CSE530 Distributed Systems
Winter 2023
Project

Deadline: April 30, 2023 - 11:59 pm IST

Problem Statement

In this project, you will be required to implement all the components of the MapReduce framework by yourself without using an existing distributed computing framework such as Apache Hadoop or Apache Spark.

Since it may not be possible to implement a generic MapReduce framework which can be used for running any application in a distributed manner, you should ensure that your implementation supports the following three applications/queries:

1. Word Count
2. Inverted Index [Record-Level Inverted index]
3. Natural Join [A good tutorial]

You are free to have a separate implementation for supporting each of the above-mentioned applications or you may have a single implementation which supports all three applications.

Note: As this is a project (not assignment), we have purposefully kept it a bit open-ended so that you have sufficient freedom to make reasonable assumptions and design choices. At the same time, we have tried to provide all the essential details which are required for you to complete the project.

Learning Objectives

The objective of this project is to gain a deeper understanding of how MapReduce-like frameworks are actually implemented and how such frameworks can be used to process large datasets in a distributed computing environment.

By implementing all the components of MapReduce and running multiple mappers and reducers in parallel without using an existing distributed computing framework such as Apache Hadoop or Apache Spark, you will gain a solid understanding of distributed computing and parallel processing for big data analytics. This project is a great way to gain hands-on experience with the MapReduce programming model and prepare yourself for working on big data projects.

Setup:

1. For the sake of simplicity, we will deploy the entire MapReduce framework on one machine. Hence, each mapper and each reducer as well as the master will be a separate process on the same machine. Additionally, each mapper and each reducer will store their intermediate/output data in a separate file directory in the local file system.
    a. Mapper persists intermediate data
    b. Reducer persists final output data
2. **Use of gRPC for RPCs is mandatory for this project.**
3. You may use any programming language you wish to use. We recommend using Python.

Components to be implemented:

Your implementation should necessarily have the following components:

1. **Master:** Master program/process is responsible for running and communicating with the other components in the system. When running the master program, the user should provide:
    a. input data location
    b. number of mappers (M)
    c. number of reducers (R)
    d. output data location
2. **Input split:** You will need to write code to divide the input data into smaller chunks that can be processed in parallel by multiple mappers. For simplicity, you may assume that the input data consists of multiple data files and each file is processed by a ~~separate~~ **single** mapper.
3. **Map:** You will need to write code to apply the Map function to each input split to generate intermediate key-value pairs. The Map function takes a key-value pair as input and produces a set of intermediate key-value pairs as output. The output of each Map function should be written to a file in the mapper's directory on the local file system. Note that each mapper will be run as a different process.
4. **Partition:** The output of the Map function (as mentioned in mapper above) also needs to be partitioned into a set of smaller partitions. In this step, you will write a function that takes the list of key-value pairs generated by the Map function and partitions them into a set of smaller partitions. The partitioning function should ensure that all key-value pairs with the same key are sent to the same partition. Each partition is then picked up by a specific reducer during shuffling and sorting.
5. **Shuffle and sort:** You will need to write code to sort the intermediate key-value pairs by key and group the values that belong to the same key. This is typically done by sending the intermediate key-value pairs to the reducers based on the key. Shuffling and sorting is done by the reducer.
6. **Reduce:** You will need to write code to apply the Reduce function to each group of values that belong to the same key to generate the final output. The Reduce function

takes a key and a set of values as input and produces a set of final key-value pairs as output. The output of each Reduce function should be written to a file in the reducer's directory on the local file system. Note that each reducer will be run as a different process.

Sample input and output files:

To have uniformity in evaluation, we will provide sample input and output files for each application. You should ensure that your code runs successfully on the provided format of sample input files, and generates the final output files in the same format as the sample output files respectively.

You may access the sample files here.

Deliverables

1. Submit a zipped file containing the entire code.

Evaluation:

TAs will run your submitted code on a set of input files for each of the three applications for which your implementation should run successfully and give the expected output. TAs will match your result with the expected output files with them. The format of input and output files will be the same as the sample files mentioned above.