Milestone 2 – Big Data

The solutions should be zipped and submitted via Canvas, before 3rd April 2022, 23:59:00.

This milestone includes four questions:

1. Load and prepare data in Spark (**20pt**)
2. Query the data using SparkSQL (**30pt**)
3. Identify foreign keys using Spark RDDs (**30pt**)
4. Identify heavy-hitters in IP-logs using Spark Streaming (**20pt**)

Input data

You will use the following data file as input:

Database.csv
RelationName, AttributeNames, AttributeValues

- RelationName is a single character that represents the relation a record belongs to.
- AttributeNames is a string of max. 20 characters which contains the names of attributes that make up a record in the database. Attribute names are separated by ";".
- AttributeValues is a string of max. 200 characters which contains the values of attributes that make up a record in the database. Attribute values are separated by ";".

A line $R, a; b; c, 1; 2; 3$ in the input file represents a record $(1,2,3)$ in a relation named R with attributes (a, b, c). You can assume that the attribute names and values of each relation always appear in the same order – at all lines of the relation. Furthermore, the order of the values in each line of the input file corresponds to the order of attributes in the same line (i.e., for the record given above, $a = 1,\ b = 2$ and $c = 3$).

There may be relations with common attribute names. For example, relations $R(a, b, c)$ and $S(b, c, d)$ have b and c as common attribute names. To unambiguously identify attributes, the 'dot notation' is used in combination with the relation name, i.e., $R.a,\ R.b$ and $R.c$ for the attributes in R, and $S.b, S.c$ and $S.d$ for the attributes in S. An attribute that is unambiguously identified using dot notation is called a *fully qualified attribute*.

The Database.csv file contains a header on the first row.

The data can be downloaded from Canvas.

Setup

The milestones in this course are set up in a way that allows all groups to test their work for correctness and performance under the same conditions. There is a server available to which each group can submit its work. This server will run your code and provide feedback about performance metrics as well as the output of your code. A document with more detailed information on how to connect to- and interact with this server in general is available in Canvas.

For this milestone, you will need to upload your application to your group's home directory on the server. As described in the general guide, the application can be in the form of a JAR file or ZIP file and must be

named `app.jar` or `app.zip`, respectively. You do not need to upload any data for this milestone. The data will be made available on HDFS and you can refer to it in your source code as `'/Database.csv'` (note the forward slash, which you may not use locally).

The stream will be available on the server at hostname "stream-host" and port 9000. For local development, a JAR file called "2ID70-2022-MS2-Stream.jar" is available on Canvas. You can run this file using "`java -jar 2ID70-2022-MS2-Stream.jar t r`" from the directory in which the JAR is located. Doing so will make a stream available at hostname "localhost" and port 9000. The parameters `t` and `r` are numbers defining the time in seconds after which the stream will time-out and the rate at which the stream provides output in number of rows per second, respectively. On the server, the values 1200 and 2000 and your group number are used automatically for `t` and `r`.

Submission in canvas

Submitting your code to the server for execution is not enough to receive a grade. You also need to submit your solution in Canvas before the deadline. Each group should submit their work in Canvas only once. You should submit a single ZIP archive. Assume that n is your group number. The ZIP archive should contain only a directory called **`group-n`** (`e.g., if you are group number 3, the directory should be called group-3`). The directory's contents depend on the programming language you chose to work with.

Java

If you built your solution in Java, the directory `group-n` should have to following contents:

- app.jar (this should be the same JAR you uploaded to the server and thus be executable there)
- src (a directory containing any .java source files you used to build app.jar)

Python

If you built your solution in Python, the directory `group-n` should have to following contents:

- main.py (this should be the same script you uploaded to the server and thus be executable there)
- src (a directory containing any other .py source files. If none exist, this directory can be omitted)

1. Load and prepare data in Spark (20pt)

Notice that each line contains a list of attribute names and values. These are separated by delimiter ";". It is more convenient for this milestone to have one record per attribute. You are therefore asked to create a Spark **RDD** based on the database – as read by Database.csv – which has the following format: $RelationName, AttributeName, AttributeValue$.

For example, the following record:
$R, a; b; c, 1; 2; 3$

should be converted to three records as follows:
$R, a, 1$
$R, b, 2$
$R, c, 3$

The creation of the RDD should occur in a file q1.java or q1.py. The resulting RDD should be saved in an RDD named $q1RDD$ and made available to the code which implements the remaining questions in this Milestone.

To enable automated testing, your code should count the number of lines in the RDD for relations R, S, and T, after the described transformation, and print them out as follows:
>> [q1: R: result1]
>> [q1: S: result2]
>> [q1: T: result3]

where result1, result2 and result3 are integers and correspond to the number of records containing these relation names. All results should be printed at stdout (from the driver code, use System.out.println in Java or print in Python). Notice that each answer is prefixed with ">>", enclosed in square brackets and ends with a new line (character "\n").

For example, this could be achieved in Java by the following lines:

```
JavaRDD q1RDD = … (call your method to load and prepare the data)
System.out.println(“>> [q1: R: “ + q1RDD.filter(row ->
row.split(“,”)[0].equals(“R”)).count());
```
(and similar lines for the other two relations).

Deliverable: Spark code (Java or Python) in a method/function named `q1()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

2. Query the data using SparkSQL (30pt)

Using $q1RDD$ of the previous question as an input, execute the following queries using SparkSQL:

1. Find the number of rows for the relation named R . This should be identical to your first answer at question 1, but now with SparkSQL. (**10pt**)
2. Find the number of fully qualified attributes for which there are at least 1000 distinct values (**10pt**)
3. Find the fully qualified attribute with the smallest number of distinct values (**10pt**)

You can assume that there exists exactly one fully qualified attribute for which the number of distinct values is strictly smaller than the number of distinct values for any other fully qualified attribute.

You should use data frames and data sets, but the computation needs to be done in Spark rather than in Java of Python. The results for each query should be stored in variables named q21, q22 and q23 respectively. The code should print these results as follows:

>> [q21: result1]
>> [q22: result2]
>> [q23: result3]

For example, a possible answer (for different input data) could look like this:

>> [q21: 4230105]
>> [q22: 21]
>> [q23: R.b]

Deliverable: Spark code (Java or Python) in a method/function named `q2()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

3. Identify inclusion relationships using Spark RDDs (30pt)

An attribute $R.a$ is called an *inclusion attribute* to an attribute $S.b$ if and only if for any value of $R.a$, there exists a record **in a different relation** S where $S.b$ takes that same value. In this example, R is called the *referencing* relation,S is called the *referenced* relation, $R.a$ is called the referencing attribute and $S.b$ is called the referenced attribute. The pair $(R.a, S.b)$ is called an inclusion relationship.

You can think of an inclusion attribute as a weakened foreign key, where it is *not* necessary that the referenced attribute is a candidate key for the referenced relation.

For this question, you are asked to find all inclusion relationships that hold on the given data set.
You only need to consider inclusion attributes with a single referencing- and single referenced attribute, i.e., you do not need to consider composite referencing/referenced attributes.

The input data should be the RDD $q1RDD$ produced in question 1. Your solution should print the result as pairs (x, y) where x and y are fully qualified attributes and x is an inclusion attribute to y. For example, if for every value of $R.a$ there exists a record in S such that $S.b$ is equal to $R.a$, then one row in the results would be:

>> [q3: R.a,S.b]

Notice that the answer is prefixed with ">>" and the character "," is used as a delimiter.

You **are not allowed to use SparkSQL** for this question.

Deliverable: Spark code (Java or Python) in a method/function named `q3()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

4. Identify heavy-hitters in IP-address log files, using Spark Streaming (20pt)

You are asked to identify and continuously print 'heavy-hitters' for streaming data consisting of IP addresses over a sliding window, using Spark Streaming. Heavy-hitters are defined as those IP addresses that have a relative frequency of at least 3% in the sliding window (relative to the number of records in the window, i.e., $rf(x) = \frac{\#occurrences\ of\ x\ in\ SW}{total\ arrivals\ in\ SW}$.

More precisely, you should implement a Spark Streaming program that does the following:

- Continuously read the data from the stream. The streaming data will be a list of IP addresses, separated by a line break. An IP address is a string of the form $w.x.y.z$ with w, x, y and z being numbers between 1 and 255 (both inclusive).
- Keep the relative frequencies over a sliding window of 20 seconds. Use a sliding interval of 4 seconds and a batch interval of 2 seconds.
- Print all those IP addresses and their corresponding relative frequencies provided that this relative frequency is at least 3%.
- You should use an inverse function for better performance.
- You should **not** use SparkSQL for this question.

If you need temporary space in the server for checkpointing, you can use the directory "/checkpoint" under HDFS.

Deliverable: Spark code (Java or Python) in a method/function named `q4()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

Frequently asked questions

Plagiarism: Plagiarism will not be tolerated. All parts participating in plagiarism will be reported and punished according to the university rules.

How will my submission be graded? The final submissions will be downloaded from Canvas. They will be executed and their results will be compared to the expected results. The tests will be automatic. This is why it is very important that you strictly follow the provided instructions.

Do I need to adhere strictly to the provided instructions (filenames, structures, etc)? Yes, for the testing suite to evaluate your solutions correctly.

Can I verify that my answer is correct, before submission? You can verify your submission on the small data set for which you will receive the expected output. Notice however that obtaining the same output on the small data set does not completely guarantee the correctness of a submission in general.

Further questions: For clarifications and questions about the milestone (or useful pointers), please use Canvas discussions so that your fellow students can also see the answers. For questions that cannot be asked publicly (e.g., the question reveals part of the answer), you can ask us privately during the instruction hours.