

Evaluation Doc

AWS Deployment

Copy the AWS CLI code and save it to `$HOME/.aws/credentials`

Run “aws configure”

1. For AWS Access Key ID, it's already configured in the previous step by creating a credentials file. Just press enter to skip this step.
2. For AWS Secret Access Key, it's also configured in the previous step. Press enter to skip this step.
3. For default region name, input “us-east-1”.
4. For default output format, input “json”.

Create an EC2 instance using AWS CLI. Open a terminal and run:

```
aws ec2 run-instances --image-id ami-0d73480446600f555
--instance-type m5a.large --key-name vockey > instance.json
```

For image id we used Ubuntu 18.04 server image provided by Canonical, at the time of writing the latest AMI (Amazon Machine Images) ID is “ami-0d73480446600f555”

The “instance.json” file will contain the details of your instance. Copy the “InstanceId” field. Now in your terminal check the status of your instance using the following command:

```
aws ec2 describe-instances --instance-id <your-instance-id>
```

Example:

```
aws ec2 describe-instances --instance-id i-07ba1758d2fe611e6
```

Get the public DNS name from the field “PublicDnsName”:

Example:

```
"ec2-3-83-14-35.compute-1.amazonaws.com"
```

Get the public IP address from the field "PublicIpAddress" which will be used by the client to send requests to the front-end:

Example:

```
"3.83.14.35"
```

Download the PEM key and set the right permission for the PEM key.

```
chmod 400 labsuser.pem
```

The other thing we need to do is to authorize port 22 (used by ssh) and port 8000 (used by our front-end service) in the default security group (you can think of security group as a virtual firewall)

```
aws ec2 authorize-security-group-ingress --group-name default  
--protocol tcp --port 22 --cidr 0.0.0.0/0
```

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

Now run ssh to access the instance:

```
ssh -i labsuser.pem ubuntu@<your-instance's-public-DNS-name>
```

Example:

```
ssh -i labsuser.pem  
ubuntu@ec2-3-83-14-35.compute-1.amazonaws.com
```

In the EC2 instance terminal, run git clone to clone the lab directory.

```
git clone  
https://github.com/umass-cs677-current/spring23-lab-3-677-lab3.git
```

Run the catalog service on one terminal:

```
CATALOG_HOST="0.0.0.0" CATALOG_PORT=6000 python3  
catalogService.py
```

Run the first order service replica on one terminal:

```
SERVICE_ID="5" ORDER_HOST="0.0.0.0" ORDER_PORT=6001  
CATALOG_HOST="0.0.0.0" CATALOG_PORT=6000 ORDER_ID="5,6,8"  
FILE_PATH="../data/" ORDER_PORTS="6001,6002,6003"  
ORDER_HOSTS="0.0.0.0,0.0.0.0,0.0.0.0" python3 orderService.py
```

Run the second order service replica on one terminal:

```
SERVICE_ID="6" ORDER_HOST="0.0.0.0" ORDER_PORT=6002  
CATALOG_HOST="0.0.0.0" CATALOG_PORT=6000 ORDER_ID="5,6,8"  
FILE_PATH="../data/" ORDER_PORTS="6001,6002,6003"  
ORDER_HOSTS="0.0.0.0,0.0.0.0,0.0.0.0" python3 orderService.py
```

Run the third order service replica on one terminal:

```
SERVICE_ID="8" ORDER_HOST="0.0.0.0" ORDER_PORT=6003  
CATALOG_HOST="0.0.0.0" CATALOG_PORT=6000 ORDER_ID="5,6,8"  
FILE_PATH="../data/" ORDER_PORTS="6001,6002,6003"  
ORDER_HOSTS="0.0.0.0,0.0.0.0,0.0.0.0" python3 orderService.py
```

Run the front-end service on one terminal:

```
CACHE_SIZE=5 FRONTEND_HOST="0.0.0.0" FRONTEND_PORT=8000  
CATALOG_HOST="0.0.0.0" CATALOG_PORT=6000  
ORDER_HOSTS="0.0.0.0,0.0.0.0,0.0.0.0" ORDER_ID="5,6,8"  
ORDER_PORTS="6001,6002,6003" python3 front-end-http-server.py
```

Run the clients on the localhost:

```
python3 client.py <PublicIPAddress> <port>
```

Example:

```
python3 client.py "3.83.14.35" 8000
```

At the end kill the instance:

```
aws ec2 terminate-instances --instance-ids <your-instance-id>
```

Example:

```
aws ec2 terminate-instances --instance-ids i-07ba1758d2fe611e6
```

Measurement results and plots

The services were run on AWS while 5 clients were running on localhost

Cache size = 5

Probability p: 1

Total stock lookup requests: 200

Average stock lookup time : 37.36081123352051ms

Total trade requests: 168

Average trade time : 51.50053188914344ms

Total order lookup requests: 168

Average order lookup time: 47.838448058991204ms

Probability p: 0.8

Total stock lookup requests: 200

Average stock lookup time : 46.8511438369751ms

Total trade requests: 118

Average trade time : 53.81907042810472ms

Total order lookup requests: 118

Average order lookup time: 40.964815576197736ms

Probability p: 0.6

Total stock lookup requests: 200

Average stock lookup time : 35.02136468887329ms

Total trade requests: 96

Average trade time : 46.86927298704783ms

Total order lookup requests: 96

Average order lookup time: 38.41234495242437ms

Probability p: 0.4

Total stock lookup requests: 200

Average stock lookup time : 35.36201357841492ms
Total trade requests: 70
Average trade time : 47.79754025595528ms
Total order lookup requests: 70
Average order lookup time: 40.84650107792446ms
Probability p: 0.2
Total stock lookup requests: 200
Average stock lookup time : 34.592087268829346ms
Total trade requests: 21
Average trade time : 44.08895401727585ms
Total order lookup requests: 21
Average order lookup time: 42.05333618890671ms
Probability p: 0
Total stock lookup requests: 200
Average stock lookup time : 35.41490435600281ms
Total trade requests: 0
Average trade time : 0ms
Total order lookup requests: 0
Average order lookup time: 0ms

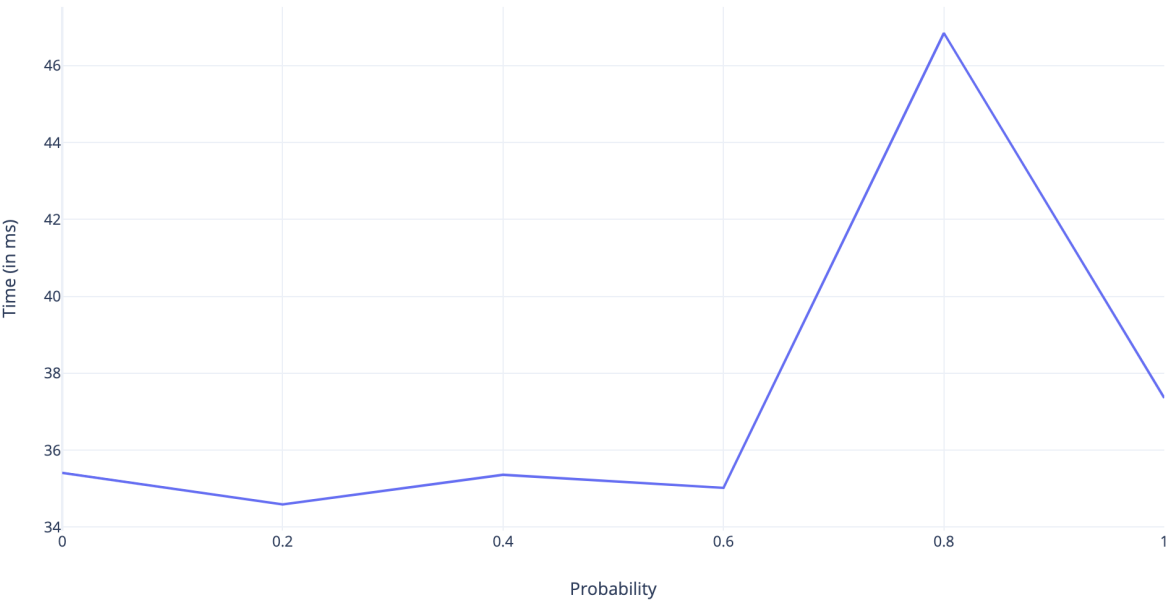
Cache size = 0

Probability p: 1
Total stock lookup requests: 200
Average stock lookup time : 37.69152760505676ms
Total trade requests: 156
Average trade time : 48.75322488638071ms
Total order lookup requests: 156
Average order lookup time: 39.372366208296555ms
Probability p: 0.8
Total stock lookup requests: 200
Average stock lookup time : 41.6025447845459ms
Total trade requests: 145
Average trade time : 47.191779366854966ms
Total order lookup requests: 145
Average order lookup time: 43.85764352206526ms
Probability p: 0.6
Total stock lookup requests: 200
Average stock lookup time : 37.962894439697266ms
Total trade requests: 83

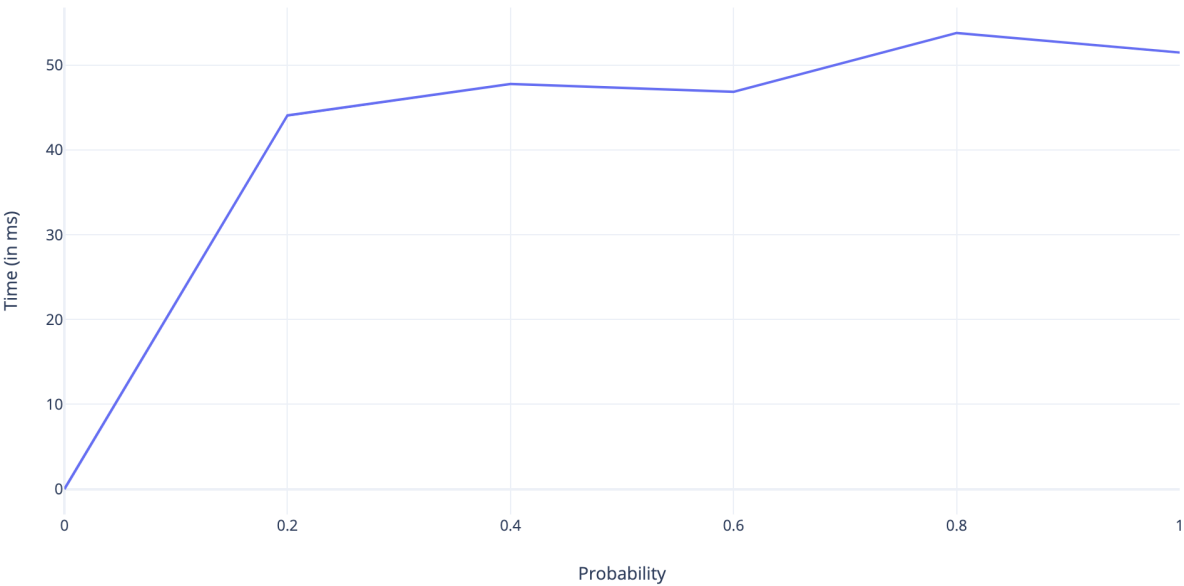
Average trade time : 52.68632360251553ms
Total order lookup requests: 83
Average order lookup time: 38.94853591918945ms
Probability p: 0.4
Total stock lookup requests: 200
Average stock lookup time : 40.159642696380615ms
Total trade requests: 59
Average trade time : 66.22393656585176ms
Total order lookup requests: 59
Average order lookup time: 49.39359325473591ms
Probability p: 0.2
Total stock lookup requests: 200
Average stock lookup time : 44.671605825424194ms
Total trade requests: 43
Average trade time : 55.79930128053177ms
Total order lookup requests: 43
Average order lookup time: 43.16348253294479ms
Probability p: 0
Total stock lookup requests: 200
Average stock lookup time : 42.48074173927307ms
Total trade requests: 0
Average trade time : 0ms
Total order lookup requests: 0
Average order lookup time: 0ms

- **Plots for times measured with cache:**

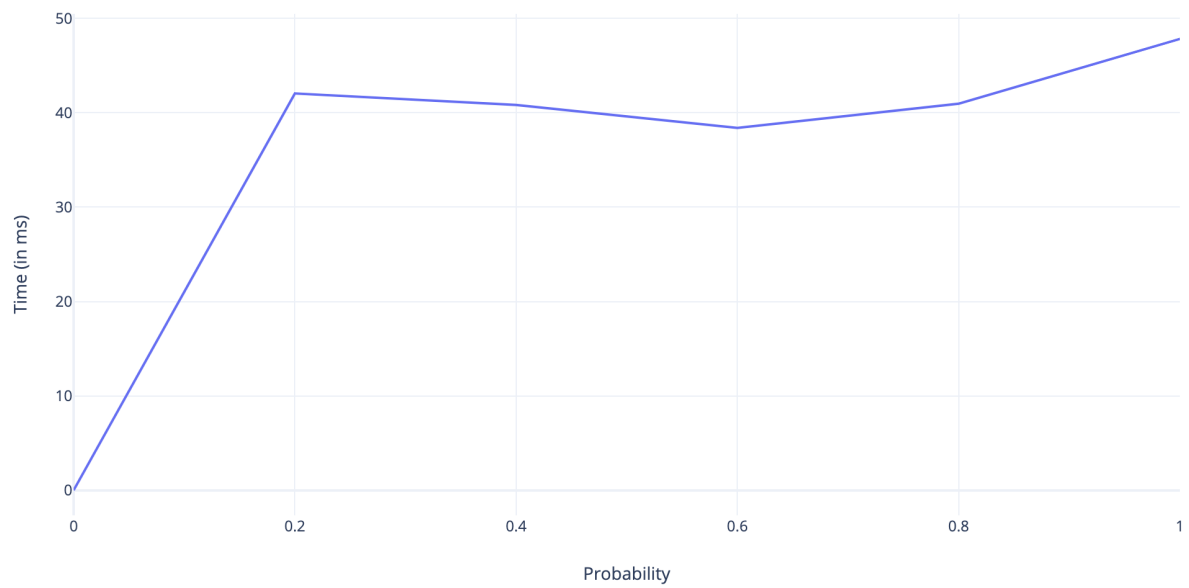
Average Stock Lookup Time with cache



Average Trade Time with cache

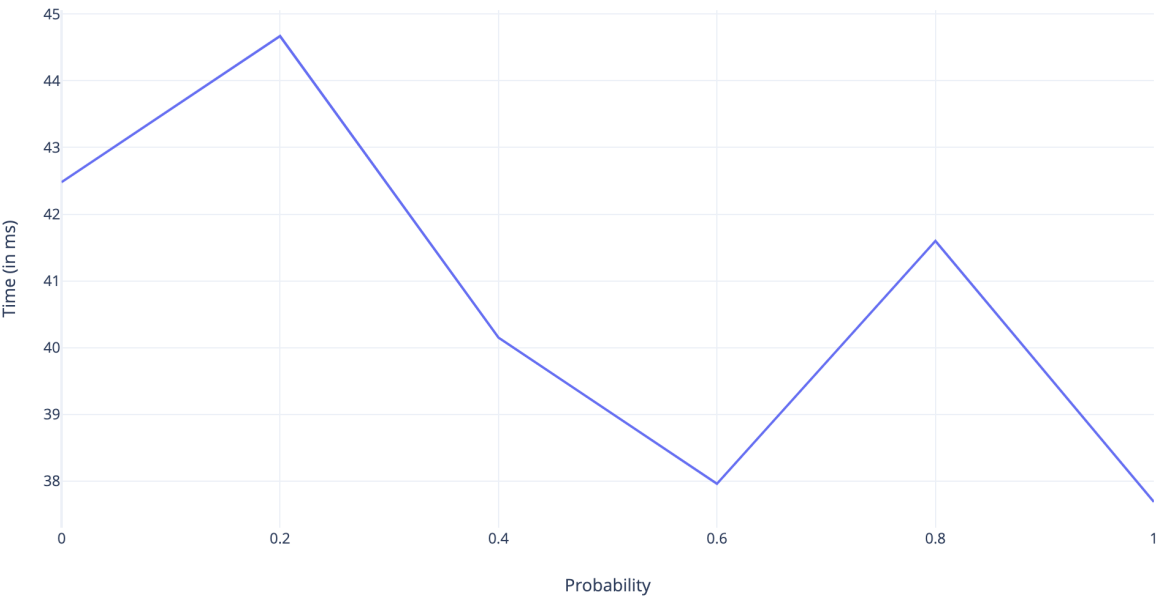


Average Order Lookup Time with cache

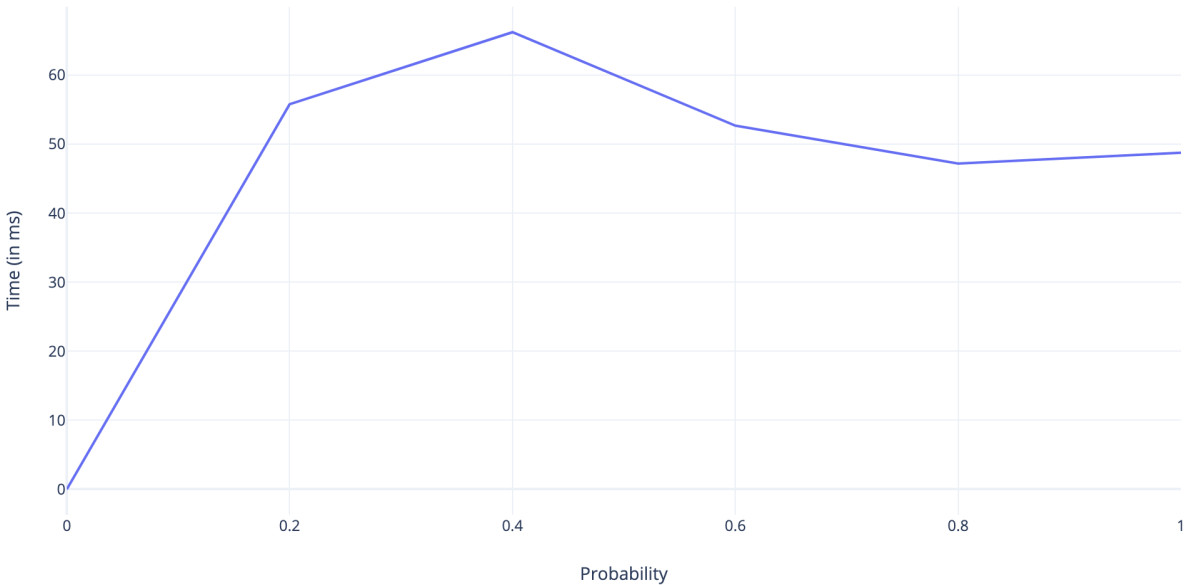


- **Plots for times measured without cache:**

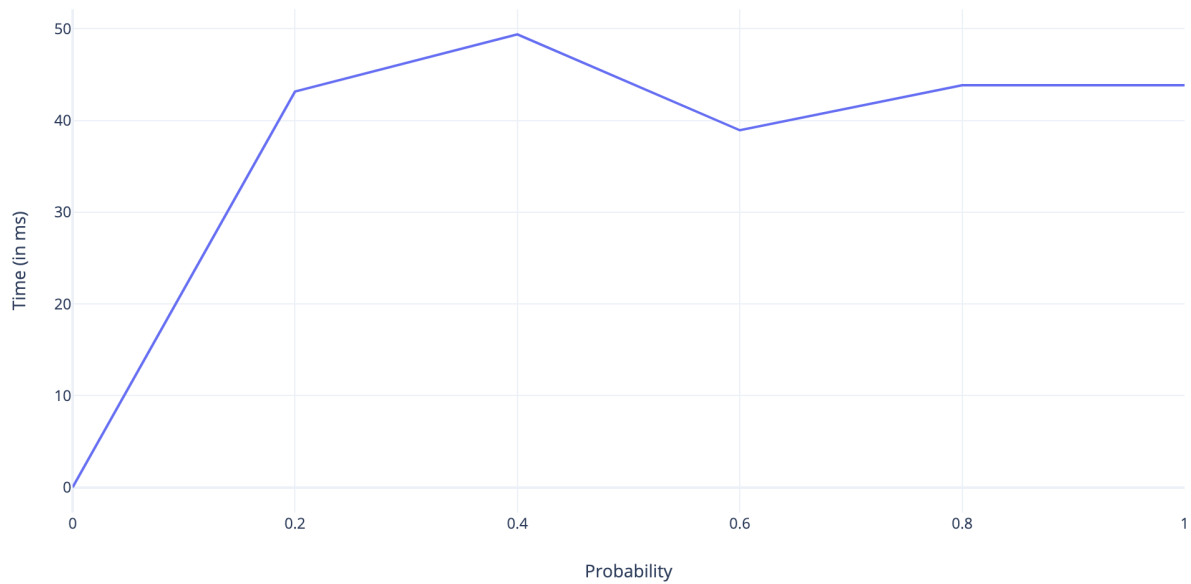
Average Stock Lookup Time without cache



Average Trade Time without cache



Average Order Lookup Time without cache



Observations:

1. Trade time was significantly lower when cache was used.
2. Stock lookup time was lower using cache than no cache.
3. Order lookup was slower when cache was not used.

Questions

Q1.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?

Ans: Tested by abruptly killing leader and follower services, and restarting them. The clients didn't notice any failures, as failures were taken care of by electing new leader and continuing serving requests using the new leader. Failures are transparent to the clients.

When the killed service was restarted, its database was updated by communicating with replicas and once updated it continued being synchronized with the service leader.

```
data:
{'data': {'name': 'Wheely', 'price': 64.2300033569336, 'quantity': 1969}}
Stock Quantity: 1969
Sending Trade request for stockname: Wheely , type: sell
response.status, response.reason, response.version :
200 OK 11
data:
{'data': {'transaction_number': 9886}}
Sending Lookup request for stockname: InvalidStockName
404 Not Found 11
data:
{'error': {'code': 404, 'message': 'stock not found'}}
Sending Lookup request for stockname: CarCo
200 OK 11
data:
{'data': {'name': 'CarCo', 'price': 45.0, 'quantity': 2002}}
Stock Quantity: 2002
Sending Trade request for stockname: CarCo , type: buy
response.status, response.reason, response.version :
200 OK 11
```


The order service replica database after it was stopped:

```
9859 - Stockname: ConStock Quantity: 1 Order: sell
9860 - Stockname: MenhirCo Quantity: 1 Order: buy
9861 - Stockname: BoarCo Quantity: 1 Order: buy
9862 - Stockname: Wheely Quantity: 1 Order: sell
9863 - Stockname: FishCo Quantity: 1 Order: sell
9864 - Stockname: GameStart Quantity: 1 Order: buy
9865 - Stockname: MenhirCo Quantity: 1 Order: buy
9866 - Stockname: GameStart Quantity: 1 Order: sell
9867 - Stockname: MenhirCo Quantity: 1 Order: sell
9868 - Stockname: FishCo Quantity: 1 Order: buy
9869 - Stockname: TheaterCo Quantity: 1 Order: buy
9870 - Stockname: MenhirCo Quantity: 1 Order: buy
9871 - Stockname: Wheely Quantity: 1 Order: buy
9872 - Stockname: BoarCo Quantity: 1 Order: sell
9873 - Stockname: MenhirCo Quantity: 1 Order: sell
9874 - Stockname: MenhirCo Quantity: 1 Order: buy
9875 - Stockname: CarCo Quantity: 1 Order: sell
9876 - Stockname: MenhirCo Quantity: 1 Order: buy
```

The order service replica database after it was restarted:

```
9870 - Stockname: MenhirCo Quantity: 1 Order: buy
9871 - Stockname: Wheely Quantity: 1 Order: buy
9872 - Stockname: BoarCo Quantity: 1 Order: sell
9873 - Stockname: MenhirCo Quantity: 1 Order: sell
9874 - Stockname: MenhirCo Quantity: 1 Order: buy
9875 - Stockname: CarCo Quantity: 1 Order: sell
9876 - Stockname: MenhirCo Quantity: 1 Order: buy
9877 - Stockname: CarCo Quantity: 1 Order: sell
9878 - Stockname: MenhirCo Quantity: 1 Order: buy
9879 - Stockname: GameStart Quantity: 1 Order: buy
9880 - Stockname: TheaterCo Quantity: 1 Order: sell
9881 - Stockname: TheaterCo Quantity: 1 Order: sell
9882 - Stockname: GameStart Quantity: 1 Order: sell
9883 - Stockname: MenhirCo Quantity: 1 Order: buy
9884 - Stockname: MenhirCo Quantity: 1 Order: sell
9885 - Stockname: MenhirCo Quantity: 1 Order: buy
9886 - Stockname: Wheely Quantity: 1 Order: sell
9887 - Stockname: CarCo Quantity: 1 Order: buy
9888 - Stockname: CarCo Quantity: 1 Order: buy
9889 - Stockname: BoarCo Quantity: 1 Order: buy
9890 - Stockname: MenhirCo Quantity: 1 Order: buy
9891 - Stockname: ConStock Quantity: 1 Order: sell
```

The killed replica synchronizes with other replicas after starting and starts updating its database with the updates it gets from the leader.

In our implementation all the replicas ended up with the same contents in database file, even when a service was killed and then restarted after some time.