COL733: Fundamentals of Cloud Computing
Semester I, 2023-2024

Lab-3: Stream processing
Deadline: 20 September 2023, 11:59PM

Submission Instructions

1. You can **only** use Python, Pandas, and Redis for this Lab. **Use of any other libraries** will lead to zero marks in the Lab.
2. You will submit the source code in **zip** format to Moodle (Lab 3). The naming convention of the zip file should be <Entry_Number>_<First_Name>.zip. Additionally, you need to submit a **pdf** for analysis questions on Gradescope.
3. The Lab would be **auto-graded**. Therefore, **follow** the same naming conventions described in the Deliverables section. Failing to adhere to these conventions will lead to zero marks in the Lab.
4. You should write the code **without** taking help from your peers or referring to online resources except for documentation. The results reported in the report should be **generated from Baadal-VM**. Not doing any of these will be considered a breach of the honor code, and the consequences would range from zero marks in the Lab to a disciplinary committee action.
5. You can use **Piazza** for any queries related to the Lab.
6. Please use the **same VM** you have created earlier for Lab-1. We suggest you **start early** to avoid the last-minute rush.

Dataset Description

The dataset is available at [1]. Each CSV file contains 7 attributes, following is a brief description of each attribute:

- ***tweet_id:*** A unique, anonymized ID for the Tweet. Referenced by response_tweet_id and in_response_to_tweet_id.
- ***author_id:*** A unique, anonymized user ID. @s in the dataset have been replaced with their associated anonymized user ID.

- ***inbound:*** Whether the tweet is "inbound" to a company doing customer support on Twitter. This feature is useful when re-organizing data for training conversational models.
- ***created_at:*** Date and time when the tweet was sent.
- ***text:*** Tweet content. Sensitive information like phone numbers and email addresses are replaced with mask values like __email__.
- ***response_tweet_id:*** IDs of tweets that are responses to this tweet, comma-separated.
- ***in_response_to_tweet_id:*** ID of the tweet this tweet is in response to, if any.

Problem Statement

With constant efforts and determination, you have designed a multi-threaded and fault tolerant "word counting" application for the Hogwarts school of witchcraft and wizardry. Professor McGonagall realized that a lot of wizards are posting magical "tweets" every day. As a result, the prior approach of dumping all the data into one place and processing them is becoming quite cumbersome for the school. Moreover, she suspects that if one of the threads processes the tweets slower due to resource contention, then it may have an adverse performance impact.

She wishes to extend your word count application to process the incoming magical "tweets" on the fly. In particular, she wishes to see the top 3 words at any time of tweet content (***text*** attribute). Moreover, you need to handle load balancing between the threads. For this lab, you can find the starter code here.

Deliverables

- ***Source code***: You need to provide the source code for the word counting application implemented using the python ***multiprocessing*** library. The source code should be in a .zip format and should be uploaded to moodle. A sample source code folder structure is shown below:

```
directory:  2020CSZ2445_Abhisek
            2020CSZ2445_Abhisek/client.py
            2020CSZ2445_Abhisek/base.py
            2020CSZ2445_Abhisek/constants.py
            2020CSZ2445_Abhisek/mrds.py
            2020CSZ2445_Abhisek/worker.py
```

```
        2020CSZ2445_Abhisek/__init__.py
        2020CSZ2445_Abhisek/mylib.lua
        2020CSZ2445_Abhisek/requirements.txt
```

When we unzip the submission then we should see the above files in the aforementioned structure.

- Your word-count application should be named ***client.py*** and runnable by the following command. *Note:* All the relevant information necessary for the word count application is available in ***constants.py***

  ```
  python3 client.py
  ```

- ***We will change the client.py and constants.py file with appropriate values during evaluation. Therefore, do not change constants.py file.***
- ***The evaluation script will load the redis function. You need not write any code to load it.***

- *Analysis:* Answer the following questions on Gradescope (Lab 1-3: Analysis):
  - Describe the load balancing mechanism used by your word counting application.
  - To understand the effect of faulty or slow workers on the overall performance, you need to perform the following experiments with a suitable value of *m* and *n*. For consistency, you need to spawn 7 workers and inject *m* files (each of size ~3.5MB) every second:
    - Measure the time taken to process a file starting from the timestamp the file was inserted into the stream till the count of unique words is updated in Redis while the workers are fully functional. Then, kill *n* workers and monitor the time taken to process a file until the time taken to process a file saturates (Refer to Section 6.2 of the Spark Streaming paper).
    - Perform the above experiment, where instead of killing the workers, you need to slow down *n* workers using the cpulimit command. You can limit the CPU usage of a worker in the range of [30,50,70] (Refer to Section 6.2 of the Spark Streaming paper).
    - We expect you to draw figures like the Figure 10 in the Spark Streaming paper.

Rubrics (25 marks)

1. 2 marks: Correctness of word-count application with workers=8.
2. 8 marks: This has relative grading. The faster programs for multiple workers will receive higher marks.
    a. 4 marks: In the presence of faulty workers
    b. 4 marks: In the presence of stragglers
3. 15 marks: Justifications and analysis as requested in the deliverables.

References

[1]: https://www.kaggle.com/thoughtvector/customer-support-on-twitter