Lab1 - ID2221 HT20-1 Data-Intensive Computing

-Mattia Zoffoli & Varrun Varatharajan

Code Summary

The main idea of this exercise is to print out top ten users based on the given reputation score. We use TreeMap, which reads input from HDFS to find top ten records, which will then be split and output to reduce phase to be written in HBase.

First, we import all necessary libraries necessary for all operations of HDFS, HBase, and MapReduce.

```
package id2221.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;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.filter.FirstKeyOnlyFilter;
```

Inside the public class called TopTen, the mapper helps parse the input in .xml to a map with xml.trim(). This trims the required parts of a data using substring (5), (length-3), and “\””. Here, we get reputation string alone from parsing, which is the ultimate goal.

```
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;
    }
```

Now we iterate through the input records using get(), put() to get the parsed records. The rows with a null ID or reputation will be ignored. Put() adds the iterated records to the map, with the reputation as a key. The entire map is a dictionary of key: value pairs. Finally, we want only the top ten reputation values. Hence, the record size is set to 10, and the lowest key (lowest reputation) among the bunch is automatically not included in the top 10. This is represented by firstKey() which has the lowest value.

```
public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
    // Stores a map of user reputation to the record
    TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map<String, String> parsevalue = TopTen.transformXmlToMap(value.toString());
    if (parsevalue != null) {
        System.out.println("Values found");
    }
    else {
        System.out.println("Null value found, returning");
        return;

    }

    String userId = parsevalue.get("AccountId");
    String reputation = parsevalue.get("Reputation");
    if (userId == null || reputation == null) {
        if (userId == null) {
            System.out.println("The accountid was found to be empty");
            return;
        }
        else if ( reputation == null) {
            System.out.println("The reputation was found to be empty");
            return;
        }


    }

    // Add to map. Reputation is key
    repToRecordMap.put(Integer.parseInt(reputation), new Text(value));

    // Remove lowest reputation outside top ten
    if (repToRecordMap.size() > 10) {
    repToRecordMap.remove(repToRecordMap.firstKey());
    }
}
```

The Mapper processes key:value pairs one at a time, and writes it to intermediate results. To know the top 10, entire block needs to be iterated, hence context.write() in cleanup is used.

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


{
    for (Map.Entry<Long, String> recordEnt : repToRecordMap.entrySet())
    {

        int count = recordEnt.getKey();
        String name = recordEnt.getValue();

        context.write(count, new Text(name));
    }
}
}
```

Similar in logic to the mapper phase, the reducer also processes key:value pair one by one. Here, the final storage is done by reputation as key. The records are finally written and stored on HBase.

```
//Reducer
    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 {
       for (Text value : values) {
        Map<String, String> parsevalue = TopTen.transformXmlToMap(value.toString());
        String reputation = parsevalue.get("Reputation");

        //Add record. Reputation is key
        repToRecordMap.put(Integer.parseInt(reputation), new Text(value));

        // Lowest reputation is removed after top ten
        if (repToRecordMap.size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
        }
        }
```

Now all the keys, values have been mapped. AccountID and reputation score from TreeMap should be stored in HBase. It is stored as columns.

```
    for (Text val : repToRecordMap.descendingMap().values()) {
    Map<String, String> parsevalue = TopTen.transformXmlToMap(val.toString());
    String reputation = parsevalue.get("Reputation");
    String accountid = parsevalue.get("AccountId");

    Put insertHBase = new Put(reputation.getBytes());

    //Insert and write to HBase
    insertHBase.addColumn(Bytes.toBytes("value"), Bytes.toBytes("reputation"), Bytes.toBytes(reputation));
    insertHBase.addColumn(Bytes.toBytes("value"), Bytes.toBytes("accountid"), Bytes.toBytes(accountid));
    context.write(null, insertHBase);
    }
}
public void cleanup(Context context) throws IOException, InterruptedException
{

    for (Map.Entry<string, String> entry : repToRecordMap.entrySet())
    {

        String reputation = entry.getKey();
        String Values = entry.getValue();
        context.write( new Text(reputation), new Text(name));
    }
}

}
```

The final driver code sets the job instance to be executed. Initiating job.setOutputKeyClass(NullWritable.class); sets the types expected from output of map and reduce.

setNumReduceTasks(1) ensures one input batch for the reducer containing all mapped top 10 records. Reducer stores all these in TreeMap to be stored in “topten” in HBase.

```java
public static void main(String[] args) throws Exception {
    Configuration conf = HBaseConfiguration.create();
//Job and class are set
Job job = Job.getInstance(conf, "topten");
job.setJarByClass(TopTen.class);
job.setMapperClass(TopTenMapper.class);
job.setReducerClass(TopTenReducer.class);

TableMapReduceUtil.initTableReducerJob("topten", TopTenReducer.class, job);

job.setOutputKeyClass(NullWritable.class);
job.setOutputValueClass(Text.class);
job.setNumReduceTasks(1);

FileInputFormat.addInputPath(job, new Path(args[0]));
System.exit(job.waitForCompletion(true) ? 0 : 1);

}
```

Running the code

The file TopTen.java contains the code, and users.xml contains input records.

First start NAMENODE_DIR and DATANODE_DIR:

```
$HADOOP_HOME/bin/hdfs --daemon start namenode

$HADOOP_HOME/bin/hdfs --daemon start datanode
```

Create folder for HDFS input and upload files:

```
$HADOOP_HOME/bin/hdfs dfs -mkdir -p topten_input

$HADOOP_HOME/bin/hdfs dfs -put data/users.xml

$HADOOP_HOME/bin/hdfs dfs -ls topten_input
```

Start HBase:

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

$HBASE_HOME/bin/hbase shell
```

Set environment variables:

```
export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)

export HBASE_CLASSPATH=$($HBASE_HOME/bin/hbase classpath)

export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_CLASSPATH
```

Change directory to /src and make directory topten_classes for the compiled files:

```
mkdir topten_classes

javac -cp $HADOOP_CLASSPATH -d topten_classes topten/TopTen.java

jar -cvf topten.jar -C topten_classes/ .
```

Run application and check result in HBase shell:

```
$HADOOP_HOME/bin/hadoop jar topten.jar id2221.topten.TopTen topten_input topten_output

scan 'topten'
```