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

Lab-3: Network partition
Deadline: 06 March 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 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 (Lab3: Analysis).
3. You are allowed to form a group of not more than 3 members.
4. 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.
5. You should write the code **without** taking help from your peers except for group members 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.
6. You can use **Piazza** for any queries related to the Lab.
7. The group will use the **same VM's** created earlier for Lab-1. We suggest you **start early** to avoid the last-minute rush. In case the size of the team is less than 3, you can request additional VM's to make sure you have 3 VM's for this lab.
8. You may need **sudo** access for some parts of this assignment. We have added the *student* user to the sudoer list. Please verify the same. We also suggest you to use the sudo privileges cautiously. Note that, if we are not able to run your code due to some unintended usage of the sudo privilege (such as revoking the access of the *baadalvm* account) you'll be awarded **zero** in this assignment.

RabbitMQ and Redis configuration:

To access the rabbitmq throughout the cluster of 3 nodes, the following operations need to be performed as a rabbitmq user:

1. Update the NODE_IP_ADDRESS config parameter in the /etc/rabbitmq/rabbitmq-env.conf file with the Node Public IP (10.x.x.x).
2. Restart the rabbitmq server service with the following command:
```
service rabbitmq-server restart
```
3. Create another user in rabbitmq as user 'guest' can only connect via localhost rabbitmqctl add_user <username> <password>
```
rabbitmqctl  add_user  <username> <password>
```
4. Set the permission for the user to access the rabbitmq vhost '/':
```
rabbitmqctl  set_permissions -p / <username> '.*' '.*' '.*'
```

To set up the Redis use the redis.conf file attached here. You also need to do the following steps:

1. In redis.conf there would be a line “bind 10.17.x.x -::1”, update the IP with your machine’s IP.
2. Start the redis server with the following command
```
redis-server redis.conf
```

Dataset Description

The dataset is available at ~/data directory. Each CSV file contains 7 attributes; the 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 contributions in designing a scalable and fault-tolerant application to find the top 10 frequent words in tweets. To handle more requests, Garima is thinking of deploying her application on a cluster of 3 nodes, thus achieving a 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 Garima values throughput, she wishes to build an eventually consistent system, i.e, the system is always available but may be temporarily inconsistent.

You are provided with 3 machines with only the following packages: Celery, RabbitMQ and Redis. The rabbitmq is running on a single node, and the Redis server is running on each node. It is the Redis servers that may get partitioned from each other[1].

She has created the following setup for you: a template for the application is available here. You need to spawn 24 celery workers (8 workers on each VM) and *process 1 file* in each celery task. Each celery worker processes the provided input file set and stores the word count results in Redis.

The *config.py* script automates the creation, cleanup and restart of appropriate resources and services, you need to just run *client.py* to execute the application with a network partition. We may further evaluate the worker failures by triggering the cold shutdown of a worker and kill Redis server. The *trend.py* script can be used for validating the output.

---

[1] Note that in this lab, we will not partition workers from RabbitMQ or kill the RabbitMQ instance. We could make our system fault tolerant to RabbitMQ using quorum queues.

Redis should be available: as long as writes to two Redis instances succeed, the write should be considered successful. After network partition heals, we would want to achieve strong eventual consistency across Redis instances.

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/myrds.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 cluster can be initialized by the following command:

```
python3 config.py
```

  - 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 (Lab3: Analysis):
- AvailableRedis:
  - Give reasons for the correctness of your implementation. Cover all failure scenarios in your analysis: celery worker failure, Redis instance failure, and Redis network partitions.

    - Plot the time taken to achieve strong eventual consistency vs the input size (N) after the network partition is healed.
    - What happens if there is a full partition, i.e, all 3 VMs can't talk to Redis instance on each other?
- ConsistentRedis:
    - If you were instead using a consistent Redis, which implementation performs better in the absence of network partitions?
    - Which implementation is expected to perform better when there is a majority-minority partition, i.e, only 2 Redis instances can talk to each other but 1 can't talk to any of the other 2?

Rubrics (35 marks)

1. 2 marks: Correctness of word-count application without any network partition/failures.
2. 3 marks: The word-count application is fault-tolerant to celery worker failures.
3. 5 marks: The word-count application is fault-tolerant to Redis instance failures. In this lab, we will assume that at most one Redis instance can be killed during an experiment (if a write made it to 2 replicas, write can be considered successful).
4. 7 marks: This has relative grading. The faster programs will receive higher marks.
5. 8 marks: This has relative grading. The programs that can achieve eventual consistency quicker after a network partition heals will receive higher marks.
6. 10 marks: Justifications and analysis as requested in the deliverables.

References

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