Homework #3 Report

Name: Tianyu Yang

GW ID: G38878678

Date: 3/4/2019

Objective:

1. This homework is to use MapReduce function in Hadoop to run BFS algorithm

2. Try to use Colonial One machine to run nodes and edges with 1, 4, 8, 16

Procedure 1: Install Hadoop on Ubuntu

1. Install jdk on Ubuntu and configure the environment

   (1) Update the repositories

   Sudo apt-get update

   (2) Start java installation

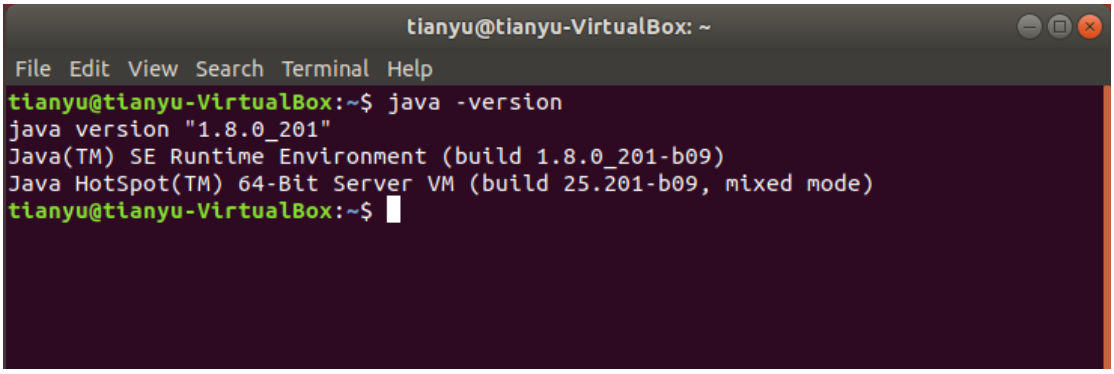
   Sudo apt-get purge openjdk*

   Sudo add-apt-repository ppa:webupd8team.java

   Sudo apt-get update

   Sudo apt-get install oracle-java8-installer

   Java -version



   Sudo echo “export JAVA_HOME=/usr” >> /etc/profile

   Source /etc/profile

   (3) Disable ipv6 since Hadoop does not support it that Hadoop support only ipv4

   Sudo nano /etc/syssctl.conf

   Move to the end and append the code

#Disable IPv6

Net.ipv6.conf.all.disable_ipv6 = 1

Net.ipv6.conf.default.disable_ipv6 = 1

Net.ipv6.conf.lo.disable_ipv6 = 1

Then reboot the system

Sudo reboot

(4) Add a group for Hadoop

Sudo addgroup hadoopgroup

Sudo adduser -ingroup hadoopgroup tyuser

(5) Install ssh

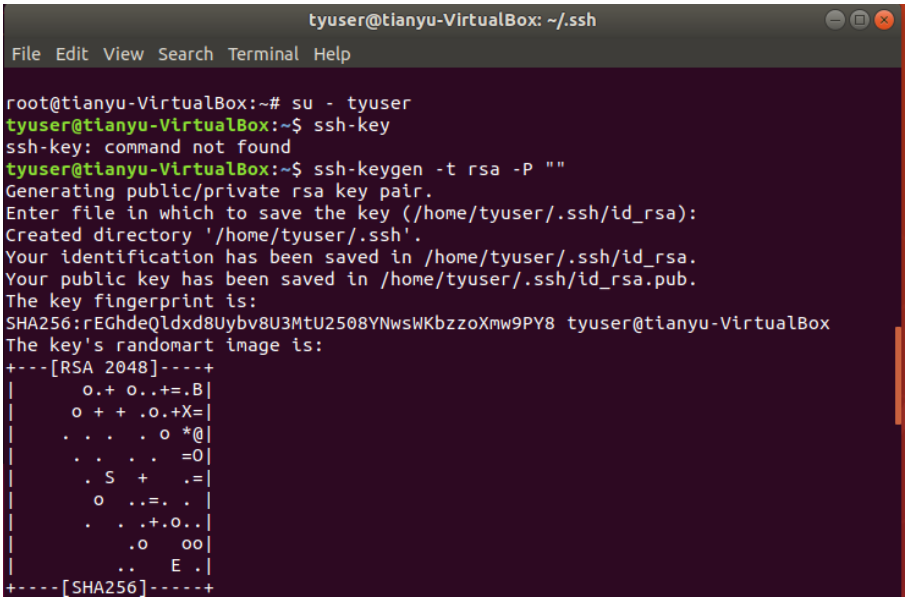
Sudo apt-get install ssh

Sudo systemctl enable ssh

Sudo systemctl start ssh

Su – tyuser

Ssh-keygen -t rsa -P “”



Cat /home/tyuser/.ssh/id_rsa.pub >> /home/tyuser/.ssh/authorized_keys

Cd .ssh/

Ls

Chmode 600 ./authorized_keys

Ls -l

Ssh-copy-id -i /home/tyuser/.ssh/id_rsa.pub localhost

Ssh localhost

Exit

2. Install Hadoop to Ubuntu

(1) Download Hadoop package from Oracle website

Wget http://mirror.fibergrid.in/apache/hadoop/common/hadoop-3.1.2/hadoop-3.1.2.tar.gz

Tar -xvf Hadoop-3.1.2.tar.gz

Sudo mv ./hadoop-3.1.2 /usr/local

Sudo ln -sf /usr/local/Hadoop-3.1.2/ /usr/local/Hadoop

Sudo chown -R hduster:hadoopgroup /usr/local/Hadoop-3.1.2/

Su – tyuser

(2) Configure the running environment for Hadoop

Nano ./.bashrc

Add the code in the bottom of the file

#Hadoop config

Export HADOOP_PREFIX=/usr/local/Hadoop

Export HADOOP_HOME=/usr/local/Hadoop

Export HADOOP_MAPRED_HOME=${HADOOP_HOME}

Export HADOOP_COMMON_HOME=${HADOOP_HOME}

Export HADOOP_HDFS_HOME=${HADOOP_HOME}

Export YARN_HOME=${HADOOP_HOME}

Export HADOOP_CONF_DI${HADOOP_HOME}/etc/Hadoop

#Native path

Export

HADOOP_COMMON_LIB_NATIVE_DI${HADOOP_HOME}/lib/native

Export HADOOP_OPTS=”-

Djava.library.path=$HADOOP_PREFIX/lib/native”

#Java path

Export JAVA_HOME=”/usr”

#OS Path

Export

PATH=$PATH:$HADOOP_HOME/bin:$JAVA_HOME/bine:$HADOOP_HO

ME/sbin

then

source ./.bashrc

nano /usr/local.hadoop/etc/Hadoop/Hadoop-env.sh

Add the code in the bottom of the file

Export JAVA_HOME=”/usr”

(3) Modify the xml file

Cd /usr/local/Hadoop/etc/hadoop

Nano core-site.xml

Add the code in the bottom of the file(inside <configuration>)

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9000<value>

</property>

Similarly for hdfs-site-xml

Name: dfs.replication

Value: 1

Name: dfs.name.dir

Value: file:/usr/local/Hadoop/hadoopdata/hdfs/namenode

Name: dfs.data.dir

Value: file:/usr/local/Hadoop/hadoopdata/hdfs/datanode

Similarly for mapred-site-xml

Name: mapreduce.framework.name

Value: yarn

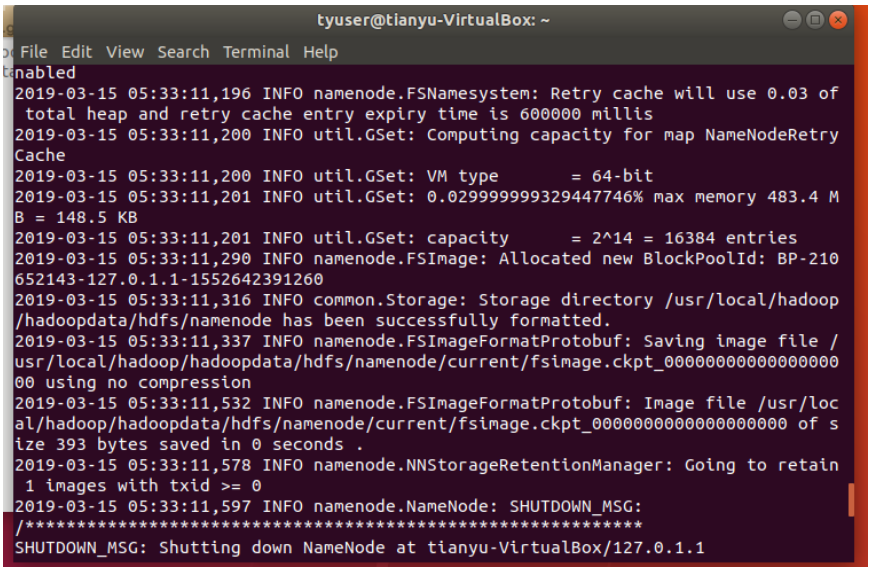
Similarly for yarn-site-xml

Name: yarn.nodemanager.aux-services

Value: mapreduce_shuffle

Then

Hdfs namenode -format

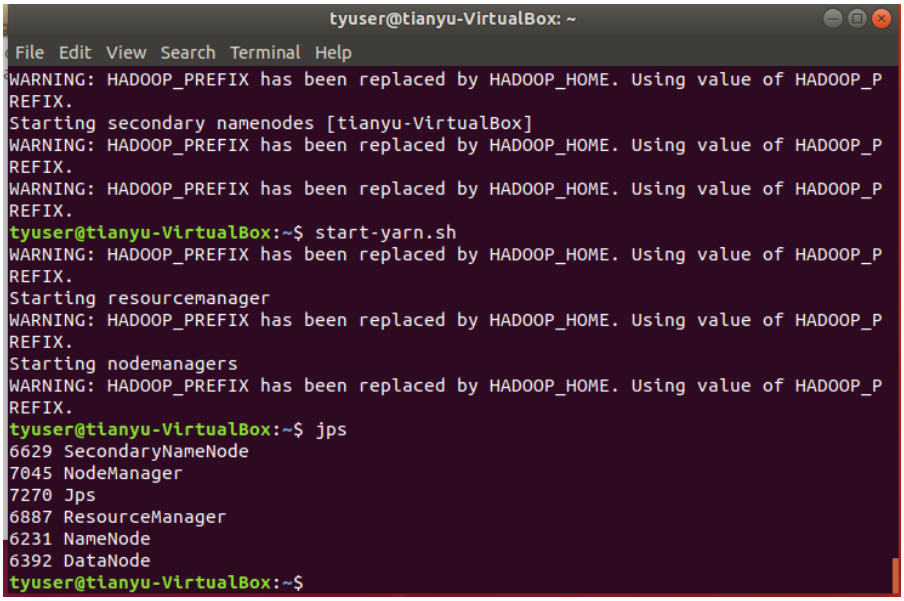


(4) Run .sh file

Start-dfs.sh

Start-yarn.sh

Jps



Procedure 2:

1. Introduction to Breadth-First Search (BFS)

   Breadth-First Search is an iterated algorithm over graphs. Fronties advances from origin by one level with each pass

2. How to use MapReduce in BFS

   Iterated passes through MapReduce – map some nodes, result includes additional nodes which are fed into successive MapReduce passes. The MapReduce method can advance the known frontier by one hop. To realize this parallel computing, the key element is independent PageRank computations in a given step. This parallelization requires to think about the minimum data partitions to transmit.

3. The running code is shown as below:

```
/*
*
* ECE 6130 Big Data and Cloud Computing
* Spring 2019
* Homework 3: BFS using MapReduce (Hadoop)
* Name: Tianyu Yang
* GW ID:G38878678
* Referenced from https://puffsun.iteye.com/blog/1905524
*
*/
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.io.LongWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;

public class BFSMapReduce extends Configured implements Tool {

   public static String OUT = "output";
   public static String IN = "inputlarger";

   public static class DijkstraMapper extends Mapper<LongWritable,
Text, LongWritable, Text> {

       public void map(LongWritable key, Text value, Context context)
             throws IOException, InterruptedException {

          //From slide 20 of Graph Algorithms with MapReduce (by
Jimmy Lin, Univ @ Maryland)
          //Key is node n
          //Value is D, Points-To
          //For every point (or key), look at everything it points
to.
          //Emit or write to the points to variable with the current
distance + 1
          Text word = new Text();
          String line = value.toString();//looks like 1 0 2:3:
          String[] sp = line.split(" ");//splits on space
          int distanceAdded = Integer.parseInt(sp[1]) + 1;
          String[] pointsTo = sp[2].split(":");
          for (String distance : pointsTo) {
             word.set("VALUE " + distanceAdded);//tells me to look
at distance value
             context.write(new
LongWritable(Integer.parseInt(distance)), word);
             word.clear();
          }
          //pass in current node's distance (if it is the lowest
distance)
          word.set("VALUE " + sp[1]);
          context.write(new LongWritable(Integer.parseInt(sp[0])),
```

```
word);
            word.clear();

            word.set("NODES " + sp[2]);//tells me to append on the
final tally
            context.write(new LongWritable(Integer.parseInt(sp[0])),
word);
            word.clear();

        }
    }

    public static class DijkstraReducer extends Reducer<LongWritable,
Text, LongWritable, Text> {
        public void reduce(LongWritable key, Iterable<Text> values,
Context context)
                throws IOException, InterruptedException {

            //From slide 20 of Graph Algorithms with MapReduce (by
Jimmy Lin, Univ @ Maryland)
            //The key is the current point
            //The values are all the possible distances to this point
            //we simply emit the point and the minimum distance value

            String nodes = "UNMODED";
            Text word = new Text();
            int lowest = 10009;//start at infinity

            for (Text val : values) {//looks like NODES/VALUES 1 0
2:3:, we need to use the first as a key
                String[] sp = val.toString().split(" ");//splits on
space
                //look at first value
                if (sp[0].equalsIgnoreCase("NODES")) {
                    nodes = null;
                    nodes = sp[1];
                } else if (sp[0].equalsIgnoreCase("VALUE")) {
                    int distance = Integer.parseInt(sp[1]);
                    lowest = Math.min(distance, lowest);
                }
            }
            word.set(lowest + " " + nodes);
            context.write(key, word);
            word.clear();
```

```
        }
    }

    //Almost exactly from
http://hadoop.apache.org/mapreduce/docs/current/mapred_tutorial.html
    public int run(String[] args) throws Exception {

//http://code.google.com/p/joycrawler/source/browse/NetflixChallenge/
src/org/niubility/learning/knn/KNNDriver.java?r=242
        //make the key -> value space separated (for iterations)
        getConf().set("mapred.textoutputformat.separator", " ");

        //set in and out to args.
        IN = args[0];
        OUT = args[1];

        String infile = IN;
        String outputfile = OUT + System.nanoTime();

        boolean isdone = false;
        boolean success = false;

        HashMap<Integer, Integer> _map = new HashMap<Integer,
Integer>();

        while (!isdone) {

            Job job = new Job(getConf(), "Dijkstra");
            job.setJarByClass(ParallelDijkstra.class);
            job.setOutputKeyClass(LongWritable.class);
            job.setOutputValueClass(Text.class);
            job.setMapperClass(DijkstraMapper.class);
            job.setReducerClass(DijkstraReducer.class);
            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(infile));
            FileOutputFormat.setOutputPath(job, new Path(outputfile));

            success = job.waitForCompletion(true);

            //remove the input file
            //http://eclipse.sys-con.com/node/1287801/mobile
            if (!infile.equals(IN)) {
```

```
            String indir = infile.replace("part-r-00000", "");
            Path ddir = new Path(indir);
            FileSystem dfs = FileSystem.get(getConf());
            dfs.delete(ddir, true);
        }

        infile = outputfile + "/part-r-00000";
        outputfile = OUT + System.nanoTime();

        //do we need to re-run the job with the new input file??
        //http://www.hadoop-blog.com/2010/11/how-to-read-file-from-
hdfs-in-hadoop.html
        isdone = true;//set the job to NOT run again!
        Path ofile = new Path(infile);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br = new BufferedReader(new
InputStreamReader(fs.open(ofile)));

        HashMap<Integer, Integer> imap = new HashMap<Integer,
Integer>();
        String line = br.readLine();
        while (line != null) {
            //each line looks like 0 1 2:3:
            //we need to verify node -> distance doesn't change
            String[] sp = line.split(" ");
            int node = Integer.parseInt(sp[0]);
            int distance = Integer.parseInt(sp[1]);
            imap.put(node, distance);
            line = br.readLine();
        }
        if (_map.isEmpty()) {
            //first iteration... must do a second iteration
regardless!
            isdone = false;
        } else {
            //http://www.java-examples.com/iterate-through-values-
java-hashmap-example
            //http://www.javabeat.net/articles/33-generics-in-java-
50-1.html
            for (Integer key : imap.keySet()) {
                int val = imap.get(key);
                if (_map.get(key) != val) {
                    //values aren't the same... we aren't at
convergence yet
```

```
                    isdone = false;
                }
            }
        }
        if (!isdone) {
            _map.putAll(imap);//copy imap to _map for the next
iteration (if required)
        }
    }

    return success ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    System.exit(ToolRunner.run(new ParallelDijkstra(), args));
  }

}
```

4. Run a sample BFS tree and obtain the result.

Input data:

```
1 0 2:3:
2 10000 1:4:5:
3 10000 1:
4 10000 2:5:
5 10000 2:4:
```

Output shell result:

```
03/03/19 20:08:03 INFO input.FileInputFormat: Total input paths to
process : 1
03/03/19 20:09:04 INFO mapred.JobClient: Running job:
job_201307131656_0001
03/03/19 20:09:05 INFO mapred.JobClient:  map 0% reduce 0%
03/03/19 20:10:05 INFO mapred.JobClient:  map 100% reduce 0%
03/03/19 20:15:13 INFO mapred.JobClient:  map 100% reduce 100%
03/03/19 20:15:16 INFO mapred.JobClient: Job complete:
job_201307131656_0001
03/03/19 20:15:16 INFO mapred.JobClient: Counters: 32
03/03/19 20:15:16 INFO mapred.JobClient:   File System Counters
03/03/19 20:15:16 INFO mapred.JobClient:     FILE: Number of bytes
read=183
03/03/19 20:15:16 INFO mapred.JobClient:     FILE: Number of bytes
```

```
written=313855
03/03/19 20:15:16 INFO mapred.JobClient:    FILE: Number of read
operations=0
03/03/19 20:15:16 INFO mapred.JobClient:    FILE: Number of large
read operations=0
03/03/19 20:15:16 INFO mapred.JobClient:    FILE: Number of write
operations=0
03/03/19 20:15:16 INFO mapred.JobClient:    HDFS: Number of bytes
read=173
03/03/19 20:15:16 INFO mapred.JobClient:    HDFS: Number of bytes
written=53
03/03/19 20:15:16 INFO mapred.JobClient:    HDFS: Number of read
operations=2
03/03/19 20:15:16 INFO mapred.JobClient:    HDFS: Number of large
read operations=0
03/03/19 20:15:16 INFO mapred.JobClient:    HDFS: Number of write
operations=1
03/03/19 20:15:16 INFO mapred.JobClient:  Job Counters
03/03/19 20:15:16 INFO mapred.JobClient:    Launched map tasks=1
03/03/19 20:15:16 INFO mapred.JobClient:    Launched reduce tasks=1
03/03/19 20:15:16 INFO mapred.JobClient:    Data-local map tasks=1
03/03/19 20:15:16 INFO mapred.JobClient:    Total time spent by all
maps in occupied slots (ms)=38337
03/03/19 20:15:16 INFO mapred.JobClient:    Total time spent by all
reduces in occupied slots (ms)=159310
03/03/19 20:15:16 INFO mapred.JobClient:    Total time spent by all
maps waiting after reserving slots (ms)=0
03/03/19 20:15:16 INFO mapred.JobClient:    Total time spent by all
reduces waiting after reserving slots (ms)=0
03/03/19 20:15:16 INFO mapred.JobClient:  Map-Reduce Framework
03/03/19 20:15:16 INFO mapred.JobClient:    Map input records=5
03/03/19 20:15:16 INFO mapred.JobClient:    Map output records=20
03/03/19 20:15:16 INFO mapred.JobClient:    Map output bytes=383
03/03/19 20:15:16 INFO mapred.JobClient:    Input split bytes=112
03/03/19 20:15:16 INFO mapred.JobClient:    Combine input records=0
03/03/19 20:15:16 INFO mapred.JobClient:    Combine output records=0
03/03/19 20:15:16 INFO mapred.JobClient:    Reduce input groups=5
03/03/19 20:15:16 INFO mapred.JobClient:    Reduce shuffle bytes=179
03/03/19 20:15:16 INFO mapred.JobClient:    Reduce input records=20
03/03/19 20:15:16 INFO mapred.JobClient:    Reduce output records=5
03/03/19 20:15:16 INFO mapred.JobClient:    Spilled Records=40
03/03/19 20:15:16 INFO mapred.JobClient:    CPU time spent (ms)=3970
03/03/19 20:15:16 INFO mapred.JobClient:    Physical memory (bytes)
snapshot=240209920
```

```
03/03/19 20:15:16 INFO mapred.JobClient:    Virtual memory (bytes)
snapshot=2276737024
03/03/19 20:15:16 INFO mapred.JobClient:    Total committed heap
usage (bytes)=101519360
```

Procedure 3

Result analysis

The input tree has three levels and five nodes. The total time spent by all maps in occupied slots is 38337 ms. The total time spent by all reduces in occupied slots is 159310 ms. For the MapReduce, the input records is 5 and the output records is 20. The CPU time spent is 2290 ms.