COSC–254 Data Mining
Project 01 – Mining Itemsets in MapReduce
Due: Wednesday, March 20, 2019, 1.59pm

In this project you will implement and evaluate a MapReduce algorithm for mining (approximately) the frequent itemsets from a dataset.

Collaboration reminder: Please do not share your code with anyone and do not ask to see anyone's code. Please use the forum for *all* discussions, so any doubt can be clarified for everybody, and so everybody is on the same page.

Implementation

The algorithm is described in Section 6.4.4 of the MMDS book.

User interface Please call the "main" class of your implementation `MRMiner`, and call your JAR file `mr.jar`. It must be possible to run your work as

```
$ hadoop jar mr.jar MRMiner MINSUPP CORR TRANS_PER_BLOCK PATH_TO_INPUT
PATH_TO_FIRST_OUT PATH_TO_FINAL_OUT
```

where:

- `MINSUPP`, an integer, is the minimum support threshold;
- `CORR`, an integer, is the "correction" to the minimum support threshold: the first Map function will mine the set of transactions it receives with a minimum support threshold of `MINSUPP-CORR`.
- `TRANS_PER_BLOCK`, an integer, is the number of transactions per "block" of the dataset, which are simultaneously given in input to the first Map function (see below, where implementation details are discussed).
- `PATH_TO_INPUT` is the path to the HDFS input directory containing the input dataset (format described below);
- `PATH_TO_FIRST_OUT` is the path to the HDFS output directory for the first round;
- `PATH_TO_FINAL_OUT` is the path to the HDFS final output directory (format described below).

Implementation details Implement Apriori as the algorithm to mine the blocks of transactions in the first Map. The size $|\mathcal{I}|$ of the set $\mathcal{I}$ of items will be small enough that you won't have to worry about memory problems (such as those described in Section 6.3 of MMDS).

To obtain the split of the dataset into blocks of consecutive lines required in the first Map function you should use, as the InputFormatClass of your Job object, the class MultiLineInputFormat, whose implementation you can find in the support files that can be obtained from `http://bit.ly/254s19P01F` (put the source file in your source code directory). You can set the InputFormatClass with

```
job.setInputFormatClass(MultiLineInputFormat.class);
```

where `job` is your Job object.

When compiling, you have to specify `MultiLineInputFormat.java` on your command line together with your other files.

To set the number of transactions that are given in input to each invocation of the first Map function (this quantity is specified as `LINES_PER_BLOCK` on the command line), use the static method

```
org.apache.hadoop.mapreduce.lib.input.NLineInputFormat.setNumLinesPerSplit(Job
 job, int numLines)
```

while setting up your Job object. Due to Hadoop's limitations, it is not always the case that each first Map invocation receives exactly `LINES_PER_BLOCK` lines in input: from time to time, a Mapper may receive slightly fewer lines, but we do not expect these fluctuations to have any major impact.

To have the output of the first round (which is written to the HDFS path `PATH_TO_FIRST_OUT` given on the command line) available in the Mappers of the second round, add it to the distributed cache.

Input and Output Formats The input dataset is a plaintext file containing one transaction per line. Each transaction is a sequence of non-negative integers separated by a white space, such as

$$11\ 8\ 9\ 20$$

The output should contain the frequent itemsets in the input dataset with support at least `MINSUPP`, one frequent itemset per line with items separated by a space, and followed (on the same line) by the support of the itemset. For example

$$20\ 8\ 204532$$

would denote that the itemset $\{20, 8\}$ has support 204532.

You can find an example input file `in.txt`, and expected output `output.txt` for `MINSUPP` equal to 3 in the `examples` directory of the support files that can be obtained from `http://bit.ly/254s19P01F`.

Evaluation

Please do not underestimate how long this part would take.

Using `/user/mriondato/apriori` as `PATH_TO_INPUT` and `1600000` (1.6×10^6) as `MINSUPP`, evaluate:

- the running time; and
- the output size (number of frequent itemsets found)

of your implementation for different values of `CORR` and `LINES_PER_BLOCK` (try at least 4 reasonably different values for each parameter (e.g., 1×10^5, 5×10^5, 1×10^6, 2×10^6 for `LINES_PER_BLOCK` (these may not be optimal values, try others!)) and all their combinations).

Write a report on your findings, with plots showing the behavior of the running time and the output size, and discuss this behavior.

When preparing your submission, place your report in the same directory of your source code.

How to submit

Submit your work at `https://www.cs.amherst.edu/submit` or via `cssubmit` from romulus/remus, as a *single* archive file with name `username.ext` where `username` is your user name and `ext` is one of `.zip`, `.tar.bz2`, or `.tar.gz` (no `.rar`, please).

The archive must contain a *single* directory with name `username`. This directory must contain a subdirectory with name `X` for each Exercise `X`, or a single subdirectory with name `1` in case of projects. All files (source code or otherwise) for each exercise (or project) must be in the directory for that exercise. Directories containing source code should contain a `README.txt` file explaining how to run the code in that directory. For non-code answers, please submit a `.pdf` or a `.txt` (no `.doc(x)`, please). You can find an example archive at `http://bit.ly/DM19sub`.

Please post to the Moodle forum if you have problems with the submission.