VE572 — Methods and Tools for Big Data

Midterm Rubric — Summer 2019

TA: Yihao and Yanjun (UM-JI)

This document should be kept secret.

Exercise 1 — Big data or not big data? [20 marks]

This part will count at most 20 marks in total.

1. What is the data size? [2.5+5=7.5 marks]

1. 1G-10G: Spark without Hadoop
2. 10G-100G: Mapreduce with Hadoop
3. 100G+: Spark with Hadoop

2. What is the data type? [2.5+5=7.5 marks]

1. Real time: Spark
2. Batch: Mapreduce

3. What does the company want to do with the data? [2.5+5=7.5 marks]

1. Search: Drill
2. Implement algorithms: Spark

4. Other reasonable answers [5 marks]

Exercise 2 — MapReduce [40 marks]

1. Determine all the FOFs in the following toy example. [(2/3)*15=10 marks]

```
Ali Ben 1
Ali Col 1
Ali Gil 2
Ben Dan 1
Ben Eve 1
Ben Fin 1
Col Dan 1
Col Gil 1
Col Han 1
Dan Eve 1
Dan Han 2
```

```
Eve Fin 1
Eve Gil 1
Fin Gil 1
Fin Han 1
```

2. Write the Hadoop pseudocode for the first MapReduce Job. Assume a simple input text file with a list of names on each line, the user as first field followed by all his friends. For the output we expect a simple text file where each line is composed of a user and a FOF followed by the number of friends they have in common. [15 marks]

- The first MapReduce [5 marks]
- The second MapReduce [5 marks]
- The overall design [5 marks]

```
package com.ve572.e1;

import org.apache.commons.text.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.ArrayList;

public class FindFOF {
    private static String formPair(final String a, final String b) {
        if (a.compareTo(b) < 0) return a + "," + b;
        return b + "," + a;
    }

    public static class Map1 extends Mapper<Object, Text, Text, Text> {
        private Text resultKey = new Text();
        private Text resultValue = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
        ↪  InterruptedException {
            StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",");
```

```
            resultKey.set(tokenizer.nextToken());
            while (tokenizer.hasNext()) {
                resultValue.set(tokenizer.nextToken());
                context.write(resultKey, resultValue);
            }
        }
    }


    public static class Reduce1 extends Reducer<Text, Text, Text, IntWritable> {
        private Text resultKey = new Text();
        private IntWritable resultValue = new IntWritable();

        public void reduce(Text key, Iterable<Text> values, Context context) throws
        ↪ IOException, InterruptedException {
            ArrayList<String> arrayList = new ArrayList<>();
            resultValue.set(0);
            for (Text val : values) {
                resultKey.set(formPair(key.toString(), val.toString()));
                context.write(resultKey, resultValue);
                arrayList.add(val.toString());
            }
            resultValue.set(1);
            for (int i = 0; i < arrayList.size(); i++) {
                for (int j = i + 1; j < arrayList.size(); j++) {
                    resultKey.set(formPair(arrayList.get(i), arrayList.get(j)));
                    context.write(resultKey, resultValue);
                }
            }
        }
    }

    public static class Map2 extends Mapper<Object, Text, Text, IntWritable> {
        private Text resultKey = new Text();
        private IntWritable resultValue = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException,
        ↪ InterruptedException {
            StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",");
            resultKey.set(tokenizer.nextToken() + " " + tokenizer.nextToken());
            resultValue.set(Integer.parseInt(tokenizer.nextToken()));
            context.write(resultKey, resultValue);
        }
    }
```

```
72
73      public static class Reduce2 extends Reducer<Text, IntWritable, Text, IntWritable> {
74          private IntWritable resultValue = new IntWritable();
75
76          public void reduce(Text key, Iterable<IntWritable> values, Context context)
            ↪  throws IOException, InterruptedException {
77              int count = 0;
78              for (IntWritable val : values) {
79                  if (val.get() == 0) return;
80                  count += val.get();
81              }
82              resultValue.set(count);
83              context.write(key, resultValue);
84          }
85      }
86
87      public static void main(String[] args) throws Exception {
88          Configuration conf = new Configuration();
89          conf.set("mapred.textoutputformat.separator", ",");
90  //          conf.set("mapreduce.output.textoutputformat.separator", ",");
91
92          Job job1 = Job.getInstance(conf, "ve572e1ex2.2");
93          job1.setJarByClass(FindFOF.class);
94          job1.setMapperClass(Map1.class);
95          job1.setReducerClass(Reduce1.class);
96
97          job1.setMapOutputValueClass(Text.class);
98          job1.setOutputKeyClass(Text.class);
99          job1.setOutputValueClass(IntWritable.class);
100
101         FileInputFormat.addInputPath(job1, new Path("data.txt"));
102         Path outputPath1 = new Path("output-1");
103         FileSystem fileSystem = outputPath1.getFileSystem(conf);
104         if (fileSystem.exists(outputPath1)) {
105             fileSystem.delete(outputPath1, true);
106         }
107         FileOutputFormat.setOutputPath(job1, outputPath1);
108
109         boolean exitCode = job1.waitForCompletion(true);
110         if (!exitCode) System.exit(1);
111
112         conf.set("mapred.textoutputformat.separator", " ");
113
114         Job job2 = Job.getInstance(conf, "ve572e1ex2.2");
```

```
115         job2.setJarByClass(FindFOF.class);
116         job2.setMapperClass(Map2.class);
117         job2.setReducerClass(Reduce2.class);
118
119         job2.setOutputKeyClass(Text.class);
120         job2.setOutputValueClass(IntWritable.class);
121
122         FileInputFormat.addInputPath(job2, outputPath1);
123         Path outputPath2 = new Path("output-2");
124         if (fileSystem.exists(outputPath2)) {
125             fileSystem.delete(outputPath2, true);
126         }
127         FileOutputFormat.setOutputPath(job2, outputPath2);
128         exitCode = job2.waitForCompletion(true);
129
130         System.exit(exitCode ? 0 : 1);
131     }
132
133 }
```

3. Write the Hadoop pseudocode for the second MapReduce job. Assume the previous output file as input, and as output a simple text file where each line is composed of a user and all his FOF ordered with respect to the number of common friends; for each FOF also display the number of common friends. [15 marks]

- Mapper [5 marks]
- Reducer [10 marks]

```
1  package com.ve572.e1;
2
3  import org.apache.commons.lang3.tuple.ImmutablePair;
4  import org.apache.commons.lang3.tuple.Pair;
5  import org.apache.commons.text.StringTokenizer;
6  import org.apache.hadoop.conf.Configuration;
7  import org.apache.hadoop.fs.FileSystem;
8  import org.apache.hadoop.fs.Path;
9  import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.mapreduce.Job;
11 import org.apache.hadoop.mapreduce.Mapper;
12 import org.apache.hadoop.mapreduce.Reducer;
13 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15
16 import java.io.IOException;
17 import java.util.ArrayList;
```

```java
import java.util.Comparator;
import java.util.stream.Collectors;

public class CountFOF {

    public static class Map extends Mapper<Object, Text, Text, Text> {
        private Text resultKey = new Text();
        private Text resultValue = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
        ↪  InterruptedException {
            StringTokenizer tokenizer = new StringTokenizer(value.toString(), " ");
            String a = tokenizer.nextToken();
            String b = tokenizer.nextToken();
            String count = tokenizer.nextToken();
            resultKey.set(a);
            resultValue.set(b + " " + count);
            context.write(resultKey, resultValue);
            resultKey.set(b);
            resultValue.set(a + " " + count);
            context.write(resultKey, resultValue);
        }
    }

    public static class Reduce extends Reducer<Text, Text, Text, Text> {
        private Text resultValue = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context) throws
        ↪  IOException, InterruptedException {
            ArrayList<Pair<String, Integer>> arrayList = new ArrayList<>();
            for (Text val : values) {
                String[] arr = val.toString().split(" ");
                arrayList.add(new ImmutablePair<>(arr[0], Integer.parseInt(arr[1])));
            }
            arrayList.sort(Comparator.comparing(Pair<String,
            ↪  Integer>::getValue).reversed());
            resultValue.set(arrayList.stream().map(
                    x -> x.getKey() + " " +
                    ↪  x.getValue().toString()).collect(Collectors.joining(", "))
            );
            context.write(key, resultValue);
        }
    }

```

```
58      public static void main(String[] args) throws Exception {
59          Configuration conf = new Configuration();
60          conf.set("mapred.textoutputformat.separator", " ");
61  //          conf.set("mapreduce.output.textoutputformat.separator", ",");
62
63          Job job = Job.getInstance(conf, "ve572e1ex2.3");
64          job.setJarByClass(CountFOF.class);
65          job.setMapperClass(CountFOF.Map.class);
66          job.setReducerClass(CountFOF.Reduce.class);
67
68          job.setOutputKeyClass(Text.class);
69          job.setOutputValueClass(Text.class);
70
71          FileInputFormat.addInputPath(job, new Path("output-2"));
72          Path outputPath = new Path("output-3");
73          FileSystem fileSystem = outputPath.getFileSystem(conf);
74          if (fileSystem.exists(outputPath)) {
75              fileSystem.delete(outputPath, true);
76          }
77          FileOutputFormat.setOutputPath(job, outputPath);
78
79          boolean exitCode = job.waitForCompletion(true);
80          System.exit(exitCode ? 0 : 1);
81      }
82  }
```

Exercise 3 — Course questions [35 marks]

1. HDFS [15 marks]

a) What is the default replication level in HDFS? [1 mark]

The default replication factor is 3.

b) Parallel is often seen as more efficient than serial. When writing a file in HDFS, blocks are first sent to a DataNode which forwards them to another, which sends them to another, and so on...Why is this process not parallelised, i.e. send blocks to all the DataNodes at the same time? [4 marks]

- If the blocks are sent to all the DataNodes at the same time, it will be slow due to the limited throughput. [2 marks]
- Thus when writing a file in HDFS, the blocks are sent to a DataNode at a time. Once the blocks are written in a DataNode, it is no need to forward them to other DataNodes. [2 marks]

c) Explain how to find a file in HDFS. [3 marks]

1. Each datanode announces the blocks it has. [1 mark]
2. The namenode keeps all the information in its memory. [1 mark]
3. When a write occurs an entry is added to the edit log. [1 mark]

d) What are the namespace image and edit log? [3 marks]

- Namespace image: Stroing the entire file system namespace, including the mapping of blocks to files and file system properties. [1.5 marks]
- Edit log: Recording every change that occurs to file system metadata, such as creating a new file in HDFS and changing the replication factor of a file. [1.5 marks]

e) Is it possible to have several NameNodes in a cluster? If so explain how it works. [4 marks]

- It is possible. [1 mark]
- We can use HDFS federation. [1 mark]
- Split the filesystem over several independent namenodes. Each namenode has a namespace and its own pool of blocks [1 mark]
- A namespace with a block pool is called namespace volume and a datanode is not attached to a specific namespace volume. [1 mark]

2. YARN [8 marks]

a) Explain how an application is launched and run using YARN. [3 marks]

1. Application client submit a YARN application to Resource Manager.
2. Resource Manager contacts Node Manager to launch a new container and run Application Master in it. [1 mark]
3. Application Master asks Resource Manager for allocating the resources. [1 mark]
4. Application manager gets the resources information from Resource Manager and it launches the container through other Node Manager. [1 mark]

b) Would you recommend the fair or capacity scheduler? Explain the when and why. [5 marks]

- – Fair scheduling is a method of assigning resources to applications such that all apps get, on average, an equal share of resources over time. [1 mark]
  – It is a good default for small to medium sized clusters [0.5 mark]
  – since it is more flexible and allows for jobs to consume unused resources in the cluster. [1 mark]
- – Capacity scheduler is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster. [1 mark]
  – It's generally used on large clusters with lots of different workloads with different needs [0.5 mark]

  - since it can give each organization capacity guarantees.[1 mark]

3. Briefly explain the similarities and differences between MapReduce and Spark. [7 marks]

- Similarity: Both are highly scalable and can be used in a cluster. [1 mark]
- MapReduce:
  - MapReduce takes two stages to process data, Map and Reduce. It reads and writes from disk and thus slower and has high latency. [1 mark]
  - It uses replication for fault tolerance, which significantly increase the completion times for operations with a single failure. [2 marks]
- Spark:
  - Spark is lightning fast cluster computing tool. It is much faster than MapReduce and has low-latency computing. [1 mark]
  - It uses RDDs and DAG for fault tolerance. If an RDD is lost, it is easy to recompute a new one by using the original transformations. [2 marks]

4. Drill [5 marks]

a) What is Zookeeper, and why is it a core component of Drill's strategy? [2 marks]

- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. [1 mark]
- Drill uses ZooKeeper to maintain cluster membership and health-check information. [1 mark]

b) Explain what is a Drillbit and how it functions. [3 marks]

- A Drillbit is the process running on each active Drill node that coordinates, plans, and executes queries, as well as distributes query work across the cluster to maximize data locality. [1 mark]
- The Drillbit receives the query from a client. A SQL parser in the DrillBit parses the SQL and form a logical plan. [0.5 mark]
- The Drillbit sends the logical plan into a optimizer to optimize and convert the logical plan into a physical plan that describes how to execute the query. [0.5 mark]
- A parallelizer in the Drillbit transforms the physical plan into multiple phases, called major and minor fragments. [0.5 mark]
- These fragments create a multi-level execution tree that rewrites the query and executes it in parallel against the configured data sources, sending the results back to the client or application. [0.5 mark]

Exercise 4 — Simple Hadoop questions [5 marks]

1. Why is ssh needed on the master and workers? How to configure it? [1 mark]

- The master uses ssh protocol to send commands to the workers. [0.5 mark]

- The worker should add the master's public key in ssh configurations. [0.5 mark]

Which Java version is needed by Hadoop, why? [1 mark]

- Java version 8 is needed. [0.5 mark]
- Some APIs are deprecated in the new versions of Java and Hadoop hasn't altered them yet. [0.5 mark]

Why should Hadoop's home be the same across the whole cluster? [1 mark]

Because the master use the same configuration file to find the Hadoop's home on every worker.

How to use `hdfs dfs` command line interface to [2 marks]

(i) list a directory [1 mark]

`hdfs dfs -ls <hdfs path>`

(ii) upload or download a file [1 mark]

- `hdfs dfs -put <local path> <hdfs path>` [0.5 mark]
- `hdfs dfs -get <hdfs path> <local path>` [0.5 mark]