COVID-19 Contact Tracing
Based on Hadoop MapReduce

Yelei Wu, Weitao Chen

1.Abstract

As we all know, since March 2020, the Covid-19 has spread extremely rapidly, and the number of infected people is endless. Since the outbreak, the number of infected people has continued to rise. As of November this year, the number of new infections worldwide in a single day has exceeded 600,000 (as shown in figure 1.1).

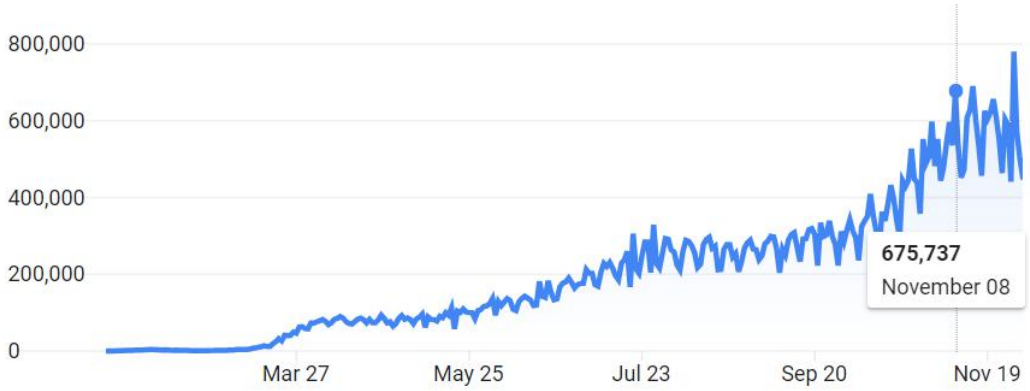


Figure 1.1

In addition, covid-19 has also brought serious economic problems. On the one hand, the spread of the epidemic has accelerated globally, which has led to a frustration of investor confidence, which has triggered turmoil in the financial and capital markets; on the other hand, countries have strictly restricted the movement of people and transportation to control the spread of the epidemic, pressing the pause button on the economy. The government's actions put pressure on the economic operation from both the consumer side and the production side. The sharp decline in the World trade volume and World GDP (as shown in Figure 1.2) will not only impact enterprises but also affect employment and the livelihoods of countless people. Therefore, how to quickly and effectively control the spread of the epidemic has become our current top priority.

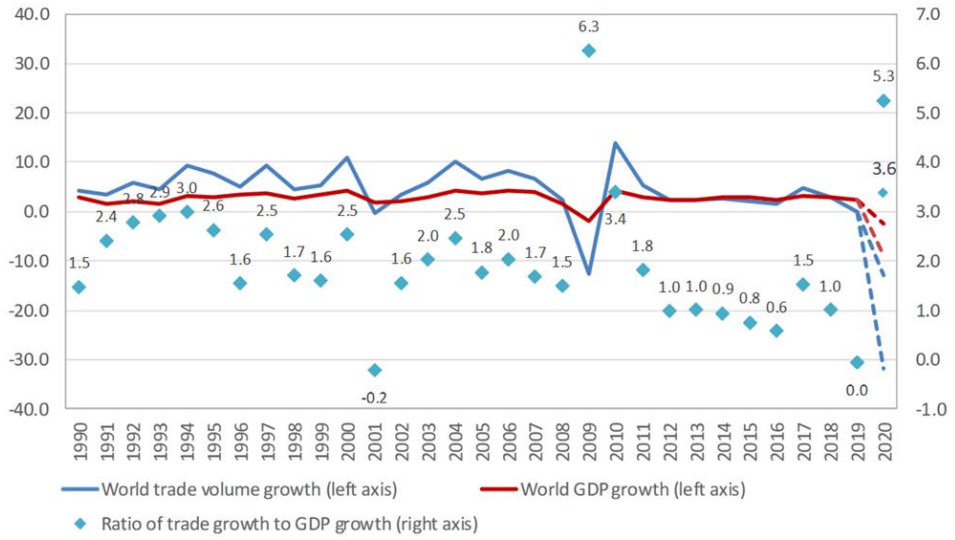


Figure1.2

The government was determined to control it, but due to the large number of infected people, it was impossible to track and test susceptible people. As the number of infected people increases, the most direct recommended method is to iterate all susceptible people and test them. For each patient, test his friends, his roommate, his colleagues is necessary. This method is feasible, but the data set that needs to be calculated at the same time is too large, and we need a scalable method.

As we all know, unlike the traditional data processing system, using parallel distributed algorithm clusters on the server, MapReduce can implement for processing huge amount of data in short (as shown in Figure 1.3) . Therefore, we propose a hypothesis whether it is possible to use a MapReduce-based big data algorithm to track the susceptible people contacted by each infected person, and to test in order according to the frequencies they contact with the patients.

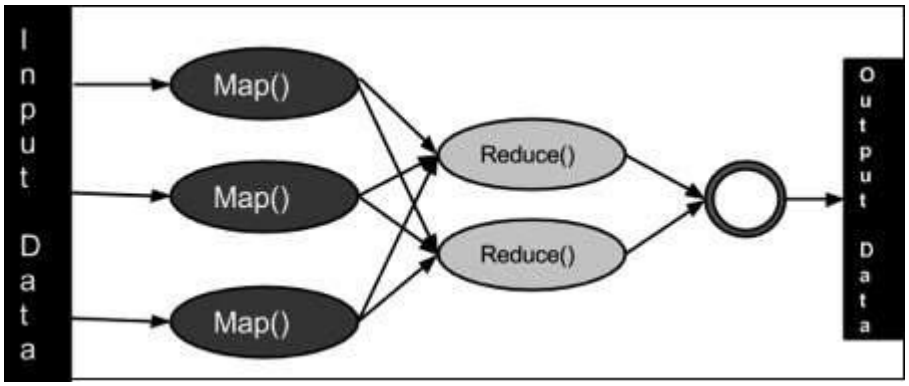


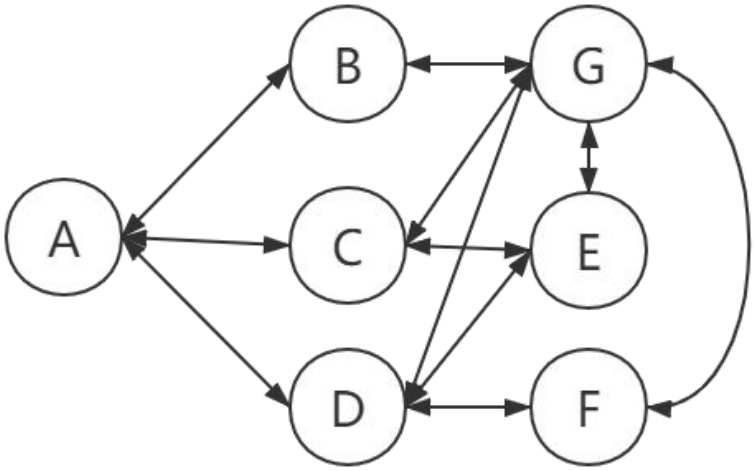
Figure1.3

Notice that the Precondition is that we have already found out those people who have been in contact with COVID-19 patients. And then in this program we need to arrange them in order according to the frequencies they contact with the patients.

2. Solution

For people who have direct contact with infected people, we test them first. For people who have indirect contact, we will test them in order according to the 'value'.

For example, suppose A is the patient, and the relationship is as follows:



And the input show that B,CD direct contact WITH A so we first TEST BCD.

But the question is that, how can we determine who should be tested first among E F G after testing bcd?

The input shows that A and E have been indirectly contact through C, so Let A-E be 1. And also, A and E also indirectly contacted again through D, So Let A-E be 1 again.

After that, we add the same relationship together, and then we can get A-E as 2.Finally we sort them from largest to smallest, and we know that A-G is 3 and G should be tested first after BCD.

3. Research Methodology

3.1MapReduce

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It consists of two parts, the storage part and the other part of the data processing part. The storage part is called HDFS, and the processing part is called MapReduce.

For the storage part, HDFS distribute the data among multiple data nodes. It is highly reliable since data are replicated to cope with node failure. The HDFS structure is a Namenode storing metadata and a number of DataNodes storing contents/blocks of files, which allows schedule computation close to data.

For MapReduce, it builds on the observation that many tasks have the same structure. It has 3 tasks: map, reduce, and shuffle. As following figure shows, map

task performs data transformation. Reduce task combines results of map tasks. Shuffle task sends output of map tasks to right reduce tasks. Key/value pairs form the basic data structure in MapReduce. In our project, key means the people who is likely to get COVID-19, while value means the probability.

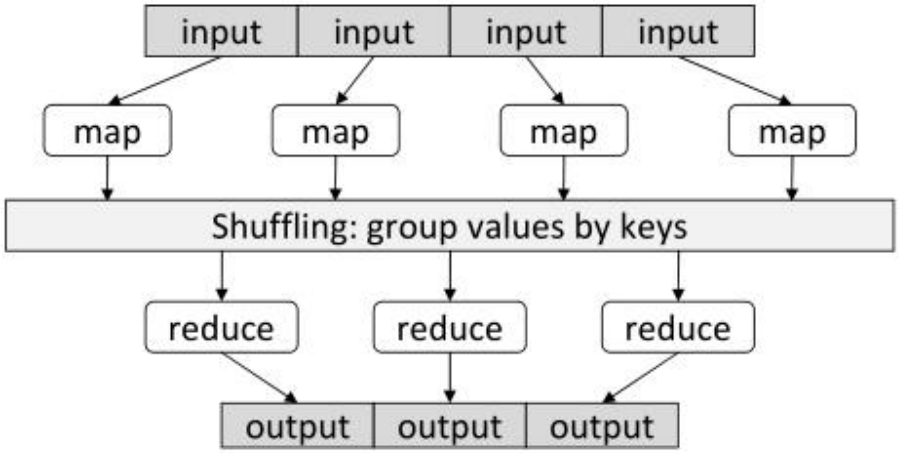


Figure 1: Illustration of the MapReduce framework

The advantage of Hadoop:

- Large-scale:
  - Handle a large amount of data and computation
- Distributed:
  - Distribute data & work across a number of machines
- Batch processing
  - Process a series of jobs without human intervention

3.2 EC2

In our project, we will use AWS EC2 for running our Hadoop MapReduce framework. According to the documentation, Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud.

Amazon EC2 provides the following features (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html)

— Virtual computing environments, known as *instances*

— Preconfigured templates for your instances, known as *Amazon Machine Images (AMIs)*, that package the bits you need for your server (including the operating system and additional software)

— Various configurations of CPU, memory, storage, and networking capacity for your instances, known as *instance types*

— Secure login information for your instances using *key pairs* (AWS stores the public key, and you store the private key in a secure place)

— Storage volumes for temporary data that's deleted when you stop, hibernate, or terminate your instance, known as *instance store volumes*

— Persistent storage volumes for your data using Amazon Elastic Block Store (Amazon EBS), known as *Amazon EBS volumes*

— Multiple physical locations for your resources, such as instances and Amazon EBS volumes, known as *Regions* and *Availability Zones*

— A firewall that enables you to specify the protocols, ports, and source IP ranges that can reach your instances using *security groups*

— Static IPv4 addresses for dynamic cloud computing, known as *Elastic IP addresses*

— Metadata, known as *tags*, that you can create and assign to your Amazon EC2 resources

— Virtual networks you can create that are logically isolated from the rest of the AWS Cloud, and that you can optionally connect to your own network, known as *virtual private clouds* (VPCs)

4. Algorithm and technology

PAIRWISE SIMILARITY ALGORITHM:

This algorithm is based on the famous paper: (*Elsayed T , Lin J J , Oard D W . Pairwise Document Similarity in Large Collections with MapReduce[C]*)

It is a MapReduce algorithm for computing pairwise document similarity in large document collections. There are two steps in this algorithm, index and pairwise similarity.

Index: For each document , we build a standard inverted index, where each term is associated with a list of docid's for documents that contain it and the associated term weight. Mapping over all documents, the mapper, for each term in the document, emits the term as the key, and a tuple consisting of the docid and term weight as the value. The MapReduce runtime automatically handles the grouping of these tuples, which the reducer then writes out to disk, then generating the postings

Pairwise similarity: Mapping over each posting, the mapper generates key tuples corresponding to pairs of docids in the postings. These key tuples are associated with the product of the corresponding term weights—they represent the individual term contributions to the final inner product. The MapReduce runtime sorts the tuples and then the reducer sums all the individual score

contributions for a pair to generate the final similarity score.
Here is the figure shows the process of the algorithm:

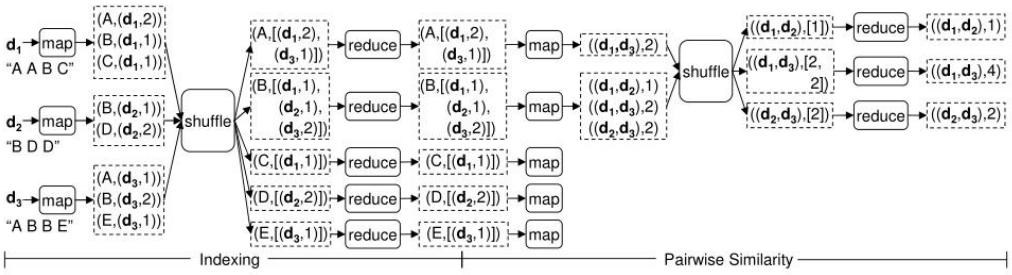
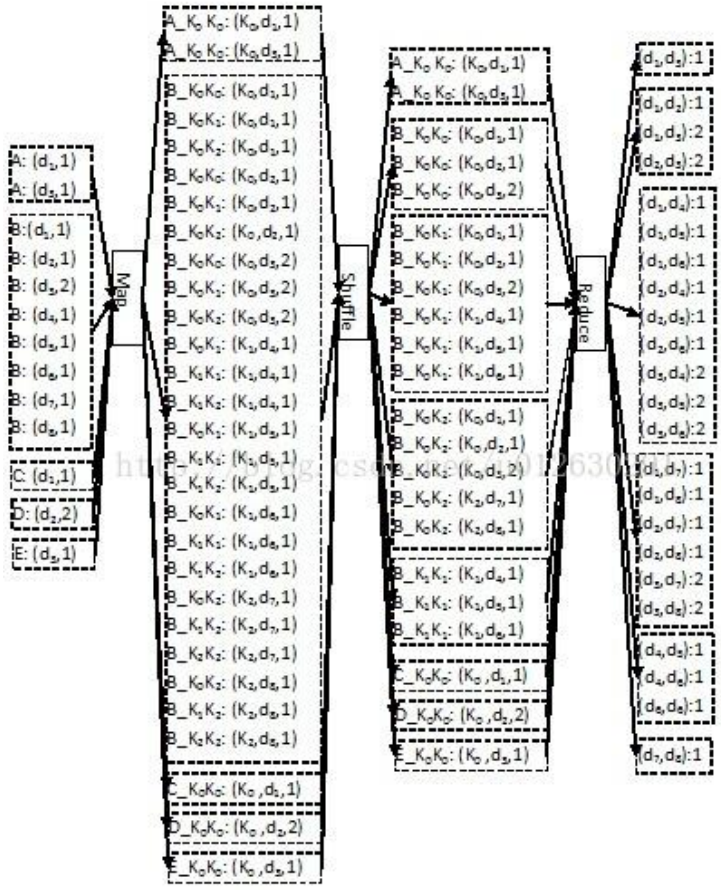


Figure: framework of the algorithm

Now we can take an easy example to show this:



In general, the possibility of getting the virus is related to the time they spent with the people got. Suppose we have COVID-19 datasets: for confirmed cases, we have the people they get touch with for each hour. Therefore, we can just apply the pairwise similarity algorithm.

5. Detailed Implementation

5.1 Experimental evalution

In the experiment, we plan to use Hadoop version 0.16.0 and plan to run the program on a cluster of 20 computers. Each machine has two single-core processors (running at 3.6GHz frequency), 4GB RAM and 100GB disk.

By reviewing the paper 'Pairwise Document Similarity in Large Collections with MapReduce', we know that the use of mapreduce technology will greatly improve the efficiency of our algorithm. In the experiment for stopword removal, the author adopted a 99% cut, which means that the most frequent 1% of terms were discarded. This makes the running time increase linearly with the collection size and the running space is also greatly reduced, which not only reflects a very ideal attribute, but also reflects the superiority of mapreduce technology for big data processing.

5.2 Procedure

For example, the input file INPUT (as shown in Figure 5.1) stores information about 9 people contacting each other. Each line shows two names, which means that two people have had direct contact. First, we assign an ID number to each person, which is stored in the database in a one-to-one correspondence. In the example shown in Figure 5.1, the ID of ANDREW ADAMS is i1, the ID of WILLIAM MITCHELL is i2, the ID of OLIVIA CLARK is i3, the ID of ISABELLA HALL is i4, the ID of SAMANTHA SMITH is i5, and the ID of CHRISTOPHER LEE is i6, the ID of ABIGAIL PEREZ is i7, the ID of JOSEPH SMITH is i8, and the ID of JOSHUA LEWIS is i9.

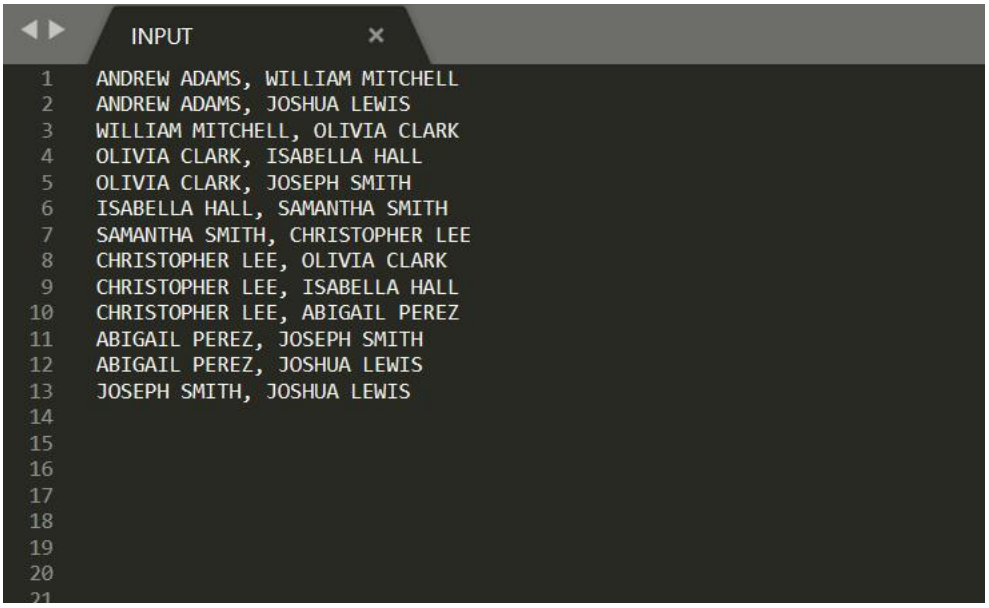


Figure5.1

This algorithm requires two shuffles and calls the idea of MapReduce twice in the process. In the first MapReduce process, for the Map part, suppose our input is'i8, i9', then our output is key ='H' and value ='H, I'; key ='I' and value ='H, I'. Both sets of data indicate that i8 and i9 had close contact. We will deal with the redundant part of the data in the next process.

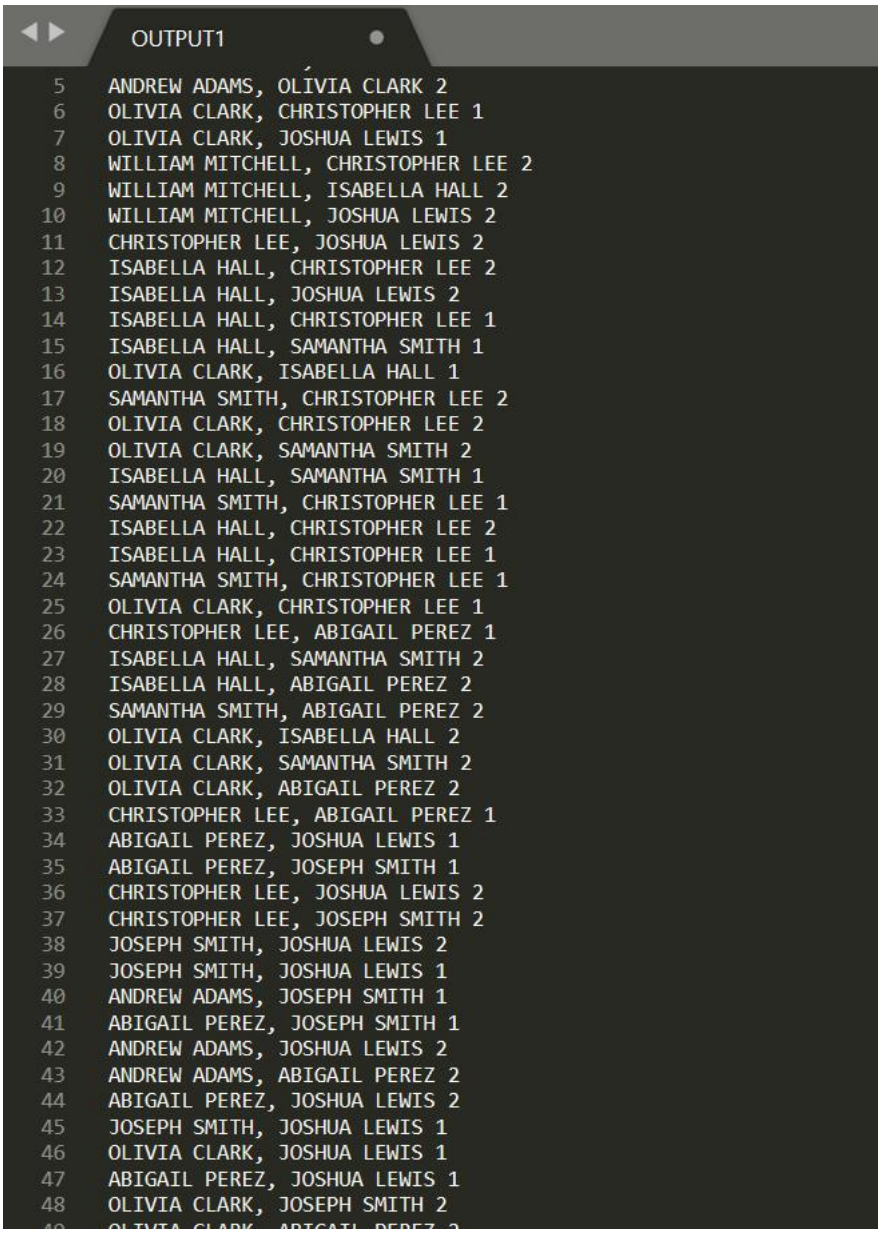


Figure5.2

Then referring to the above steps, we can get a total of 9 key values. For example, after the Reduce process, we know that the value of key = i9 is {'i3, i9','i7, i9','i8, i9'}. In fact, after the Reduce process, we know who i9 has directly contacted. At the same time, according to i3, i7 and i8 have all been in contact with i9, we can say that i3, i7 and i8 have had indirect contact through i9. All in all, we have traversed everyone's contacts. For two people in direct contact, we set the value between them to 1. For two people in indirect contact, we set the value between them to 2. (as shown in Figure 5.2).

After successfully listing the level of contact between two people, our work has not yet been completed. It can be seen that some people have both 'direct contact' and 'indirect contact', which is obviously contradictory. And we will solve this problem in the second round of MapReduce.

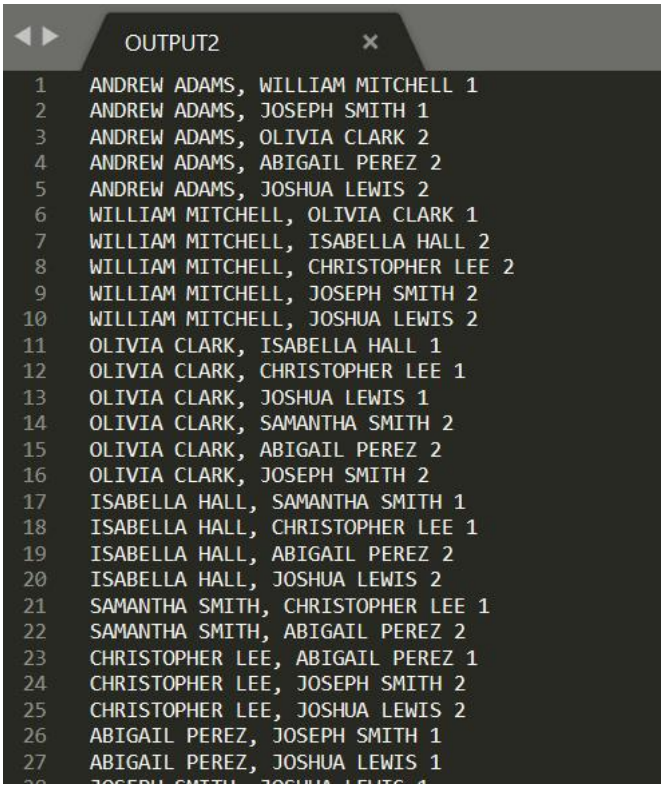


Figure5.3

The input of the second round of MapReduce is based on the output of the first round of MapReduce. In the second round of MapReduce, if two people have had direct contact, then they cannot have an indirect contact relationship. Therefore, in this case, we discard the item with value of 2. And if there is only an item with value of 2 between two people, it means that they have had only indirect contact between each other, and we only keep the item with value of 2(as shown in Figure 5.3).

For example, according to OUTPUT1, the value between i4 and i6 is 2 in line 12, and the value is 1 in line 14. Therefore, we discard the item whose value is 2 and judge that there has been direct contact between i4 and i6. For i1 and i3, there is only an item with value of 2 between them, indicating that there has only been indirect contact between them.

Finally, we can know who have direct contact and who have indirect contact in the crowd. Once it is detected that a certain person in the crowd was infected with COVID-19, we can test the others in order according to the 'value' we got before. In the above example, only indirect contact through one person is involved. In fact, we also plan to include indirect contact through two people in the experiment, and set the value between them to 3, which just need to add a MapReduce process. The general process is consistent with the above.

6. Conclusion

6.1 Conclusion

It has been more than a month since the project started. It can be said that I have used all the knowledge about MapReduce I learned this semester in this project as much as possible. Similarly, in the process of gradually completing the project, I also reviewed the knowledge that I have mastered before but have been a little forgotten. It can be said that I have learned a lot.

With the continuous integration and development of the global information industry, the scale of network resources and data is also increasing. Especially in the fields of scientific research, computer simulation, Internet applications, and e-commerce, the amount of data is showing a trend of rapid growth. Traditional data analysis technology has become increasingly unsuitable for the current needs of intensive mass data processing. The implementation of the Map/Reduce model in cloud computing proposed by Google Labs solves these problems.

The project uses MapReduce, a knowledge that will be practically applied in various enterprises. Even if you have had contact before, it is still a huge challenge to have two people complete a large project. It took a lot of time to learn MapReduce-related knowledge, and the content in the class was still slightly insufficient for specific project operations.

Although the project itself is not perfect, this is due to my lack of practical experience and defects. However, I will make unremitting efforts to maintain the spirit of continuous learning in my future work, and strive to increase my professional knowledge and practical experience, and strive to make the project better and better.

6.2 Future work

Spark is another generation of computing framework proposed after Hadoop, and it also focuses on offline batch processing. However, Spark has achieved a performance increase of 10-100 times on the basis of Hadoop's native computing engine MapReduce, thus surpassing the native MapReduce in the Hadoop ecosystem and is gradually being reused.

Spark inherits the characteristics of Hadoop MapReduce and is a typical Master/worker architecture. This architecture divides the computing tasks and assigns them to multiple Slaves, that is, Map. After the Slaves complete the tasks assigned to them, they are then aggregated on the Master, which is Reduce. This

is the idea of MapReduce.

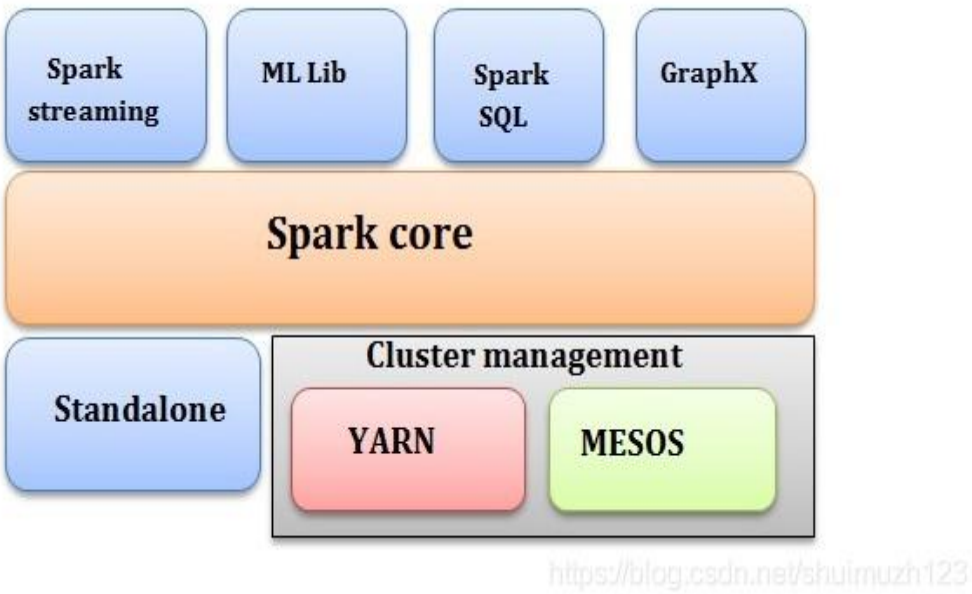


Figure6.1

Hadoop components can be used with Spark in the following ways (as shown in Figure 6.1):

- **HDFS**：Spark can run on top of HDFS to take advantage of distributed storage.
- **MapReduce**：Spark can be used with MapReduce in the same Hadoop cluster, or it can be used separately as a processing framework.
- **YARN**：Spark applications can be run on YARN (Hadoop NextGen).
- **Batch processing and real-time processing**：MapReduce and Spark are used together. MapReduce is used for batch processing and Spark is used for real-time processing.

Although both Hadoop Mapreduce and Spark are big data frameworks, current technology trends show that MapReduce will be replaced by a new generation of big data processing platforms such as Spark. Therefore, we plan to expand our horizons, learn about big data processing knowledge such as Spark, and continue to optimize this project to strive for early success.

7 Contribution

1.Data input
Weitao Chen(Lead)
Yelei Wu

2.Analysis of algorithm
Yelei Wu(Lead)
Weitao Chen

3.Database
Yelei Wu(Lead)
Weitao Chen

4.Code implementation
Yelei Wu(Lead)
Weitao Chen

5.Report
Weitao Chen(Lead)
Yelei Wu

REFERENCE

[1] Learn the Concept of Key-Value Pair in Hadoop MapReduce, (AUGUST 22, 2019). Retrieved from https://data-flair.training/blogs/key-value-pair-in-hadoop-mapreduce/

[2] MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, Donald Miner, (May 27, 2019). Retrieved from 'MapReduce Design Patterns'

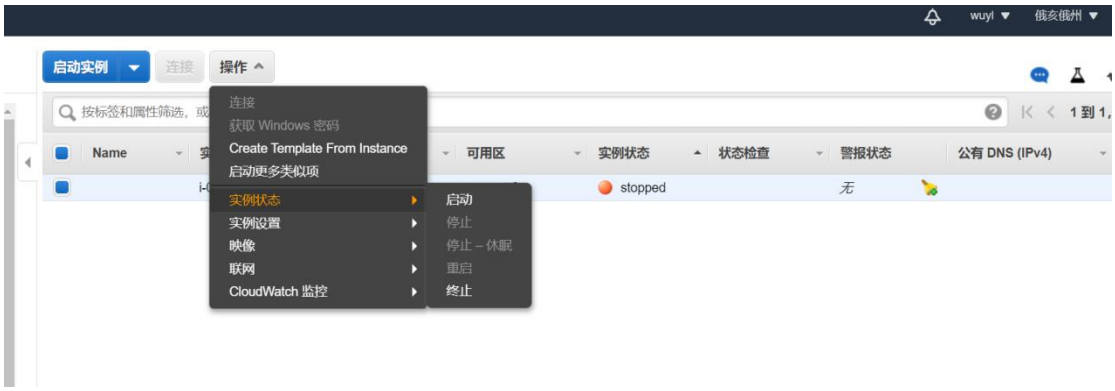
[3] How MapReduce Works?,(FEB16, 2017). Retrieved from https://www.educba.com/how-mapreduce-work/

[4] MapReduce Tutorial, (AUGUST 22, 2019). Retrieved from https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[5] Hadoop Big Data Analytics Market: Impact of COVID-19, Market Watch. Retrieved from https://www.marketwatch.com/press-release/hadoop-big-data-analytics-market-impact-of-covid-19-on-global-demand-trend-analysis-top-companies-and-opportunity-outlook-2025-2020-09-25?tesla=y

APPENDIX

1. Starting EC2 instance:



Connecting EC2 by SSH:

Install the Hadoop package

– Log into your EC2 instance and then execute:

- wget http://apache.cs.utah.edu/hadoop/common/hadoop-3.1.4/hadoop-3.1.4.tar.gz
- tar xvf hadoop-3.1.4.tar.gz
- sudo yum install java-1.8.0-devel
- Edit ~/.bashrc by adding the following:
  - export JAVA_HOME=/usr/lib/jvm/java
  - export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
  - export HADOOP_HOME=/home/ec2-user/hadoop-3.1.2
  - export PATH=${JAVA_HOME}/bin:${HADOOP_HOME}/bin:${PATH}

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
export JAVA_HOME=/usr/lib/jvm/java
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
export HADOOP_HOME=/home/ec2-user/hadoop/hadoop-3.1.4
export JAVA_HOME=/usr/lib/jvm/java

export PYSPARK_PYTHON=python3 # needed to use Python 3

export PATH=${JAVA_HOME}/bin:${HADOOP_HOME}/bin:${PATH}
[2;3R
```

- Comment out /etc/hadoop/core-site.xml

```
<configuration>
<!--property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
</property-->

</configuration>
```

Execute Hadoop MapReduce:

- hadoop com.sun.tools.javac.Main Count.java
- jar cf wc.jar Count*.class
- hadoop jar wc.jar Count input-hello output-hello

2. Code:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class Sum {



    public static class SumMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private IntWritable result = new IntWritable();
         @Override
         protected void map(Object offset, Text rows, Context context) throws
IOException, InterruptedException {

              String[] cols = rows.toString().split(",");
            String ss = cols[2].replace('\'', ' ');

              if (Float.parseFloat(ss)>=20.0) {
                 String sss = cols[3].replace('\'', ' ');
                 sss=sss.trim();
                 int ii=Integer.parseInt(sss);

                    context.write(new Text(cols[0]), new IntWritable(ii));
              }
```

```
        }

    }

    public static class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();
        protected void reduce(Text province, Iterable<IntWritable> nursing, Context context)
                    throws IOException, InterruptedException {
                    int sum = 0;
                for (IntWritable val : nursing) {
                    sum += val.get();
                }
                result.set(sum);
                context.write(province, result);

        }
    }


    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        for (int i = 0; i < otherArgs.length; i++) {
            System.out.println(otherArgs[i]);
        }
        Job job = Job.getInstance(conf, "Sum");
        job.setJarByClass(Sum.class);
        job.setMapperClass(SumMapper.class);
        job.setReducerClass(SumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        for (int i = 0; i < otherArgs.length - 1; ++i) {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
```

```
            }
            FileOutputFormat.setOutputPath(job,
                    new Path(otherArgs[otherArgs.length - 1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class covid19{


      public static class SumMapper extends Mapper<Object, Text, Text,
IntWritable> {
            private IntWritable result = new IntWritable();
             @Override
             protected void map(Object offset, Text rows, Context context) throws
IOException, InterruptedException {

                    String[] cols = rows.toString().split(",");
                String ss = cols[2].replace('\'', ' ');

                  if (Float.parseFloat(ss)>=20.0) {
                     String sss = cols[3].replace('\'', ' ');
```

```
                sss=sss.trim();
                int ii=Integer.parseInt(sss);

                    context.write(new Text(cols[0]), new IntWritable(ii));
            }

        }

    }

    public static class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();
        protected void reduce(Text province, Iterable<IntWritable> nursing, Context context)
                throws IOException, InterruptedException {
                String[] sum= val.toString().split("\t");
    if (sum[0].equals(key.toString())) {
    hisFriends.add(sum[1]);
    context.write(val, new Text("tag1"));
    }
    if (sum[1].equals(key.toString())) {
    hisFriends.add(sum[0]);
    context.write(val, new Text("tag1"));
    }
    }
    for(int i = 0; i < hisFriends.size(); i++)
    {
    for(int j = 0; j < hisFriends.size(); j++)
    {
    if (hisFriends.elementAt(i).compareTo(hisFriends.elementAt(j)) < 0) {
    Text reduce_key = new Text(hisFriends.elementAt(i)+"\t"+hisFriends.elementAt(j));
    context.write(reduce_key, new Text("tag2"));
        }
    }
```

```
        }
 }

public static class Map2 extends Mapper<Object, Text, Text, Text>
{

protected void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
String[] line = value.toString().split("\t");
      if (line.length == 3) {
      Text map2_key = new Text(line[0]+"\t"+line[1]);
      Text map2_value = new Text(line[2]);
      context.write(map2_key, map2_value);
      }
}
}

public static class Reduce2 extends Reducer<Text, Text, Text, Text>
      {

protected void reduce(Text key, Iterable<Text> values, Context context)
      throws IOException, InterruptedException {
      boolean isdeg1 = false;
      boolean isdeg2 = false;
      int count = 0;
      for(Text val : values)
      {
if (val.toString().compareTo("tag1") == 0) {
      isdeg1 = true;
}
if (val.toString().compareTo("tag2") == 0) {
isdeg2 = true;
count++;
}
}
      if ((!isdeg1) && isdeg2) {
      context.write(new Text(String.valueOf(count)),key);
      }
```

```
    }
    }

     public static void main(String[] args) throws Exception {
          Configuration conf = new Configuration();
          String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
          for (int i = 0; i < otherArgs.length; i++) {
               System.out.println(otherArgs[i]);
          }
          Job job = Job.getInstance(conf, "Sum");
          job.setJarByClass(Sum.class);
          job.setMapperClass(SumMapper.class);
          job.setReducerClass(SumReducer.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(IntWritable.class);

          for (int i = 0; i < otherArgs.length - 1; ++i) {
               FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
          }
          FileInputFormat.addInputPath(job1, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job1, new Path(otherArgs[1]));
    if (job1.waitForCompletion(true)) {
         Job job2 = new Job(conf, "Deg2friend");
         job2.setJarByClass(deg2Friend.class);
         job2.setMapperClass(Map2.class);
         job2.setReducerClass(Reduce2.class);
         job2.setOutputKeyClass(Text.class);
         job2.setOutputValueClass(Text.class);
         FileInputFormat.addInputPath(job2, new Path(otherArgs[1]));
         FileOutputFormat.setOutputPath(job2, new Path(otherArgs[2]));
         System.exit(job2.waitForCompletion(true)? 0 : 1);
    }
          System.exit(job.waitForCompletion(true) ? 0 : 1);
     }
}
```