Programming Assignment 0

Creating Hadoop Clusters and Running a Simple Analytics Job

Due: Sept. 6, 2021 5:00PM
Submission: via Canvas, individual submission

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

- Setting up a Hadoop cluster
- Hadoop standalone operations
- Running a MapReduce example

1. Introduction

The objective of this assignment is to gain experience with Hadoop clusters. This includes,
- Configuring your own Hadoop cluster in the CS120 lab
- Learning how to manage files using Hadoop standalone operations
- Running a MapReduce example

You must use 5 ~ 10 machines to create your Hadoop cluster. To ensure the availability of node and port range, please follow the specified port and node assignments given in Node and PortAssignment.pdf

2. Cluster Setup

Before we start with the actual assignment, this prerequisite exercise is intended to give you an introduction on Apache Hadoop in its simplest form i.e. Single Node Mode (running all the hadoop daemons in a single jvm instance). After this prerequisite exercise, one should be able to run distinct-nodes example using mapreduce programming model (prefer CS machine)

Setup:
- Before you come to recitation, please download the hadoop **binary** (version 3.1.2) [Link]. (If possible, verify your download using the checksum file provided.)
- And, install JAVA (1.7 or above) [Link] (If In CS120 machine it is already there so please use : **export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk"** in .bashrc)

Note: Windows user should run a Linux distro using VirtualBox. [Link] [Not necessary if CS120 machine is being used]

Installation guide:

<u>Step 1:</u> Unpack the downloaded Hadoop distribution. Setup Hadoop path just like you did with java path above.

(export HADOOP_HOME=<*path to Hadoop distribution*>) (example:

export HADOOP_HOME="/usr/local/hadoop/3.1.2")

<u>Step 2:</u> Add java to hadoop classpath using the command

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

After setting up above environment variables, to apply to the existing session, run "source ~/.bash_profile" (OS X) or "source ~/.bashrc" (Linux) based on your operating system.

<u>Step 3:</u> In the distribution, edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows
(this step is not necessary if you are doing in CS machine):

```
# set to the root of your Java installation
 export JAVA_HOME=<path to your java installation>
```

<u>Step 4:</u> Create a directory in your Home folder.

```
mkdir ~/example
```

<u>Step 5:</u> Inside "example" directory, copy-paste the following code to create a java program. Name it DistinctNodes.java

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

public class DistinctNodes {

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

        private Text node ;

        @Override
        protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                String s = itr.nextToken();
                node = new Text( s );
                context.write(node, NullWritable.get());
            }
        }
    }
public static class CountReducer extends Reducer<Text, NullWritable, IntWritable, NullWritable>
```

```
    {
        private Set<String> distinctNodes ;

        @Override
        protected void setup(Context context) {
            distinctNodes = new HashSet<String>();
        }

        @Override
        protected void reduce(Text key, Iterable<NullWritable> values, Context context) {

            distinctNodes.add(key.toString());

        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            IntWritable noOfNodes = new IntWritable(distinctNodes.size());
            context.write(noOfNodes, NullWritable.get());
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "distinct nodes new");
        job.setJarByClass(DistinctNodes.class);
        job.setMapperClass(DistinctNodes.TokenizerMapper.class);
        job.setReducerClass(DistinctNodes.CountReducer.class);
        job.setNumReduceTasks(1);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Step 6: Inside "example" directory, compile your DistinctNodes.java using the java compiler added in hadoop classpath:

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

Step 7: You should be able to see class files for DistinctNodes. Create a jar file named DistinctNodes.jar using following command:

```
jar cf DistinctNodes.jar DistinctNodes*.class
```

Step 8: Download this and unpack it into "example/input" directory. Then run your jar with the following command:

```
$HADOOP_HOME/bin/hadoop jar  ~/example/DistinctNodes.jar DistinctNodes   ~/example/input   ~/example/result
```

Note: Make sure "example/result" directory does not exist, at the time of running the command.

Step 9: Check the output:

```
cat  ~/example/result/*
```

3. Setting up a Hadoop cluster

The first part of this assignment involves setting up your Hadoop Cluster using 5-10 machines on CS120 lab. **Make sure to use only those machines assigned to you (**Table 1-Node and port range assignment**).**

Please click on this [Link] to see a detailed installation guide for your Hadoop cluster.

4. Hadoop Standalone Operations

The following are a few standalone operations that would help you manage files in HDFS and perform certain necessary Hadoop operations (NOTE: First setup your cluster):

- The command to see the contents of your HDFS is similar to the ls command you use in UNIX

  ```
  $HADOOP_HOME/bin/hadoop fs -ls /
  ```

  For instance, if you have a directory called tmp in your HDFS, in order to look into tmp, just type

  ```
  $HADOOP_HOME/bin/hadoop fs -ls /tmp
  ```

- In order to create a folder (let's call it dataLoc) in HDFS where you want to put that data, use the following command:

  ```
  $HADOOP_HOME/bin/hadoop fs -mkdir /dataLoc
  ```

  Now to upload the data into dataLoc, run the following command:

  ```
  $HADOOP_HOME/bin/hadoop fs -put <SOURCE> /dataLoc
  ```

5. Example

You will be using the simple MapReduce program for counting distinct nodes in a social network. Please use the code given above. This time make sure you are reading the input text file from hdfs and that the result is also saved in hdfs.

In order to run your program on a cluster, follow the following steps:

a. Create a Java Project, compile and create the application jar.
b. Put the same text file used in Step 8 into HDFS using commands mentioned above. This will be your input file such that you have to provide the HDFS path to this file as an argument while running your program on the cluster.
c. Run your JAR using the command mentioned in the Hadoop Installation Guide document.

[Optional] If you use IDE to export the project into a jar, in that case, you need all dependent hadoop jars for compiling the application. Hint: Pull hadoop artifact from maven repo OR add required hadoop jars [Link] in your project built path.

6. Submission

This assignment requires an **individual submission**. Please submit your tarball of configuration files and output files via Canvas. You will use the submitted configuration files for demo. Do not miss any file. For your demo, you are not allowed to use any file outside of your Canvas submission except the Hadoop software.

7. Grading

Each of the submissions will be graded based on the demonstration of your software to GTA. During the demonstration, you should present:
Step 1. Initializing Hadoop cluster (1 point)
Step 2. Running Hadoop standalone operations: store input file, find input file, delete input file, and read output file (1 point)
Step 3. Running MapReduce example (1 point)

Demo includes short interviews about your software design and implementation details. Each questions and answers will count for your score. This assignment will account for 3% of your final grade. The grading will be done on a 3-point scale.

You are required to work alone for this assignment.

8. Late policy

Please check the late policy posted on the course web page.