Evaluation Document

By: Aishwarya Malgonde, Sahil Jindal
Course: CS 677 | Professor Prashant Shenoy

Lab 3: Asterix and Double Trouble -- Replication, Caching, and Fault Tolerance

April 29, 2023

1. Should provide steps in your eval docs about how you deployed your application on AWS. Include scripts in your repo if needed (5%).

   Change `~/.aws/credentials` and download the pem file according to lablet 5

   Use `chmod 400 labsuser.pem`

   Use `aws configure` command to configure the aws and set region name and output format

   Now in the root directory use the command `aws ec2 run-instances --image-id ami-0044130ca185d0880 --instance-type m5a.large --key-name vockey > instance.json`. This command will start a m5a.large instance and install ubuntu 22.04 LTS on it. Here `ami-0044130ca185d0880` is the ubuntu 22.04 image for us-east-1 region.

   Open the port `aws ec2 authorize-security-group-ingress --group-name default --protocol tcp --port 8000 --cidr 0.0.0.0/0`

   Use `aws ec2 describe-instances --instance-id <your-instance-id>` to get the public DNS

   Using the public dns of the instace ssh to it using the command `ssh -i labsuser.pem ubuntu@<your-instance's-public-DNS-name>`

   Now install softwares

   ```
   sudo apt update
   ```

```
sudo apt install redis-server
```

```
sudo nano /etc/redis/redis.conf
```

Inside the file, find the supervised directive. This directive allows you to declare an init system to manage Redis as a service, providing you with more control over its operation. The supervised directive is set to no by default. Since you are running Ubuntu, which uses the systemd init system, change this to systemd

```
sudo systemctl status redis
```

The redis server should start on 6379 port

Use the following command to restart redis service `sudo systemctl restart redis.service`

install pip

```
sudo apt install python3-pip
```

Install the libraries

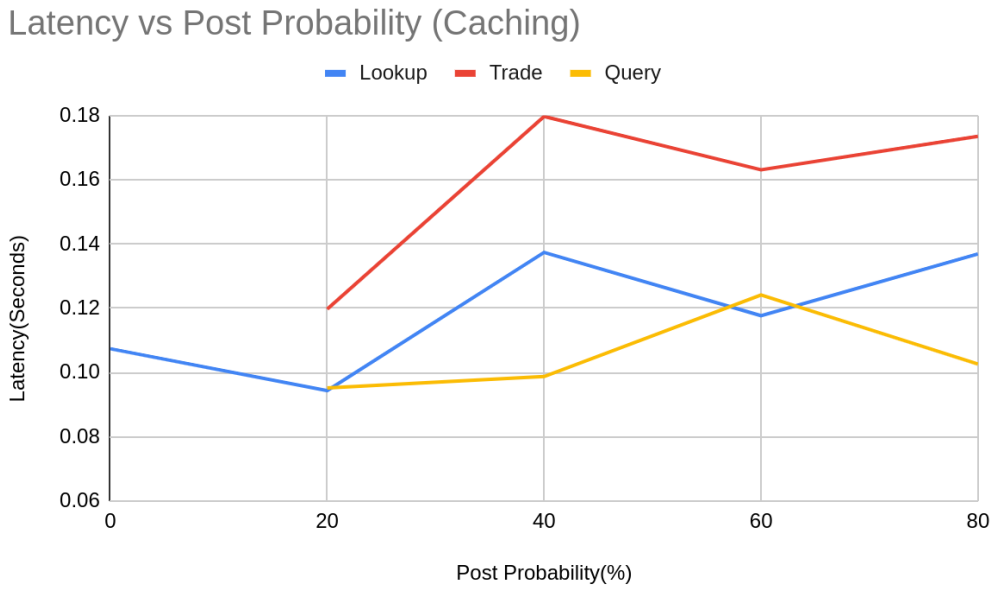
```
pip3 install Pyro5 pip3 install pandas sudo pip install redis
```

We can save the image of this instance so that we dont have to create it from scratch.

2. Next, deploy your application on an m5a.xlarge instance in the us-east-1 region on AWS. We will provide instructions on how to do this in lablet 5. Run 5 clients on your local machine. Measure the latency seen by each client for different types of requests. Change the probability p of a follow up trade request from 0 to 80%, with an increment of 20%, and record the result for each p setting. Make simple plots showing the values of p on the X-axis and the latency of different types of request on the y-axis. Also do the same experiments but with caching turned off, estimate how much benefits does caching provide by comparing the results.

For Post Probability = 0, Trade and Query have undefined latency

probability	lookup	trade	query
0	0.1073943853	nan	nan
20	0.09435426831	0.1197237402	0.09521059622
40	0.1373630655	0.1796973315	0.09874436488
60	0.1176870196	0.1630898475	0.1241363448
80	0.1368824637	0.1735046694	0.1026228504

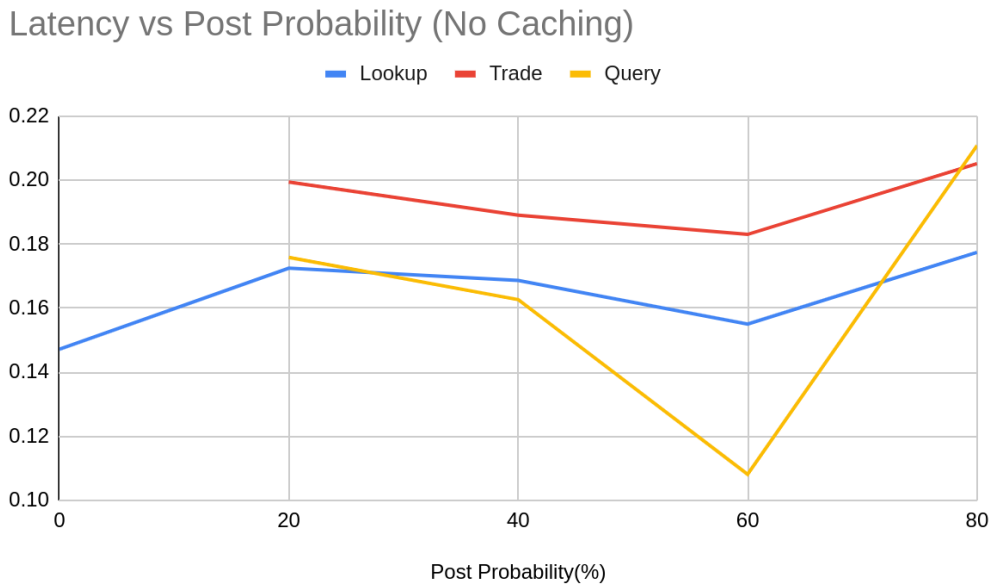


The above graph is the latency vs post probability plot when the caching is turned on.

probability	lookup	trade	query

4

0	0.1471792748	nan	nan
20	0.1724728284	0.1993552981	0.1758616613
40	0.1686723135	0.1890045994	0.1626488476
60	0.1550333898	0.1830307982	0.1081307854
80	0.1774384751	0.2051112908	0.2107870109



The above graph is the latency vs post probability plot when the caching is turned off.

With caching is turned on we get less average latency . We see an approximate 100 ms latency improvement with caching in lookup and trade requests. This shows that caching is really beneficial since the frontend does not have to communicate with the backend for duplicate requests.

3. Finally, simulate crash failures by killing a random order service replica while the client is running, and then bring it back online after some time. Repeat this experiment several times and make sure that you test the case when the leader is killed. Can the clients notice the failures? (either during order requests or the final order checking phase) or are they transparent to the clients? Do all the order service replicas end up with the same database file?

a. The clients cannot notice these failures, they are transparent. We have handled all the error handling on the frontend side, including the scenario where the leader fails. The frontend tries choosing a leader at least three times before sending an 500 error message to the client.

b. Yes, all the order replicas end up with the same db file. It fails in one case, where the leader fails after successfully processing an order trade, but before notifying the other replicas. In the design doc, we have clearly mentioned that we are assuming that the leader does not fail in such a way. It either fails before executing any order, or after notifying all the replicas about an order.