Lab 4

Objectives

- Learn how to develop Maven proejct on remote server.
- Write and play with a simple MapReduce program.
- Customize the MapReduce program by accepting user input.
- Run Hadoop MapReduce programs in standalone and distributed modes.

Prerequisites

- Setup the development environment as explained in Lab #1.
- Download two `tsv` files in Lab3(nasa_19950801.tsv, nasa_19950630.22-19950728.12.tsv), and put them to your `cs167` virtual machine home directory (decompress if needed).
- Remote-access in remote-access.

Lab Work

I. Setup - In-home (30 minutes)

This part will be done in `cs167` server.
Every steps are assuming you have already login into `cs167` server and in your home directory `/home/cs167`.

0. Makesure the namenode and all datanodes in your group are alive (you can use `screen` or `tmux` to keep them running in the backend).

1. Create a new empty project using Maven for Lab4. See previous Labs (Lab #1, Lab #2, Lab #3) for more details.

2. Import your project into IntelliJ IDEA.

3. Copy the file `$HADOOP_HOME/etc/hadoop/log4j.properties` to your project directory under `src/main/resources`. This allows you to see internal Hadoop log messages when you run in IntelliJ IDEA.

    - Manually create `src/main/resources` if it does not exist.

4. Place the two sample files in your project home directory, i.e., next to the `pom.xml` file.

5. In `pom.xml` add the following dependencies.

```
<properties>
  <!-- Change the version number below to match your installed Hadoop. -->
  <hadoop.version>3.3.6</hadoop.version>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
```

```
</properties>

<dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-hdfs</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-common</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-core</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
</dependencies>
```

II. Simple Filter Program - In-home (30 minutes)

[TODO] Specify the locations. I assume student will do this experiment on the server.

This part will be done on `cs167` server.
In this part, you will need to write a MapReduce program that produces the lines that have a specific response code in them (similar to Lab #3). We will provide you with a sample code to help you understand MapReduce procedure in Hadoop.

0. If `nasa_19950801.tsv` does not exist in your home directory, download `nasa_19950801.tsv` and put it to your virtual environment home directory. You can use the following command to check whether you have the file:

   ```
   ls ~/ | grep nasa
   ```

   Note: You should see `nasa_19950801.tsv` in the output.

1. Take a few minutes to look into the sample file and understand its format. You can use the following command:

   ```
   less nasa_19950801.tsv
   ```

Note: You can press J or K on your keyborad to scroll down or up. Press Q when you want to exist viewing the file.

2. Create a new class named Filter in package edu.ucr.cs.cs167.[UCRNetID] with the following content:

```
// Replace [UCRNetID] with your netid
package edu.ucr.cs.cs167.[UCRNetID];

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

/**
* Filter log file by response code
*/
public class Filter {
    public static void main(String[] args) throws Exception {
        String inputPath = args[0];
        String outputPath = args[1];
        // String desiredResponse = args[2];
        Configuration conf = new Configuration();
        // TODO pass the desiredResponse code to the MapReduce
program
        Job job = Job.getInstance(conf, "filter");
        job.setJarByClass(Filter.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setNumReduceTasks(0);
        job.setInputFormatClass(TextInputFormat.class);
        Path input = new Path(inputPath);
        FileInputFormat.addInputPath(job, input);
        Path output = new Path(outputPath);
        FileOutputFormat.setOutputPath(job, output);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

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

        @Override
        protected void setup(Context context)
                throws IOException, InterruptedException {
```

```
            super.setup(context);
            // TODO add additional setup to your map task, if
needed.
        }

        public void map(LongWritable key, Text value, Context
context)
                throws IOException, InterruptedException {
            if (key.get() == 0) return; // Skip header line
            String[] parts = value.toString().split("\t");
            String responseCode = parts[5];
            // TODO Filter by response code
        }
    }
}
```

3. Take some time to understand the code and answer the following questions.
    - ***(Q1) What do you think the line `job.setJarByClass(Filter.class);` does?***
    - ***(Q2) What is the effect of the line `job.setNumReduceTasks(0);`?***
    - ***(Q3) Where does the `main` function run? (Driver node, Master node, or an executor node).***
4. We will slightly modify the `Filter` class to filter all lines with response code `200`.
   Add the following code below comment `// TODO Filter by response code` in `map` function:

```
// TODO Filter by response code
if (responseCode.equals("200")){
    context.write(NullWritable.get(), value);
}
```

   Note: we use `String#equals` rather than the operator `==` since `String` is not a primitive value in Java.

5. Go to your project directory `workspace/[UCR_NetID]_lab4`, use the following command to build your `jar` file:

```
mvn clean package
```

6. Run your program by `hadoop jar` command, and specify the class `Filter`:

```
hadoop jar target/[UCRNetID]_lab4-1.0-SNAPSHOT.jar
edu.ucr.cs.cs167.[UCRNetID].Filter nasa_19950801.tsv
filter_output_dir
```

Note: Since we didn't specify the `mainClass` in `pom.xml`, we need to manually specify which class to be run.

7. After running this command, an ourput directory will be generated in HDFS.
   Check the output directory by using `hdfs dfs -ls` command:

   ```
   hdfs dfs -ls filter_output_dir
   ```

   Note: You should be able to see two files. One is called `_SUCCESS`, which indicates that your MapReduce job successfully finished.

8. Check the content in the other file you found.
   You can use the following command to see how many lines are in the MapReduce output file:

   ```
   # Replace [filter_output] to be the other file name you find
   hdfs dfs -cat filter_output_dir/[filter_output]
   ```

   Note: `hdfs dfs -cat` will show all contents of input file in HDFS.

- **(Q4) How many lines do you see in the output?**

  Note: You can use the following command:

  ```
  hdfs dfs -cat filter_output_dir/[filter_output] | wc -l
  ```

III. Take User Input For the Filter - In-lab Part (20 minutes)

This part will be run on `cs167` server.

In this part, we will customize our program by taking the desired response code from the user as a command line argument.

1. Uncomment the line `// String desiredResponse = args[2];` in the `main` function.
   Now, the variable `desiredResponse` will store a string which indicates the response code we want to filter.
   Note: Now your program needs **three** parameters to run.

2. Add the desired response code to the job configuration using the method `Configuration#set`.
   Use a user-defined configuration entry with the key `target_code`:

   ```
   conf.set("target_code", desiredResponse);
   ```

   Note: Now, `conf` will store a <key-value> pair: <`target_code`, desiredResponse>.

3. In `TokenizerMapper` class, declear a class-wide variable called `target_code`. Then, in the `setup` function, read the value of `target_code` key from the job configuration. Store it in `target_code`. Below is an example structure you can refer to:

```
// ... indicates some other codes, *this code is not directly
runnable*.
public static class TokenizerMapper extends
        Mapper<LongWritable, Text, NullWritable, Text> {

    String target_code; // The class-wide variable

    @override
    protected void setup (Context context){
        ...
        target_code =
context.getConfiguration().get("target_code"); // Read value from
key `target_code`
    }
    ...
}
```

   Note: Use `org.apache.hadoop.mapreduce.Mapper.Context` and `Configuration#get`.

4. Modify the `map` function to use `target_code` rather than the hard-coded response code that we used in Part II:
   Note: You just need to replace `200` with variable `target_code`.

5. Run your program again to filter the lines with response code `200`. This time, you will need to pass it as a third command-line argument. You can refer to the following example to run your code:

```
# Replace [output-dir-name] with the directory name you want to
store the filter result.
hadoop jar target/[UCRNetID]_lab4-1.0-SNAPSHOT.jar
edu.ucr.cs.cs167.[UCRNetID].Filter nasa_19950801.tsv [output-dir-
name] 200
```

   Note: You may need to give the output directory a new name for multiple runs.

6. Try on both files `nasa_19950801.tsv` and `nasa_19950630.22-19950728.12.tsv`.

   - ***(Q5) How many files are produced in the output for each of the two files?***
   - ***(Q6) Explain this number based on the input file size and default block size in HDFS.***
   - *Hint:* Think about how may blocks are needed to store to two files, respectively.

IV. Run in Distributed Mode (45 minutes)

This part will be done on `cs167` server.

To run your MapReduce program in distributed mode, we will need to configure Hadoop to use YARN and start YARN instances.

Note: YARN stands for Yet Another Resource Negotiator and is the default cluster manager that ships with Hadoop.

1. Login to your CS167 machine.

2. Among your group members that are present in lab, choose the node with the smallest number as the master node.

3. Configure Hadoop to run MapReduce programs with YARN. Edit the file `$HADOOP_HOME/etc/hadoop/mapred-site.xml` and add the following part.

   ```
   <!-- Put all properties inside configuration!!! -->
   <property>
       <name>mapreduce.framework.name</name>
       <value>yarn</value>
   </property>

   <property>
     <name>yarn.app.mapreduce.am.env</name>
     <value>HADOOP_MAPRED_HOME=/home/cs167/cs167/hadoop-3.3.6</value>
   </property>
   <property>
     <name>mapreduce.map.env</name>
     <value>HADOOP_MAPRED_HOME=/home/cs167/cs167/hadoop-3.3.6</value>
   </property>
   <property>
     <name>mapreduce.reduce.env</name>
     <value>HADOOP_MAPRED_HOME=/home/cs167/cs167/hadoop-3.3.6</value>
   </property>
   ```

   Note: If you do not have a `mapred-site.xml` file, make a copy of `mapred-site.xml.template` and name it `mapred-site.xml`.

4. Edit the file `$HADOOP_HOME/etc/hadoop/yarn-site.xml` and add the following part.

   ```
   <!-- Put all properties inside configuration!!! -->
   <property>
       <name>yarn.nodemanager.aux-services</name>
       <value>mapreduce_shuffle</value>
   </property>
   <property>
       <name>yarn.resourcemanager.hostname</name>
       <value>class-###</value>
   </property>
   ```

Note: Replace `class-###` with the name of the master node. If you want to run YARN on your local machine, replace `class-###` with `localhost`.

5. We need to re-start the cluster to apply the changes. Do the following steps:
   (a) Stop all datanodes.
   (b) Stop the namenode.
   (c) Start the HDFS namenode (on the namenode machine).
   (d) Start all datanodes.
   Note: Check the bottom of this page for some common problems that you might face.

6. On the master node, and preferably in `screen` or `tmux`, start the resource manager by running the command:

   ```
   yarn resourcemanager
   ```

   Note: If you met error when running this command, check the common issues at the bottom of instruction.

7. On each data node, and preferably in `screen` or `tmux`, start the node manager (worker) by running the command:

   ```
   yarn nodemanager
   ```

8. Put both test files to your HDFS home directory using the command:

   ```
   hdfs dfs -put nasa_19950801.tsv nasa_19950801_[UCRNetID].tsv
   ```

   Make sure to replace `[UCRNetID]` with your UCR Net ID. This ensures that your group members will not accidentally overwrite your file since you all share the same HDFS home directory. Repeat the same for the other test file to put that in HDFS.

   Note: Makesure your home directory `.` exists in HDFS. If you do not have one, use:

   ```
   hdfs dfs -mkdir -p .
   ```

9. Run your JAR file using the command `yarn jar <*.jar> <main class> <input> <output> <code>`, for example:

   ```
   yarn jar [UCRNetID]_lab4-1.0-SNAPSHOT.jar edu.ucr.cs.cs167.
   [UCRNetID].Filter nasa_19950801_[UCRNetID].tsv
   filter_output_[UCRNetID].tsv 200
   ```

V. Write an Aggregate Program (30 minutes)

[TODO] Keep Yarn or not?

In this part, we will create another MapReduce program that computes the total bytes for each response code. That is the sum of the column `bytes` grouped by the column `response`.

1. Create a new class `Aggregation` based on the following stub code.

```
package edu.ucr.cs.cs167.[UCRNetID];

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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 java.io.IOException;


public class Aggregation {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "aggregation");
        job.setJarByClass(Aggregation.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setNumReduceTasks(2);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

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

        private final IntWritable outKey = new IntWritable();
        private final IntWritable outVal = new IntWritable();
        public void map(LongWritable key, Text value, Context
context)
                throws IOException, InterruptedException {
            if (key.get() == 0)
                return;
            String[] parts = value.toString().split("\t");
            int responseCode = Integer.parseInt(parts[5]);
            int bytes = Integer.parseInt(parts[6]);
            // TODO write <responseCode, bytes> to the output
        }
    }

    public static class IntSumReducer extends
            Reducer<IntWritable, IntWritable, IntWritable,
IntWritable> {

        private final IntWritable result = new IntWritable();

        public void reduce(IntWritable key, Iterable<IntWritable>
values,
                                Context context)
                throws IOException, InterruptedException {
            // TODO write <key, sum(values)> to the output
        }
    }
}
```

2. Implement the **TODO** items to make the desired logic. Hint: look at the WordCount example.

3. Run your program on the file `nasa_19950801.tsv` and check the output directory. You can run it locally first in IntelliJ to test the logic. Once you're satisfied with the result, recompile into a new JAR file, copy it to your CS167 machine, and run as follows on the CS167 machine:
   [TODO]: I got the following error:

```
2025-01-25 07:54:46,553 INFO mapreduce.Job: Job
job_1737791655291_0001 failed with state FAILED due to: Application
application_1737791655291_0001 failed 2 times due to AM Container
for appattempt_1737791655291_0001_000002 exited with  exitCode: 1
Failing this attempt.Diagnostics: [2025-01-25
07:54:45.894]Exception from container-launch.
Container id: container_1737791655291_0001_02_000001
Exit code: 1
```

   And I cannot open the web UI of yarn.
   I check the local log, and the root error is:

```
Caused by: java.lang.reflect.InaccessibleObjectException: Unable to
make protected final java.lang.Class
java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,j
ava.security.ProtectionDomain) throws java.lang.ClassFormatError
accessible: module java.base does not "opens java.lang" to unnamed
module @1e14e2e7
```

I added the following code to `hadoop-env.sh` and `yarn-env.sh`, and restart HDFS and Yarn resource/node managers:

```
export HADOOP_OPTS="$HADOOP_OPTS --add-opens
java.base/java.lang=ALL-UNNAMED"
```

```
yarn jar [UCRNetID]_lab4-1.0-SNAPSHOT.jar edu.ucr.cs.cs167.
[UCRNetID].Aggregation nasa_19950801_[UCRNetID].tsv
aggregation_nasa_19950801_output_dir_[UCRNetID] 200
```

- ***(Q7) How many files are produced in the output directory and how many lines are there in each file?***
- ***(Q8) Explain these numbers based on the number of reducers and number of response codes in the input file.***
- *Note:* The hash function of the class `IntWritable` is its integer value. The default hash partitioner assigns a record to a partition using the function `hashCode mod #reducers`.

4. Run your program on the file `nasa_19950630.22-19950728.12.tsv`.

```
yarn jar [UCRNetID]_lab4-1.0-SNAPSHOT.jar edu.ucr.cs.cs167.
[UCRNetID].Aggregation nasa_19950630.22-19950728.12_[UCRNetID].tsv
aggregation_large_output_dir_[UCRNetID].tsv
```

- ***(Q9) How many files are produced in the output directory and how many lines are there in each file?***
- ***(Q10) Explain these numbers based on the number of reducers and number of response codes in the input file.***

5. Run your program on the output of the `Filter` operation with response code `200` on the file `nasa_19950630.22-19950728.12.tsv`.

   1. Re-run `Filter` program on the file `nasa_19950630.22-19950728.12.tsv`.

```
yarn jar [UCRNetID]_lab4-1.0-SNAPSHOT.jar edu.ucr.cs.cs167.
[UCRNetID].Filter nasa_19950630.22-19950728.12_[UCRNetID].tsv
filter_large_output_[UCRNetID].tsv 200
```

2. Run `Aggregation` program on the output **directory** of `Filter`: filter_nasa_19950630_output_[UCRNetID].tsv

```
yarn jar [UCRNetID]_lab4-1.0-SNAPSHOT.jar edu.ucr.cs.cs167.
[UCRNetID].Aggregation filter_large_output_[UCRNetID].tsv
aggregation_filter_large_output_[UCRNetID].tsv
```

- ***(Q11) How many files are produced in the output directory and how many lines are there in each file?***
- ***(Q12) Explain these numbers based on the number of reducers and number of response codes in the input file.***

VI. Submission (15 minutes)

1. Add a `README.md` file with all your answers (template).
2. Add a `run.sh` script that runs compiles and runs your filter operation on the sample input file with response code `200`. Then, it should run the aggregation method on the same input file. The output files should be named `filter_output` and `aggregation_output` accordingly. Try the `run.sh` file before submission to make sure your code is correct.

Submission file format:

```
[UCRNetID]_lab4.{tar.gz | zip}
  - src/
  - pom.xml
  - README.md
  - run.sh
```

Requirements:

- The archive file must be either `.tar.gz` or `.zip` format.
- The archive file name must be all lower case letters. It must be underscore '_', not hyphen '-'.
- The folder `src` and three files `pom.xml`, `README.md` and `run.sh` must be the exact names.
- The folder `src` and three files `pom.xml`, `README.md` and `run.sh` must be directly in the root of the archive, do not put them inside any folder.
- Do not include any other files/folders, otherwise points will be deducted.

See how to create the archive file for submission at here.

Rubrics

- Q/A: +12 points (+1 point for each question)
- Code: +2 points
  - +1 for completing filter class
  - +1 for completing aggregate class
- Following submission instructions: +1 point

Useful Hadoop Filesystem Commands

Check all Hadoop filesystem commands from https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

Specifically, you would be interested in the following commands:

- Upload file: put
- Create a directory: mkdir
  - To create directory with parent directoriesl use argument -p.
- Delete file: rm
  - To delete a directory recursively, use argument -r.
- List files: ls
- Print text file content: cat
  - Do not use this command on large files, otherwise your terminal may freeze.
- Print the first few lines of a text file: head
- Print the last few lines of a text file: tail

Example commands

```
# Upload file to hdfs root
hadoop fs -put nasa_19950801.tsv /

# Create a directory
hadoop fs -mkdir -p /dir1/dir2

# Upload file to hdfs under some directory
hadoop fs -put nasa_19950630.22-19950728.12.tsv /dir1/dir2/

# List directory contents
hadoop fs -ls /dir1/dir2

# Delete a directory
hadoop fs -rm -f -r /dir1
```

Common Errors

- Error: When I run my program on YARN, I see an error message similar to the following.

  ```
  Failing this attempt.Diagnostics: [...]Container
  [pid=xxx,containerID=xxx] is running beyond virtual memory limits.
  ```

```
Current usage: xxx MB of yyy GB physical memory used; xxx TB of yyy
GB virtual memory used. Killing container.
```

- Fix: Add the following configuration to your `$HADOOP_HOME/etc/yarn-site.xml`.

  ```
  <property>
      <name>yarn.nodemanager.vmem-check-enabled</name>
      <value>false</value>
  </property>
  ```

  See also https://stackoverflow.com/questions/21005643/container-is-running-beyond-memory-limits

- Error: When I run any HDFS command, I get an error related to safemode

```
Cannot create file/user/cs167/nasa_19950630.22-
19950728.12.tsv._COPYING_. Name node is in safe mode.
```

- Fix: Run the following command

```
hdfs dfsadmin -safemode leave
```

- Error: When I run the datanode, I get the following error:

```
java.io.IOException: Incompatible clusterIDs in
/home/cs167/hadoop/dfs/data: namenode clusterID = CID-ca13b215-c651-
468c-9188-bcdee4ad2d41; datanode clusterID = CID-d9c134b6-c875-4019-
bce0-2e6f8fbe30d9
```

- Fix: Do the following steps to ensure a fresh start of HDFS:

1. Stop the namenode and all data nodes.
2. Delete the directory `~/hadoop/dfs` on *the namenode and all datanodes*. `rm -rf ~/hadoop/dfs`.
3. Reformat HDFS using the command `hdfs namenode -format`.
4. Start the namenode using the command `hdfs namenode`.
5. Start the datanode using the command `hdfs datanode`.

- Error: When I run `yarn resourcemanager`, I got the following error:

```
error: Caused by: java.lang.reflect.InaccessibleObjectException: Unable
to make protected final java.lang.Class
java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.s
ecurity.ProtectionDomain) throws java.lang.ClassFormatError accessible:
module java.base does not "opens java.lang" to unnamed module @1e14e2e7
```

- Fix: add the following command to the very bottom of `$HADOOP_HOME/etc/hadoop/hadoop-env.sh`:

```
export HADOOP_OPTS="$HADOOP_OPTS --add-opens java.base/java.lang=ALL-
UNNAMED"
```