Lab1 - MapReduce, HDFS, and HBase

Group: Sandra_Anna

Sandra Pico Oristrell <sandrapo@kth.se>, Anna Hedström <annaheds@kth.se>

September 21, 2018

In this assignment, we demonstrate our solution for a MapReduce program written using HDFS and Hbase. For this, we are given a list of user records (from StackOverflow) in a users.xml file. To retrieve the top ten users with respect to their reputation, we had to create a mapper class and a reducer class that enabled us to parse and print the results. Please find a snippet from the xml data below.

Code 1: XML Data

```
<?xml version="1.0" encoding="utf-8"?>
<users>
  <row Id="-1" Reputation="1" CreationDate="2014-05-13T21:29:22.820"
  DisplayName="Community" LastAccessDate="2014-05-13T21:29:22.820"
  WebsiteUrl="http://meta.stackexchange.com/"  Location="on the server farm"
  AboutMe="&l...." Views="0" UpVotes="749"  DownVotes="221"  AccountId="-1" />
  ...
```

1. Implementation

The following section comments on the important parts of the source code. The full source code can be found in the Appendix 1: Source code.

1.1 Mapper Class

Below one can find the mapper class. The idea is to have the mapper reading the input data from HDFS and thereafter output the top ten records to the reduce phase, as communicated via the cleanup method.

- First, we use a TreeMap to store the input records of user reputation, named repToRecordMap. The TreeMap data type is convenient to use since it allow us to sort on key.
- Next, the map method will get the "Id" and "Reputation" data as a single String value. As suggested, we use the transformXmlToMap helper function to parse the users.xml file. Here, we also make sure to skip the rows that do not contain any user data, i.e., where userString == null.

Code 2: Mapper Class

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

    TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

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

        Map<String, String> mapString = transformXmlToMap(value.toString());
        String userString = mapString.get("Id");
        String reputationString = mapString.get("Reputation");

        if (userString == null) {
                return;
        }

        Integer reputationInt = Integer.parseInt(reputationString);

        repToRecordMap.put(reputationInt, new Text(value));

        if (repToRecordMap.size() > 10) {
                repToRecordMap.remove(repToRecordMap.firstKey());
        }
    }

     protected void cleanup(Context context) throws IOException, InterruptedException {

                for (Text information : repToRecordMap.values()) {
                        context.write(NullWritable.get(), information);
                }
        }
}
```

- Now we can convert the data to the correct format, namely `Integer` and `Text` which `put` into the `TreeMap`.
- Since we only want to output ten top records, we remove the excessive keys from the `repToRecordMap`.
- Before exiting the task, the `cleanup` method will be called, so to output the ten records to the reducer with a `null` key.

1.2 Reducer Class

Below one can find the `reducer class`. The idea here is to have the reducer to write the result in a `Hbase` table and thereafter print the top ten users using the `scan` method in the `Hbase` shell.

- First, the `reducer class` looks very similar to the `mapper class`. We use a `TreeMap` to store the input records of user reputation, which we again name `repToRecordMap`. We also convert to appropriate data types, i.e., `Integer` and `Text`.
- Next, in the `reduce` method, given that we only have one reducer (we configured our job so that we only have one input group, using `job.setNumReduceTasks(1)`, we can loop over *all* the `Text` values.

Code 3: Reducer Class

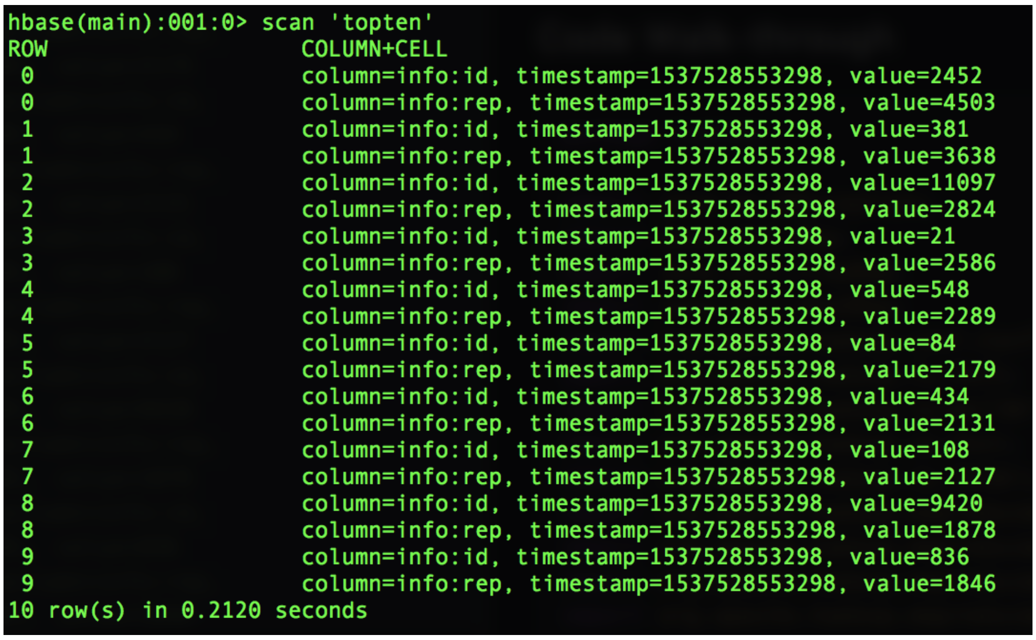
```
public static class TopTenReducer extends TableReducer<NullWritable, Text, NullWritable> {

        private TreeMap<Integer, Text> repToRecordMap =
        new TreeMap<Integer, Text>();

        public void reduce(NullWritable key, Iterable<Text> values,
        Context context) throws IOException, InterruptedException {

        try {
                for(Text user_value: values){

                        Map<String, String> mapString =
                        transformXmlToMap(user_value.toString());
                        String userString = mapString.get("Id");
                        String reputationString = mapString.get("Reputation");

                        Integer reputationInt = Integer.parseInt(reputationString);
                        Text userText = new Text(userString);

                        repToRecordMap.put(reputationInt, userText);

                        if (repToRecordMap.size() > 10) {
                                repToRecordMap.remove(repToRecordMap.firstKey());
                        }
                }

                Integer count = 0;

                for (Integer data: repToRecordMap.descendingKeySet()){

                        Put insHBase = new Put(Integer.toString(count).getBytes());

                        insHBase.addColumn(Bytes.toBytes("info"),
                        Bytes.toBytes("rep"),
                        Integer.toString(data).getBytes());
                        insHBase.addColumn(Bytes.toBytes("info"),
                        Bytes.toBytes("id"),
                        repToRecordMap.get(data).getBytes());

                        context.write(NullWritable.get(), insHBase);
                        count += 1;
                }
        }
        catch(Exception e) {
                e.printStackTrace();
        }
}
}
```

- In our `repToRecordMap` we put the ”Reputation” as a key and the ”Id” as value. Again, we only want the top ten records.
- Next step is to write the data to our `HBase` table, which we named `topten`. We loop through the user records via `repToRecordMap.descendingKeySet())` and maintain a `count` for each of the iterations. In this loop, the first thing we do is to create a `Put Hbase` table.
- Next, using the `addColumn` method, we add the top ten records data to the particular columns, i.e., `info:rep` and `info:id` into our `Hbase` table.
- Lastly, we can write the data to the `Hbase` table. We use the method `context.write` to insert the `insHbase` table data into our `topten` table.

2. Results

In the figure 1 below, one can view the final results as generated in the `Hbase` shell. As seen, there are ten row instances displayed, one carrying the information about the `info:id` and one about the `info:rep`. As seen, the highest ”Reputation” value is 4503 for user 2452, and the lowest is 1846 for user 836.

Figure 1: Final Output: Top ten users by reputation


```
hbase(main):001:0> scan 'topten'
ROW                       COLUMN+CELL
 0                        column=info:id, timestamp=1537528553298, value=2452
 0                        column=info:rep, timestamp=1537528553298, value=4503
 1                        column=info:id, timestamp=1537528553298, value=381
 1                        column=info:rep, timestamp=1537528553298, value=3638
 2                        column=info:id, timestamp=1537528553298, value=11097
 2                        column=info:rep, timestamp=1537528553298, value=2824
 3                        column=info:id, timestamp=1537528553298, value=21
 3                        column=info:rep, timestamp=1537528553298, value=2586
 4                        column=info:id, timestamp=1537528553298, value=548
 4                        column=info:rep, timestamp=1537528553298, value=2289
 5                        column=info:id, timestamp=1537528553298, value=84
 5                        column=info:rep, timestamp=1537528553298, value=2179
 6                        column=info:id, timestamp=1537528553298, value=434
 6                        column=info:rep, timestamp=1537528553298, value=2131
 7                        column=info:id, timestamp=1537528553298, value=108
 7                        column=info:rep, timestamp=1537528553298, value=2127
 8                        column=info:id, timestamp=1537528553298, value=9420
 8                        column=info:rep, timestamp=1537528553298, value=1878
 9                        column=info:id, timestamp=1537528553298, value=836
 9                        column=info:rep, timestamp=1537528553298, value=1846
10 row(s) in 0.2120 seconds
```

3. Getting Started

This section is dedicated to describe the steps needed to run the program. For this, we assume a successful installation of `HDFS` and `Hbase`. To begin with, we set some environmental variables that was needed to make the `Hadoop native library` work. We run our program on `macOS` so extra configuration was required.

Code 4: Run the program

```
// Create environmental variables for Hadoop native library.

    > export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
    > export OPENSSL_ROOT_DIR="/usr/local/opt/openssl"
    > export LDFLAGS="-L${OPENSSL_ROOT_DIR}/lib"
    > export CPPFLAGS="-I${OPENSSL_ROOT_DIR}/include"
    > export PKG_CONFIG_PATH="${OPENSSL_ROOT_DIR}/lib/pkgconfig"
    > export OPENSSL_INCLUDE_DIR="${OPENSSL_ROOT_DIR}/include"
    > export HADOOP_OPTS="-Djava.net.preferIPv4Stack=true-Djava.library.path  ...
    > =/Users/.../hadoop-2.9.0/lib/native"

    // Check that native now works.

    > hadoop checknative
      Native library checking:
      hadoop:  true Users/.../hadoop-dist/target/...//native/libhadoop.dylib
      zlib:    true /usr/lib/libz.1.dylib
      snappy:  true /usr/local/lib/libsnappy.1.dylib

    // Start the HDFS namenode and datanode.

    > $HADOOP_HOME/sbin/hadoop-daemon.sh start namenode
    > $HADOOP_HOME/sbin/hadoop-daemon.sh start datanode

    // Start the Hbase shell.

    > $HBASE_HOME/bin/start-hbase.sh
```

```
// Create a Hbase table 'topten' and add 'info' column-family.
// Then we exit the shell.

> $HBASE_HOME/bin/hbase shell
        > create 'topten', 'info'
        > exit

// Go to src folder and in HDFS, create topteninput folder.
// Add the users.xml data to the topteninput folder just created.

> cd lab1/src/topten
> $HADOOP_HOME/bin/hdfs dfs -mkdir -p /topteninput
> $HADOOP_HOME/bin/hdfs dfs -put topten/- data/users.xml /topteninput

// Compile the java file and place it in the toptenclasses folder.
// Generate the jar file and run the program.

> javac -cp $HADOOP_CLASSPATH -d toptenclasses topten/TopTen.java
> jar -cvf topten.jar -C toptenclasses/ .
> $HADOOP_HOME/bin/hadoop jar topten.jar topten.TopTen /topteninput /toptenoutput

// Check that the user records were successfully ritten to the Hbase table.
// Use scan 'topten' to view the content of the rows.

> $HBASE_HOME/bin/hbase shell
        > scan 'topten'
```

4. Appendix: Code

Code 5: Source Code

```
package topten;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;
import java.util.HashMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
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.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
```

```java
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.filter.FirstKeyOnlyFilter;

public class TopTen {

// This helper function parses the stackoverflow into a Map for us.
public static Map<String, String> transformXmlToMap(String xml) {
        Map<String, String> map = new HashMap<String, String>();
        try {
                String[] tokens = xml.trim().substring(5,
                xml.trim().length() - 3).split("\"");
                for (int i = 0; i < tokens.length - 1; i += 2) {
                        String key = tokens[i].trim();
                        String val = tokens[i + 1];
                        map.put(key.substring(0, key.length() - 1), val);
                }
        } catch (StringIndexOutOfBoundsException e) {
                System.err.println(xml);
        }
        return map;
}

//Class to implement the map function.
public static class TopTenMapper extends
Mapper<Object, Text, NullWritable, Text> {

        // Stores a map of user reputation.
        TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

        // The map function received the whole users.xml in the Text value.
        // However, MapReduce handle everything in a "parallel" way. Then,
        // the valuethat we will receive here,it will actually be only one
        //  row of the xml file.
        public void map(Object key, Text value, Context context) throws
        IOException, InterruptedException {

                // First we need to use the transformXmlToMap function:
                // The map method will get the users.xml as a single string: value.
                Map<String, String> mapString = transformXmlToMap(value.toString());

                //As it is only one id-reputation pair, we just need to
                // read it by "Id" and "Reputation"
                String userString = mapString.get("Id");
                String reputationString = mapString.get("Reputation");

                // Skip the rows that do not contain the Id.
                if (userString == null) {
                        return;
                }

                // Convert to integer and text.
                Integer reputationInt = Integer.parseInt(reputationString);

                // Now we need to put it into the TreeMap, with Integer and Text.
                // TreeMap needs the reputation as a key. However, as a Text, we
                // are sending the whole information. So, being the value that
                // we received through the map function.
                repToRecordMap.put(reputationInt, new Text(value));
                // We only want the ten top records so we remove the last key,
                // given it is in descending order.
                if (repToRecordMap.size() > 10) {
                        repToRecordMap.remove(repToRecordMap.firstKey());
                }
        }

        protected void cleanup(Context context) throws IOException,
        InterruptedException {
                // Output our ten records to the reducers with a null key.
                // The ten outputs that we will send will contain the whole
                // information: being the id and the resolution part of it.
                for (Text information : repToRecordMap.values()) {
                        context.write(NullWritable.get(), information);
                }
        }
}

public static class TopTenReducer extends
TableReducer<NullWritable, Text, NullWritable> {

        // Stores a map of user reputation to the record.
        private TreeMap<Integer, Text> repToRecordMap =
        new TreeMap<Integer, Text>();

        public void reduce(NullWritable key, Iterable<Text> values,
        Context context) throws IOException, InterruptedException {

        try{
        // We only have one reducer so we can iterate for the whole
        // Text values and save it into our HBase Table structure.

                // The reducer needs to store again the 10 received values
                // into a TreeMap.
                for(Text user_value: values){

                        Map<String, String> mapString =
                        transformXmlToMap(user_value.toString());

                        String userString = mapString.get("Id");
                        String reputationString = mapString.get("Reputation");

                        // Convert to integer and text.
                        Integer reputationInt = Integer.parseInt(reputationString);
                        Text userText = new Text(userString);

                        // In this tree, we will save the reputation as a key, and the
                        // user id as the value.
                        repToRecordMap.put(reputationInt, userText);

                        if (repToRecordMap.size() > 10) {
                                repToRecordMap.remove(repToRecordMap.firstKey());
                        }

                }

                Integer count = 0;

                // Going to the TreeMap:
                for (Integer data: repToRecordMap.descendingKeySet()){
                        // Create Hbase put.
                        Put insHBase = new Put(Integer.toString(count).getBytes());

                        //Add the particular columns (rep and id).
                        insHBase.addColumn(Bytes.toBytes("info"), Bytes.toBytes("rep"),
                        Integer.toString(data).getBytes());
                        insHBase.addColumn(Bytes.toBytes("info"), Bytes.toBytes("id"),
                        repToRecordMap.get(data).getBytes());

                        // Write data to Hbase Table.
                        context.write(NullWritable.get(), insHBase);
                        count += 1;
                }

        } catch(Exception e) {
                e.printStackTrace();
        }
        }

        }

        public static void main(String[] args) throws Exception {

                // Configure new job.
                Configuration conf = new Configuration();

                Job job = Job.getInstance(conf,"Topten reputation");
                job.setJarByClass(TopTen.class);

                // Set the Mapper, Output- and Reduce classes.
                job.setMapperClass(TopTenMapper.class);
                job.setMapOutputKeyClass(NullWritable.class);
                job.setMapOutputValueClass(Text.class);
                job.setReducerClass(TopTenReducer.class);
                job.setNumReduceTasks(1);

                //Input file: users.xml
                FileInputFormat.addInputPath(job, new Path(args[0]));

                //Output: Define HBase Table.
                TableMapReduceUtil.initTableReducerJob("topten",
                TopTenReducer.class, job);

                job.waitForCompletion(true);


        }
}
```