1. Setting up an HBase cluster in Azure

Creating a cluster

- Sign into the azure portal (portal.azure.com)
- Click "+" in the upper left corner
- Choose "Intelligence + Analytics" then "HDInsight"
- Give the cluster a unique name: e.g. "darkomahbase"
- In the "Select Cluster Type" choose "HBase" and a standard Cluster Tier (Finish with pressing "select")
- In the credentials provide:
  - cluster login username and password (used for the admin interface - not important for now)
  - ssh username and password (remember these, we will use them to connect to the cluster)
- In the Data Source provide:
  - Name for the storage account (e.g. "darkomahbasestorage")
    - If you have already created an HBase cluster previously and want to reuse your previous storage use "select existing"
  - Name for the storage container (e.g. "hbasecontainer")
    - **Important note:** If you are reusing a previously created storage make sure you use the same container name (otherwise you will not see the same data)
  - For location select "West Europe" it will be easiest and cheapest for the file transfers
  - Finish with the Data Source by pressing "Select"
- In the Pricing provide:
  - Select 2 region nodes
  - For Region node size and Head node size you can leave the default one
  - Exit the Pricing page by pressing "Select"
- Choose your resource group e.g. "Exercise05"
- Choose pin to dashboard and click Create
- Wait for 20 mins so that your cluster is ready
**Accessing your cluster**

- To access your cluster via ssh use the following command (inside a terminal):
  ```
  nbuser@nbserver:~$ ssh <ssh_user_name>@<cluster_name>-ssh.azurehdinsight.net
  ```

- If you are using Linux or MacOSX you can use your standard terminal
- If you are using Windows you can use
  - Putty SSH Client (get it at http://www.putty.org/)
  - the Notebook server terminal (Click on the Jupyter logo and the goto New -> Terminal)

**Running HBase commands**

- To run hbase commands use the hbase shell:
  ```
  sshuser@hn-darkom0:~$ hbase shell
  ```

- Try out sample commands
  - create a table named 'mytable' with two column families 'colfamily1' and 'colfamily2':
    ```
    hbase(main):001:0> create 'mytable', 'colfamily1', 'colfamily2'
    ```

- Insert rows in the table
  ```
  hbase(main):001:0> put 'mytable', 'row1', 'colfamily1:col1', 'c11val1'
  hbase(main):001:0> put 'mytable', 'row1', 'colfamily1:col2', 'c12val1'
  hbase(main):001:0> put 'mytable', 'row1', 'colfamily2:col2', 'c22val1'
  hbase(main):001:0> put 'mytable', 'row2', 'colfamily1:col1', 'c11val2'
  ```

- Scan the table
  ```
  hbase(main):001:0> scan 'mytable'
  ```

- See table schema
  ```
  hbase(main):001:0> describe 'mytable'
  ```

- Apply filters to the scan
  ```
  hbase(main):001:0> scan 'mytable', {COLUMNS => 'colfamily1:col1'}
  hbase(main):001:0> scan 'mytable', {COLUMNS => 'colfamily1:col1', ROWPREFIXFILTER => 'row'}
  hbase(main):001:0> scan 'mytable', {COLUMNS => 'colfamily1:col1', FILTER => "ValueFilter(=, 'substring:val1')"}
  ```

- Check examples online for more info on commands:
  - [https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/](https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/)

- You can exit the shell using CTRL+C
2. Setting up Wikipedia Dataset

We have prepared a subset of the English Wikipedia dataset (200,000 articles) with some metadata for you to explore in HBase. Follow the below steps to set this dataset up in your HBase instance:

- First log in to your cluster using ssh as described above.
- Then download the dataset from our storage account using wget, and extract it

```
myuser@hn0-darkom:~$ wget https://ethzbigdata2016exercise.blob.core.windows.net/exercise04/200k_wiki.tar.gz
myuser@hn0-darkom:~$ tar -xzvf 200k_wiki.tar.gz
```

You will see 4 csv files (author.csv, comment.csv, text.csv, timestamp.csv) each of which is in the format "", where 'key' is the title of the wikipedia article and 'value' is author, comment, text and timestamp respectively for each file.

```
myuser@hn0-darkom:~$ ls -lh
total 3.5G
-rw-rw-r-- 1 darkoma darkoma 885M Okt 12 18:53 200k_wiki.tar.gz
-rw-rw-r-- 1 darkoma darkoma 5.7M Okt 12 17:57 author.csv
-rw-rw-r-- 1 darkoma darkoma 16M Okt 12 18:46 comment.csv
-rw-rw-r-- 1 darkoma darkoma 2.6G Okt 12 18:35 text.csv
-rw-rw-r-- 1 darkoma darkoma 7.6M Okt 12 18:48 timestamp.csv
```

Now upload these files to the HDFS (Azure) storage:

```
myuser@hn0-darkom:~$ hadoop dfs -copyFromLocal *.csv /tmp/
```

Now create two tables (both of which will be populated with the same dataset).

- One table will contain only one column family for all columns.
- The other table will have one column family dedicated for the text (the article's content) and another column family dedicated to the rest of the columns (the author, comment and timestamp metadata)

```
myuser@hn0-darkom:~$ echo "create '200k_wiki_1cf', 'data'" | hbase shell
myuser@hn0-darkom:~$ echo "create '200k_wiki_2cf', 'content', 'metadata'" | hbase shell
```

Note that the above commands are using the bash shell to execute commands in the hbase shell

- Next populate the first table with the data uploaded to your hdfs

```
myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:text" 200k_wiki_1cf wasbs:///tmp/text.csv
myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:author" 200k_wiki_1cf wasbs:///tmp/author.csv
myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:comment" 200k_wiki_1cf wasbs:///tmp/comment.csv
myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:timestamp" 200k_wiki_1cf wasbs:///tmp/timestamp.csv
```
• Do the same for the second table

myuser@hn0-darkom:$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, content:text" 200k_wiki_2cf wasbs:///tmp/text.csv

myuser@hn0-darkom:$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:author" 200k_wiki_2cf wasbs:///tmp/author.csv

myuser@hn0-darkom:$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:comment" 200k_wiki_2cf wasbs:///tmp/comment.csv

myuser@hn0-darkom:$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:timestamp" 200k_wiki_2cf wasbs:///tmp/timestamp.csv

3. Storage Models

2.1. Write the following queries for the two Wikipedia tables we have set up above:
   • Select all article titles and author names (data:author and metadata:author) where the row name starts with ‘ETH’
   • Select all article titles and author names (where the author contains the substring ‘john’) 

2.2 Execute your queries on the two tables (more than once) and observe the query execution times

2.3 Prepare a third schema where all the metadata attributes also have their own column family. Populate this third schema with the same dataset, execute the queries and observe the execution times.

2.4. What are advantages and disadvantages of a pure row store?
2.5. What are advantages and disadvantages of a pure column store?
2.6. What are advantages and disadvantages of wide column stores?
2.7. What are advantages and disadvantages of denormalization?

Solution:

2.1.

- Selects all article titles and author names where row starts with ETH

scan ‘200k_wiki_2cf’, {COLUMNS => ’metadata:author’, ROWPREFIXFILTER => ’ETH’}
scan ‘200k_wiki_1cf’, {COLUMNS => ’data:author’, ROWPREFIXFILTER => ’ETH’}

- Selects all article titles and author names where author name contains the substring ‘john’

scan ‘200k_wiki_2cf’, {COLUMNS => ’metadata:author’, FILTER => “ValueFilter(=, ’substring:john’)
scan ‘200k_wiki_1cf’, {COLUMNS => ’data:author’, FILTER => “ValueFilter(=, ’substring:john’)”}
2.2.

- For ROWPREFIXFILTER queries on both tables should be quick because all rows are indexed by row key.
- For the ValueFilter of a separate column following times are expected:
  * 200k_wiki_2cf: First run (cold caches) : 3 seconds, subsequent runs (0.3 seconds)
  * 200k_wiki_1cf: First run (cold caches) : 145 seconds, subsequent runs (1.5 seconds)

2.3.

create '200k_wiki_4cf', 'text', 'author', 'timestamp', 'comment'

myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,
-Dimporttsv.columns="HBASE_ROW_KEY, text:text" 200k_wiki_4cf wasbs:///tmp/text.csv

myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,
-Dimporttsv.columns="HBASE_ROW_KEY, author:author" 200k_wiki_4cf wasbs:///tmp/author.csv

myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,
-Dimporttsv.columns="HBASE_ROW_KEY, comment:comment" 200k_wiki_4cf wasbs:///tmp/comment.csv

myuser@hn0-darkom:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,
-Dimporttsv.columns="HBASE_ROW_KEY, timestamp:timestamp" 200k_wiki_4cf wasbs:///tmp/timestamp.csv

- Queries should be even quicker than 200k_wiki_2cf

Extra:
- Compare to local (single-threaded) substring seach:

(HBase – warm caches – 2 nodes – 16 threads)
scan '200k_wiki_1cf', {COLUMNS => 'data:text', FILTER => "ValueFilter(=, 'substring:Uetliberg')")

(Local – warm caches – 1 thread)
time grep -i 'Uetliberg' text.csv

2.4.

Pure row store:
- Advantages:
  ◦ Good for workloads with point lookups and updates. Retrieving (updating) a single row is efficient as the row is collocated
- Disadvantages:
  ◦ Inflexible schema (when attribute names are not stored for each row)
  ◦ Scans are more expensive (whole row is always retrieved)
2.5. Pure column store:
  • Advantages:
    ◦ Scans are very efficient (only specific columns can be retrieved)
  • Disadvantages:
    ◦ Inflexible schema
    ◦ To retrieve (or update) a whole row many random accesses need to be performed

2.6. Wide column store:
  • Advantages:
    ◦ Column families offer a 'middle ground' between pure row and column. Columns frequently accessed together can be colocated, very wide columns (affecting scan speed) can be separated
    ◦ Flexible schema (column names stored for each row) offer flexibility for cases where schema is not known upfront (or in cases of sparse columns)
  • Disadvantages
    ◦ Performance penalties, point lookups not as fast as pure row store, scans not as fast as pure column store
    ◦ Storage overhead

2.7. Denormalization:
  • Advantages:
    ◦ Everything is a scan or a table lookup. No need for expensive joining of multiple relations (all data is colocated or easily mapped)
  • Disadvantages:
    ◦ It's difficult to enforce (maintain) consistency in cases of updates
    ◦ Storage (memory) overhead
    ◦ Scan processing can be more expensive

4. HBase Architecture

4.1. Assuming an uncompressed format of the HFile as discussed in the lecture and the following sizes of fields: (32bit key length and value length, 16bit row length, 8bit column-family length and type and 64bit timestamp)
  • What is total size of a single row in an HFile if row keys, column names and values are all 4 bytes each.
  
  **Solution:**
  
  See the documentation (values with * determined by question):
  
  • KeyLength: 4B
  • ValueLength: 4B
  • Key:
• RowLength: 2B
• Row: 4B*
• ColumnFamilyLength: 1B
• ColumnFamily: 4B*
• ColumnQualifier: 4B*
• Timestamp: 8B
• Type: 1B
• Value: 4B*
Total: 36B
• What would be the equivalent size in a pure column oriented store (with a fixed schema and fixed value sizes)?
  • **Solution:**
    • 4 byte row key and 4 byte value = 8 bytes (column name is not stored, no need for any offsets or sizes)

4.2. Which of the following statements is correct?
• A store contains values of subset of rows for a single column family  
  • **Correct**
• A row has at least one value for each column family  
  • **False**
• Cells in each store (HFile) are sorted first by row name and then by column name.  
  • **Correct**
• An HFile contains at most one key-value for each (row key, column name) pair  
  • **False** (There may be multiple versions)
• HMaster stores and serves the region locations to clients  
  • **False**
• A HFile contains cells from different column families  
  • **False**

4.3. Take a look at the storage container of your HBase instance. Locate the Store Files and observe the naming scheme.

- Data is stored in
  - `<storage_countainer> / hbase / data / default / <table_name> / <node/part ID> / <column_family> /`

**Important:** delete your HDInsight cluster as soon as you are done, as it can quickly eat up your credit. You can recreate the cluster later using the same storage account (your tables will still be there) using the procedure described above.
5. Bloom Filters

**Lookup** the properties of bloom filters and answer the following questions, assuming ideal hash functions.

5.1 What is the probability of false positives as a function of the number of elements, number of hash functions and bloom filter size (number of bits)?

**Solution:**
For 'n' number of elements, 'k' number of hash functions and 'm' number of bits, the probability is:
\[
\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.
\]

5.2 The optimal number of hash functions, that minimizes false positives for a given number of elements and size of a bloom filter, is linearly proportional to?

- a) The number of elements in the bloom filter  
  **False**
- b) The ratio of the bloom filter size (number of bits) to the number of elements  
  **True**
- c) The size of the bloom filter (for fixed number of elements)  
  **True**

5.3 Now compare bloom filters to just checking existence using a hash table. Assume an ideal hash table size, which is the number of elements multiplied by the size of each element.

- a) Calculate maximum space savings (i.e. ratio of hash table size to bloom filter size) for a desired false positive probability of 1% with respect to different element sizes (2B, 64B, 1KB)

  **Solution:**

<table>
<thead>
<tr>
<th>Element Size</th>
<th>Space Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B</td>
<td>1.67</td>
</tr>
<tr>
<td>64</td>
<td>53.4</td>
</tr>
<tr>
<td>1KB</td>
<td>854</td>
</tr>
</tbody>
</table>

- b) What is the optimal number of hash functions in this case?

  **Solution:**
  6.64 Hash functions
c) What is the false positive probability if you use a non-optimal number of hash functions (e.g. 1, 2, 4) for the same ratio?

**Solution:**

<table>
<thead>
<tr>
<th>k</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.099</td>
</tr>
<tr>
<td>2</td>
<td>0.035</td>
</tr>
<tr>
<td>3</td>
<td>0.0194</td>
</tr>
<tr>
<td>4</td>
<td>0.013</td>
</tr>
<tr>
<td>6</td>
<td>0.0101432</td>
</tr>
<tr>
<td>6.64</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>0.0100392</td>
</tr>
</tbody>
</table>