MapReduce on AWS

Advanced Concepts of Cloud Computing

2024

Abstract

In this lab assignment, you will get hands-on experience running MapReduce on AWS. You will also learn how to fit problems/algorithms into the MapReduce paradigm. In the first part of the assignment, you will have to run a simple wordcount problem similar to what you saw during the lectures. Next, you will apply your newly acquired skills to answer few business intelligence questions by designing map/reduce operations for solving the friend recommendation problem.

1 Introduction

MapReduce is a programming model and processing technique designed for distributed computing on large datasets. It simplifies the processing of vast amounts of data across clusters of computers by breaking tasks into two main phases: the Map phase, which filters and sorts data, and the Reduce phase, which summarizes the results. In the context of AWS, MapReduce is primarily implemented through Amazon EMR (Elastic MapReduce), allowing users to process and analyze big data efficiently and cost-effectively in the cloud.

For this assignment, you will use Hadoop and Spark, benchmark their performance against each other on the same problem, and finally implement your own simplified EMR.

It is important to stop EC2 instances when you are not working on them. You will have 50 CAD credits provided by AWS for three assignments of this course. You should manage your expenses carefully. The overall goals of this lab assignment are:

- Get hands-on experience running MapReduce on AWS.
- Run big data operations on AWS using Hadoop and Spark.
- Use the codebase you have developed for your 1st assignment to setup your instances.
- Learn how to fit problems/algorithms into the MapReduce paradigm.

2 Big Data Processing

2.1 What is Hadoop

Apache Hadoop is an open-source framework designed for distributed storage and processing of large datasets across clusters of computers. It's built on two core components: the

Hadoop Distributed File System (HDFS) for storage and MapReduce for processing. HDFS distributes data across multiple nodes in a cluster, providing high-availability and fault-tolerance. MapReduce, Hadoop's processing engine, breaks tasks into smaller subtasks that can be executed in parallel across the cluster. Hadoop is particularly well-suited for batch processing and can handle structured, semi-structured, and unstructured data.

2.2 What is Spark

Apache Spark is a more recent open-source, distributed computing system that was developed to address some of Hadoop's limitations. Spark provides in-memory processing capabilities, which significantly speeds up data processing compared to Hadoop's disk-based processing. It offers a more flexible and easier-to-use programming model with support for Java, Scala, Python, and R. Spark includes libraries for various tasks such as SQL and structured data processing (Spark SQL), machine learning (MLlib), graph processing (GraphX), and stream processing (Spark Streaming).

2.3 Hadoop vs Spark

The main differences between Hadoop and Spark lie in their processing speed, ease of use, and specific use cases. Spark is generally faster due to its in-memory processing, while Hadoop is better for very large datasets that don't fit in memory. Spark offers a more user-friendly API and is more versatile, supporting both batch and real-time processing, whereas Hadoop is primarily designed for batch processing. Hadoop includes its own distributed file system (HDFS), while Spark does not, though it can work with HDFS and other storage systems. In practice, Spark is often used on top of Hadoop, leveraging HDFS for storage while using Spark's processing capabilities for faster computation.

2.4 What is Amazon EMR?

Amazon EMR is a cloud-based big data platform service provided by AWS. It's designed to process and analyze massive amounts of data quickly and cost-effectively using open-source tools such as Apache Hadoop, Apache Spark, and other big data frameworks. EMR allows users to easily scale their compute resources up or down based on processing needs, while AWS handles the complexity of setting up, configuring, and managing the cluster, reducing operational overhead. It integrates well with other AWS services like S3 for storage, EC2 for compute, and IAM for access control. Common applications of EMR include log analysis, web indexing, data transformations (ETL), machine learning, financial analysis, and scientific simulation. The service offers cost-effectiveness by allowing users to spin up clusters as needed and terminate them when the job is done, paying only for the resources used. EMR also supports various programming languages and interfaces, making it accessible to data scientists, analysts, and engineers.

3 Why Learn MapReduce

MapReduce is a powerful programming model and processing technique designed to efficiently handle large-scale data processing tasks across distributed computing environments. At its core, MapReduce breaks down complex data processing jobs into two main phases:

- The Map phase, which filters and sorts data.
- The Reduce phase, which summarizes and aggregates results.

Any operation that requires processing data, can be framed and solved as a MapReduce problem. Learning to think in MapReduce provides several advantages: First, it provides a foundational understanding of distributed computing principles, which are crucial for any work that is done on large amounts of data. Second, it teaches a problem-solving methodology that can be applied beyond just data processing tasks. The MapReduce paradigm encourages developers to think in terms of parallelizable operations and efficient algorithm design for large-scale problems; of breaking down large tasks, into semi-independent operations that can be carried out regardless of the underlying problem's scale. This can be useful for multiple fields, from software engineering to data science, as the ability to decompose complex problems into simpler, manageable parts that can be processed concurrently will allow you to design fast, scalable programs.

4 Counting Words

The WordCount problem is the "Hello World" of big data processing. Given a large corpus of text, our goal is to count the number of times each word has been encountered. Given that this problem is well-known, we will use it as a template to experiment with Hadoop, Spark, and a simple bash scripting command to see the benefits of MapReduce paradigm.

4.1 Instructions on Setting Up Hadoop

You will find detailed instructions about Hadoop installation at the following website: Installing Hadoop on Ubuntu

4.2 Instructions on Setting Up Spark

You will find detailed instructions about Spark's installation at the following website: How to install Spark on Ubuntu

Note that to install Spark you must first have installed Hadoop as Spark uses functionalities from HDFS and YARN.

4.3 Experiments with WordCount

4.3.1 Running WordCount.java on HDFS

In order to process a text file with Hadoop, you first need to download the file to a directory in your Hadoop account, then copy it to the Hadoop Distributed File System (HDFS) so that

the Hadoop Name Node and Data Nodes can share it. Download a copy of James Joyce's Ulysses book available at this link: Gutenberg textual data. Create an "input" directory in the Hadoop Distributed File System (HDFS) and copy the data file pg4300.txt to it:

```
bin/hadoop jar hadoop-mapreduce-examples-<ver>.jar wordcount /input/
   pg4300.txt /output
# After job completion the output will be in the output directory
# Concatenate the results and store them in an output.txt
hadoop fs -cat /output/part-r-00000 > output.txt
```

Listing 1: Running Hadoop's own WordCount implementation

Where `<ver>` is the version of the Hadoop you have installed. You can view Hadoop's own documentation about how its WordCount solution works in here: MapReduce Tutorial

4.3.2 Running WordCount with Linux

```
cat pg4300.txt | tr ' ' '\n' | sort | uniq -c
```

Listing 2: Running Wordcount with bash commands

This bash command opens the contents of pf4300, replaces spaces with newer lines, sorts the words alphabetically, and counts the occurrence of each unique word.

4.3.3 Apache Spark

Similar to Hadoop, Spark also has its own implementation of the WordCount problem. You can simply run their code on 'pg4300.txt' to see how easy it is to use spark as the backend for your MapReduce problems.

- Spark's Java WordCount implementation
- Spark's Python WordCount implementation

By now, you should have counted the words on 'pg4300.txt' using Hadoop, Spark, and Linux.

4.4 Who Wins?

Create one T2.large Linux Ubuntu instance and set up Hadoop and Spark on it. Make sure that all required packages are installed. You can use Java or Python (PySpark).

Use below datasets for all your experiments:

- https://tinyurl.com/4vxdw3pa
- https://tinyurl.com/kh9excea
- https://tinyurl.com/dybs9bnk
- https://tinyurl.com/datumz6m
- https://tinyurl.com/j4j4xdw6

- https://tinyurl.com/ym8s5fm4
- https://tinyurl.com/2h6a75nk
- https://tinyurl.com/vwvram8
- https://tinyurl.com/weh83uyn

Run WordCount program both on Hadoop and Spark machines 3 times on all above datasets and measure the execution time, plot all your data points and compare/report the average. You can launch the program in Hadoop by giving the JAR file as an argument and submit a work on Spark using the built-in bash script similar to what we did above.

4.5 How does this relate to your 1st assignment

In your first assignment, you practiced setting up your clusters and installing applications on them using code and running commands on your machines remotely. Here, the underlying principles are the same:

- Setting up the machines using code is the same.
- You don't need to set up a loadbalancer.
- Instead of installing fastAPI, you will install Hadoop and Spark.
- Instead of sending requests, here you will run commands for processing your scripts. Same way of installing Hadoop and Spark.
- You will report the metrics the same way but just use the outputs of your MapReduce script instead of using CloudWatch.

5 Let's think MapReduce

Write a MapReduce program in you preferred programming language that implements a simple "People You Might Know" social network friendship recommendation algorithm. The key idea is that if two people have a lot of mutual friends, then the system should recommend that they connect with each other. For this problem, you only need to design and implement the mapping algorithm (creating key, value pairs) and the reduce algorithm (processing the key, value pairs). You do not need to use Hadoop or Spark.

5.1 Instructions

5.1.1 Input Data

Download the input file from Moodle. The input file contains the adjacency list and has multiple lines in the following format: `<User><TAB><Friends>`. Here, `<User>` is a unique integer ID corresponding to a unique user and `<Friends>` is a comma separated list of

unique IDs corresponding to the friends of the user with the unique ID `<User>`. Note that the friendships are mutual (i.e., edges are undirected): if A is friend with B then B is also friend with A. The data provided is consistent with that rule as there is an explicit entry for each side of each edge.

5.2 Algoritm

For each user U, recommends N = 10 users who are not already friends with U, but have the most number of mutual friends in common with U.

5.2.1 Output

The output should contain one line per user in the following format:

`<User><TAB><Recommendations>`.

Where `<User>` is a unique ID corresponding to a user and `<Recommendations>` is a comma separated list of unique IDs corresponding to the algorithm's recommendation of people that `<User>` might know, ordered in decreasing number of mutual friends. Even if a user has less than 10 second-degree friends, output all of them in decreasing order of the number of mutual friends. If a user has no friends, you can provide an empty list of recommendations. If there are recommended users with the same number of mutual friends, then output those user IDs in numerically ascending order.

6 Automation and Infrastructure as Code

Your solution should be end-to-end automated. This means that during your demo, running the commands for download and setting up Hadoop/Spark, running the benchmarks, retrieving the results, and running your friend recommendation solution **SHOULD** be done by running a simple bash script. As such, you will need to develop your solutions using AWS' SDKs depending on the programming language that you prefer: AWS SDKs

7 Working in Groups

You should work in groups of two, three, or four.

8 Report

One submission per group is required for this assignment. To present your findings and answers questions, you have to write a lab report on your results using LATEX template. In your report should write:

- Experiments with WordCount program.
- Performance comparison of Hadoop vs. Linux.

- Performance comparison of Hadoop vs. Spark on AWS.
- Describe how you have used MapReduce jobs to solve the social network problem.
- Describe your algorithm to tackle the social network problem.
- Presents your recommendations of connection for the users with following user IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993.
- Summary of results and instructions to run your code.

One submission per group is required for this assignment. You must submit only one ZIPPED file named assignment1.zip which includes:

- report.pdf: Contains the name of the team members and all the material of the report. You can design the format and sections at your convenience.
- Your code: This will include all the necessary commands to execute and show the results of your benchmarking. It should have enough comments and descriptions to understand every single section of it.