Practical#7

Name: Saurin Anilkumar Prajapati

Roll No.: 19BCE239

Course Code and Name: 2CS702 Big Data Analytics

Batch: D1

Aim:

Analyse impact of different number of mapper and reducer on same definition as practical 4.

Implement any one of the analytic algorithm using mapreduce by handling larger datasets in main memory.

- **PCY/Multi-Hash/SON algorithm**
- **Regression**
- **K-means Clustering**

Algorithm Chosen:

PCY Algorithm

PCY algorithm was developed by three Chinese scientists **Park, Chen, and Yu**. This is an algorithm used in the field of big data analytics for the frequent itemset mining when the dataset is very large.

Consider we have a huge collection of data, and in this data, we have a number of transactions. For example, if we buy any product online it's transaction is being noted. Let, a person is buying a shirt from any site now, along with the shirt the site advised the person to buy jeans also, with some discount. So, we can see that how two different things are made into a single set and associated. The main purpose of this algorithm is to make frequent item sets say, along with shirt people frequently buy jeans.

Code:

I have implemented this algorithm using 2 passes.

FirstPass.java

```
package FrequentItem;


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class FirstPass {
    private static class PCYFirstValue {
        private ArrayList<Integer> mItems;

        public PCYFirstValue() {
            mItems = new ArrayList<>();
        }

        public ArrayList<Integer> getItems() {
            return mItems;
        }
    }

    private static class PCYFirstMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private static final int HASHTABLE_SIZE = 100007;
        private Text word = new Text();

        @Override
        protected void map(Object key, Text value, Context context) throws IOException, In
terruptedException {
```

```
            String[] tokens = value.toString().split(" ");
            PCYFirstValue outValue = new PCYFirstValue();


            for (String token : tokens) {
                outValue.getItems().add(Integer.parseInt(token));
            }
            ArrayList<Integer> items=outValue.getItems();
            Collections.sort(items);

            for(int item:items) {
              word.set(Integer.toString(item));
              context.write(word, one);
            }
            for (int i = 0; i < items.size(); ++i) {
                for (int j = i + 1; j < items.size(); ++j) {
                    int candidates[] = new int[] { items.get(i), items.get(j) };
                    int h=Hash.hash(candidates, HASHTABLE_SIZE);
                    context.write(new Text('H'+Integer.toString(h)), one);

                }
            }
        }

    }

    private static class PCYFirstReducer extends Reducer<Text,IntWritable,NullWritable,Tex
t> {

        private MultipleOutputs<NullWritable, Text> mMultipleOutputs;
        private int mThreshold;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
            super.setup(context);
            Configuration conf = context.getConfiguration();
            mMultipleOutputs = new MultipleOutputs<>(context);
            mThreshold = conf.getInt("threshold", Integer.MAX_VALUE);
        }
        private Text word = new Text();
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) thr
ows IOException, InterruptedException {
          int sum = 0;
            for (IntWritable val : values) {
              sum += val.get();
            }

            context.write( NullWritable.get(), key);
        }

    }

    public static void run(Configuration conf, int n, int threshold, String input) throws
```

```
 IOException, ClassNotFoundException, InterruptedException {
        Configuration configuration = new Configuration(conf);
        Configuration jConf = new Configuration(configuration);
        jConf.setInt("threshold", threshold);
        Job job = Job.getInstance(jConf, "Frequent Itemsets Pass 1");

        job.setJarByClass(FrequentItemsetsMain.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(PCYFirstMapper.class);
        job.setReducerClass(PCYFirstReducer.class);
        job.setNumReduceTasks(1);
        FileInputFormat.addInputPath(job, new Path(input));
        FileOutputFormat.setOutputPath(job, new Path("frequent-itemsets-1"));

        job.waitForCompletion(true);
    }
}
```

SecondPass.java

```
package FrequentItem;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;

public class SecondPass {
    private static class PCYKey implements WritableComparable<PCYKey> {
        private ArrayList<Integer> mKeys;
```

```
        public PCYKey() {
            mKeys = new ArrayList<>();
        }

        public ArrayList<Integer> getKeys() {
            return mKeys;
        }

        @Override
        public int compareTo(PCYKey o) {
            if (mKeys.size() != o.mKeys.size()) {
                return Integer.compare(mKeys.size(), o.mKeys.size());
            } else {
                for (int i = 0; i < mKeys.size(); ++i) {
                    if (!mKeys.get(i).equals(o.mKeys.get(i))) {
                        return Integer.compare(mKeys.get(i), o.mKeys.get(i));
                    }
                }
                return 0;
            }
        }

        @Override
        public void write(DataOutput out) throws IOException {
            out.writeInt(mKeys.size());
            for (int key : mKeys) {
                out.writeInt(key);
            }
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            mKeys = new ArrayList<>();
            int size = in.readInt();
            for (int i = 0; i < size; ++i) {
                mKeys.add(in.readInt());
            }
        }
    }

    private static class PCYValue implements Writable {
        private ArrayList<Integer> mValues;

        public PCYValue() {
            mValues = new ArrayList<>();
        }

        public ArrayList<Integer> getValues() {
            return mValues;
        }

        @Override
        public void write(DataOutput out) throws IOException {
            out.writeInt(mValues.size());
```

```
            for (int value: mValues) {
                out.writeInt(value);
            }
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            mValues = new ArrayList<>();
            int size = in.readInt();
            for (int i = 0; i < size; ++i) {
                mValues.add(in.readInt());
            }
        }
    }

    private static class SolverMapper extends Mapper<Object, Text, PCYKey, IntWritable> {
        private static final int HASHTABLE_SIZE = 10007;

        private HashSet<Integer> mHashes;
        private HashSet<Integer> mFrequentItems;
        private ArrayList<Integer> mElements;
        private Context mContext;
        private int mK;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
            super.setup(context);
            mHashes = new HashSet<>();
            mFrequentItems = new HashSet<>();
            initialization(context);
//            initializeHashes(context);
//            initializeFrequentItems(context);
        }

        @Override
        protected void map(Object key, Text value, Context context) throws IOException, In
terruptedException {
            String[] tokens = value.toString().split(" ");
            mContext = context;
            mElements = new ArrayList<>();
            for (String token : tokens) {
                mElements.add(Integer.parseInt(token));
            }
            Collections.sort(mElements);
//            mElements.sort(Integer::compareTo);
            generateSets(0, 0, new ArrayList<>());
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            super.cleanup(context);
        }

      private void initialization(Context context) {
```

```
            try {
                FileSystem fileSystem = FileSystem.get(context.getConfiguration());
                FileStatus[] fileStatuses = fileSystem.listStatus(new Path("frequent-items
ets-1"));
                for (FileStatus fileStatus : fileStatuses) {
                    String[] fileName = fileStatus.getPath().toString().split("/");
                    if (fileName[fileName.length - 1].contains("part-r-00000")) {
                        BufferedReader reader = new BufferedReader(
                                new InputStreamReader(fileSystem.open(fileStatus.getPath
()))
                        );
                        String line;
                        while ((line = reader.readLine()) != null) {
                          if(line.charAt(0)=='H')
                            mHashes.add(Integer.parseInt(line.substring(1)));
                          else
                            mFrequentItems.add(Integer.parseInt(line));
                        }
                        reader.close();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

      private final static IntWritable one = new IntWritable(1);
        private void generateSets(int now, int index, ArrayList<Integer> elements) throws
 IOException, InterruptedException {
            if (now == 2) {
                PCYKey newKey = new PCYKey();
                int hashValue = Hash.hash(elements, HASHTABLE_SIZE);
                if (mHashes.contains(hashValue)) {
                    PCYValue newValue = new PCYValue();
                    newKey.getKeys().addAll(elements);

                    boolean broke=false;
                    for (int value : elements) {
                            if (! mFrequentItems.contains(value)) {
                                return;
                            }
                       }
                       mContext.write(newKey, one);

                }
                return;
            }
            for (int i = index; i < mElements.size(); ++i) {
                if (mFrequentItems.contains(mElements.get(i))) {
                    elements.add(mElements.get(i));
                    generateSets(now + 1, i + 1, elements);
                    elements.remove(elements.size() - 1);
                }
            }
```

```
        }

    }

    private static class SolverReducer extends Reducer<PCYKey, IntWritable, NullWritable,
 Text> {
        private static final int HASHTABLE_SIZE = 10007;

        private MultipleOutputs<NullWritable, Text> mMultipleOutputs;

        private int mThreshold;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
            super.setup(context);
            Configuration conf = context.getConfiguration();
            mMultipleOutputs = new MultipleOutputs<>(context);
            mThreshold = conf.getInt("threshold", Integer.MAX_VALUE);
        }

        @Override
        protected void reduce(PCYKey key, Iterable<IntWritable> values, Context context) t
hrows IOException, InterruptedException {
          int sum = 0;
            for (IntWritable val : values) {
              sum += val.get();
            }
           context.write(NullWritable.get(), new Text(Integer.toString(sum)));
            if (sum >= mThreshold) {
                String output = "";
                boolean isFirst = true;
                for (int element : key.getKeys()) {
                    if (!isFirst) {
                        output += " ";
                    } else {
                        isFirst = false;
                    }
                    output += String.valueOf(element);
                }
                mMultipleOutputs.write("itemsets", NullWritable.get(), new Text(output));
            }
        }


//
    }

    public static void run(Configuration conf,  int n, int threshold, String input) throws
IOException, ClassNotFoundException, InterruptedException {
        Configuration configuration = new Configuration(conf);
        Configuration jConf = new Configuration(configuration);
        jConf.setInt("threshold", threshold);
        Job job = Job.getInstance(jConf, "Frequent Itemsets Second Pass ");
        job.setJarByClass(FrequentItemsetsMain.class);
```

```
        job.setMapOutputKeyClass(PCYKey.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(SolverMapper.class);
        job.setReducerClass(SolverReducer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setNumReduceTasks(1);
        FileInputFormat.addInputPath(job, new Path(input));
        FileOutputFormat.setOutputPath(job, new Path("frequent-itemsets-second"));
        MultipleOutputs.addNamedOutput(job, "candidates", TextOutputFormat.class, NullWrit
able.class, Text.class);
        MultipleOutputs.addNamedOutput(job, "itemsets", TextOutputFormat.class, NullWritab
le.class, Text.class);
        job.waitForCompletion(true);
    }
}
```

Hash.java

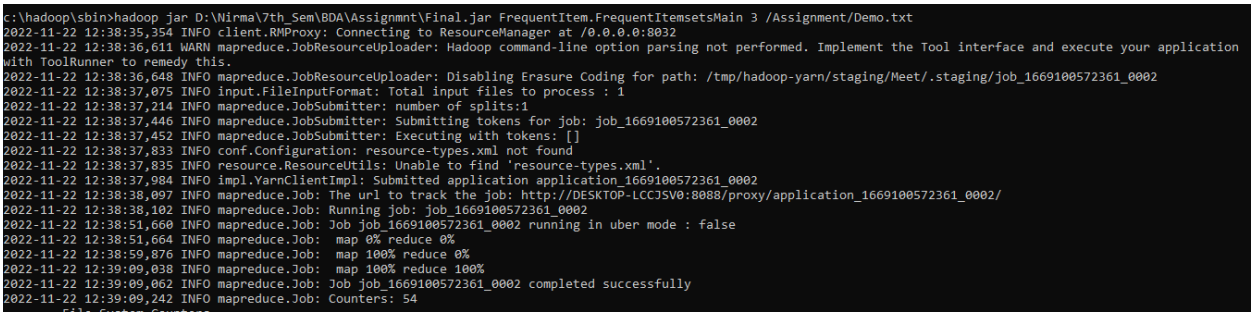
```
package FrequentItem;

import java.util.List;

public class Hash {
    public static int hash(int[] items, int mod) {
        int value = 1;
        int base = 1;
        for (int item : items) {
            value += item * base;
            base = base * 31;
        }
        value = value % mod;
        return Math.abs(value);
    }

    public static int hash(List<Integer> items, int mod) {
        int value = 1;
        int base = 1;
        for (int item : items) {
            value += item * base;
            base = base * 31;
        }
        value = value % mod;
        return Math.abs(value);
    }
}
```

FrequentItemsetsMain.java

```
package FrequentItem;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.Tool;



import java.io.IOException;

public class FrequentItemsetsMain {
    public static void main(String[] args) {
         int Threshold = Integer.parseInt(args[0]);
         Configuration conf=new Configuration();

         try {
           FirstPass.run(conf,100000,Threshold, args[1]);
           SecondPass.run(conf,100000,Threshold, args[1]);
         }
         catch(Exception e) {
           System.out.print(e);
         }

    }
}
```

Execution

[Note: Final.jar parameter → threshold & input file]



```
2022-11-22 12:39:09,318 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2022-11-22 12:39:09,345 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application
with ToolRunner to remedy this.
2022-11-22 12:39:09,355 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Meet/.staging/job_1669100572361_0003
2022-11-22 12:39:09,417 INFO input.FileInputFormat: Total input files to process : 1
2022-11-22 12:39:09,920 INFO mapreduce.JobSubmitter: number of splits:1
2022-11-22 12:39:10,031 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1669100572361_0003
2022-11-22 12:39:10,033 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-11-22 12:39:10,309 INFO impl.YarnClientImpl: Submitted application application_1669100572361_0003
2022-11-22 12:39:10,328 INFO mapreduce.Job: The url to track the job: http://DESKTOP-LCCJSV0:8088/proxy/application_1669100572361_0003/
2022-11-22 12:39:10,329 INFO mapreduce.Job: Running job: job_1669100572361_0003
2022-11-22 12:39:27,742 INFO mapreduce.Job: Job job_1669100572361_0003 running in uber mode : false
2022-11-22 12:39:27,743 INFO mapreduce.Job:  map 0% reduce 0%
2022-11-22 12:39:34,894 INFO mapreduce.Job:  map 100% reduce 0%
2022-11-22 12:39:43,030 INFO mapreduce.Job:  map 100% reduce 100%
2022-11-22 12:39:44,059 INFO mapreduce.Job: Job job_1669100572361_0003 completed successfully
2022-11-22 12:39:44,113 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=3526
                FILE: Number of bytes written=483139
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2818
                HDFS: Number of bytes written=123
                HDFS: Number of read operations=13
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
```