20151203:0    20151204:0    20151205:1    20151206:0    20
```

Each line must have 31 days, even if there were zero page views for any particular day.

Troubleshooting EMR
The following video covers common troubleshooting scenarios for EMR:

Video: Troubleshooting a Streaming Program on Elastic MapReduce

Grading

After completing the above exercise, you are expected to do some analysis of the results and answer certain questions for this week's project module. The questions are present in the file `/home/ubuntu/Project1_2/runner.sh`. You can verify and submit your results using the given auto-grader in the AMI. To use the auto-grader, do the following:

1. Go to the auto-grader folder located at `/home/ubuntu/Project1_2`.
2. The auto-grader consists of three files, `runner.sh`, `submitter` and `references`. You have permissions to edit `runner.sh` and `references` files.
3. If you have completed mapper and reducer, you can submit your files through the auto-grader executable `submitter`. Run the executable using the command `./submitter` from the auto-grader folder.
4. The `submitter` Usage:

   ```
   ./submitter –a andrewId –l python,java
   ```

   If you write your mapper and reducer in python, place mapper.py and reducer.py on the same folder as `submitter`, and then run

   ```
   ./submitter –a msakr –l python
   ```

   If you write your mapper and reducer in java, place Mapper.java and Reducer.java in the same folder as `submitter`, and then

   ```
   ./submitter –a msakr –l java
   ```

5. The first column will show 20 points if you have a correct mapper and reducer in a submission. It is of course possible that your code may pass our auto-grader but yet fail when run on the full dataset using EMR, mainly due to configuration issues.
6. If you get your filtered output from EMR, please aggregate them into a file called `output` and place in the same folder as `submitter`. You will need to work on it to answer some questions in `runner.sh`.
7. Edit the script `runner.sh` to include the commands/code used to answer the checkpoint questions. Using bash scripting is recommended. Do not move any of the provided files. If you are using any external scripts, ensure that you are calling the correct scripts from `runner.sh`. Please ensure that you are placing all your code in the same folder and also assume that the dataset is present in the current folder. Please read the dataset in your code assuming it's present in the current folder (i.e do not use the absolute path to access the dataset).
8. Edit the text file `references` to include all the links that you referred to for completing this project. Also include the Andrew IDs of all the other students who you might have discussed general ideas with when working on this project in the same file.
9. You can run the auto-grader by typing `./runner.sh` from the autog-rader folder. Running this script should print out the answers to all the questions. Please ensure that the answers are printing correctly before using submitter.
10. Once you have completed all the questions, you can submit the answers to the evaluation system using the auto-grader executable `submitter` again. You should be able to see your scores on the website in a few minutes. There is no limit on the number of submissions allowed before the project deadline. However, each submission must be separated by at least 60 seconds.
11. It is important to know that we may test your answer on different data sets. You can only get full marks if you pass all of our tests.

Good job!!! You have solved the AssessMes for this week.