CSci455/655 Assignment 3

Due: Nov 27th, 11:59pm.

In this group programming assignment (with group size 2), you will practice parallel programming, in particular, you will gain experience with MapReduce programming. MapReduce has been widely used for processing of large data sets (terabytes of data) and computationally expensive tasks. Hadoop is an open-source version of MapReduce written in Java. Penalties will be given for late submissions (20% deduction per day).

Objectives

The goal of this programming assignment is to enable you to gain experience in:

1. Basic features of Hadoop distributed file system and MapReduce
2. Writing a program for finding common neighbors between all pairs of nodes/peers in a large P2P Network.

Language:

You are required to use Java as the implementation language.

Deliverables

You should hand in a (1) report which presents your implementation, (2) a README file explaining how to run your program, (3) your source code, and (4) a 5-minute video demo that shows how your system works. Please do not upload the video, instead, upload it to a Cloud e.g., YouTube, Google Drive, OneDrive, etc., and provide the link.

Total Marks: 100

MapReduce Programming

Description

In your previous assignment, you have implemented a Gnutella P2P network. You would know that in a large-scale P2P network, there are normally lots of peers. The task of this assignment is to implement a MapReduce program to identify common neighboring peers among all pairs of peers. Let P be a set of all peers: {P1, P2, ..., Pn}. Then the goal is to find common neighbors for every pair of peers (Pi, Pj) where $i \neq j$.

The input files have the following format:

P_A P_B P_C P_D P_E

P_B P_A P_C P_D

P_C P_A P_B
P_D P_A P_B P_E
P_E P_A P_D
where the first token in the line is the peer and the remaining tokens in the line are the neighbors of this peer. So, for line 1, P_A has four neighbors: P_B, P_C, P_D, and P_E. For example, P_A and P_B have P_C and P_D as their common neighbors. Also, P_A P_E have only P_D as their common neighbor.

Use the p2p network topology provided that is attached with this assignment as input to your program. The network has been partitioned into three files: file01, file02, file03.

The output will be stored in a file in the output directory.

Below is a tutorial helping you getting started with the Hadoop installed in the department cluster. You can also install your own Hadoop in your personal computer.

Getting Started with Hadoop

Logging In

First, make sure you can log in to the head node with SSH, currently at zoidberg.cs.ndsu.nodak.edu. You can log in to this server with your CS Domain password or your Blackboard password.

Example: WordCount

Before we jump into the details, lets walk through an example MapReduce application to get a flavor for how they work.

Source Code

```
import java.io.IOException;
import java.util.StringTokenizer;

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;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
```

```
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Usage

Set environment variables:

```
export JAVA_HOME=/path/to/your/jdk (path where java jdk is installed)

e.g., export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Compile `WordCount.java` and create a jar:

```
$ bin/hadoop com.sun.tools.javac.Main WordCount.java

(or hadoop com.sun.tools.javac.Main WordCount.java)

$ jar cf wc.jar WordCount*.class
```

Setting Up Input Files

This program can use the Hadoop Distributed File System (HDFS) that is set up in the CS department. This file system spans all the Linux lab machines and provides distributed storage for use specifically with Hadoop.

You can work with HDFS with UNIX-like file commands. The list of file commands can be found here.

First, make a directory to store the input for the program (use your username).

```
yourName@zoidberg:~$ hadoop fs -mkdir /user/yourName/wordcount

yourName@zoidberg:~$ hadoop fs -mkdir /user/yourName/wordcount/input
```

To set up input for the WordCount program, create two files as follows:

file01:

```
Hello World Bye World
```

file02:

```
Hello Hadoop Goodbye Hadoop
```

Save these to your home folder on the head node. To move them into HDFS, use the following commands:

```
yourName@zoidberg:~$ hadoop fs -copyFromLocal /home/yourName/file01
/user/yourName/wordcount/input/file01

yourName@zoidberg:~$ hadoop fs -copyFromLocal /home/yourName/file02
/user/yourName/wordcount/input/file02
```

Again, use your username where applicable.

The syntax here is “hadoop fs -copyFromLocal <LOCAL_FILE> <HDFS_FILE>”, in this case we're going to copy file01 from the local system into HDFS under our HDFS user directory into the wordcount/input/ directory.

Running the WordCount Program

You can now run the WordCount program using the following command:

```
hadoop jar wc.jar WordCount /user/yourName/wordcount/input
/user/yourName/wordcount/output
```

The command syntax is: “hadoop jar <JARFILE> <CLASS> <PARAMETERS…>”

In this case, we use the wc.jar JAR file, running the class 'WordCount' with two parameters, an input directory and an output directory. The output directory must not already exist in HDFS, it will be created by the program.

View Output

You can check the output directory with:

```
hadoop fs -ls /user/yourName/wordcount/output/
```

You should then see something similar to:

```
yourName@zoidberg:~$ hadoop fs -ls /user/yourName/wordcount/output

Found 2 items

-rw-r--r--   3 yourName nogroup         41 2011-11-08 11:23
/user/yourName/wordcount/output/part-r-00000
```

The 'part-r-00000' file contains the results of the word counting. You can look at the file using the 'cat' command.

```
yourName@zoidberg:~$ hadoop fs -cat /user/yourName/wordcount/output/part-r-00000

Bye 1

Goodbye 1

Hadoop 2
```

```
Hello 2

World 2
```

Another Example: Bigram

The following example may help you write your assignment program.

Bigrams are simply sequences of two consecutive words. For example, the previous sentence contains the following bigrams: "Bigrams are", "are simply", "simply sequences", "sequences of", etc.

A Bigram Counting application that can be composed as a two-stage Map Reduce job.

- The first stage counts bigrams.
- The second stage MapReduce job takes the output of the first stage (bigram counts).

You can read and run the attached Bigram source code to learn. (To run the program, you need to create input file(s).