COL733: Fundamentals of Cloud Computing
Semester II, 2021-2022

Lab-2: Stream processing
Deadline: 6 Feb 2022, 11:59pm

Submission Instructions

1. You can **only** use Python for this Lab. You are restricted to using only the following components: Celery, RabbitMQ, and Redis, already installed in the virtual machine. **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 2). 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 (Lab2: Analysis).
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 ~/data directory. Each CSV file contains 7 attributes, following are 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

Garima is thankful for your contribution in designing a scalable and fault-tolerant word counting application. However, with a lot of people tweeting every day, dumping all the data into one place and processing them is becoming quite cumbersome for her.

She wishes to extend this word count application to process the incoming tweets on the fly. In particular, she wishes to see the top 10 words at any time of tweet content (***text*** attribute). You are provided with a machine that has only the following packages: Celery, RabbitMQ and Redis.

She has created the following setup for you, a template for the application is available here. She wrote scripts *init.py* and *config.py*, which creates three Redis streams[2] namely *tweet*, *w_0* and *w_1*. New incoming tweets will come to the *tweet* Redis stream. Subsequently, your application will tokenize those tweets, run *hash(word)%2* to find its appropriate stream and dump the words into *w_0* and *w_1* streams. Next, your application processes the words from the *w_0* and *w_1* streams and stores the word counts in a Redis sorted set called "*words*".

Note that you can not create new Redis streams or other Redis data structures. You must use ALL 3 Redis streams (*tweets, w_0, w_1*) and the *words* sorted set and NOT use other Redis data structures. Failing to adhere to this will break evaluation scripts and will lead to zero marks in the Lab.

Deliverables

- ***Source code***: You are provided with a starter code available here. You need to add the necessary processing logic for the word counting application. 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/tasks.py
              2020CSZ2445_Abhisek/init.py
              2020CSZ2445_Abhisek/config.py
  ```
  When we unzip the submission then we should see the above files in the aforementioned structure.

  The python file containing celery tasks should be named ***tasks.py***.
  - The streams should be initialized by the following command:
    ```
    python3 init.py
    ```
  - Your celery tasks should be runnable by the following command:
    ```
    celery -A tasks worker --loglevel=INFO --concurrency=4 -n
    task@%h
    ```
  - Your word-count application should be named ***client.py*** and runnable by the following command, where DATA_DIR (*e.g.:* ~/data/) points to the directory containing the tweets.
    ```
    python3 client.py <DATA_DIR>
    ```

- ***Analysis:*** Answer the following questions on Gradescope (Lab2: Analysis):
  - For a fixed input size, measure how the rate at which messages are processed from the '*tweet*' stream varies with an increase in worker threads allocated to the application. Explain your observations.
    Note: Use a single celery worker node and vary the concurrency option for the experiment between 4 and 8.
  - Given there are two streams namely *w_0* and *w_1* which store words, how does the designed solution handle load imbalance, such as *w_0* starts

receiving a lot of messages whereas *w_1* is not receiving any new messages?
  - Describe how your code is tolerant to celery worker failures. In other words, describe why your code is guaranteed to provide the same answer even if a worker crashes.
  - Describe how your code can be made tolerant to network partitions between a celery worker and RabbitMQ. A network partition occurs when a celery worker is unable to talk to RabbitMQ. It may be able to talk to Redis.

Rubrics (30 marks)

1. 2 marks: Correctness of word-count application with concurrency=8.
2. 3 marks: The word-count application is fault-tolerant.
3. 8 marks: This has relative grading. The faster programs on concurrency=8 will receive higher marks.
4. 7 marks: This has relative grading. The faster programs with an imbalanced input on concurrency=8 will receive higher marks.
5. 10 marks: Justifications and analysis as requested in the deliverables.

References

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

[2]: https://redis.io/topics/streams-intro