Asterix and the Multi-Trader trouble
Design Document

Group:

Prachiti Parkar
Sai Pranav Kurly

Design Description

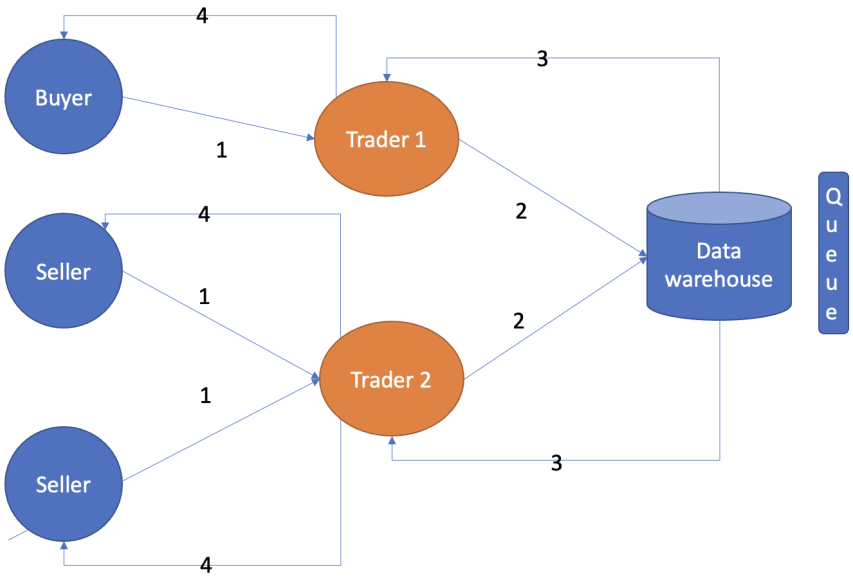
Buyers, sellers, data warehouse and the traders are peers, The traders can contact the data warehouse and the buyers/sellers can contact the traders.

Design Decisions:

1. Traders = 2
2. Peers >= 4 [Since we need min 1 buyer and 2 seller and we have implemented the approach as per min 2 traders]
3. Ng = 5 and Tg = 5 [usually except for experiments]
4. Seller - Initially starts with 0 quantity of products in its inventory and then goes on to restock Ng quantity every Tg seconds with the same product. If it sells more than 1 product, then it restocks every 1 product with Ng goods every Tg seconds.
5. Buyer - Only 1 unit of product bought per buy request.
6. The seller for the product which the buyer wants to buy always exists in the system. The quantity might become 0 and would need to be restocked but the seller would always be present.
7. In the cache-based approach, the data warehouse informs the trader only in the case of overselling and if the trader gets this message, it informs the buyer later.
8. In case of fault tolerance only 2 traders are considered and we have simulated that Trader 1 would fail and Trader 2 would then take up its responsibility.

Synchronized approach

1 -> Buyer/Seller makes a buy/sell request to the trader.
2 -> Trader contacts the Data warehouse, making addProductRequest() in case of sell request [always success] and removeProductRequest() in case of buy request. [If Data warehouse has enough products, it completes the transaction, Else, it does not]
3 -> The data warehouse informs trader whether the transaction was successful or not (result)
4 -> The Trader informs buyer/seller the result



Synchronous - A simulation of 1 buyer and 2 sellers with 2 traders

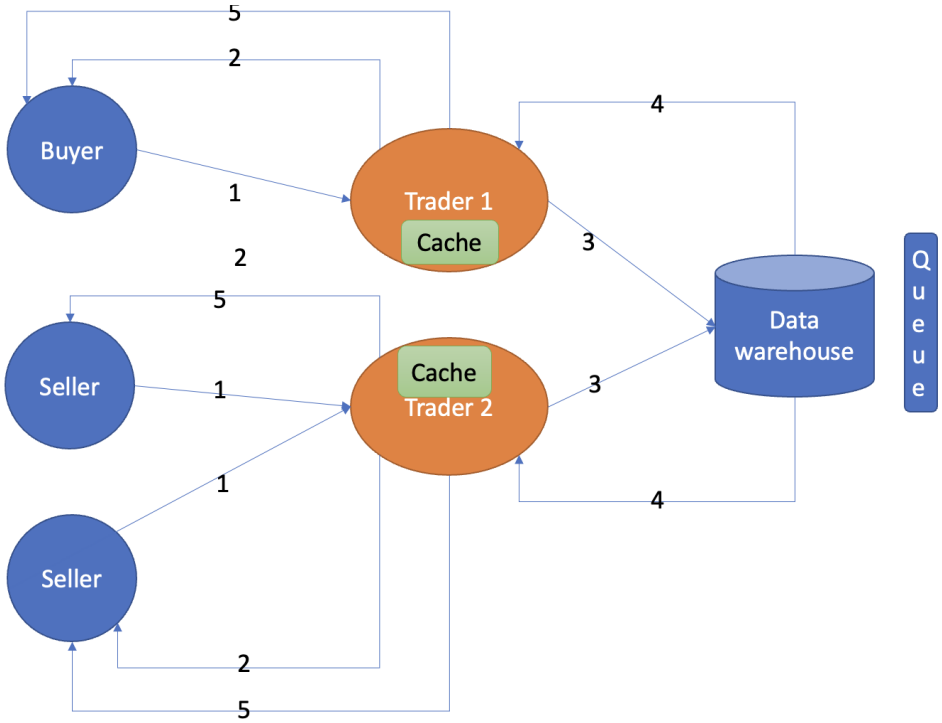
Approach using inventory information caches

We have implemented **Causal consistency**.
If TS(a) < TS(b), then all writes must be processed in that order for all processes. Concurrent writes may be seen in different order on different processes.

Basic approach
Any update request (write by definition) - buy or sell

1. Before any add or remove, update cache wrt from the data warehouse (TS(a) < TS(b) - satisfied)
2. Then check with cache (Ignore the possibility that if somewhere, some update is happening at similar time, you don't need to order that) [only in case of buy]
3. Then go update the warehouse again with only the changes

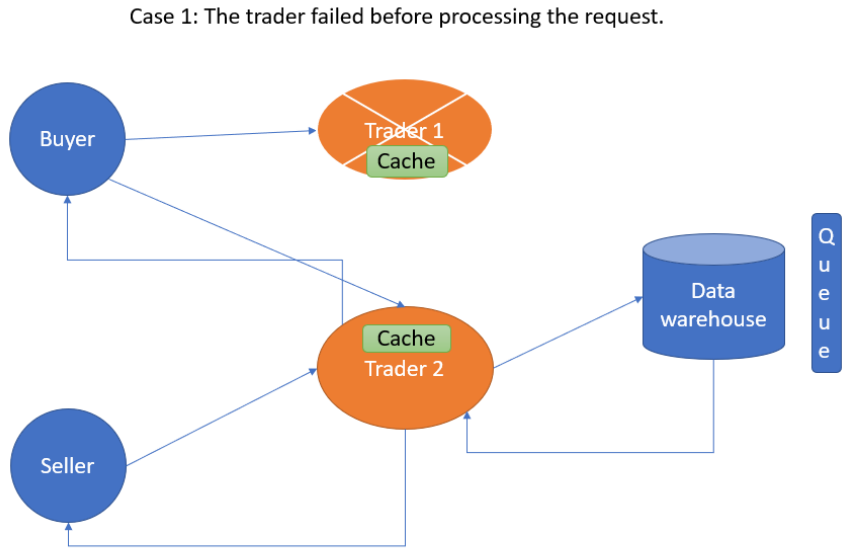
1 -> Buyer/Seller makes a buy/sell request to the trader.
If sell request – the trader adds it to the cache [always successful], If buy request – the trader checks if it contains the item, if it does, it completes the request else it does not.
2 -> If transaction is successful, the trader informs buyer/seller if it is successful or not.
Note: [3,4,5 will occur only if transaction is successful]
3 -> Trader contacts the Data warehouse, making addProductRequest() in case of sell request [always success] and removeProductRequest() in case of buy request. [If Data warehouse has enough products, it completes the transaction, Else, it does not]
4 -> The data warehouse informs trader if the transaction was not successful
5 -> The Trader informs buyer/seller the result (if received) from the data warehouse



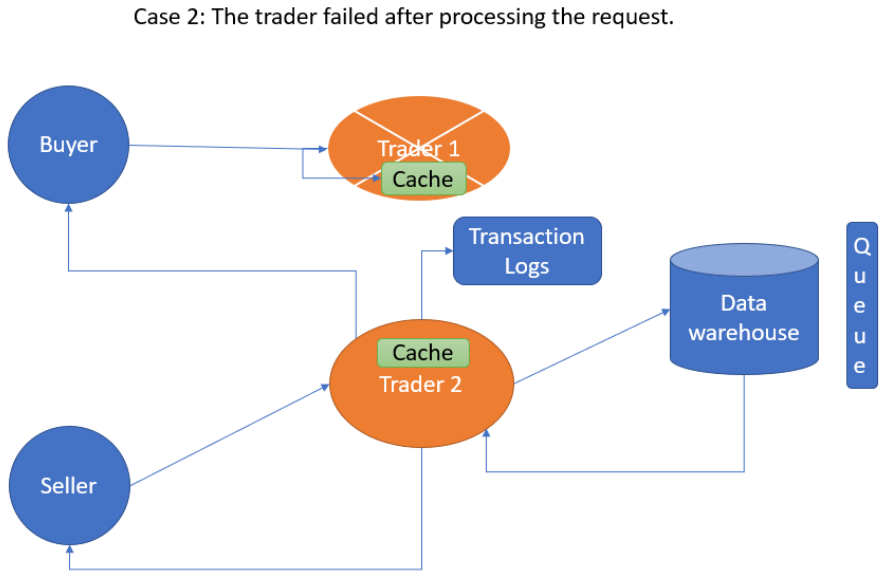
In case of oversell (i.e data warehouse does not have the item already sold by the trader), the data warehouse informs the trader and the trader informs the buyer. The buyer functions as it was functioning before and ignores the request which resulted in an oversell.

Fault tolerance

Case 1: The trader failed before processing the request. Here, another request is made to the new trader trader and the request is processed and transaction completed.



Case 2: The trader failed after processing the request and removing the product from cache but failed to respond back to the buyer and update the data warehouse. Here, the new Trader, trader 2 takes over by checking transaction logs (transactions.csv) and informs the buyer that the transaction is successful and also updates the data warehouse after the transaction is done.



Trade Offs

1. Since we are using HTTP, it is not secure. The server's public key might not be registered with the CA i.e no SSL certificate thus, there is no guarantee that the transmission is secure. If we were also supposed to authenticate payments for Bazaar, the peer servers could have been a victim of the Man-In-The-Middle attack.
2. Total ordering of transactions can be done by using clocks. This may result in adoption of a stricter consistency model than causal consistency and thus due to the overheads, the throughput would be lower than our approach.

Possible Improvements

1. Trader fails after request is processed and informs the seller that good is sold
2. Separate threads could have also been used to perform helper functions like lookup or registerProduct
3. If all buyers try to buy a product which isn't present in the system i.e. no seller has it, the system will reach a deadlock stage.
4. The data warehouse should be able to tell the new trader if the past trader failed after passing the request to the data warehouse.

How to run

Note: Peer_0.txt is the data warehouse
Usually Peer_1.txt and Peer_2.txt are traders
And rest are buyers or sellers

1. python3 peer.py 'Synchronous'
2. python3 peer.py 'Cached'
3. python3 test.py 'Cached' #testNo
4. python3 fault_tolerance.py 'Cached' 'T'

Tests

Please use test.py to test the below

1. If less than four peers is passed as input (at least one buyer,seller and 2 traders):

```
OMPSCI 677 - Distributed Systems/Lab/Lab-1/Cloned/CS677/Lab3$ /mnt/c/Users/SaiPranav/anaconda3/python.exe t
est.py 'Cached' 1
Cached marketplace is live! Check Peer_X.txt for logging!

Less than 4 peers passed!
```

2. If there is a least one buyer:

```
pranav@Pranav-Sai-PC:/mnt/c/Users/SaiPranav/OneDrive - Cleantech Energy Corporation Limited/Desktop/UMass/C
OMPSCI 677 - Distributed Systems/Lab/Lab-1/Cloned/CS677/Lab3$ /mnt/c/Users/SaiPranav/anaconda3/python.exe t
est.py 'Cached' 2
[1, 2]
Cached marketplace is live! Check Peer_X.txt for logging!

Enter atleast 1 buyer and seller!
```

3. If there is a least one seller:

```
pranav@Pranav-Sai-PC:/mnt/c/Users/SaiPranav/OneDrive - Cleantech Energy Corporation Limited/Desktop/UMass/C
OMPSCI 677 - Distributed Systems/Lab/Lab-1/Cloned/CS677/Lab3$ /mnt/c/Users/SaiPranav/anaconda3/python.exe t
est.py 'Cached' 3
[1, 2]
Cached marketplace is live! Check Peer_X.txt for logging!

Enter atleast 1 buyer and seller!
```

4. Seller continues to restock every 5 seconds:
   Below we can see that Peer 4 is a seller and restocks every 5 seconds.

Peer_4.txt

```
4 restocking Boar by 5 more
2022-12-11 16:04:15.415449 4 sold an item.
4 restocking Salt by 5 more
2022-12-11 16:04:29.493884 4 sold an item.
2022-12-11 16:04:36.535300 4 sold an item.
4 restocking Boar by 5 more
2022-12-11 16:04:39.575609 4 sold an item.
2022-12-11 16:04:39.776660 4 sold an item.
2022-12-11 16:04:48.676731 4 sold an item.
2022-12-11 16:04:49.868694 4 sold an item.
4 restocking Salt by 5 more
2022-12-11 16:04:53.709760 4 sold an item.
2022-12-11 16:04:56.916773 4 sold an item.
2022-12-11 16:05:03.978548 4 sold an item.
2022-12-11 16:05:06.782689 4 sold an item.
4 restocking Boar by 5 more
2022-12-11 16:05:11.057561 4 sold an item.
2022-12-11 16:05:11.830138 4 sold an item.
2022-12-11 16:05:14.094424 4 sold an item.
4 restocking Salt by 5 more
```

5. If good not present
   Below we see that the request is not processed be the trader since the good is not present with it

Peer_2.txt
```
2022-12-11 16:13:56.904053 Added to trader 2 cache Product:  Boar Quantity:  5
2 Item is not present!
2022-12-11 16:14:07.876577 Added to trader 2 cache Product:  Boar Quantity:  5
2 Item is not present!
2022-12-11 16:14:11.957090 Added to trader 2 cache Product:  Boar Quantity:  5
2 Item is not present!
2 Item is not present!
```

6. Overselling [Cache-based]
   Data warehouse does not have the item present, and realizes that the trader did an oversell. It informs the trader.

Peer_0.txt
```
2022-12-11 19:10:03.343073  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:07.385065  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:10.406416  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:16.389552  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:18.406638  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:18.441217  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:19.398454  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:19.402848  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:20.430823  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:22.410991  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:22.465782  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:23.417988  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:24.423686  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:24.430296  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:25.457260  Item present. Informing Trader. {'4_Boar': {'seller': {'peerId': 4, 'hos
2022-12-11 19:10:26.434051  Oversell! Item not present. Informing Trader.1
```

The trader receives the message from the data warehouse that it did an oversell

Peer_1.txt
```
2022-12-11 19:10:03.354430 Added to trader 1 cache Product:  Boar Quantity:  5
2022-12-11 19:10:07.378939 Product Boar of buyer 5 sold to seller 4
1 Item is not present!
2022-12-11 19:10:14.358486 Added to trader 1 cache Product:  Salt Quantity:  5
1 Item is not present!
1 Item is not present!
2022-12-11 19:10:18.373263 Added to trader 1 cache Product:  Fish Quantity:  5
2022-12-11 19:10:18.435826 Product Fish of buyer 5 sold to seller 6
2022-12-11 19:10:19.390387 Product Fish of buyer 9 sold to seller 6
2022-12-11 19:10:19.396924 Product Salt of buyer 8 sold to seller 4
2022-12-11 19:10:20.422498 Product Fish of buyer 10 sold to seller 6
2022-12-11 19:10:22.402952 Product Boar of buyer 7 sold to seller 4
2022-12-11 19:10:22.457040 Product Fish of buyer 5 sold to seller 6
2022-12-11 19:10:26.426717 Product Fish of buyer 7 sold to seller 6
2022-12-11 19:10:26.434327  Message received from data warehouse, I did an oversell!
```

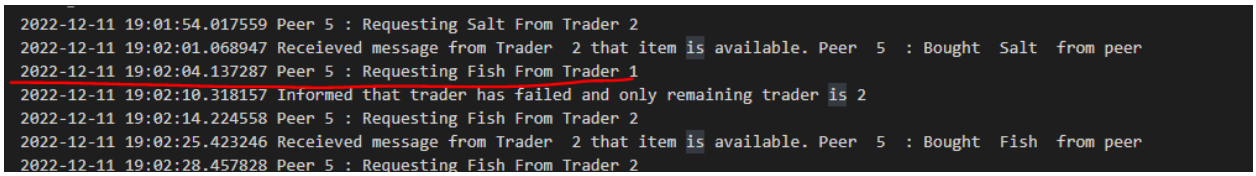
Please use fault_tolerance.py to test the below

Note: In both cases it is assumed that the database server is running and the requests are sent to the database server before failure.

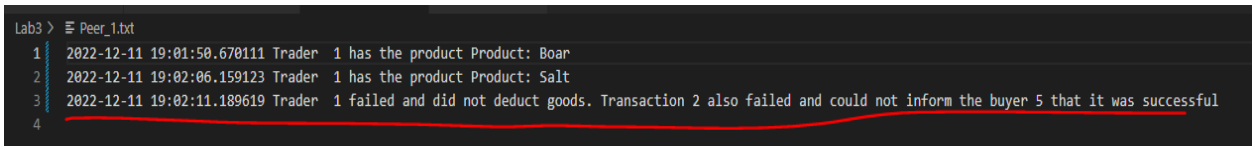
1. Trader fails when request is sent to trader that failed and request is not processed at all

   Command to test: python3 fault_tolerance.py 'Cached' 'T'

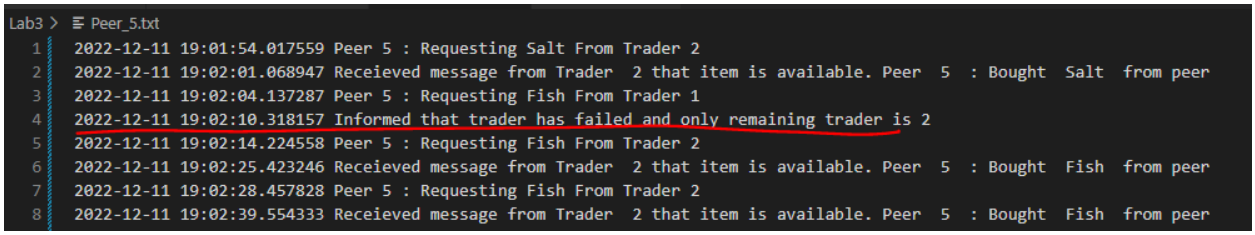
   Initially peer 5 sends a buy request to Peer 1 (Trader 1):



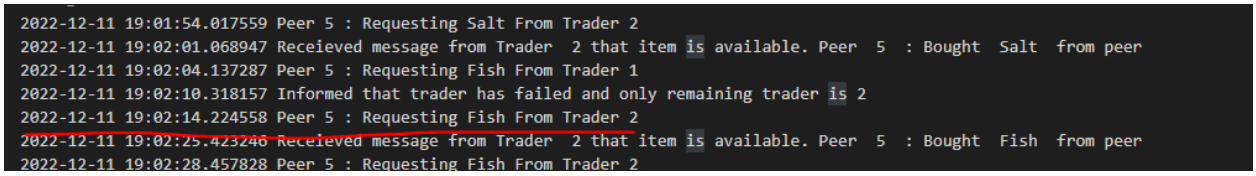
   Peer 1 (Trader 1) receives this request but fails immediately.



   Peer 5 is informed by Peer 2 (Trader 2) that Peer 1 is the only trader now::



   Peer 5 sends a buy request to Peer 2 (Trader 2) and receives the product:



2. Trader fails after request is processed and does not informs the buyer that good has been bought
   Command to test: python3 fault_tolerance.py 'Cached' 'F'

   Initially peer 3 sends a buy request to Peer 1 (Trader 1):

Lab3 > Peer_3.txt

```
2022-12-11 16:32:22.788214 Peer 3 : Requesting Fish From Trader 2
2022-12-11 16:32:29.841143 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:32.881377 Peer 3 : Requesting Fish From Trader 1
2022-12-11 16:32:36.932247 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:36.939228 Informed that trader has failed and only remaining trader is 2
2022-12-11 16:32:43.971976 Peer 3 : Requesting Fish From Trader 2
2022-12-11 16:32:51.032728 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
```

Peer 1 (Trader 1) receives this request and processes it. However, after processing the request it fails for some reason:

Lab3 > Peer_1.txt

```
2022-12-11 16:32:28.860719 Trader  1 has the product Product: Salt
2022-12-11 16:32:35.926679 Trader  1 has the product Product: Salt
2022-12-11 16:32:40.937519 Trader  1 failed but deducted goods from the cache. The transaction 2  failed and could not inform the buyer 3 that it was successful
```

Peer 2 (Trader 2) finds out that Peer 1 (Trader 1) is down because it is not responding to the heartbeat and decides to forward the request itself.

Lab3 > Peer_2.txt

```
2022-12-11 16:32:24.811078 Trader  2 has the product Product: Boar
2022-12-11 16:32:36.927573 Trader  2  informed the buyer  3  that transaction  2 is successful
2022-12-11 16:32:38.943764 Trader  2 has the product Product: Salt
2022-12-11 16:32:46.022398 Trader  2 has the product Product: Salt
```

We can also see this in transactions.csv. Transaction 2 from Peer 3 was forwarded again:

	A	B	C	D	E	F	G
1		tradeCoun	traderId	productNa	buyerId	completed	
2	0	1	2	Fish	3	FALSE	
3	0	1	1	Salt	5	FALSE	
4	0	1	2	Fish	3	TRUE	
5	0	1	1	Salt	5	TRUE	
6	0	2	1	Fish	3	FALSE	
7	0	2	1	Fish	3	FORWARD	
8	0	2	2	Fish	5	FALSE	
9	0	2	2	Fish	5	TRUE	
10	0	3	2	Fish	3	FALSE	
11	0	3	2	Fish	3	TRUE	

We can also see in Peer 3's logs that the request was sent to Peer 1 but Peer 3 informed it later.

```
2022-12-11 16:32:22.788214 Peer 3 : Requesting Fish From Trader 2
2022-12-11 16:32:29.841143 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:32.881377 Peer 3 : Requesting Fish From Trader 1
2022-12-11 16:32:36.932247 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:36.939228 Informed that trader has failed and only remaining trader is 2
```

Finally, the data warehouse is also informed and the other nodes are then informed by Peer 2 (Other trader) that the first trader has failed. The 2nd trader i.e. Peer 2 informs all

other peers in the network. Below is the log of one of the peer:

```
2022-12-11 16:32:22.788214 Peer 3 : Requesting Fish From Trader 2
2022-12-11 16:32:29.841143 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:32.881377 Peer 3 : Requesting Fish From Trader 1
2022-12-11 16:32:36.932247 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:36.939228 Informed that trader has failed and only remaining trader is 2
2022-12-11 16:32:43.971976 Peer 3 : Requesting Fish From Trader 2
2022-12-11 16:32:51.032728 Receieved message from Trader  2 that item is available. Peer  3  : Bought  Fish  from peer
2022-12-11 16:32:54.087285 Peer 3 : Requesting Fish From Trader 2
```

The requests are now only sent to Peer 2 since it is the sole trader in the network.

Experimental Results

1. Comparing throughput of the cache-less and cache-based approaches

Throughput is the number of goods shipped successfully sent per second.

Hyperparameters like No. of buyers = 6, sellers = 2, traders = 2 and Ng=Tg=5 were kept the same so that we can compare the results accurately.

Cache-less approach:

```
prachitiparkar@prachiti Lab3 % python3 peer3.py Synchronous
Synchronous marketplace is live! Check Peer_X.txt for logging!

Average throughput of system is 0: 0.33189746744641774 (req/sec)
```

Cache-based approach:

```
prachitiparkar@prachiti Lab3 % python3 peer3.py Async
Cached marketplace is live! Check Peer_X.txt for logging!

Average throughput of system is 0: 0.5552339948720587 (req/sec)
```

Analysis: Cache-based approach has a greater throughput than cache-less approach. We expected this because in the cache-less approach all requests are sent to the data warehouse whereas in the cache-based approach, many requests are denied by the caches itself and therefore, less requests (expected to be successful by the traders) reach the data warehouse. To achieve this throughput, the cache-based approach suffers from overselling and underselling. This is the trade off of the cache-based approach.

Performance gains of cache based approach

- Increasing throughput: Same hyperparameters but buyers = 5 and sellers = 3

```
prachitiparkar@prachiti Lab3 % python3 peer3.py Asynchronous
Cached marketplace is live! Check Peer_X.txt for logging!

Average throughput of system is 0: 0.7197227104161344 (req/sec)
```

The throughput is even higher in this case. We expected this because there are more sellers and goods are getting restocked comparatively faster than the buy requests. The throughput would increase as we decrease the buyer seller ratio i.e increase the number of sellers keeping the number of buyers the same or decrease number of buyers keeping the number of sellers the same.

- Decreasing throughput: Same hyperparameters but buyers = 6 and sellers = 1

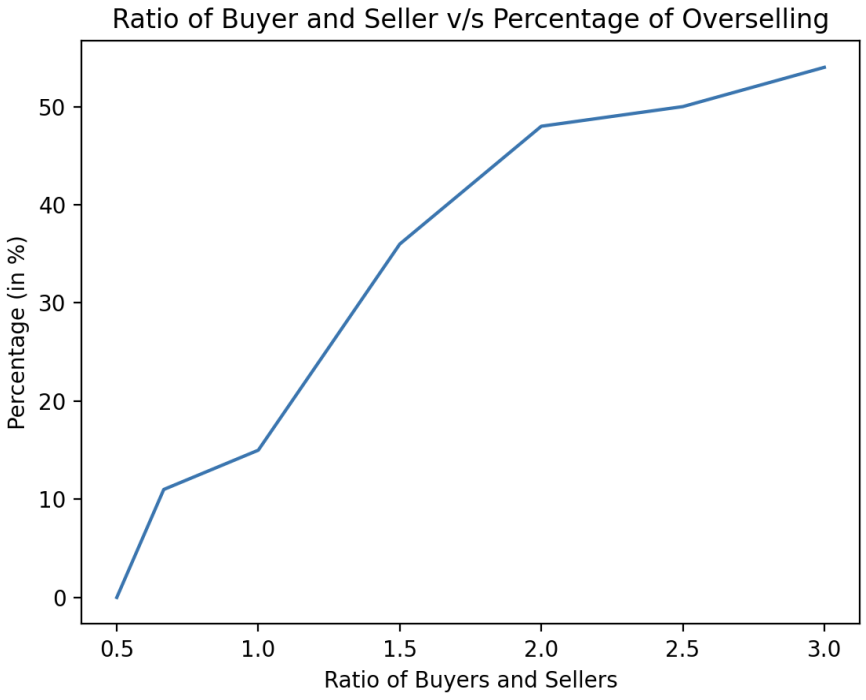
```
prachitiparkar@prachiti Lab3 % python3 peer3.py Asynchronous
Cached marketplace is live! Check Peer_X.txt for logging!

Average throughput of system is 0: 0.3217341822469903 (req/sec)
```

We can see that this is almost similar to the cache-less approach since there is only 1 seller in the system, goods get restocked comparatively slower than the number of buy requests.

2. Rate of over-selling for your cache-based implementation

We fixed the Ng = 5 and Tg = 5 for this simulation and the number of traders = 2. We ran this experiment for various buyer and seller ratio for approximately 2 minutes [was decided after trial and error].

Buyers	1	2	2	3	4	5	6
Sellers	2	3	2	2	2	2	2
Buyers/Sellers	½ = 0.5	⅔ = 0.666	2/2 = 1	3/2 = 1.5	4/2 = 2	5/2 = 2.5	3
Percentage	0	11	15	36	48	50	54



How to run: python3 overselling.py

Insights/Analysis:

We realized that as the buyer seller ratio increases, the rate of overselling also increases. The buyer seller ratio is directly proportional if we keep the Ng = Tg = 5 (i.e constant), traders = 2 and run the experiment for a specific period of time for all. We kept the number of sellers the same (in some cases) so that we can derive the insights efficiently. This helped us understand that we get this plot since there are less number of goods available in the market (as per the cache - outdated information) and the buyers want the same product.

Note: Please uncomment the line of code on the fifth last line if you wish to see the overselling rate. It will be printed on the console.

```
def removeProduct(self,seller,productName,traderPeerId):
    if self.programType == 'Synchronous':
        tot_prod = 0
        for i in self.tradeList:
            if self.tradeList[i]['productName'] == productName:
                    tot_prod+=self.tradeList[i]['productCount']
        sellerList = []
        for peerId,sellerInfo in self.tradeList.items():
            if sellerInfo["productName"] == productName:
                sellerList.append(sellerInfo["seller"])
        if len(sellerList) > 0 and tot_prod>0:
            with open('Peer_'+str(self.peerId)+".txt", "a") as f:
                    f.write(" ".join([str(datetime.datetime.now()),' Item present. Informing Trader.','\n']))
            seller = sellerList[0]
            self.tradeList[str(seller['peerId'])+'_'+str(seller['productName'])]["productCount"]  = self.tradeList[str(seller['peerId'])+'_'+str(seller['productN
            self.calculateThroughput(self.timeStart,datetime.datetime.now())
            return [1,seller]
        else:
            with open('Peer_'+str(self.peerId)+".txt", "a") as f:
                    f.write(" ".join([str(datetime.datetime.now()),' Item not present. Informing Trader.' + str(traderPeerId),'\n']))
            return [-1,seller]
    else:
        tot_prod = 0
        for i in self.tradeList:
            if self.tradeList[i]['productName'] == productName:
                    tot_prod+=self.tradeList[i]['productCount']
        sellerList = []
        for peerId,sellerInfo in self.tradeList.items():
            if sellerInfo["productName"] == productName:
                sellerList.append(sellerInfo["seller"])
        if len(sellerList) > 0 and tot_prod > 0:
            with open('Peer_'+str(self.peerId)+".txt", "a") as f:
                    f.write(" ".join([str(datetime.datetime.now()),' Item present. Informing Trader.',str(self.tradeList),'\n']))
            seller = sellerList[0]
            self.tradeList[str(seller['peerId'])+'_'+str(seller['productName'])]["productCount"]  = self.tradeList[str(seller['peerId'])+'_'+str(seller['productN
            self.calculateThroughput(self.timeStart,datetime.datetime.now())
            return 1
        #check once
        else:
            self.oversellCount +=1
            #uncomment only if you want to see oversell percentage
            # print("Oversell! The incidence of over-selling as a percentage of total buy requests: (in %) ",(self.oversellCount/self.totalBuyRequest)*100)
            with open('Peer_'+str(self.peerId)+".txt", "a") as f:
                    f.write(" ".join([str(datetime.datetime.now()),' Oversell! Item not present. Informing Trader.' + str(traderPeerId),'\n']))
            with open('Peer_'+str(traderPeerId)+".txt", "a") as f:
                    f.write(" ".join([str(datetime.datetime.now()),' Message received from data warehouse, I did an oversell!','\n']))
```

3. Throughput of your system after a trader fails

```
prachitiparkar@prachiti Lab3 % python3 fault_tolerance.py "Cached" "T"
Cached marketplace is live! Check Peer_X.txt for logging!

Average throughput of system is 0: 0.05160566378352704 (req/sec)
Average throughput of system is 0: 0.07573521665611296 (req/sec)
Average throughput of system is 0: 0.1094486190174564 (req/sec)
Average throughput of system is 0: 0.1196030041404168 (req/sec)
Average throughput of system is 0: 0.14514926540827672 (req/sec)
Average throughput of system is 0: 0.14824155130632677 (req/sec)
Average throughput of system is 0: 0.16875265845705875 (req/sec)
Average throughput of system is 0: 0.16840865775436822 (req/sec)
Average throughput of system is 0: 0.18554782810700765 (req/sec)
Average throughput of system is 0: 0.18336656577288118 (req/sec)
Average throughput of system is 0: 0.17866275830513426 (req/sec)
Average throughput of system is 0: 0.17493745949377792 (req/sec)
Average throughput of system is 0: 0.17190013770920032 (req/sec)
Trader failed
Trader failed
Trader failed
Trader failed
Average throughput of system is 0: 0.16937011206929967 (req/sec)
Average throughput of system is 0: 0.16724951822940026 (req/sec)
Average throughput of system is 0: 0.16538520011671234 (req/sec)
Average throughput of system is 0: 0.1638021303411322 (req/sec)
Average throughput of system is 0: 0.16242971170061707 (req/sec)
Average throughput of system is 0: 0.1612320723619045 (req/sec)
Average throughput of system is 0: 0.16016339228627477 (req/sec)
Average throughput of system is 0: 0.15919320881771182 (req/sec)
Average throughput of system is 0: 0.15833719004343505 (req/sec)
Average throughput of system is 0: 0.1575662022962328 (req/sec)
Average throughput of system is 0: 0.156859763737301 (req/sec)
Average throughput of system is 0: 0.1562197080222413 (req/sec)
Average throughput of system is 0: 0.1556317211878306 (req/sec)
Average throughput of system is 0: 0.1550901251118932 (req/sec)
Average throughput of system is 0: 0.15458769061283364 (req/sec)
Average throughput of system is 0: 0.15411164610762396 (req/sec)
Average throughput of system is 0: 0.15368073505536559 (req/sec)
Average throughput of system is 0: 0.1532769797496981 (req/sec)
Average throughput of system is 0: 0.15290231870728047 (req/sec)
Average throughput of system is 0: 0.15255794460474945 (req/sec)
Average throughput of system is 0: 0.15222487966567297 (req/sec)
Average throughput of system is 0: 0.15192026894401806 (req/sec)
```

We can see that the throughput was higher when there were 2 traders and it decreased as the trader failed. This is expected since the requests were split between 2 traders and were handled comparatively faster than the approach with only 1 trader. According to us, the number of

traders will have a larger effect on the throughput if there are many traders initially and more than half of them start failing.

Conclusion: Gauls should use a cache-based approach since it leads to higher throughput even though it involves overselling and underselling. We can minimize overselling by decreasing the buyer seller ratio. We have used causal consistency but sequential consistency can provide better or similar output and will help to minimize underselling and overselling, but since it is stricter than causal it would lead to overheads and complicated implementation and might decrease the throughput.