Big Data Analytics
Stony Brook University
CSE545 - Spring 2020

Assignment 1
Assigned: 2/7/2020; Due: 2/21/2020 11:59pm

Overview

Goal: Gain experience working with streaming algorithms as well as MapReduce workflow systems. Gain experience simulating working with a distributed file system.

Requirements. You must use Python version 3.5 or later. You do not need a distribution of MapReduce for this assignment as we will use our own basic simulator (see task II.A.) -- future assignments will use MapReduce on a cluster.

Templates and Python Libraries. Template code is provided for each part of the assignment. Each template includes all of the data science, machine learning, or statistics libraries that you may import. Other data science, machine learning, or statistics related libraries are prohibited unless listed below -- **ask if unsure.** The intention is for you to implement the algorithms we have gone over and problem solve in order to best understand the concepts of this course and their practical application.

Within the templates, all provided method names and classes must be used as provided with the same parameters. However, you may also use additional methods to keep your code clean.

Additional approved libraries that are not in the template will be listed here (if any):

```
datetime
random.shuffle
```

Copying code from other students, online or other resources is prohibited and will result in at least a zero on the assignment and report to graduate program director with possibility for more consequences. Please see syllabus for additional policies.
A word to the wise: As is tradition in CSE545 at SBU, parts of this assignment are completely novel, never given before.

Part I. Streaming (35 points)

Here, you will implement and evaluate the runtime of the typical multi-level sampler as well as the "streaming" sampler that we went over in class. We call it a "multi-level sampler" because the level of analysis we wish to sample (e.g. user level) is not the same level with which the data arrives (e.g. transaction level).

Data: to complete these tasks, you be provided transaction data of the following form:

```
record_id, date, user_id, amount
```

record_id -- unique id for the transaction
date -- date of the transaction
user_id -- id for the user making the transaction
amount -- the amount of the transaction
The data is provided in three files

1) `transactions_small.csv`
2) `transactions_medium.csv` [zip]
3) `transactions_large.csv` [zip]

Task I.A)**Typical non-streaming multi-level sampler (15 points)**
Implement the standard non-streaming sampling method
Step 1: read file to pull out unique user_ids from file
Step 2: subset to random pct% of user_ids (shuffle then take top pct%)
Step 3: read file again to pull out records from the pct% user_id
and compute mean and standard deviation within
Note: We're assuming the file is too big to simply load the whole thing (all columns) into memory and simply shuffle it

Task I.B)**Streaming multi-level sampler (15 points)**
Implement the streaming multi-level sampling code we went over in class which uses hash functions on the user_id to read through the file once. Technically, we are simulating a web stream of data by instead taking a single pass at a file but you should see the advantage of this algorithm even for sampling from files. Record the information that is needed in order to compute the mean and standard deviation. Your sample should correspond to 2% and .5% of the user_ids in each file (approximate, especially in the case of the small file). Make sure to use a streaming mean and std-dev (see rules in method description).

Task I.C)**Timing (5 points)**
Time wall-clock processing time, in milliseconds, over different sizes of data: small(10,000) medium(1,000,000) and large (100,000,000)
Report runtimes and results for both implementations above, using percentages of .02 and .005 for each of the three files (small may not have an adequate sample at .005: that's ok).

Template Code for Part I. A template to be filled in with your code is provided here:
`sampler_lastname_id_v02.py`

Part II. MapReduce (65 points)

Here, you will complete a back-end for a MapReduce system and test it on a couple MapReduce jobs: word count (provided), and meanCharsMR (you must implement). Template code is provided. Specifically, you must complete:

Task II.A)**PartitionFunction (10 points)**
Complete the partition function, making sure to use a hash that can handle: integers and strings.

Task II.B)**RunSystem (20 points)**
Complete the "runSystem(self)" method which divides the data into chunks and schedules the running of mapTasks and reduceTasks. The are two places to complete:
(1) Divide up the data into chunks according to num_map_tasks, and launch a map task per chunk.
(2) Send each key-value pair to its assigned reducer.

Task II.C)**Combiner (15 points)**
Edit the "MapTask" method to add support for running a Combiner. Look for "# <<COMPLETE>>" within the method. Remember, a combiner runs the reduce task at the end of the map task in order to save communication cost of sending to multiple reducers. Note: main will run the WordCountBasicMR to test with and without the combiner. It is recommended that you look over the WordCountBasicMR to understand what it is doing. You can assume your combiner code will only run on reducers that are both commutative and associative (see hint at bottom).

Task II.D)**Mean CharsMR (20 points)**
Edit the "map" and "reduce" methods of "MeanCharsMR" to implement a map-reduce computation of the mean and standard deviation of the number of each character (a-z, case insensitive) per document (i.e. the mean is across all documents; the count of each character is per document). Consider each record (i.e. single key value pairs arriving at mapper) to be a single document. Reduce can return more than the mean, and standard deviation (hint: including other items will be helpful for the combiner to run).
Example: if one had three documents: ['a bacd a', 'cda', 'bcd'], then the mean of each char would be:

```
('a': {'mean': 1.333 = (3+1+0) / 3, 'std-dev': 1.52 = sqrt(((3-
1.333)^2 + (1-1.333)^2 + (0 - 1.333)^2)/2)}),
('b': {'mean': 0.666 = (1+0+1)/3; … }), …
```

```
**Do not use self.data from the mappers or reducers: they need to work with
the key values that they are provided.**
```

Template Code for Part II. A template to be filled in with your code is provided here:
`MRSystemSimulator2020_lastname_id.py`

Submission

Please use blackboard to submit two files each with your lastname and student id:

1. `sampler_<lastname>_<id>.py`
2. `MRSystemSimulator2020_<lastname>_<id>.py`

Do not upload a zip file. Double-check that your files are there and correct after uploading and make sure to submit. Uploading files that are zips or any other type than python code files will result in the submission being considered invalid. Partially uploaded files or non-submitted files will count as unsubmitted.

Questions: Please post questions to the course piazza page.

Hints
As questions come in this location will be used to track suggestions.

- Combiners require a reduce function that is both commutative and associative:
  f(v1, v2) = f(v2, v1)   and   f(f(v1, v2), v3) = f(v1, f(v2, v3))