Assignment 3: HBase

Arash Vahdat

February 24, 2015

Readings You are highly recommended to read the following chapters while doing this assignment:

- Chapter 1: Introduction, “HBase, The Definitive Guide”, Lars George.
- Chapter 13: HBase, “Hadoop, The Definitive Guide”, Tom White, the third edition.
- Chapter 3: Client API, The Basics, “HBase, The Definitive Guide”, Lars George.
- Chapter 4: Client API, Advanced Features, “HBase, The Definitive Guide”, Lars George.
- Chapter 7: MapReduce Integration, “HBase, The Definitive Guide”, Lars George.

Provided files An archive file with all necessary files for this assignment can be found at the following URL:

```
http://www.cs.sfu.ca/~avahdat/personal/files/cmpt732_assignment3.zip
```

Download it and extract its content.

1 HBase Basics

HBase is a column-oriented distributed database built on top of Hadoop Distributed File System (HDFS). In this assignment, we are going to learn the basics of storing and processing large collections of data using HBase. To start, we will use command-line and Java programs to interact with HBase databases and later on we will develop more advanced systems using MapReduce to process data residing in HBase databases.

1.1 HBase shell

HBase shell can be used to administrate databases. Similar to Linux shell, HBase shell has a set of predefined commands that can be used to view and update a database. SSH to rcg-hadoop server and try the following command to start the shell:

```
$ hbase shell
```

This will bring up a JRuby IRB(Interactive Ruby Shell) interpreter that has some HBase-specific commands added to it. Type the following to get a general help on all commands:

```
$ help
```

or type the following to get help on specific command groups:

```
$ help “Command_Group”
```

or type the following to get help on a particular command:

```
$ help “create”
```

In order to create a table, you need to define a name and a schema (which includes the table attributes and the list of table column families). A schema can be later edited by off-lining the table using the “disable” command, making the necessary alternations using “alter”, then, putting the table back online with the “enable” command.

The current HBase installation on our cluster is not secured. In other words, every student can alter every table on the database. To avoid any conflict 1) always name your table with names that start with a unique prefix. (Your SFU computing ID or a unique prefix that instructors will assign to you in class) 2) never change other students tables.

Let’s create a test table called `“UniquePrefix_test”`[1] with a column family name `‘data’`

```
$ create ‘test’, ‘data’
```

To check whether the table is created or not use:

```
$ list
```

Let’s put some data in our table. Here, we will create three different rows with different columns under the same column family:

```
$ put ‘test’, ‘row1’, ‘data:name’, ‘alex’
$ put ‘test’, ‘row2’, ‘data:age’, ‘37’
$ put ‘test’, ‘row3’, ‘data:city’, ‘vancouver’
```

Now you can use the “scan” command to list all entries in your table:

```
$ scan ‘test’
```

You can also count the number of rows in the table using the following command without actually scanning the whole table:

```
$ count ‘test’
```

In order to remove the table you must “disable” it before “dropping” it:

```
$ disable ‘test’
$ drop ‘test’
```

So far, we have created a table named `‘test’` with one column family, `‘data’`. We added three rows with three different columns. Run the same commands again and create the same table. Try to answer the following questions:

[1] After this we remove “UniquePrefix_” from the name of tables for clarity.

Question 1 Write down a command that will add age to 'alex'.

Question 2 How can we update the age in row2?

Question 3 How can we use 'scan' to show all rows with a value under the column 'data:age'?

Question 4 Imagine the entry in row3 changes from Vancouver to New York. How can we add a new city in 'data:city' for row3 without losing the previous city?

Question 5 How can we use 'scan' to show all versions in all cells?

Question 6 Imagine a user made a mistake when entering the new city and misspelled the word 'New York' for row3. How can we change the last version of 'data:city' without creating a new version?

Question 7 Try adding gender information using the following command:

```
$ put 'test', 'row3', 'personal:gender', 'male'
```

Explain what went wrong and how the problem can be fixed.

1.2 Java Client

HBase shell used in the previous part is based on JRuby. In this section, we will develop Java programs to interact with HBase databases. Let's start with setting up the Eclipse environment. First, we need to download HBase and Hadoop packages from Apache:

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.4.0/hadoop-2.4.0.tar.gz
$ tar -xvzf hadoop-2.4.0.tar.gz
$ wget http://archive.apache.org/dist/hbase/hbase-0.98.7/hbase-0.98.7-hadoop2-bin.tar.gz
$ tar -xvzf hbase-0.98.9-hadoop2-bin.tar.gz
```

Note, that if you already have Hadoop packages from assignment 1, you do not need to download them again. Similar to what you did in the assignment 1, to setup Eclipse, create a new project called CMPT732A3-HBase and follow the instructions from assignment 1 to add a build.xml file to automate building Jar archives. When you are adding external libraries to your build path, you need to add the following jar files:

- `hadoop-2.4.0/share/hadoop/common/hadoop-common-2.4.0.jar`
- `hadoop-2.4.0/share/hadoop/hdfs/hadoop-hdfs-2.4.0.jar`
- `hadoop-2.4.0/share/hadoop/MapReduce/hadoop-MapReduce-client-common-2.4.0.jar`
- `hadoop-2.4.0/share/hadoop/MapReduce/hadoop-MapReduce-client-core-2.4.0.jar`
- `hbase-0.98.7-hadoop2/lib/hbase-common-0.98.7-hadoop2.jar`
- `hbase-0.98.7-hadoop2/lib/hbase-client-0.98.7-hadoop2.jar`

- `hbase-0.98.7-hadoop2/lib/hbase-server-0.98.7-hadoop2.jar`
- `hbase-0.98.7-hadoop2/lib/hbase-protocol-0.98.7-hadoop2.jar`
- `hbase-0.98.7-hadoop2/lib/high-scale-lib-1.1.1.jar`
- `hbase-0.98.7-hadoop2/lib/htrace-core-2.04.jar`
- `hbase-0.98.7-hadoop2/lib/hbase-hadoop-compat-0.98.7-hadoop2.jar`
- `hbase-0.98.7-hadoop2/lib/protobuf-java-2.5.0.jar`

Question 1 Locate the file `ExampleClient.java` in the archive files provided with this assignment. Add this class to your working project and try building and running it on rcg-hadoop cluster. For building, you can simply use (*Ctrl+B*) after setting up your build.xml file. For running the client, ssh to rcg-cluster, and run the following lines in the directory that contains the built ExampleClient.jar.

```
$ sh
$ CLASS_PATH=$(hbase classpath)
$ java -cp ${CLASS_PATH}:ExampleClient.jar org.CMPT732A3.ExampleClient
```

Review the client code in ExampleClient.java. In line 39, the result obtained from a “get” command is printed to the standard output. While running it, try to locate the output line associated with this command in the standard output. Modify the print command such that it prints a structured output with the following format:

```
rowKey,  columnFamily:qualifier, value
```

Question 2 Modify the ExampleClient.java to scan over the whole table. Your scanner should print each row in the format mentioned above.

Question 3 Let’s use a small database containing the weather data from 2013 and try a few techniques for loading data from a database. You can find this database in HBase under the name “avahdat_weather”. Check a few entries of this table. Report the row key format, column family and qualifiers used in this table.

Question 4 Write a Java client program that interacts with the weather database. Your program should find the day and the station with the maximum snowfall in 2013. Note that the amount of snowfall is stored in millimeters under the “SNOW” column.

Program this part with and without a qualifier filter. Report the run time and briefly explain why the qualifier filter speeds up the process.

Question 5 Did we get the whole row (with all columns), when we specified a qualifier filter in the previous section? Does this effect the running time?

Question 6 If you check some rows in the database manually, you will notice that most snowfall entries contain zero values. One way to speed up the program is to define a value filter that filters out these entries. Modify your code from the previous question and scan only the entries that contain nonzero values under the “SNOW” column using a SingleColumnValueFilter. How fast is your client now?

Question 7 You probably noticed how a SingleColumnValueFilter can further speed up scanning the whole database by focusing on nonzero entries. In some cases, we may need to combine multiple filters to further prune undesired entries. For this purpose, we can use the FilterList class in HBase to pass a list of filters to a scanner. Assume we have some prior knowledge that the maximum snowfall was recorded in one of "USC" stations. Create a FilterList combining two filters. The first filter should select nonzero entries of the "snow" column and the second filter should find rows that correspond to USC. Report the run time in this case.

Question 8 Another way to speed-up scanning of a database is to enable caching results on the client side. This will reduce the number of RPC calls to the HBase server. Enable caching for your scanner and examine different caching values. What was the best timing that you obtained by tuning your caching amount?

2 Movie Recommendation System

HBase tables can be integrated easily with Hadoop MapReduce framework. This gives a great flexibility for a data analyst to process the whole data stored in a big HBase database in a batch mode. Consider the case of running an online service. You may gather different information from your users while they are interacting with your website. The information can be stored in an HBase database and daily data process can be done using MapReduce tasks.

In this part, we are going to design a movie recommendation system by processing movie ratings provided by users[2]. We are going to use HBase as a mean to store rating data and we will create several MapReduce jobs that process user ratings. We are looking for movie similarity measures that measure how two movies are similar. We can use these measures to suggest movies for a given user profile.

In this part, we are going to use ratings from the Movie Tweeting dataset[3]. The dataset was collected by Simon Dooms[4] by processing public tweets on Twitter and it contains 100,000 anonymous ratings of approximately 10,000 movies made by 16,000 users. The rating files can be found in the archive provided with this assignment in rating.zip.

rating.zip contains two files: *movies.dat* that stores movie ID, movie title and its genres in each line. This file uses the format *ID::Title::Genres* e.g.:

```
0004936::The Bank (1915)::Comedy|Short
0004972::The Birth of a Nation (1915)::Drama|History|Romance|War
0005078::The Cheat (1915)::Drama
0006684::The Fireman (1916)::Short|Comedy
0006689::The Floorwalker (1916)::Short|Comedy
0007264::The Rink (1916)::Comedy|Short
```

Note that titles are identical to IMDB titles, therefore, you can check these movies by browsing through: http://us.imdb.com/M/title-exact?Movie_Title. For example, you can check the first movie in the above-

[2]The idea of this exercise was first originated on a web-log post by Edwin Chen: http://blog.echen.me/2012/02/09/movie-recommendations-and-more-via-mapreduce-and-scalding/

[3]https://github.com/sidooms/MovieTweetings

[4]http://crowdrec2013.noahlab.com.hk/papers/crowdrec2013_Dooms.pdf

mentioned list using http://us.imdb.com/M/title-exact?The Bank (1915)

The second file in the provided archive is *ratings.dat*. This file stores ratings in the format *user_id::movie_-id::rating::timestamp*. These ratings are scaled from 0 to 10 and timestamp is measured in UNIX seconds since 1/1/1970 UTC. You can find some examples of *rating.dat* below:

```
1::1074638::7::1365029107
1::1853728::8::1366576639
2::0104257::8::1364690142
2::1259521::8::1364118447
2::1991245::7::1364117717
3::1300854::7::1368156300
```

Movie ratings can be considered as a sparse matrix. In this matrix, rows correspond to movies and columns to users. The ij entry of the matrix represents the rating from the j^{th} user to the i^{th} movie:

$$\begin{bmatrix} 1 & 9 & . & 8 & . & . & 0 \\ . & . & 4 & . & . & 7 & . \\ . & . & 3 & . & 7 & . & 4 \\ 5 & . & . & 9 & . & . & . \\ . & 1 & . & 3 & . & 6 & . \\ . & . & 2 & . & . & . & . \\ . & 2 & . & . & 3 & . & . \\ 8 & . & . & . & . & . & 5 \\ . & . & 0 & 1 & . & . & . \end{bmatrix} \tag{1}$$

Let's represent the ratings for movies X and Y using vectors x and y respectively. Assume n_1 and n_2 are the number of users who rated X and Y respectively. n refers to the number of users who rated both movies. We can compute three different similarity measurements between two movies:

Cosine Similarity:

$$Cosine(X,Y) = \frac{\sum xy}{\sqrt{\sum x^2}\sqrt{\sum y^2}} \tag{2}$$

Correlation:

$$Correlation(X,Y) = \frac{n\sum xy - \sum x \sum y}{\sqrt{n\sum x^2 - (\sum x)^2}\sqrt{n\sum y^2 - (\sum y)^2}} \tag{3}$$

Jaccard Similarity:

$$Jaccard(X,Y) = \frac{n}{n_1 + n_2 - n} \tag{4}$$

Question 1 Write a java program that reads *movies.dat* and stores all movie information in an HBase table. Let's call this table, "movieInfo".

Question 2 Write a java program that reads *ratings.dat* and stores the ratings in an HBase table. Create your table such that each row refers to a rating. Propose a schema for this table. Try to the enable client-side writer buffer using the method table.setAutoFlushTo(). Compare your running time with and without the buffer. Briefly explain why a writer buffer can speed up moving the data to a database?

HBase can be used as a data source or data sink for MapReduce jobs. As a data source, a scanner reads entries from a table and converts them to key-value pairs which are fed to map tasks. As a data sink, the reducer can emit "put" objects that will be inserted to an HBase table. In the next few questions, we will see how we can develop simple MapReduce jobs that read/write from/to HBase tables. This will be necessary to process the rating data that we put in HBase.

Question 3 Locate ExtractMeanRating.java from the files provided with this assignment. This code reads ratings from a rating table generated in Question 2 and produces a mean rating for each movie. Map emits <movieID, rating> pairs and Reduce computes the average ratings per movie. Review the code and fill in the missing parts indicated as /*?*/.

Note on running MapReduce programs: HBase libraries are not typically shipped with Hadoop MapReduce framework. In this case, we need to provide the framework with all the necessary libraries. For doing so, instead of using the `hadoop jar command`, we will use the `java` command, and, we will pass the libraries using the `-cp` option as follows:

```
$ sh
$ CLASS_PATH=$(hbase classpath)
$ java -cp ${CLASS_PATH}:YourJarFile.jar org.CMPT732A3.YourClass
```

Question 4 Using ExtractMeanRating.java, create another MapReduce program that instead of producing the average, counts the number of ratings per movie. In this part, instead of writing the output of reducer phase to a file, redirect them to an HBase table. You can store the number of ratings per movie in the table created in Question 1 ("movieInfo").

Question 5 Write a MapReduce program that generates movie similarities in two MapReduce steps. These steps are explained through Algorithm 1 to Algorithm 4.

Algorithm 1 map() function in step 1

```
  Input: a row from movie rating table
  Output: $(userID, \langle movieID, r \rangle)$
```

Algorithm 2 reducer() function in step 1

```
  Input: $(userID,\ movieRating = [\langle movieID_1, r_1 \rangle, \langle movieID_2, r_2 \rangle, \ldots, \langle movieID_m, r_m \rangle])$
  for $\langle movieID_i, r_i \rangle$ in $movieRating$ do
    for $\langle movieID_j, r_j \rangle$ in $movieRating$ do
      Output: $(\langle movieID_i, movieID_j \rangle, \langle r_i, r_j \rangle)$
    end for
  end for
```

Algorithm 3 map() function in step 2

```
Input: $(\langle movieID_i, movieID_j\rangle, \langle r_i, r_j\rangle)$
Output: $(\langle movieID_i, movieID_j\rangle, \langle r_i, r_j\rangle)$
```

Algorithm 4 reducer() function in step 2

```
Input: $(\langle movieID_1, movieID_2\rangle,\ ratingPair = [\langle r_{11}, r_{12}\rangle, \langle r_{21}, r_{22}\rangle, \ldots, \langle r_{m1}, r_{m2}\rangle])$
$\Sigma(xy) \leftarrow 0$
$\Sigma(x^2) \leftarrow 0$
$\Sigma(y^2) \leftarrow 0$
for $\langle r_1, r_2\rangle$ in $ratingPair$ do
  $\Sigma(xy) \leftarrow \Sigma(xy) + r_1 * r_2$
  $\Sigma(x^2) \leftarrow \Sigma(x^2) + r_1 * r_1$
  $\Sigma(y^2) \leftarrow \Sigma(y^2) + r_2 * r_2$
end for
$cosine\_similarity \leftarrow \frac{\Sigma(xy)}{\sqrt{\Sigma(x^2)}\sqrt{\Sigma(y^2)}}$
put $\langle movieID_1, movieID_2\rangle$, $cosine\_similarity$ in a table
```

Step 1: Algorithm 1 and Algorithm 2 correspond to the first MapReduce step. In this step , use the database created in Question 2 as input to your mapper. Map emits userID and <movieID, rating> pair for each rating. In the reducer, we will have all ratings associated with a user. Given the ratings of a user, reducer will generate all movie pairs and associated ratings pairs that are made by user. Store the output of reducer in a SequenceFile.

Step 2: In the second step , use the output of the previous step as input of map. For map, use the default *identity* mapper. After the shuffle phase, reducer will have a list of rating pairs for each movie pair. Use all these rating pairs to compute the movie similarity measures: cosine, correlation and Jaccard similarity.

Implementation Details: Note that during the reduce phase of the second step , we have access to x, y, xy, and n from the list of rating pairs. However, the number of total ratings for movie1 and movie2 is not available. These values are available in the HBase table that we updated earlier in Question 4 ("movieInfo"). In the setup() method of the second reduce, we can create a table object referring to this HBase table. This will enable reducers to retrieve the rating counts directly from database for each movie while computing the similarity measures.

Note that we are using a pair of integers as key or value in the MapReduce steps. So, you should implement your own IntegerPair class that implements WritableComparable interface. Finally, our Algorithm 4 computes cosine similarity for two movies. In the same reducer, you should be able to compute other measures.

Sample Output: Let's look at some similarity values. For example 0458339 ("Captain America: The First Avenger (2011)") and 0848228 ("The Avengers (2012)") have similarity of (Correlation, Cosine, Jaccard) = (0.8939, 0.9904, 0.0950). Or 0372784 ("Batman Begins (2005)") and 1300854 ("Iron Man 3 (2013)") have similarity of (Correlation, Cosine, Jaccard) = (0.3062, 0.9900, 0.0149).

Question 6 The reduce function in the second MapReduce step is very slow. Use MapReduce Counters to collect running time information for each part of the reducer() function. Which part of your reducer is the most time consuming part? Why?

Question 7 One way to avoid performing random access to a table in the reducer is to use multiple tables as input of MapReduce. For our problem, we can use the movie rating database (created in Question 2) and movie count database (created in Question 4) as input to our map. However, we will need to associate ratings with rating counts for each movie from two databases. This can be done in another MapReduce step as shown in Algorithm 5. The output of this phase can be directly fed to the input of Algorithm 1. Briefly, explain how we can change Algorithms 1, 2, 3, and 4 in terms of input/output key/value pairs to pass the number of ratings to the final reducer that computes the similarity measures. Implement your idea and report the running time for your last MapReduce step. Is your final MapReduce faster now?

Algorithm 5 map() function in step 0. This step associates ratings to rating counts for each movie.

```
Input: a table row
if row is from rating table then
    Output: $(movieID, \langle userID, r \rangle)$
else
    Output: $(movieID, \langle 0, n \rangle)$
end if
```

Algorithm 6 reducer() function in step 0.

```
Input: $(movieID,\ movieRating = [\langle 0, count \rangle, \langle userID_1, r_1 \rangle, \ldots, \langle userID_m, r_m \rangle])$
$n \leftarrow 0$
for $\langle userID, r \rangle$ in movieRating do
    if $userID == 0$ then
        $n \leftarrow r$
    end if
end for
for $\langle userID, r \rangle$ in movieRating do
    if $userID != 0$ then
        Output: $userID, \langle movieID, r, n \rangle$
    end if
end for
```

Question 8 Let's check some movie similarities in our database. Write a java client program that accepts a movie title as input and shows the top 10 similar movies for each different measurement. What are the most similar movies to "Pocahontas (1995)", "Batman Begins (2005)", "Captain America: The First Avenger (2011)", and "Before Sunrise (1995)"? Try your favorite movies, and tell us about interesting movie recommendations.

Question 9 Check the table you created in Question 7. Some entries under Correlation column are *NaN*. Some entries for Cosine similarities are exactly one. Why do we have such entries? Are these entries robust estimates for similarity of two movies? Is there a simple way to avoid generating such entries? Implement your idea, and use your program from the last question to check the similarity measures of some movies. Which similarity measure works best in your opinion?

Question 10 Let's try our system on a larger dataset. Download the 1 million rating dataset from: `http://files.grouplens.org/datasets/movielens/ml-1m.zip`. Locate the files *movies.dat* and *ratings.dat* in the archive file and import them to new HBase tables using scripts you wrote earlier. The format of these files is identical to the format used in the MovieTweetings dataset. Unfortunately, the dataset only contains ratings for movies released before 2000. Use your scripts to find similar movies to your favorite movies from 90s. Compare the recommended movies you get from this dataset with those you would get if you were using the small dataset from the previous questions.

Question 11: Optional If you are interested in trying a larger rating dataset that contains recent movies, try the following dataset:

`https://github.com/sidooms/MovieTweetings/tree/master/latest` with about 350K

Use your scripts to find similar movies to your favorite movies. Do you get better recommendations when the dataset is bigger?

3 Operations on multiple HBase tables

In the previous section, we developed a MapReduce program that accepted two tables as input. HBase did not have this feature until very recently (see `https://issues.apache.org/jira/browse/HBASE-3996`). Having multiple tables as input of MapReduce gives us a great flexibility to leverage information from different sources. For example, joining tables is a common operation in Standard Query Languages (SQL). This operation combines rows from two or more tables based on a common field between them. In this part, we will combine multiple tables in order to extract interesting patterns.

So far, we have created two tables: a movie rating table that relates movie IDs to user IDs with their ratings, and, a movie information table that contains title, genres and number of ratings. In this part, we are going to create a third table to store information about users.

Question 1 Import user information from the 1 million rating dataset (MovieLens from Question 10) into an HBase table. What will be the schema of this table (in terms of row key, column family and qualifiers)[5]?

Question 2 Have you ever heard that boys always favor action movies over romantic movies? Have you found elderly people to be fan of documentaries? In this part, we are interested in examining the dependency between different viewer groups and their favorite movies. We can extract this type of information from MovieLens dataset as it contains information such as user gender, occupation, and age. Let's define three sets for user groups. The first set is based on gender information and contains "male" and "female" groups. The second set is based on age information and contains "18-", "18-24", "25-34", "35-44", "45-49", "50-55" and "56+". The third set is based on occupation information and contains 20 occupations (e.g. academic, artist,

[5]Check the readme file for the format used in the users.dat file in 1M MovieLens dataset

etc.)[6].

Write a MapReduce program that given a user group and a genre, counts the number of times that users from the group rated movies with the genre. Your program should combine three tables including movie information, movie rating, and user information. The final output of MapReduce will be three comma separated tables: the first table shows the dependency between genders and genres, the second table shows the dependency between age and movie genres, and the third table shows the dependency between occupations and movie genres. The dependency is represented using an estimated conditional probability of a movie genre given a viewer group. For example, this conditional probability for the action genre and the male users can be computed as follows:

$$P(action|male) = \frac{P(action, male)}{P(male)} = \frac{N(action, male)}{N(male)} \tag{5}$$

where P represents the probability and N represents the number of ratings.

For this part briefly explain what is your MapReduce steps in terms of input/output key/value pairs. Report your conditional probability lookup tables, and any interesting pattern that you observed between movie genres and viewer groups. Is it true that male viewers prefer action movies whereas female viewers like romantic movies? Which genres show a consistent trend with age groups? Is there any significant difference between programmers and writers in terms of genre preference?

Hint: You can tackle this question with three MapReduce jobs:

1. A MapReduce job that joins the movie rating with the movie information table on movie IDs and stores the output in a temporary HBase table.
2. A MapReduce job that joins the temporary table with the user table on user IDs.
3. A MapReduce job that counts pairs of group and genres.

[6] Check the readme file in the MovieLens dataset.