ID2221 – Data Intensive Computing

Lab 1 - MapReduce, HDFS and HBASE

Course Coordinator:
Amir H. Payberah

Óttar Guðmundsson

Xin Ren

2018-9-6

In this lab assignment, we were tasked with finding the top ten highest users based on their reputation rank using MapReduce and storing them in HBASE.

How to run

1. Start both a namenode and a datanode in Hadoop (make sure to set environment variables)

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

2. Change directory to the topten folder

cd lab1/src/topten

3. Create a directory in the hdfs and put the xml file in that directory

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

4. Start up a hbase shell and create a table called topten with the column info

$HBASE_HOME/bin/start-hbase.sh
$HBASE_HOME/bin/hbase shell
create 'topten', 'info'

5. Compile the class in Java and run the file. Make sure to pass the location of the file input as argument.

$HADOOP_HOME/bin/hadoop jar topten.jar topten.TopTen topten_input

6. After the code has finished, start the HBASE shell again and scan the table

scan 'topten'

Main class

Since we were given two base examples in running a MapReduce application, we had to combine their functionality to read from a file in HDFS and then store the result in HBASE.

A new function was introduced to the job, called *setNumReduceTasks* to make sure only one reduce function would be used to finalize the list. If there would have been more than 1 reduce task, we would end up with multiple top ten lists.

Files were read using the
FileInputFormat.addInputPath(job, new Path(args[0]));
since the XML file was stored in HDFS but not in a HBASE table. So to access the data, the file path needed to be added as an argument.

Finally, the results of the reducer was saved in the table *topten* created in HBASE using the
TableMapReduceUtil.initTableReducerJob("topten", TopTenReducer.class, job);
command.

Mapper

When mapper receives a xml string which is one line from the data, it parses the string into a map of parameter to value with function *transformXmlToMap*. If the map contains parameter(key) “Id”, it means the line is a valid user record and then it is stored in a TreeMap *repToRecordMap* ordered

with “Reputation” as key. *pollfirstEntry* function is used to kept only 10 entries with highest “Reputation” in the *repToRecordMap*.

After all the lines in the data have been parsed and processed as described above, the *cleanup* function of the mapper loops over the *repToRecordMap* and writes the 10 users with highest reputation in the context, which will be received by the Reducer.

Reducer

The reducer simply does very similarly as the mapper but no need to check if the inputs from mappers contain “Id” as the inputs are users with top ten highest reputation found at every mapper. It loops over all inputs, adds them to a TreeMap *repToRecordMap* of “Reputation” to “Id”, which is ordered by “Reputation” and polls the lowest reputation everytime when it adds a new entry to keep only ten entries with highest reputation in the *repToRecordMap*.

After that, all ten entries in the *repToRecordMap* are written into the HBASE table *topten* in descending order of reputation.

Results

We ended up storing the top ten highest reputation in descending order in the HBASE. Our implementation was that the user with the highest reputation was stored as row 0, the second highest as row 1 and so on. This simplifies accessing the wanted users since those who use the table will always know that the highest reputation is in row 0, the fifth highest would be in row 4 and so on.

As seen below, these were the top ten users found.

```
hbase(main):001:0> scan "topten"
ROW                                   COLUMN+CELL
 0                                    column=info:id, timestamp=1536410315912, value=2452
 0                                    column=info:rep, timestamp=1536410315912, value=4503
 1                                    column=info:id, timestamp=1536410315912, value=381
 1                                    column=info:rep, timestamp=1536410315912, value=3638
 2                                    column=info:id, timestamp=1536410315912, value=11097
 2                                    column=info:rep, timestamp=1536410315912, value=2824
 3                                    column=info:id, timestamp=1536410315912, value=21
 3                                    column=info:rep, timestamp=1536410315912, value=2586
 4                                    column=info:id, timestamp=1536410315912, value=548
 4                                    column=info:rep, timestamp=1536410315912, value=2289
 5                                    column=info:id, timestamp=1536410315912, value=84
 5                                    column=info:rep, timestamp=1536410315912, value=2179
 6                                    column=info:id, timestamp=1536410315912, value=434
 6                                    column=info:rep, timestamp=1536410315912, value=2131
 7                                    column=info:id, timestamp=1536410315912, value=108
 7                                    column=info:rep, timestamp=1536410315912, value=2127
 8                                    column=info:id, timestamp=1536410315912, value=9420
 8                                    column=info:rep, timestamp=1536410315912, value=1878
 9                                    column=info:id, timestamp=1536410315912, value=836
 9                                    column=info:rep, timestamp=1536410315912, value=1846
10 row(s) in 0.8790 seconds
```