The Chinese University of Hong Kong

JC2104 FYP Term-End Report

Term 1

Imsong Jeon

1155087995

1 December 2021

Table of Contents

Introduction

We are living in an era of data. Enormous amount of data is produced every day, and data is a very useful and valuable resource once it is processed and analyzed properly. However, the size of these data sets, as known as big data, is far beyond the ability of commonly used computer to store, manage and process the data within a reasonable timeframe. For this reason, a distributed system is inseparable from big data, machine learning, and other cutting-edge information technologies involving big data. Distributed system is a scalable system of multiple networked computers that communicate and coordinate to perform shared tasks.

My final year project is closely related to the distributed system. The objective of my FYP is to learn how to use distributed systems for large-scale data analytics, and perform an implementation or a new development on the system side based on my learning. For this term (the 1st term), I have studied about commonly used distributed system frameworks, Hadoop and Spark, learned to use basic functions (configure, start the clusters, compile executables, and submit jobs) of Ursa framework, became familiar with high level APIs of Ursa, and started to implement few distributed algorithms on Ursa.

Background

The modern trend of analytic utilizing distributed system started to emerge with the release of the Google File System and MapReduce papers published by Google in 2003 and 2004. Then, Apache Hadoop based on these papers was developed and became popular. Since then, numerous software and frameworks for big data analytic such as Apache Spark have been released.

Among these frameworks, I am using Ursa for my FYP. Ursa is a framework with improved resource scheduling and job execution based on a new system designed by Professor James Cheng and his students. Ursa is written in C++. I was provided with a cluster account for accessing the Ursa cluster along with few basic information about the usage of Ursa.

Majority of final year project topics are about coming up with a solution of a defined problem or with a solid outcome such as a software or an application. However, there is no clearly defined problem to solve and work on for my FYP. Instead, my FYP is more about the experience and the training of the ability to self-learn from the given materials. In addition, the further objective of my FYP is to come up with a problem and try to propose a solution based on my learning.

Learning Process and Outcomes

Preparation for FYP

I studied Hadoop MapReduce and Spark as a preparation for the FYP before the start of this term. Professor advised me to do so, and I believe these are two reasons behind this preparation. First, getting familiar with the use of Hadoop MapReduce and Apache Spark is important for the FYP because studying and programming on these two interfaces help to get familiar with the common workflow of a distributed system. Second, Ursa provides data transformation APIs very similar to Spark transformations. Similarly, there are also similarities between Ursa and Hadoop MapReduce such as the serializing and deserializing functions. Overall, the preceding study of Hadoop and Spark helped me a lot during my learning of Ursa during the term.

Getting Started with Ursa

In the beginning of the term, I started the use of Ursa cluster. I followed the instruction provided by the tutor, but soon I faced some problems. First, I was not certain which directory was the Ursa project home directory. The tutor helped me to realize that h-axe folder is the project home directory. Second, I was unable to start the workers properly. This was because SSH keys were not installed so I solved this problem by installing SSH keys between the master and workers. Third, after the workers were started properly, I tried to submit some example code such as hello world.cc to check whether cluster is working fine. However, I found out that a binary file of the Job Manager (JM) executable and a binary file of the Job Process (JP) executable. I realized that the .cc files in the example folder has to be compiled into JM and JP files using CMake. With no prior experience with CMake, it took me several days to learn about CMake on internet and read through cmake files in the Ursa folder. Finally, I found out that the option for building examples in CMakeList.txt was set to OFF. I was able to build the JM and JP executables for example codes after switching this option. However, executables were not enough to submit a job to Ursa; a .ini job configuration file was also needed. I referenced some other existing job config files to create a corresponding .ini and .json file for the job. After these trials and errors, I was able to finish setting up the Ursa cluster and finally submit a job to the cluster properly.

Example Codes and High-Level APIs

The Ursa cluster included several example codes of implementation of different distributed algorithms. These examples were great material for me to learn and get familiar with high-level APIs. For each example code, I observed the structure and workflow of the program, and paid extra attention to the high-level APIs during the process. These APIs are built using various primitives. For this reason, for each API, I also had to thoroughly go

through the primitive source codes that this API is based on, such as codes in src/base and src/common directories, in order to understand how the API work. I invested a lot of time in this process of studying and getting familiar with the structure, APIs, and source codes of Ursa. I was not able to fully comprehend the low-level codes, but I was able to understand the general workflow of major APIs in high-level. For instance, the following is the high-level workflow of ReduceBy() transformation based on my understanding:

Input: *key_selector, combiner*, *n_partitions*

Output: *n* partitions of reduced dataset

1. For each partition in DatasetParition loaded from Task Context:
    i. divide this partition into *n_partitions* partitions
    ii. For *n_partitions*:
        A. Sort data based on key returned from *key_selector* function
        B. Combine data with identical key using *combiner* function
        C. Serialize data into binstream format and put it in msg of this *n*th partition
2. For *n_partitions*:
    i. Deserialize binstreams in msg of this *n*th partition into data format
    ii. Sort data based on key returned from *key_selector* function
    iii. Combine data with identical key using *combiner* function
    iv. Return this partition of reduced dataset

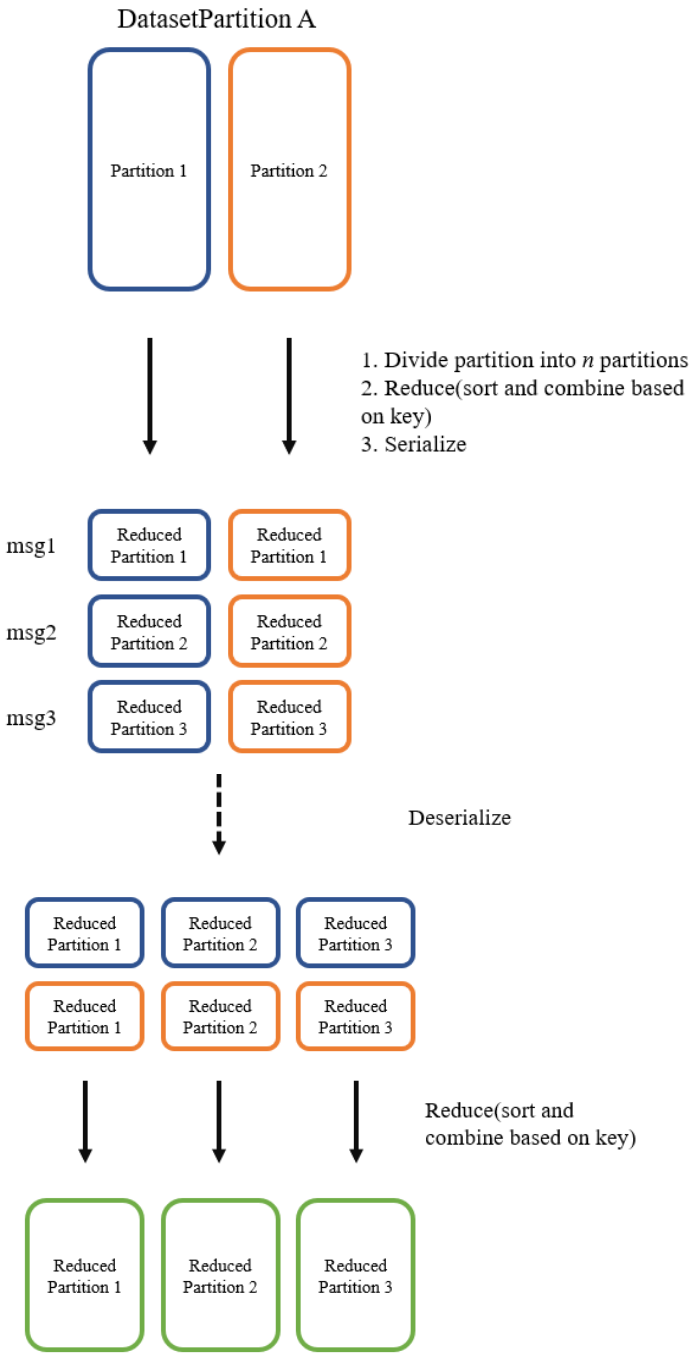

Figure 1. ReduceBy transformation ($n\_partitions = 3$)

Also, I temporally added extra lines of code to output the results into the log file so that I can check the result of an algorithm. In addition, the paper, "Improving Resource Utilization by Timely Fine-Grained Scheduling", helped to understand the system design and objectives of Ursa. Eventually, I was able to start writing my own code of distributed algorithm for Ursa

after days of learning from example codes, APIs, and source codes of Ursa.

Implementation of Distributed Algorithms

After some learning process and experience with example codes, I began the implementation of simple distributed algorithm for Ursa alongside the continuous study of example codes, APIs, and source codes. First, I implemented a word length count program based on the word count example. This program takes in lines of text from an input file and utilizes the basic MapReduce model to derive counts for each unique word length. In the original word count example, the program outputs the number of distinct words. On the other hand, the word length count program outputs the key value pair where key is the unique length of words and value is the count.

For the second implementation, I wanted to work on an algorithm that involves a graph because many graph algorithms are commonly used for big data analytics. For this reason, I decided to implement a parallel Dijkstra algorithm, which I was familiar with from CSCI4180. This program takes in an input file in the following format:

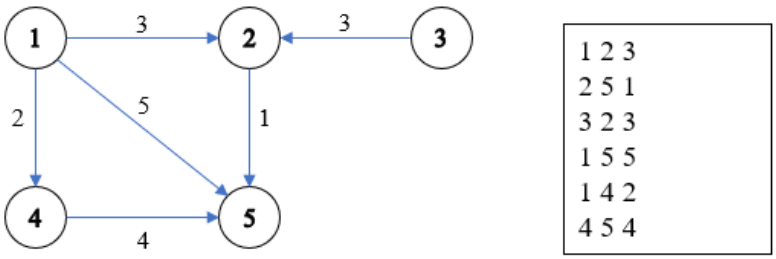

Figure 2. sample input

Each line represents an edge of weight (the third number) going out from the first number from the second number. From this input, the program uses a Node class and MapReduce to construct a directed graph in the form of an adjacency list. Then, it goes through multiple

iterations of MapReduce to find the minimum distance from the start node to other reachable nodes in the graph in the way similar to Breadth-first search. In the end, this program outputs the minimum distance from the start node to each reachable nodes together with its parent node in the following format:

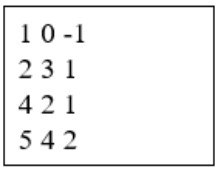

Figure 3. sample output

The first number is the node ID, the second number is the minimum distance from the start node, and the third number is the parent node ID. During the development process, I learned that it is necessary to declare operator functions for comparison, serialization, and deserialization for a class in order to utilize the class in the transformation APIs, which is similar to the implementation of Writable class in Hadoop. The process of implementation of distributed algorithms for Ursa enhanced my understanding of system design of Ursa, and I am planning on continuing this process.

Plans and Ideas

In the coming term, I would focus on implementation of more advanced distributed algorithm with knowledge and experience gained from this term. Also, I would like to be more creative and experimental to make contribution to Ursa framework if possible. To begin with, I am planning on implementing distributed principal component analysis (PCA) for Ursa. Many big data analytics handle large number of features, so distributed PCA could be utilized to reduce redundant features from the data and achieve more efficient use of resources in Ursa. In addition, I would like to design and experiment with a distributed

algorithm that can process input data in real-time if possible. It would be interesting to experiment with a distributed algorithm with real-time interaction in Ursa, where resources are scheduled efficiently.

Conclusion

I have learned not only about Ursa, distributed algorithms, C++ language, and Linux, but also about how to learn from given materials on my own, working on my FYP in this term. There were countless difficulties and obstacles during the process of learning, and I tried my best to overcome these problems by myself. Overall, I have been fascinated and inspired by the system design and implementation of Ursa and provided example codes. Also, I was motivated by the fact that I was able to comprehend and appreciate them. I am looking forward to learn more and enhance my understanding of Ursa and distributed system for large-scale data analytics by working on my FYP in the next term.