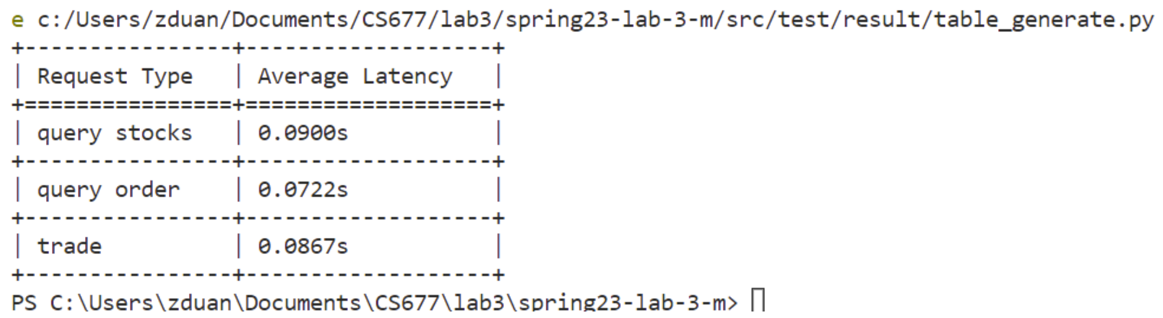
1. Load Test Results:

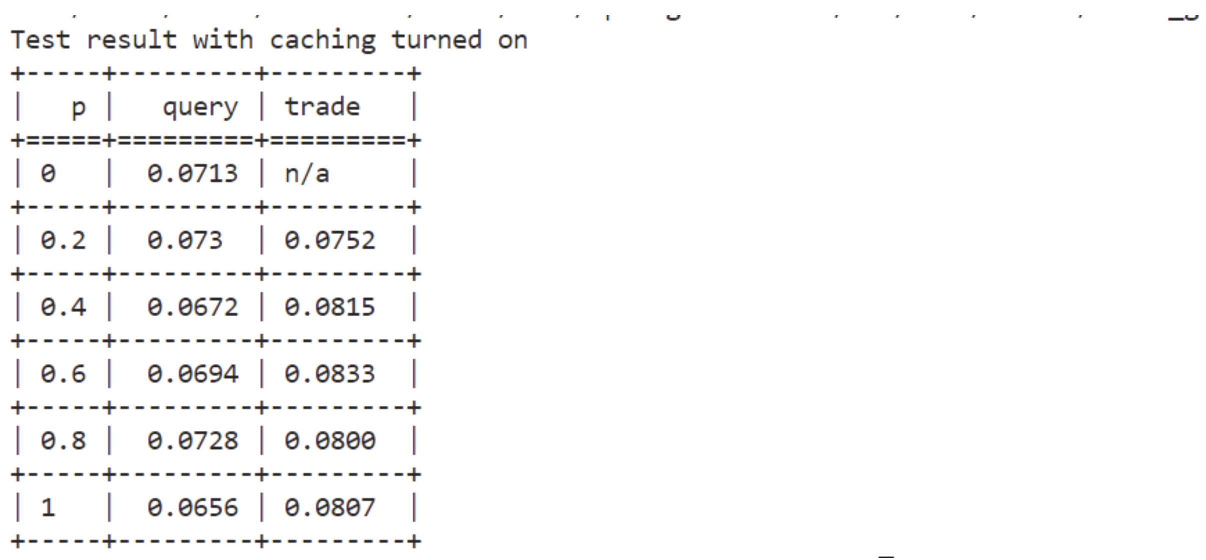
We implement the testing code to query 1000 stocks, query 1000 orders, and to perform 100 trade requests, designed according to aws processing capacity.
Python unittest measures the total latency and the average latency of each request can be calculated by dividing the total latency over the number of each request. For instance, the average latency for queries should be its overall latency over 1000. The table below shows the load test results.



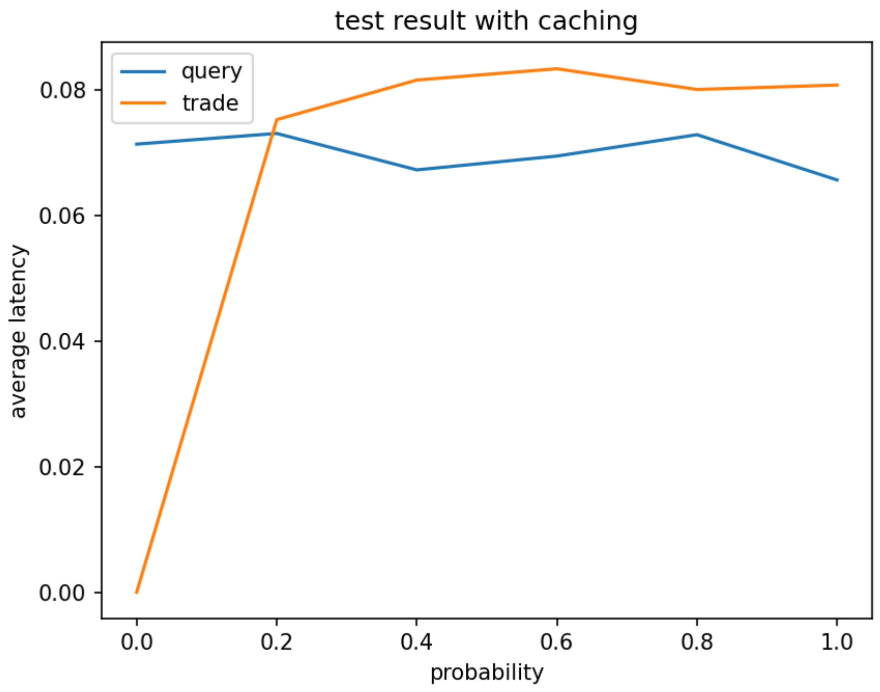
```
e c:/Users/zduan/Documents/CS677/lab3/spring23-lab-3-m/src/test/result/table_generate.py
+----------------+-------------------+
| Request Type   | Average Latency   |
+================+===================+
| query stocks   | 0.0900s           |
+----------------+-------------------+
| query order    | 0.0722s           |
+----------------+-------------------+
| trade          | 0.0867s           |
+----------------+-------------------+
PS C:\Users\zduan\Documents\CS677\lab3\spring23-lab-3-m> 
```

2. Caching Test Results

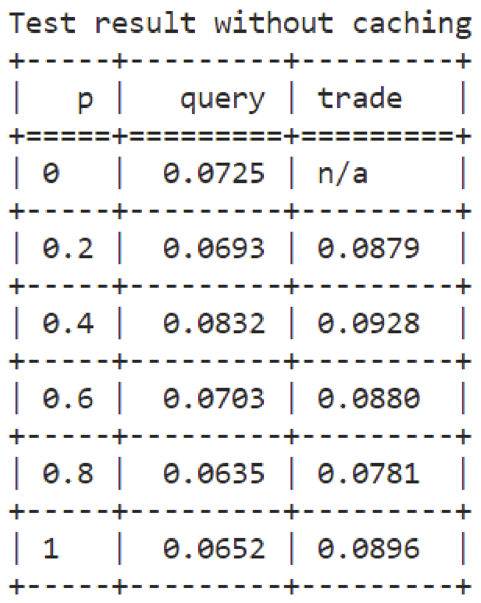
In order to see the performance improvement done by caching, we measured the latency seen by clients for query and trade requests with caching in use. We adjust the buying probability p after query from 0% to 100% with increments of 20%. The values list in the table is an average latency of repeating measurements to ensure a more accurate result.
The same experiment was performed with caching disabled. The tables below contain the measurement data.

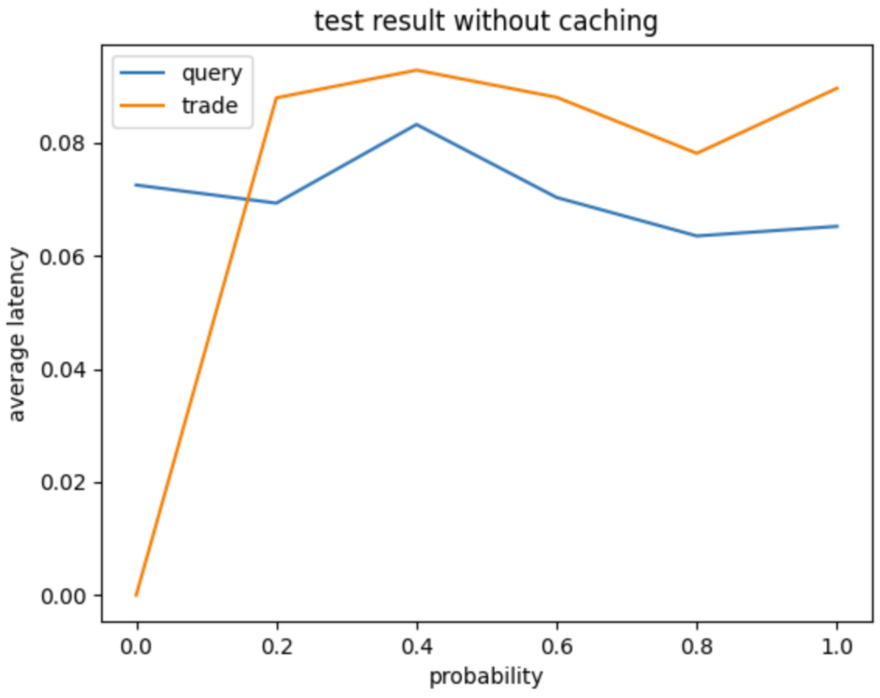


```
Test result with caching turned on
+-----+---------+---------+
|   p |   query | trade   |
+=====+=========+=========+
| 0   |  0.0713 | n/a     |
+-----+---------+---------+
| 0.2 |  0.073  | 0.0752  |
+-----+---------+---------+
| 0.4 |  0.0672 | 0.0815  |
+-----+---------+---------+
| 0.6 |  0.0694 | 0.0833  |
+-----+---------+---------+
| 0.8 |  0.0728 | 0.0800  |
+-----+---------+---------+
| 1   |  0.0656 | 0.0807  |
+-----+---------+---------+
```



The following results are from the same experiments without caching:


```
Test result without caching
+-----+---------+---------+
|   p |   query | trade   |
+=====+=========+=========+
| 0   |  0.0725 | n/a     |
+-----+---------+---------+
| 0.2 |  0.0693 | 0.0879  |
+-----+---------+---------+
| 0.4 |  0.0832 | 0.0928  |
+-----+---------+---------+
| 0.6 |  0.0703 | 0.0880  |
+-----+---------+---------+
| 0.8 |  0.0635 | 0.0781  |
+-----+---------+---------+
| 1   |  0.0652 | 0.0896  |
+-----+---------+---------+
```



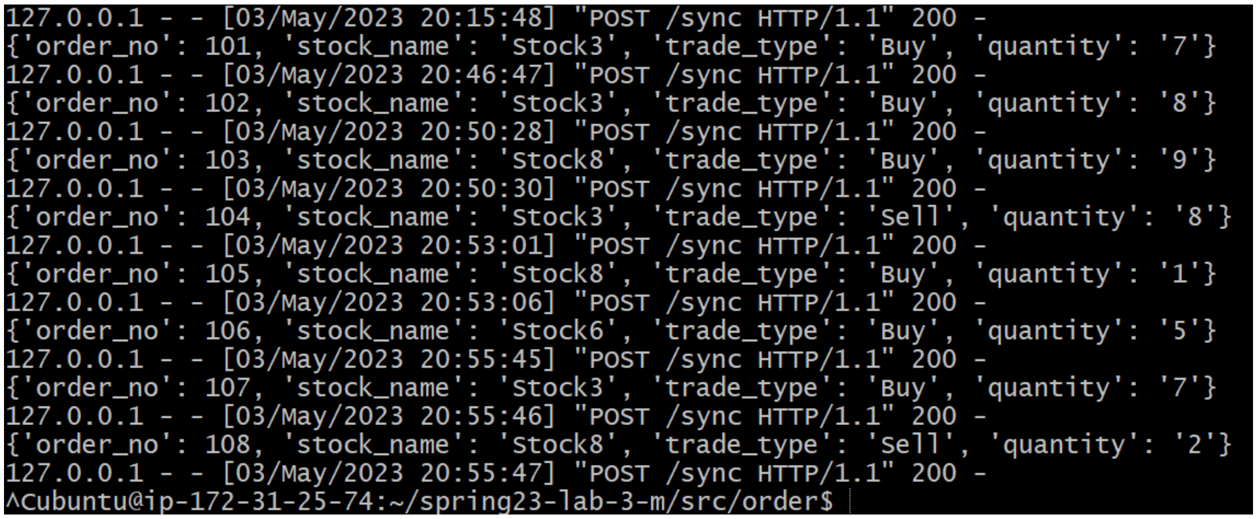
Finally, simulate crash failures by killing a random order service replica while the client is
running, and then bring it back online after some time. Repeat this experiment several times and
make sure that you test the case when the leader is killed. Can the clients notice the failures?

-The clients cannot notice the failures. The failures were handled by the server and were transparent to the clients..

(either during order requests or the final order checking phase) or are they transparent to the
clients?

-The failures are transparent to the clients.

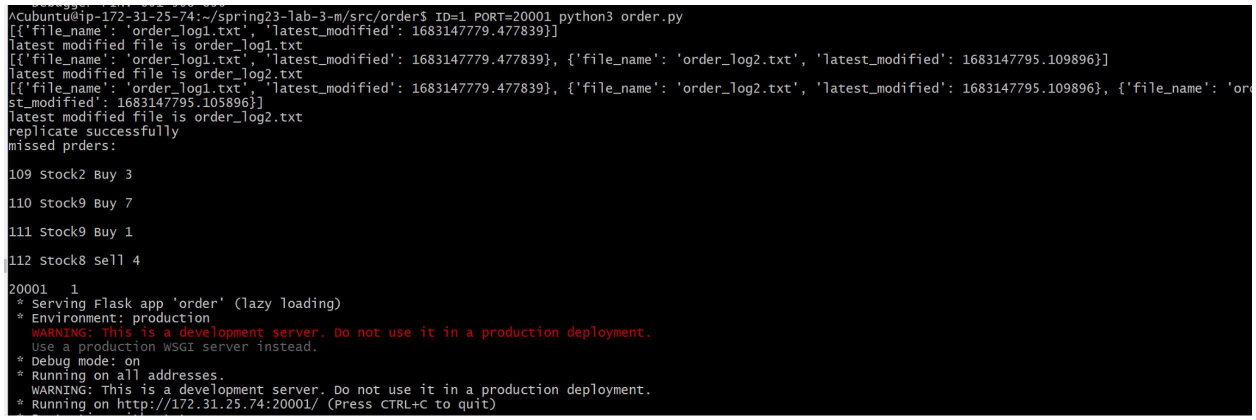
First, try to kill server 20001. Crash order server which id=1,port=20001:



```
127.0.0.1 - - [03/May/2023 20:15:48] "POST /sync HTTP/1.1" 200 -
{'order_no': 101, 'stock_name': 'Stock3', 'trade_type': 'Buy', 'quantity': '7'}
127.0.0.1 - - [03/May/2023 20:46:47] "POST /sync HTTP/1.1" 200 -
{'order_no': 102, 'stock_name': 'Stock3', 'trade_type': 'Buy', 'quantity': '8'}
127.0.0.1 - - [03/May/2023 20:50:28] "POST /sync HTTP/1.1" 200 -
{'order_no': 103, 'stock_name': 'Stock8', 'trade_type': 'Buy', 'quantity': '9'}
127.0.0.1 - - [03/May/2023 20:50:30] "POST /sync HTTP/1.1" 200 -
{'order_no': 104, 'stock_name': 'Stock3', 'trade_type': 'Sell', 'quantity': '8'}
127.0.0.1 - - [03/May/2023 20:53:01] "POST /sync HTTP/1.1" 200 -
{'order_no': 105, 'stock_name': 'Stock8', 'trade_type': 'Buy', 'quantity': '1'}
127.0.0.1 - - [03/May/2023 20:53:06] "POST /sync HTTP/1.1" 200 -
{'order_no': 106, 'stock_name': 'Stock6', 'trade_type': 'Buy', 'quantity': '5'}
127.0.0.1 - - [03/May/2023 20:55:45] "POST /sync HTTP/1.1" 200 -
{'order_no': 107, 'stock_name': 'Stock3', 'trade_type': 'Buy', 'quantity': '7'}
127.0.0.1 - - [03/May/2023 20:55:46] "POST /sync HTTP/1.1" 200 -
{'order_no': 108, 'stock_name': 'Stock8', 'trade_type': 'Sell', 'quantity': '2'}
127.0.0.1 - - [03/May/2023 20:55:47] "POST /sync HTTP/1.1" 200 -
^Cubuntu@ip-172-31-25-74:~/spring23-lab-3-m/src/order$
```

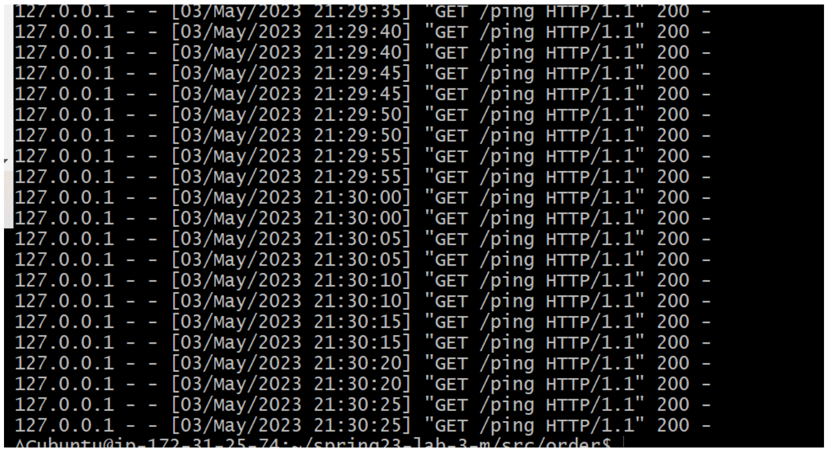
Then restart the server 20001:

We can see the missed orders during the time server 20001 was off.



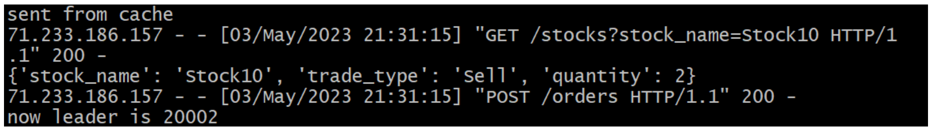
```
^Cubuntu@ip-172-31-25-74:~/spring23-lab-3-m/src/order$ ID=1 PORT=20001 python3 order.py
[{'file_name': 'order_log1.txt', 'latest_modified': 1683147779.477839}]
latest modified file is order_log1.txt
[{'file_name': 'order_log1.txt', 'latest_modified': 1683147779.477839}, {'file_name': 'order_log2.txt', 'latest_modified': 1683147795.109896}]
latest modified file is order_log2.txt
[{'file_name': 'order_log1.txt', 'latest_modified': 1683147779.477839}, {'file_name': 'order_log2.txt', 'latest_modified': 1683147795.109896}, {'file_name': 'or
st_modified': 1683147795.105896}]
latest modified file is order_log2.txt
replicate successfully
missed prders:

109 Stock2 Buy 3

110 Stock9 Buy 7

111 Stock9 Buy 1

112 Stock8 Sell 4

20001   1
 * Serving Flask app 'order' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.25.74:20001/ (Press CTRL+C to quit)
```

Take off the leader server 20003:



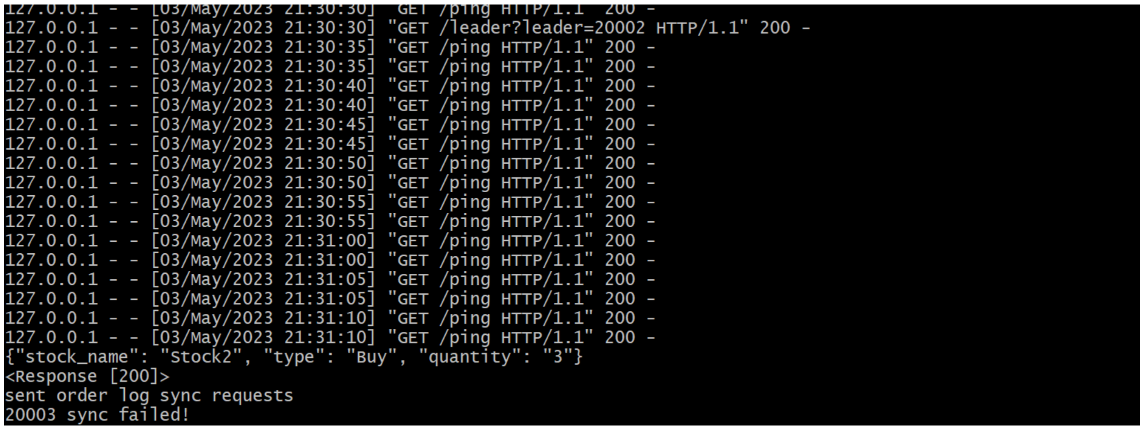
```
127.0.0.1 - - [03/May/2023 21:29:35] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:40] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:40] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:45] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:45] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:50] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:50] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:55] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:29:55] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:00] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:00] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:05] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:05] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:10] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:10] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:15] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:15] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:20] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:20] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:25] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:25] "GET /ping HTTP/1.1" 200 -
^Cubuntu@ip-172-31-25-74:~/spring23-lab-3-m/src/order$
```

Server 20002 become the new leader:



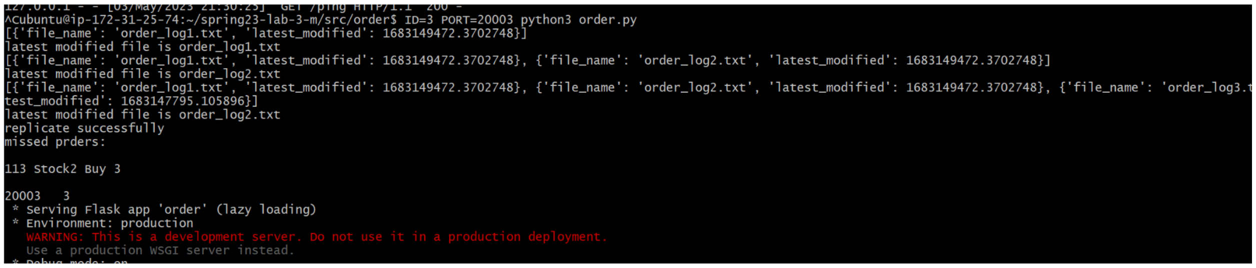
```
sent from cache
71.233.186.157 - - [03/May/2023 21:31:15] "GET /stocks?stock_name=Stock10 HTTP/1
.1" 200 -
{'stock_name': 'Stock10', 'trade_type': 'Sell', 'quantity': 2}
71.233.186.157 - - [03/May/2023 21:31:15] "POST /orders HTTP/1.1" 200 -
now leader is 20002
```

Server 20002 process the order as leader order server:



```
127.0.0.1 - - [03/May/2023 21:30:30] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:30] "GET /leader?leader=20002 HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:35] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:35] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:40] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:40] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:45] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:45] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:50] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:50] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:55] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:30:55] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:00] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:00] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:05] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:05] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:10] "GET /ping HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2023 21:31:10] "GET /ping HTTP/1.1" 200 -
{"stock_name": "Stock2", "type": "Buy", "quantity": "3"}
<Response [200]>
sent order log sync requests
20003 sync failed!
```

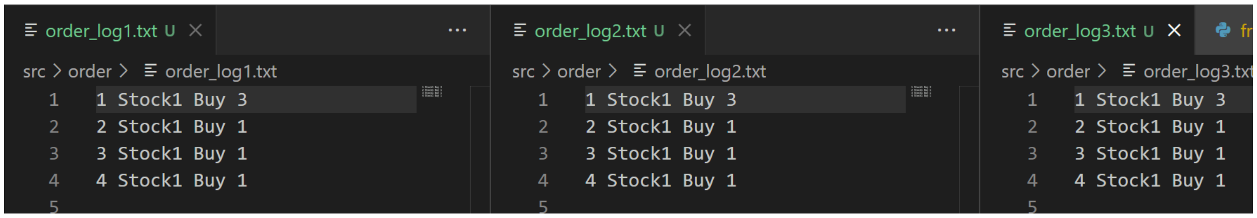
restart the previous lead server 20003:



```
127.0.0.1 - - [03/May/2023 21:30:25] "GET /ping HTTP/1.1" 200 -
^Cubuntu@ip-172-31-25-74:~/spring23-lab-3-m/src/order$ ID=3 PORT=20003 python3 order.py
[{'file_name': 'order_log1.txt', 'latest_modified': 1683149472.3702748}]
latest modified file is order_log1.txt
[{'file_name': 'order_log1.txt', 'latest_modified': 1683149472.3702748}, {'file_name': 'order_log2.txt', 'latest_modified': 1683149472.3702748}]
latest modified file is order_log2.txt
[{'file_name': 'order_log1.txt', 'latest_modified': 1683149472.3702748}, {'file_name': 'order_log2.txt', 'latest_modified': 1683149472.3702748}, {'file_name': 'order_log3.t
test_modified': 1683147795.105896}]
latest modified file is order_log2.txt
replicate successfully
missed prders:

113 Stock2 Buy 3

20003   3
 * Serving Flask app 'order' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

We can see the missed order record.

```
Do all the order service replicas end up with the same database file?
```
--Yes



```
order_log1.txt U
src > order > order_log1.txt
1 Stock1 Buy 3
2 Stock1 Buy 1
3 Stock1 Buy 1
4 Stock1 Buy 1
```

```
order_log2.txt U
src > order > order_log2.txt
1 Stock1 Buy 3
2 Stock1 Buy 1
3 Stock1 Buy 1
4 Stock1 Buy 1
```

```
order_log3.txt U
src > order > order_log3.txt
1 Stock1 Buy 3
2 Stock1 Buy 1
3 Stock1 Buy 1
4 Stock1 Buy 1
```