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

Lab-5: Network partition
Deadline: 30 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.

System Setup

1. Run “ip r” to find the IP of each VM.
2. Update the Redis bind address to the IP of the VM for each Redis instance. For example, open /etc/redis/redis.conf and add “bind 10.17.xx.yy -::1”. Replace IP with your VM’s IP. This has to be done for each VM.
3. On each VM, add “10.17.xx.yy dockervm” to /etc/hosts file.
4. In constants.py, set
   IPS = ["10.17.xx.yy", "10.17.xx.zz", "10.17.xx.aa"]
   MASTER_NODE_IP = "10.17.xx.yy"

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 acknowledges that the current design is immune to worker failures or crashes and redis instance failures. With an increase in the number of wizards, she is thinking of deploying the application on a cluster of 3 nodes, thus achieving higher throughput. However, she is worried about network partition between the replicated storage systems on the 3 nodes.

Following the CAP theorem, a partition tolerant distributed storage system can either be consistent or available. Since McGonagall values throughput, she wishes to build an

eventually consistent system, i.e, the system is always available but may be temporarily inconsistent. Redis should be available: as long as a write to even Redis instances succeed, the write should be considered successful. After the network partition heals, we would want to achieve strong eventual consistency across Redis instances.

She has created the following setup for you: 24 workers (8 workers on each VM) and 3 Redis instances (1 instance on each VM). Similar to the previous lab, instead of storing all the words of a file in Redis, we **only store the top 10 words** and their respective frequencies. You can assume that each VM will have a different set of files, and file names are unique across VMs.

We will run run_experiment.py that will copy your code on all 3 VMs, start client.py, and randomly enable/disable the communication between two nodes on port 6379 to simulate the network partitioning scenario. Each VM maintains a local Redis stream to distribute tasks among the local workers.

Finally, we will heal the network partition and repair ***the local redis instance*** to achieve strong eventual consistency across Redis instances. The starter code of the lab can be found *here*.

Deliverables

- ***Source code***: ***Only one member of the group*** needs to provide the source code for the word counting application implemented using the python ***multiprocessing*** library on Moodle. 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/run_experiment.py
```

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 run_experiment.py
```

- ***We will hold a demo for evaluating the lab submission.***
- ***The evaluation script will load the redis function. You need not write any code to load it.***

- ***Analysis:*** Answer the following questions on Gradescope (Lab5: Analysis) (Note that ***only one group member*** needs to be upload the analysis on Gradescope):
  - [5 marks] Give reasons for the correctness of your implementation. Why is it correct if we partition and heal the network many times, and do many read repairs?
  - [5 marks] Plot the time taken to achieve strong eventual consistency vs the total number of input files, after the network partition is healed.

Rubrics (20 marks)

1. 5 marks: Correctness of word-count application with many network partitions and read repairs.
2. 5 marks: This has relative grading. In this experiment, VMs stay partitioned during the entire execution. Finally, the partitions are healed and one read repair is done. The programs that can achieve eventual consistency quicker will receive higher marks.
3. 10 marks: Justifications and analysis as requested in the deliverables.

References

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