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

Lab-4: Storage Fault Tolerance
Deadline: 1 October 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 fault tolerant multi-process "word counting" application that processes magical "tweets" on the fly for the Hogwarts school of witchcraft and wizardry. Professor McGonagall acknowledged that the current design is immune to worker failures or crashes but cannot tolerate redis instance failures. *For this lab, instead of storing all the words of a file in Redis, we* ***only store the top 10 words*** *and their respective frequencies.*

To mitigate redis instance failures, she suggests two approaches:

1. Using the *BGSAVE* command in Redis to periodically create a snapshot for the redis instance. To validate the efficacy of the checkpoint-based method, you need to restart the redis instance while the jobs are running. You have to implement the `MyRedis.checkpoint` method in `mrds.py` that creates a snapshot, and `MyRedis.restart` method that restarts the redis instance. (Note: use the *systemd* utility to restart the redis service.).

2. Putting Redis into a Raft cluster. Client.py will call configure_raft.sh and provide it with (2f+1) port numbers. You need to create configure_raft.sh that starts (2f+1) redis instances and form a raft cluster with those instances. Later, client.py will call `MyRedis.restart` on a port. You need to kill the redis instance running on that port. redisraft.so is in `/home/baadalvm/redisraft/redisraft.so.`

Unfortunately, RedisRaft does not yet support FCALL and all the stream commands like XGROUP, XADD, etc. So, we partition the dataset into smaller parts, where the XXth partition is saved as "part_XX.csv." Subsequently, each worker is assigned an id between [0, N-1], where N is the total number of workers. A worker with the id *i* processes part_XX.csv if and only if *XX%N* == *i*. You can maintain the flag variable that indicates the number of workers completed. If the flag variable == N, we can safely assume that all the files are processed, exiting the client.py.

We have released `client.py` **and** `constants.py`.

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
            2020CSZ2445_Abhisek/configure_raft.sh
```

  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 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 4: Analysis):

  [1 mark] Checkpointing: Justify why your submission is guaranteed to always provide correct answers under arbitrary timing of checkpoint creation.

  Raft:
  [1 mark] Justify why your submission is guaranteed to always provide correct answers under arbitrary Redis instance crashes.
  [1 mark] Will your submission handle worker crashes? Why or why not?

  Contrast end-to-end runtime. Use three different checkpointing frequencies in all your experiments: every 1 second, and every 2 seconds.
  - [3 marks] Under no redis failures for no Redis FT, Redis FT with checkpointing, and Redis FT with Raft.
  - [8 marks] When a Redis instance is restarted every N seconds for FT with checkpointing and FT with Raft. During each restart, the instance restarts after a certain down_time seconds. Justify your findings.
    - You can vary down_time as 10ms, 1 second.
    - You can vary N as 3, 5 seconds.

Rubrics (20 marks)

1. 3 marks: Correctness of word-count application with redis instance failures under checkpointing approach.
2. 3 marks: Correctness of word-count application with redis instance failures under Raft approach.
3. 14 marks: Justifications and analysis as requested in the deliverables.

References

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