677 Lab Assignment – 3
Vikas Mehta, Purva Jhaveri

Design Description

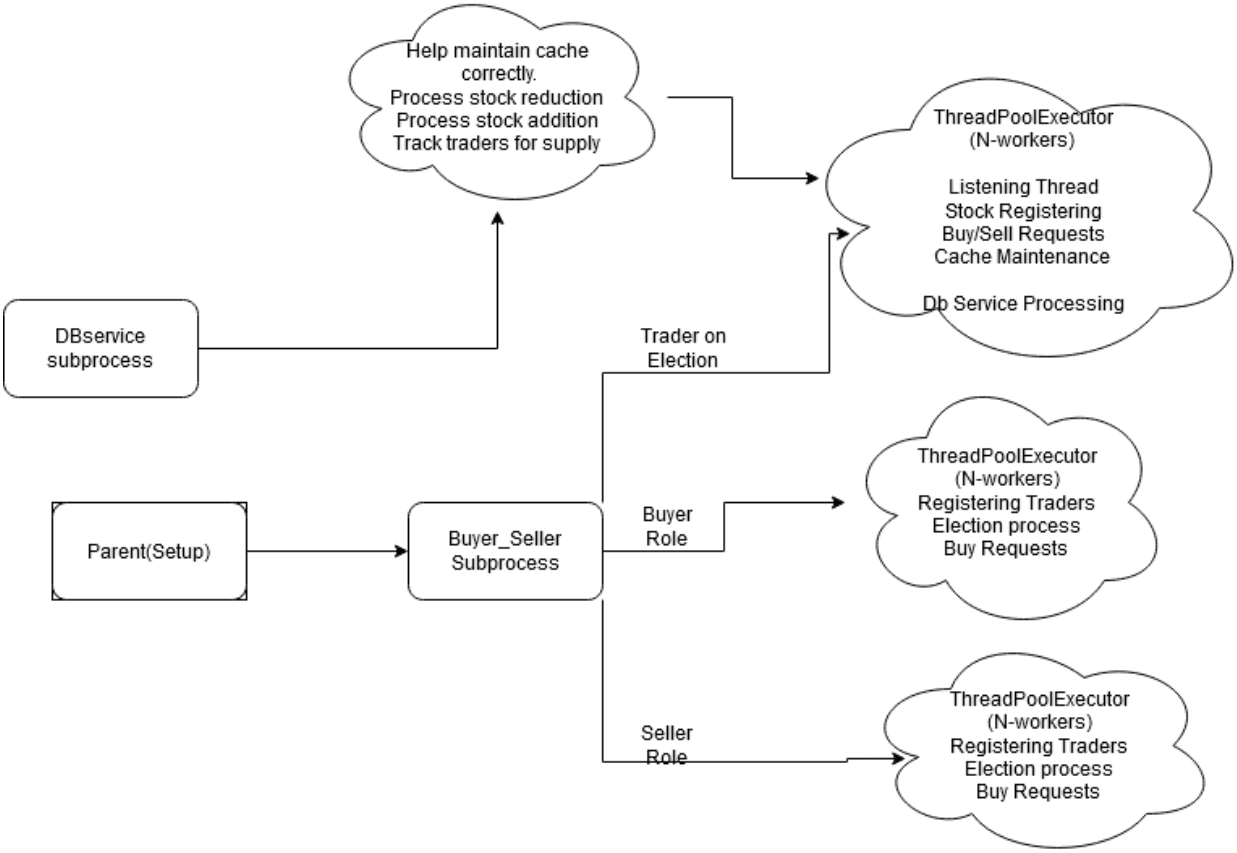


Fig 1. Design of the Bazaar

1. The setup program sets up two networks that elect their individual leaders through ring algorithm where a peer starts election process and sends the list of all visited election members node_id. For the peers we have used **the subprocess library of python** to make the processes in different directories and not share the same memory.

2. The dbService must be initialized before the network is established. The dbService is hosted at a completely different port and process and has methodology to fetch and write stock data into a file. This service has it's on listening thread

3. The buyer or seller nodes begin the process of generating requests to the leader/s only after the election state is completed.
4. The nodes maintain their list of traders after receiving a broadcast message from the leaders when they are elected. The traders are maintained in the list are requests are sent to these traders randomly.

5. In case of synchronous mode, the dbService has a threading lock that the traders need to acquire to process requests, these can be both buy and sell requests.

6. In case of cached mode, the dbService consists of a logical segment where the count of stock is taken into consideration given the number of traders and appropriately changes are made on the file to mock an upsert request when the cache is sent as a request from the traders after processing several transactions.

7. For achieving this contract the dbService in cached approach maintains the list of aliveTraders so it can keep a track of the available cache sources across the network that the nodes can leverage. The caching mechanism makes use of casual consistency to prevent abundance of overselling and underselling in case of high traffic.

8. For avoiding inconsistency in cached mode wrt stock of fish boar and salt the traders send out a message gathering the overall stock across all traders in case when the amount of stock is way lower than that requested by buyers. This causes the traders to maintain consistency.

9. In case of a trader fault the other trader needs to take up the responsibility and resume requests and even take in new requests to itself. For this reason, periodically the traders store their recovery files with the cache and the request list. The other trader periodically detects if the other trader is responding correctly or not. In case of no/incorrect response. The trader reads the logged file and appends its request list and informs all the clients of the other trader about the failure of other trader and removes the other trader from their trader list.

10. The consistency model implemented here is causal consistency. The causation of a buy request when there is no cache available for a specific requests causes the trader to call other traders, align the cache and process the complete cache write on the dbservice. This write also is followed by a read across all the Traders. This type of consistency can help reduce overselling and underselling to some extent.

The image below represents a sample scenario of the above design.

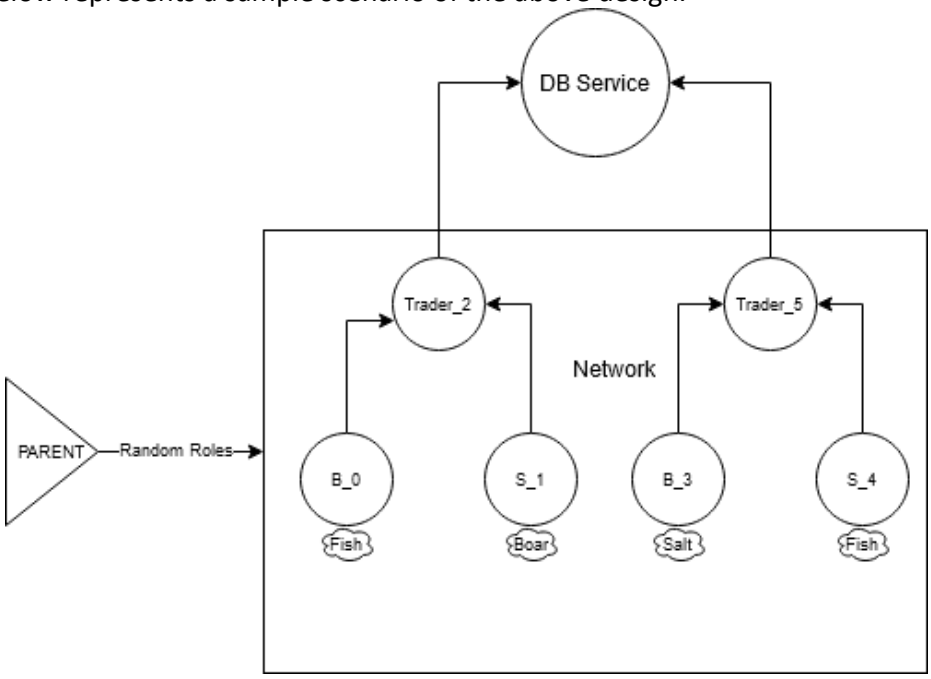


Fig 2. Sample Network

POSSIBLE IMPROVEMENTS/HOW TO RUN

To run the cache/synchronous networks. Go to the respective folders for cache based or synchronous, open 3 terminals. In one terminal start the naming server using the command:

python -m Pyro4.naming

In the other terminal start the dbService.py by running the command

python dbService.py

In the last terminal run the following command to setup the peers:

Python setup.py 6

To setup 6 peers

Improvement/Extension Scope:

1. The design suggests that in case of underselling and overselling for such a system linearized or sequential consistency can lead to better results.
   - Sketch
     {
     To address this issue, we might need an extensive implementation of clocks or keep a log of read and write requests so they can be interleaved for sequential consistency. We can use vector clocks to achieve such a consistency.
     }

2. The buy processes connect to their traders randomly. One of the scopes for extension includes the act of load balancing wrt the traders so that the load on a single trader does not throttle requests
   - Sketch
   {
   For this method of load balancing the traders need to keep a track of how busy the request queue is and using another service that routes requests from buyers can be implemented so that buyers are agnostic of the load balancing step and do not need to keep a track of the same.
   }

3. The design is structured in a circular pattern that elects leaders in the form a ring election. This restricts the network to recover from catastrophic failures
   - Sketch
   {
   To make sure that such a recovery is possible. There needs to be additional algorithm for neighbor discovery and different algorithm for leader election like a bully algorithm .
   }
4. Autoscaling with of number of leaders wrt to number of buyers and sellers and the number of requests can be one of the extensions that can be made to the system and make the system more reliable.
   - Sketch

   {

   For implementing such a system, the traders need to communicate the load that they are processing with each other and have the ability to invoke more leaders either through replication of the same trader or one of the nodes through another re-election.

   }

Description of tests:

When elections are completed for the peers in the network the message that it sends in the log file

The trader has been added to the trader list is the message that represents that a trader has been elected and now represented in the buyer_seller list and requests can now be generated for transactions to begin.

These can lead to several edge cases that need to be resolved properly and we have added these test cases to confirm that the system works and produces expected results:

Test case 1: 2 buyer 2 seller 2 traders, only 1 type of sell item: This test case presented a case with 6 nodes and appropriate transactions were made while showing that the election occurred and with the edge case that sellers sell only 1 type of item for example fish. The system works as expected and all other type of buy requests from buyers are rejected.

Test case 2: 4 buyers, 0 Sellers, 2 Traders: This test case represents a case where there are no sellers and the traders/dbService is never updated with new items to sell. In this case all buy requests are rejected.

Test case 3: 0 buyers, 4 Sellers, 2 Traders: This test case represents a case where there are no buyers and the traders/dbService is are flooded with addition

Test case 4: 1 buyer, 3 Seller, 2 Traders : This test case represents the case of only 1 buyer but multiple traders and monitors how a single buyer interacts with 2 traders.

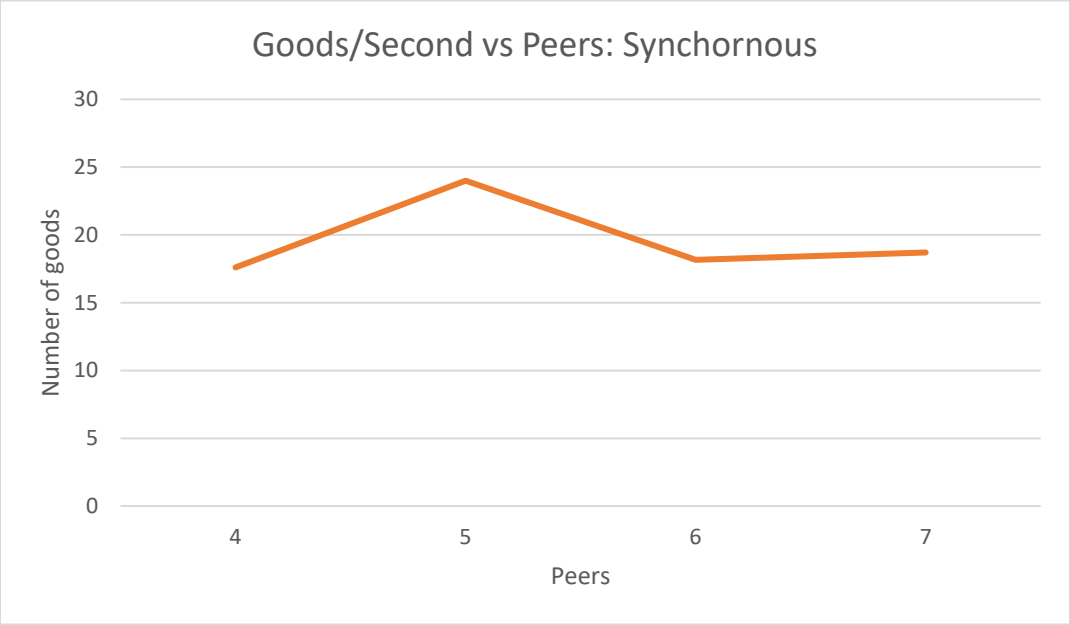
Test case 5: 2 buyer, 2 sellers, 2 traders: This represents a standard case for the network to operate in and all the necessary transactions that can take place are presents. This represents an optimistic view of the network where balancing of requests is almost not required

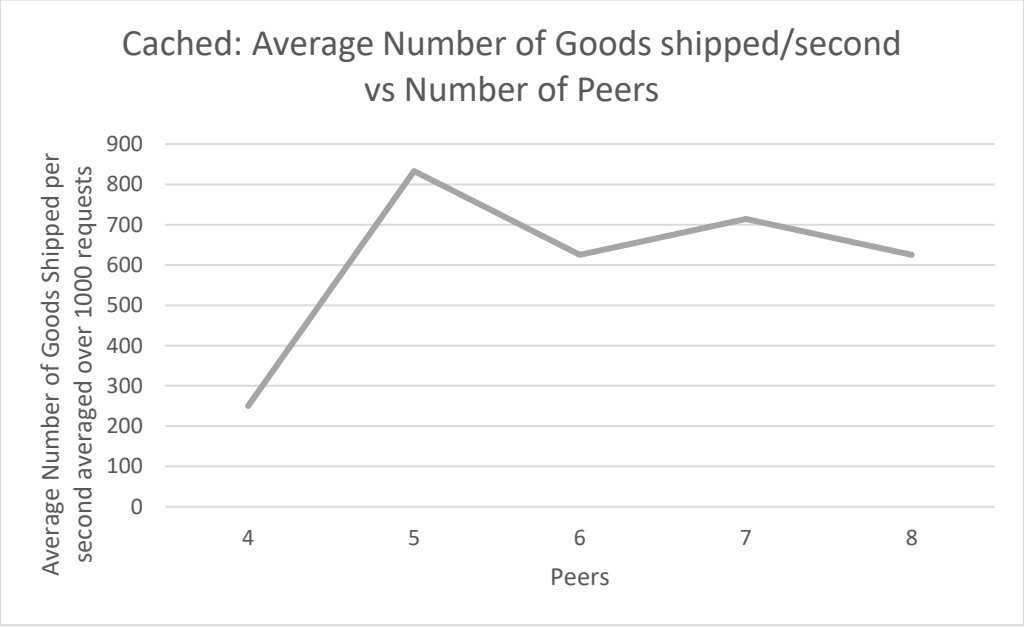
Similarly the same tests cases have been implemented for cached network and those test cases are also provided. These test cases suffice the proof of implementation for the gaul and traders based market that we have established.

Experimental Results:

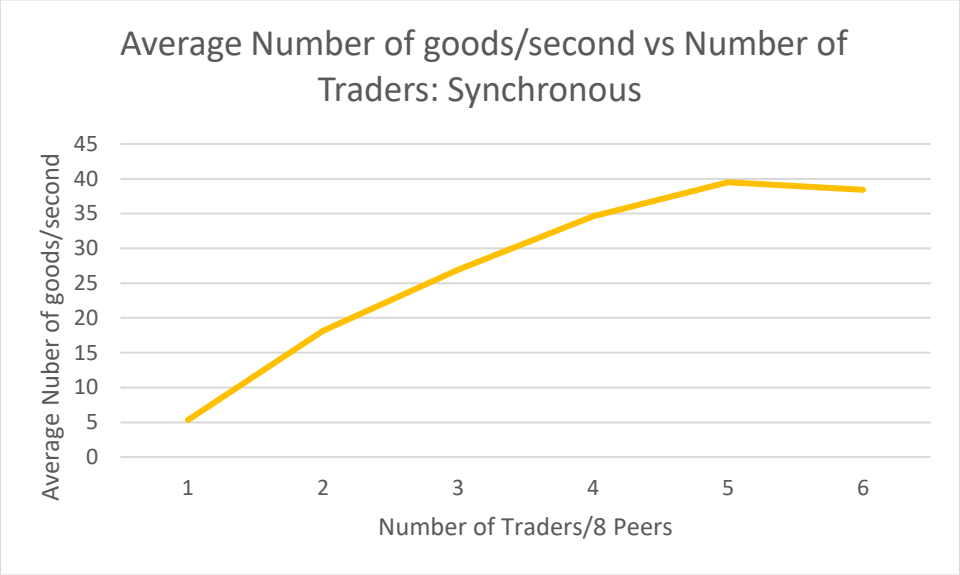
We have performed several experiments to measure the performance of the system and analyze the Performance of the system for cacheless and cache based approach.

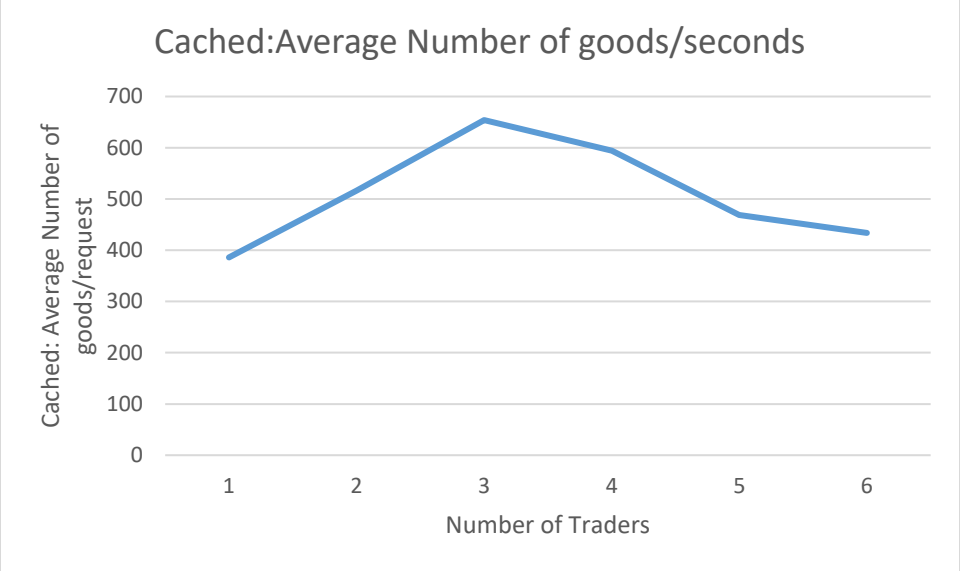
Result 1: Throughput: Average Number of Goods Shipped Cached vs Cacheless (Different number of peers but same number of traders)



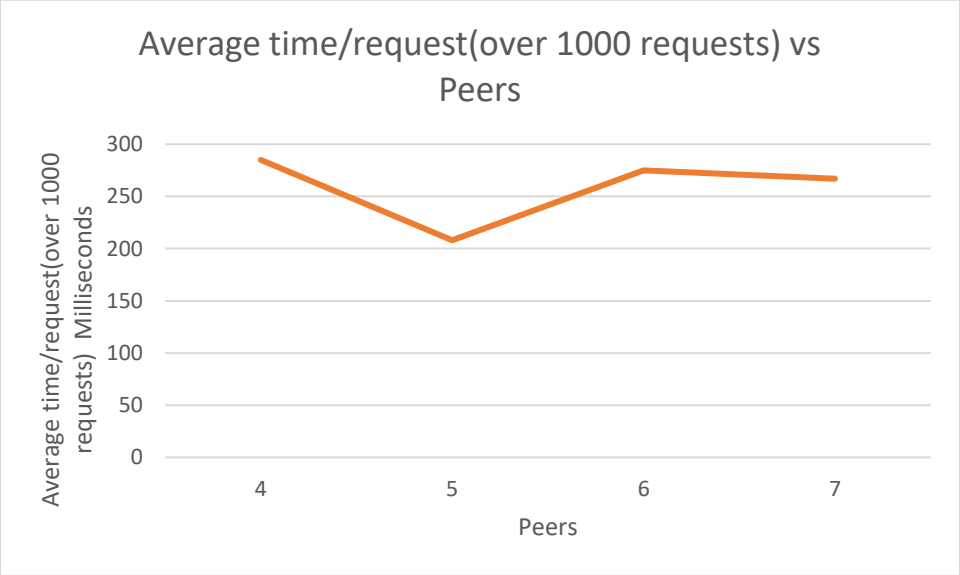


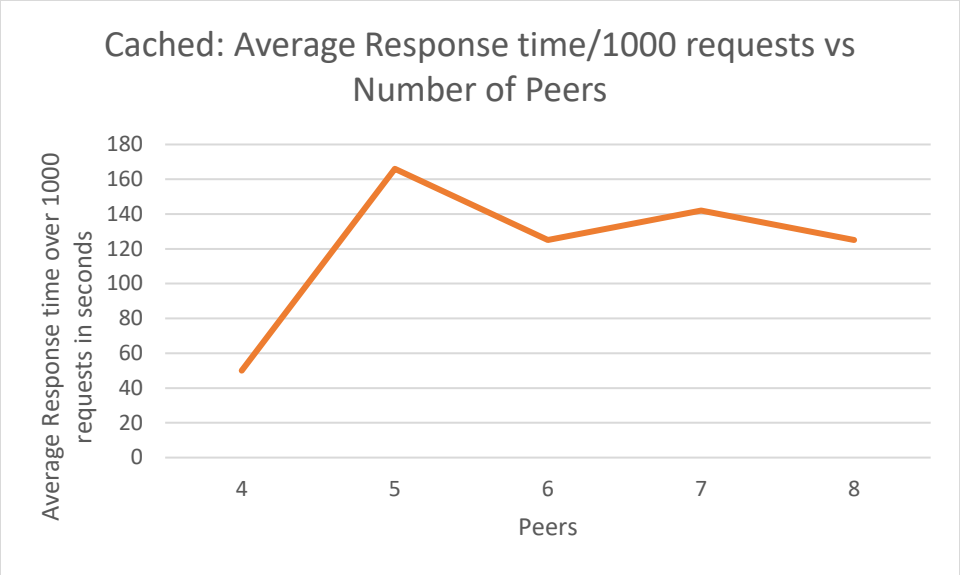
Result 2: Throughput: Average Number of Goods Shipped Cached vs Cacheless (Different number of traders but same number of peers)



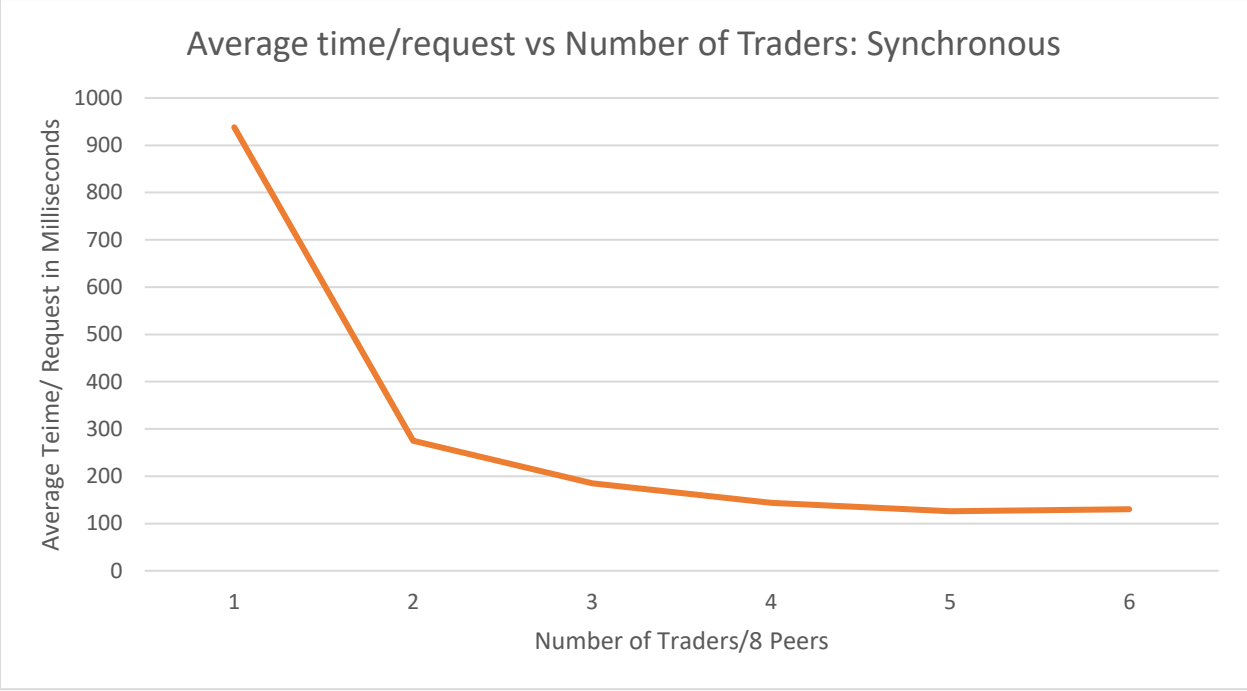


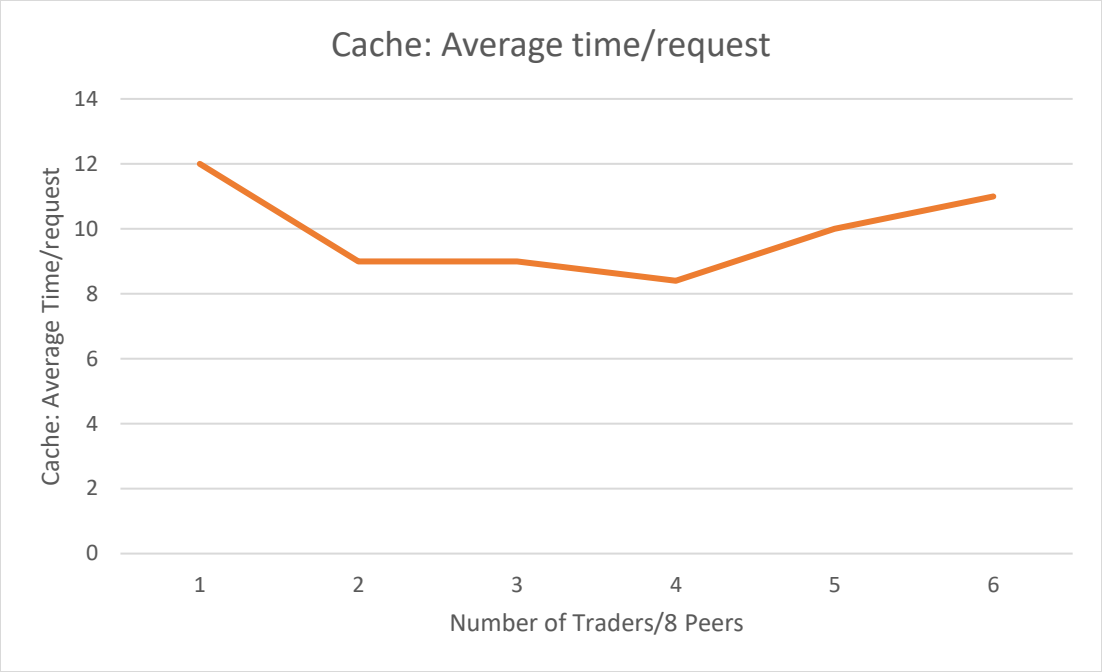
Result 3: Throughput Average response time (different number of peers same number or traders)





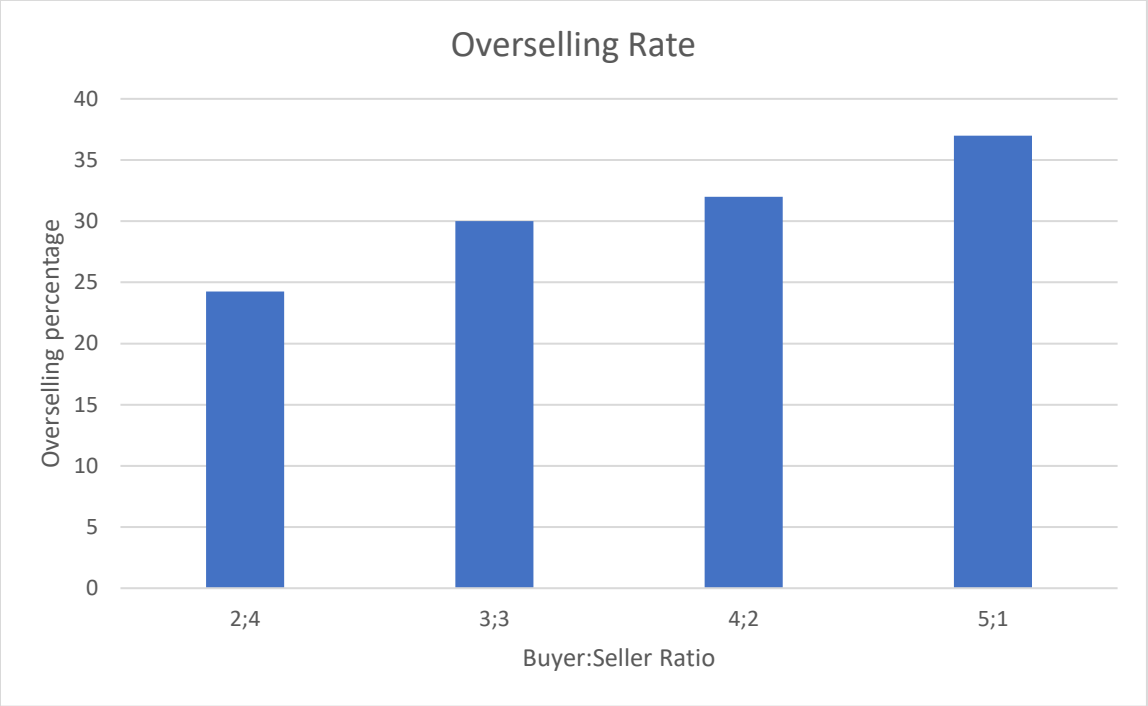
Result 4: Throughput Average response time (different number of traders same number or peers)





From the above results it is evident that the cached version has a much higher throughput than the cacheless version. Also it is important to note that as the number of traders increases, the throughput increases but plateaus after a few increments as the number of traders suffice the network traffic and further addition of traders might not work in favor of increasing throughput. For the synchronous mode we do not observe a significant growth in throughput with the number of traders since all transactions are blocking calls and several parallel requests on a blocking lock based service has almost no effect on the throughput.

Result 5: buyer/seller ratio vs Overselling incidents percentage wrt Number of Requests



In this experiment, the buyer seller ratio for 8 nodes consisting of different buyers and sellers as mentioned above but fixed number of Traders has been taken into consideration for comparing the overselling rate as described in the lab requirements. We can clearly ovserve that as the ratio increases the overselling rate also increases this follows the logically that as less sellers are available the inventory is quickly sold out among the buyers and the overselling rate increases.

Result 6: System throughput before and after fault tolerance.

Over 1000 requests were taken into consideration for calculating the throughput before and after fault tolerance

With 2 traders and 3 buyers and 3 sellers the throughput before fault tolerance kicked in was

120 milliseconds/request and 650 goods/second

After fault tolerance the response time was

210 milliseconds/request and 380 goods/second.

After analyzing the experiments it can be postulated that the gauls should use the cache based approach as the throughput of the system increases over 10 fold for certain configurations, the overselling rate although a bit worrisome can be avoided by using a consistency model like sequential consistency model which is better for such an approach and individuals are connected to each other especially for election purposes can easily agree on some total ordering like read after right and consistency can be maintained across traders. Sequential consistency can prove to be a better alternative rather than the causal consistency approach followed in this system design.