DISTRIBUTED SYSTEMS ASSIGNMENT-4
DISTRIBUTED KV STORE
REPORT

SUBMITTED BY - RAHUL GUPTA

INTRODUCTION

In this project, we will implement a distributed key-value store that offers many different consistency models. The main task is to then implement the different consistency schemes that use the underlying key-value store replicas.

DESIGN DETAILS

MESSAGING PROTOCOL

For communication between client and server, a tcp socket is bound at the server to which the client can connect to.

For communication between the servers, I have used ZeroMQ messaging service(pyzmq) with the publish subscribe model. Each individual server has a publisher socket bound to its corresponding port and a subscriber socket that is connected to the other 2 processes of the server that are bound to different ports. This is done using the config file and server id's.

EVENTUAL CONSISTENCY

The main concept behind eventual consistency according to this implementation is that Writes(set messages) are broadcasted asynchronously and both read and write operations are returned to the client immediately so that the distributed system becomes consistent eventual after some time.

Required Files in the Directory

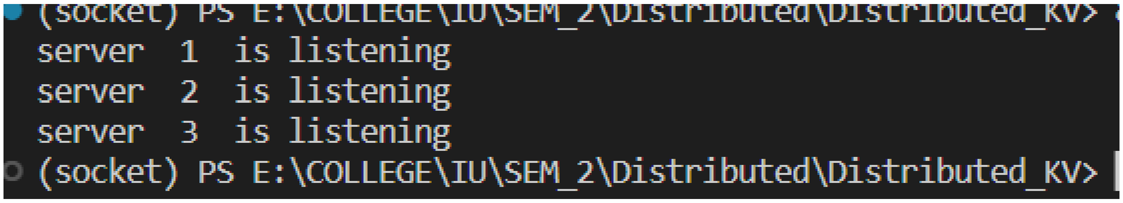
- config.yml - Common file to get ports, IP's and consistency model, set the model parameter to - 'eventual'.
- starter.py - Common to all consistency models, runs the required server processes based on consistency model
- eventual_server.py -Individual server file with eventual consistency model that is run by starter
- Client.py - python file for the client(common to all)
- Store{i}.csv - 3 store files acting as storage for 3 different processes.
- test_client.py
- client_starter.py

Working Details -

- **Config -** change the model parameter to *eventual.*
- Run starter.py
- Run client.py on different terminals to perform the test cases.

Design -

- Starter.py starts server processes passing the server id and the particular store file to be used by that server, example - server with id = 1 gets filepath of store1.csv. Wait included to avoid error of port reusage on multiple runs.
- Wait for terminal to show message for all 3 servers as shown below.



```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distributed_KV>
server  1  is listening
server  2  is listening
server  3  is listening
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distributed_KV>
```

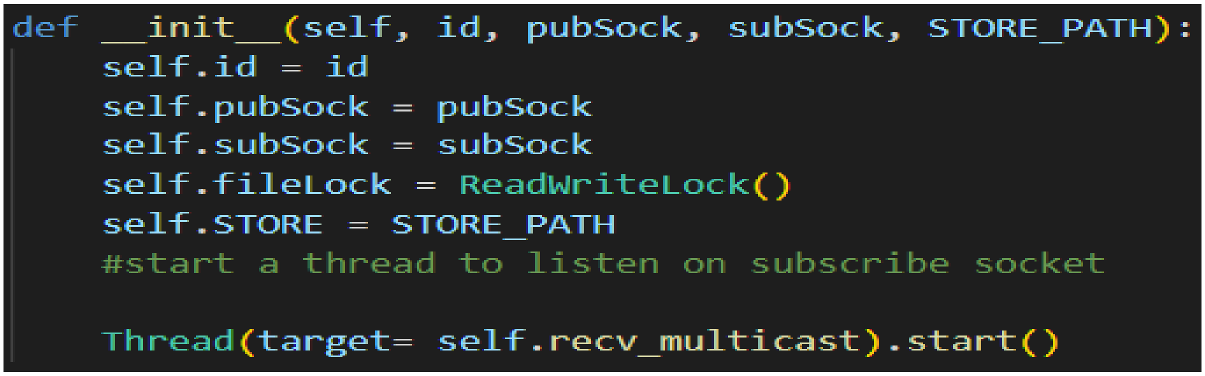
- Each server process store command line arguments - id and store path passed by starter, it reads the required ports for the particular server with the help of id and writes its own IP to config file.
- Bind publisher socket and connect subscriber socket to other 2 server processes
- An object of Store class is initialized which contains the server_id, store_path, and the socket objects along with other relevant details required by this server process.
- Next, a TCP socket is bound to a port for the server to which the client can connect to.
- I was facing problems with freeing the ports used by zmq so I created a def exit() function which takes an input and frees the ports. This function starts running before any client connection in a parallel thread.
- Now the server processes are running and client.py can be run to connect to any of ther server processes and perform get or set operations details to which are given ahead.

Starter Details-

We are building a distributed KV store with 3 servers. The starter is used to start the server processes independently.

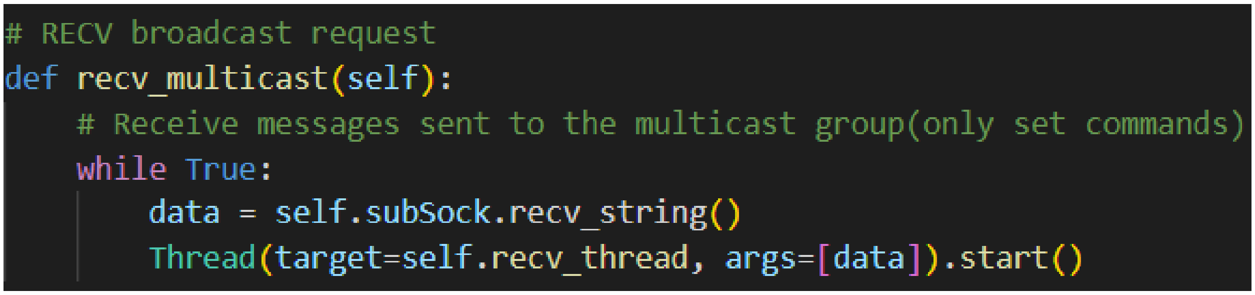
Store Details -

Each server process gets the id as command line, this id along with zmq pub-sub objects, a readwritelock variable and store path are initialized as class variables for this object. The init function is as given in the image below. We also require each process to parallely listen for published messages using subscriber socket object. This is done by starting the recv_multicast function on a new thread, details to which are given below.



```
def __init__(self, id, pubSock, subSock, STORE_PATH):
    self.id = id
    self.pubSock = pubSock
    self.subSock = subSock
    self.fileLock = ReadWriteLock()
    self.STORE = STORE_PATH
    #start a thread to listen on subscribe socket

    Thread(target= self.recv_multicast).start()
```

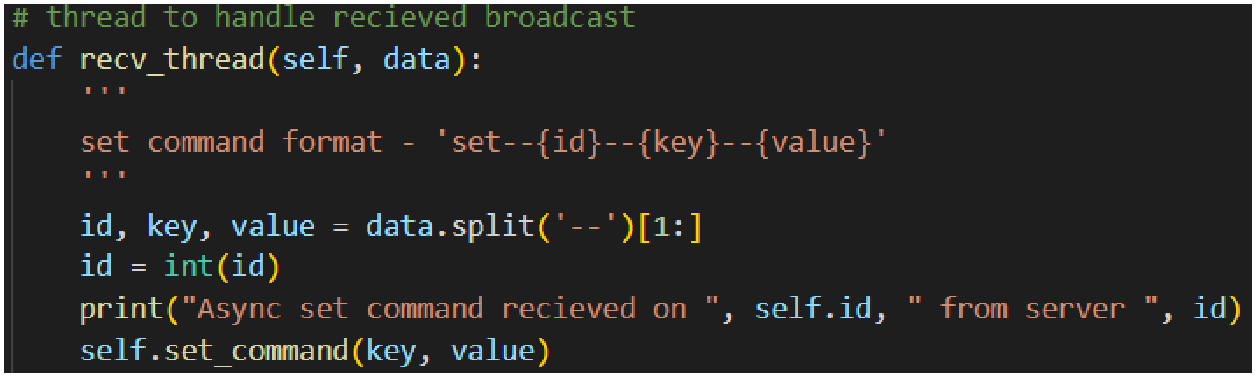
Function - recv_multicast



```
# RECV broadcast request
def recv_multicast(self):
    # Receive messages sent to the multicast group(only set commands)
    while True:
        data = self.subSock.recv_string()
        Thread(target=self.recv_thread, args=[data]).start()
```

The above function just listens for a broadcast message from any of the other servers.
On receiving a message it processes the same in recv_trhead function called on a new thread so that it can immediately go back to receiving messages in case of concurrent receives.

Function - recv_thread

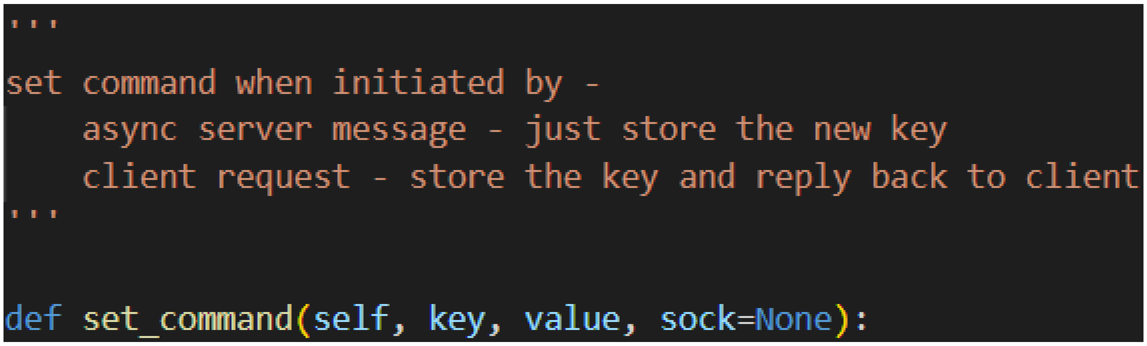


```
# thread to handle recieved broadcast
def recv_thread(self, data):
    '''
    set command format - 'set--{id}--{key}--{value}'
    '''
    id, key, value = data.split('--')[1:]
    id = int(id)
    print("Async set command recieved on ", self.id, " from server ", id)
    self.set_command(key, value)
```

The broadcast message received is processed according to the format in the image above, and the set_command function is called for this server process with the received key and value.

Function - set_command

This function is the same as of the simple KV store, it acquires a write lock (self.filelock.aqcuire_write()) and writes the key, value to the data store. There is an additional parameter in the function definition - sock.

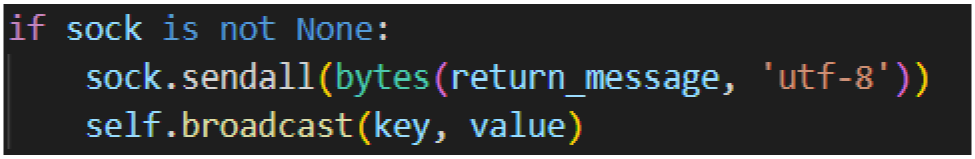


```
'''
set command when initiated by -
    async server message - just store the new key
    client request - store the key and reply back to client
'''

def set_command(self, key, value, sock=None):
```

Any server will receive a set command in the 2 following cases -

1) Directly from client, in which case after the set command an acknowledgement will have to be sent back to the client
2) An async broadcast in which only the key and value have to be stored.

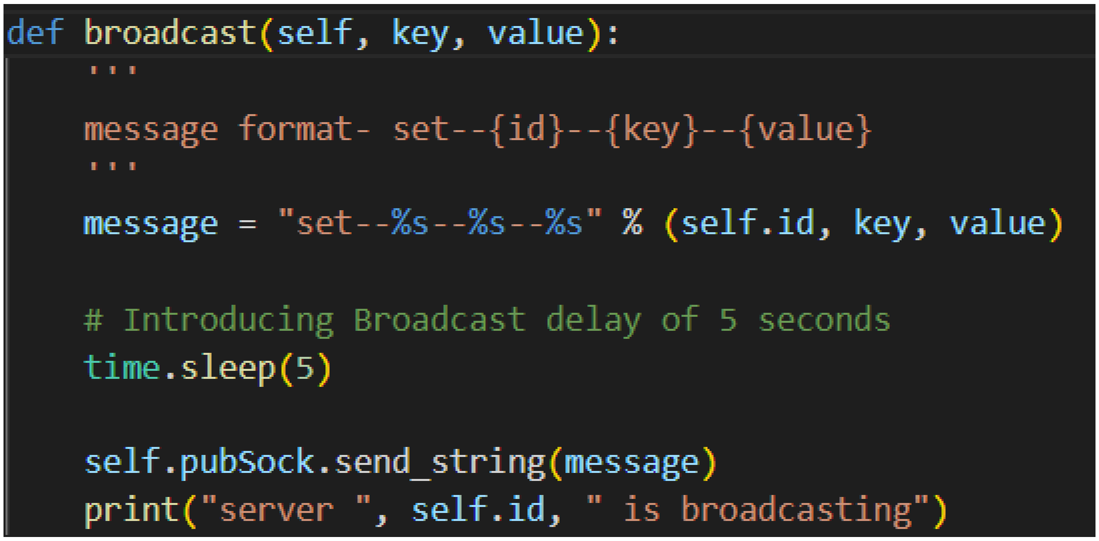
When the client request calls this set function, the sock parameter is passed as the client socket itself and in a broadcast message it remains None.



```
if sock is not None:
    sock.sendall(bytes(return_message, 'utf-8'))
    self.broadcast(key, value)
```

This if condition in the end of this function sends back acknowledgement to the client if client is connected to that particular server and initiates a broadcast message to the other servers to inform them of the write.

Function - broadcast



```
def broadcast(self, key, value):
    '''
    message format- set--{id}--{key}--{value}
    '''
    message = "set--%s--%s--%s" % (self.id, key, value)

    # Introducing Broadcast delay of 5 seconds
    time.sleep(5)

    self.pubSock.send_string(message)
    print("server ", self.id, " is broadcasting")
```

For the broadcast, the message is sent on the publisher socket with the format mentioned in the image above. A delay is also introduced for test cases and to simulate real life scenario.

Function - handle_clients

This function is called to handle client connection in a new thread from the server's TCP socket. It processes the client request and calls the required function based on the request type - set and get.

With the set command function here the client socket is also passed so that after write operation a broadcast is initiated and a reply is sent to the client.
For the get request, get_command function is called details to which are given below.

Function - get_command

This takes the key and the socket as input, as eventual consistency requires local instant reads, it just acquires a read lock over the store file, then reads and sends the value back to the client.

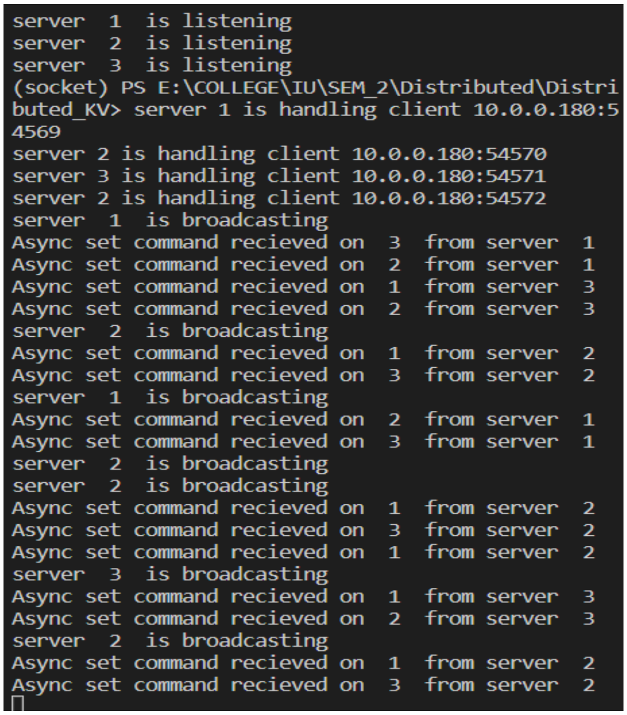
**** To make it eventually consistent in case of concurrent write requests after every 60 seconds server 1 sends its key value store to all other servers, using the broadcast so that all servers become consistent eventually.****

Client Details -

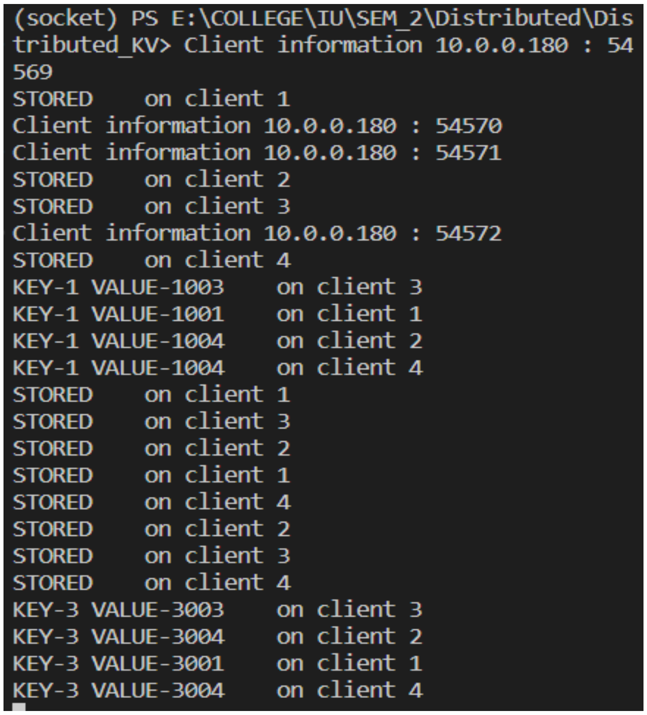
In client.py IP and port numbers of different servers are read from config files, and using a random number client connects to a server process. On connection, give the set {key} {value} and get {key} in the terminal to perform the required operation. Type exit to close the client.

Test Case 1-

Run client_starter.py with config file model parameter set to - *eventual*



```
server  1  is listening
server  2  is listening
server  3  is listening
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distri
buted_KV> server 1 is handling client 10.0.0.180:5
4569
server 2 is handling client 10.0.0.180:54570
server 3 is handling client 10.0.0.180:54571
server 2 is handling client 10.0.0.180:54572
server  1  is broadcasting
Async set command recieved on  3  from server  1
Async set command recieved on  2  from server  1
Async set command recieved on  1  from server  3
Async set command recieved on  2  from server  3
server  2  is broadcasting
Async set command recieved on  1  from server  2
Async set command recieved on  3  from server  2
server  1  is broadcasting
Async set command recieved on  2  from server  1
Async set command recieved on  3  from server  1
server  2  is broadcasting
server  2  is broadcasting
Async set command recieved on  1  from server  2
Async set command recieved on  3  from server  2
Async set command recieved on  1  from server  2
server  3  is broadcasting
Async set command recieved on  1  from server  3
Async set command recieved on  2  from server  3
server  2  is broadcasting
Async set command recieved on  1  from server  2
Async set command recieved on  3  from server  2
```



```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Dis
tributed_KV> Client information 10.0.0.180 : 54
569
STORED    on client 1
Client information 10.0.0.180 : 54570
Client information 10.0.0.180 : 54571
STORED    on client 2
STORED    on client 3
Client information 10.0.0.180 : 54572
STORED    on client 4
KEY-1 VALUE-1003    on client 3
KEY-1 VALUE-1001    on client 1
KEY-1 VALUE-1004    on client 2
KEY-1 VALUE-1004    on client 4
STORED    on client 1
STORED    on client 3
STORED    on client 2
STORED    on client 1
STORED    on client 4
STORED    on client 2
STORED    on client 3
STORED    on client 4
KEY-3 VALUE-3003    on client 3
KEY-3 VALUE-3004    on client 2
KEY-3 VALUE-3001    on client 1
KEY-3 VALUE-3004    on client 4
```

Server output client output

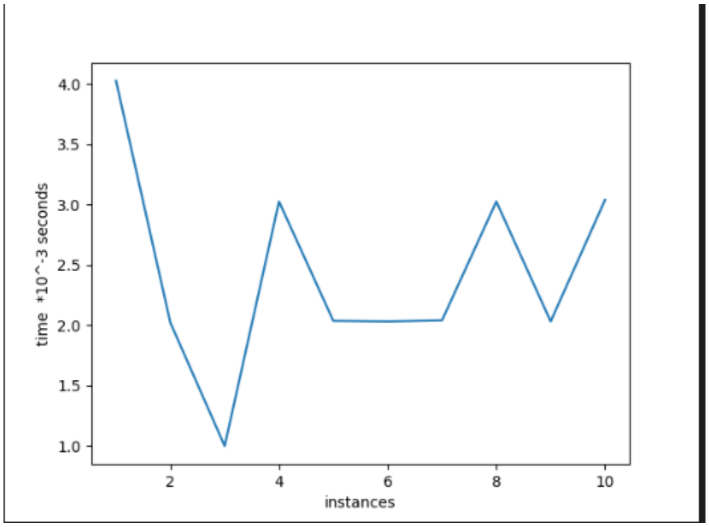
We can observer stale reads here, first set by client 1 (set 1 1001) is done and after that client 3(set 1 1003) sets the same key but the read gives us the previous value of 1001. Client 4 is the last one to set it and 2 and 4 return the value set by 4 - 1004.

More Test Cases

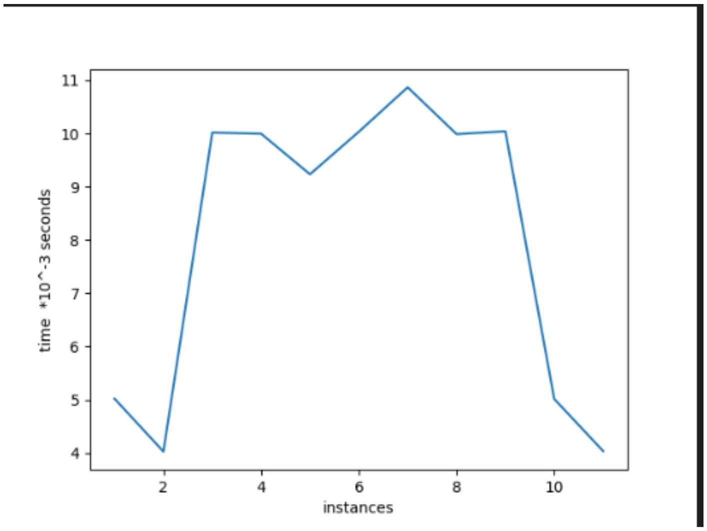
Run the client.py file in different terminals and give custom get set commands to see different scenarios.

Performance Evaluation -

X axis - number of specific request, 3 => 3rd request
Y axis - time taken to respond to client
Graph for 10 consecutive get requests



Graph for 10 consecutive set requests



SEQUENTIAL CONSISTENCY

The main concept behind sequential consistency according to this implementation is that Writes(set messages) are processed using total order multicast and Read operations are local and returned instantly.

Required Files in the Directory

- config.yml - Common file to get ports, IP's and consistency model, set the model parameter to - *sequential*.
- starter.py - Common to all consistency models, runs the required server processes based on consistency model
- sequential_server.py -Individual server file with sequential consistency model that is run by starter
- Client.py - python file for the client(common to all)
- Store{i}.csv - 3 store files acting as storage for 3 different processes.
- client_starter.py
- test_client.py

Working Details -

- **Config -** change the model parameter to *sequential.*
- Run starter.py

- Run client.py on different terminals to perform the test cases.

Design -

- The design and setup is the same as given in eventual consistency and it is same for other consistency models as well

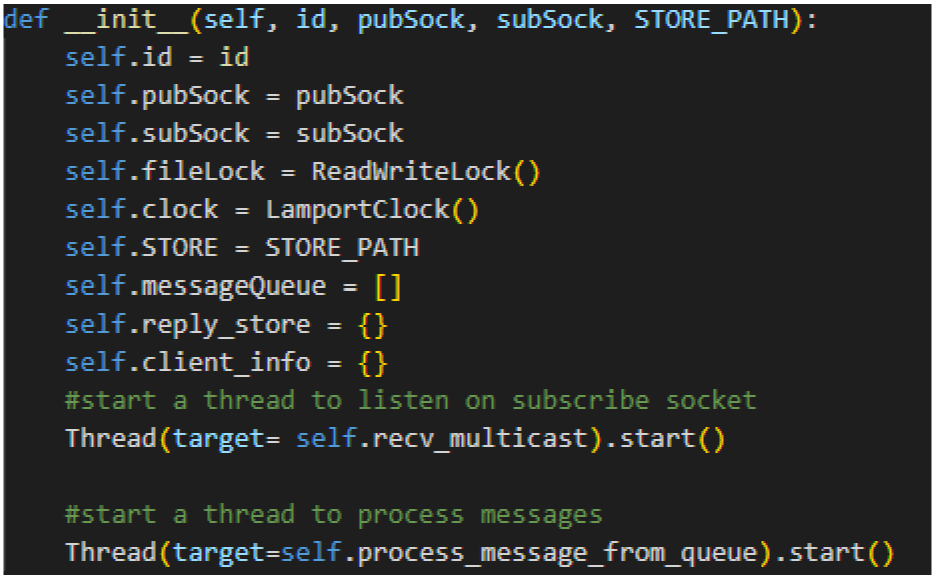
Starter Details-

We are building a distributed KV store with 3 servers. The starter is used to start the server processes independently.

Store Details -

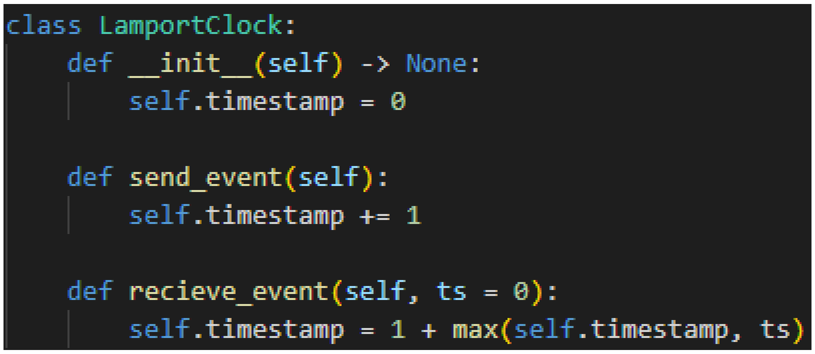
Each server process has the same parameters with some of additions.

- Each server has a Lamport clock object details to which are given below this.
- messageQueue to implement total order multicast.
- Reply store to store acknowledgements of received messages
- Client info, used to help in sending replies back to clients.
- Along with the recv_multicast, another parallel process runs after the init in a separate thread, this is to process messages from the message Queue.



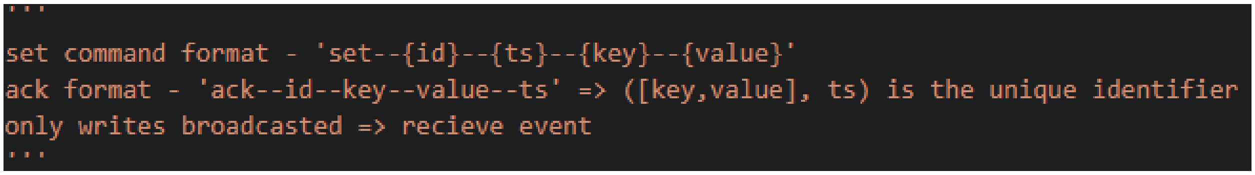
```
def __init__(self, id, pubSock, subSock, STORE_PATH):
    self.id = id
    self.pubSock = pubSock
    self.subSock = subSock
    self.fileLock = ReadWriteLock()
    self.clock = LamportClock()
    self.STORE = STORE_PATH
    self.messageQueue = []
    self.reply_store = {}
    self.client_info = {}
    #start a thread to listen on subscribe socket
    Thread(target= self.recv_multicast).start()

    #start a thread to process messages
    Thread(target=self.process_message_from_queue).start()
```

LamportClock class



```
class LamportClock:
    def __init__(self) -> None:
        self.timestamp = 0

    def send_event(self):
        self.timestamp += 1

    def recieve_event(self, ts = 0):
        self.timestamp = 1 + max(self.timestamp, ts)
```

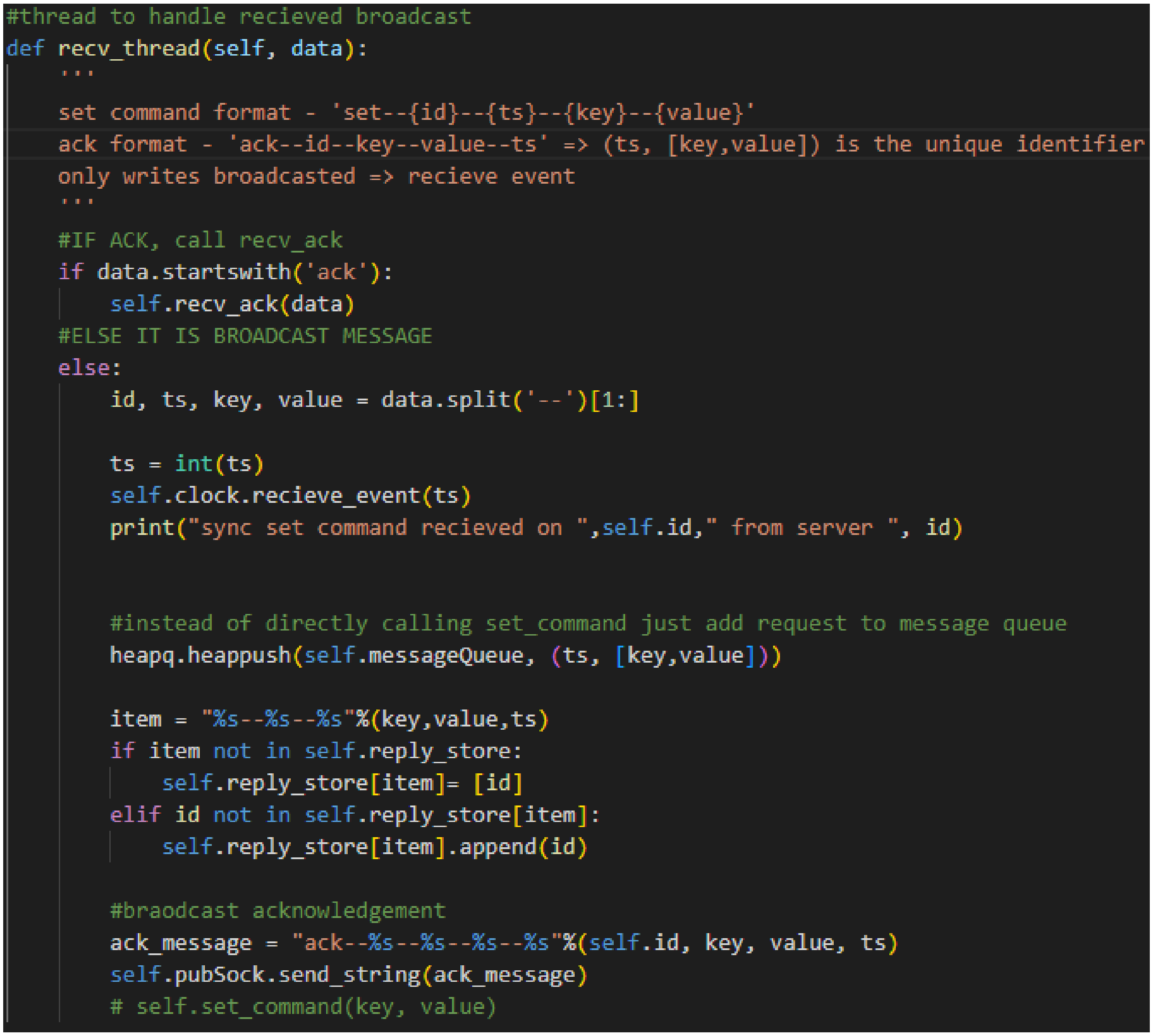
The recv_multicast function is same but there are changes in the recv thread.
An object of this class is stored in the store to easily implement lamport clock operations with send and recv events. In this implementation messages between the server are of 2 types - a broadcast containing the set command or the acknowledgement. The message formats are mentioned below



```
'''
set command format - 'set--{id}--{ts}--{key}--{value}'
ack format - 'ack--id--key--value--ts' => ([key,value], ts) is the unique identifier
only writes broadcasted => recieve event
'''
```

The timestamp is also included with each command.

Function - recv_thread

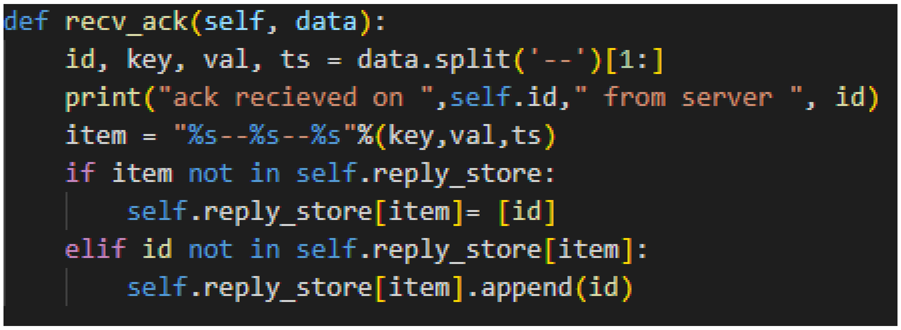


```
#thread to handle recieved broadcast
def recv_thread(self, data):
    '''
    set command format - 'set--{id}--{ts}--{key}--{value}'
    ack format - 'ack--id--key--value--ts' => (ts, [key,value]) is the unique identifier
    only writes broadcasted => recieve event
    '''
    #IF ACK, call recv_ack
    if data.startswith('ack'):
        self.recv_ack(data)
    #ELSE IT IS BROADCAST MESSAGE
    else:
        id, ts, key, value = data.split('--')[1:]

        ts = int(ts)
        self.clock.recieve_event(ts)
        print("sync set command recieved on ",self.id," from server ", id)


        #instead of directly calling set_command just add request to message queue
        heapq.heappush(self.messageQueue, (ts, [key,value]))

        item = "%s--%s--%s"%(key,value,ts)
        if item not in self.reply_store:
            self.reply_store[item]= [id]
        elif id not in self.reply_store[item]:
            self.reply_store[item].append(id)

        #braodcast acknowledgement
        ack_message = "ack--%s--%s--%s--%s"%(self.id, key, value, ts)
        self.pubSock.send_string(ack_message)
        # self.set_command(key, value)
```

For the acknowledgement messages a different function is used to handle it but for the broadcast message as we only do it for writes we have to increment the lamport clock of this server. The unique request identifier that I have used to identify messages from the queue is (ts, [key, value]). On receiving this request this server pushes this request in the message queue which is implemented using a heap and is sorted by the timestamp.

In the reply store we store the id(of the server that sent the message) of servers that already have this message. Lastly this server publishes an acknowledgement according to the format mentioned above so that other servers know that this one has received this request.

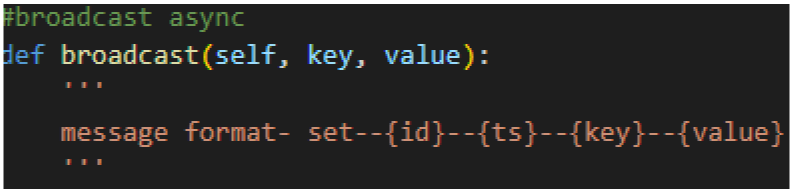
Function recv_ack()



```
def recv_ack(self, data):
    id, key, val, ts = data.split('--')[1:]
    print("ack recieved on ",self.id," from server ", id)
    item = "%s--%s--%s"%(key,val,ts)
    if item not in self.reply_store:
        self.reply_store[item]= [id]
    elif id not in self.reply_store[item]:
        self.reply_store[item].append(id)
```

On receiving an ack we just have to add the id of the sender server to our reply store.

The broadcast function is the same, just the timestamp is added to the broadcast message.

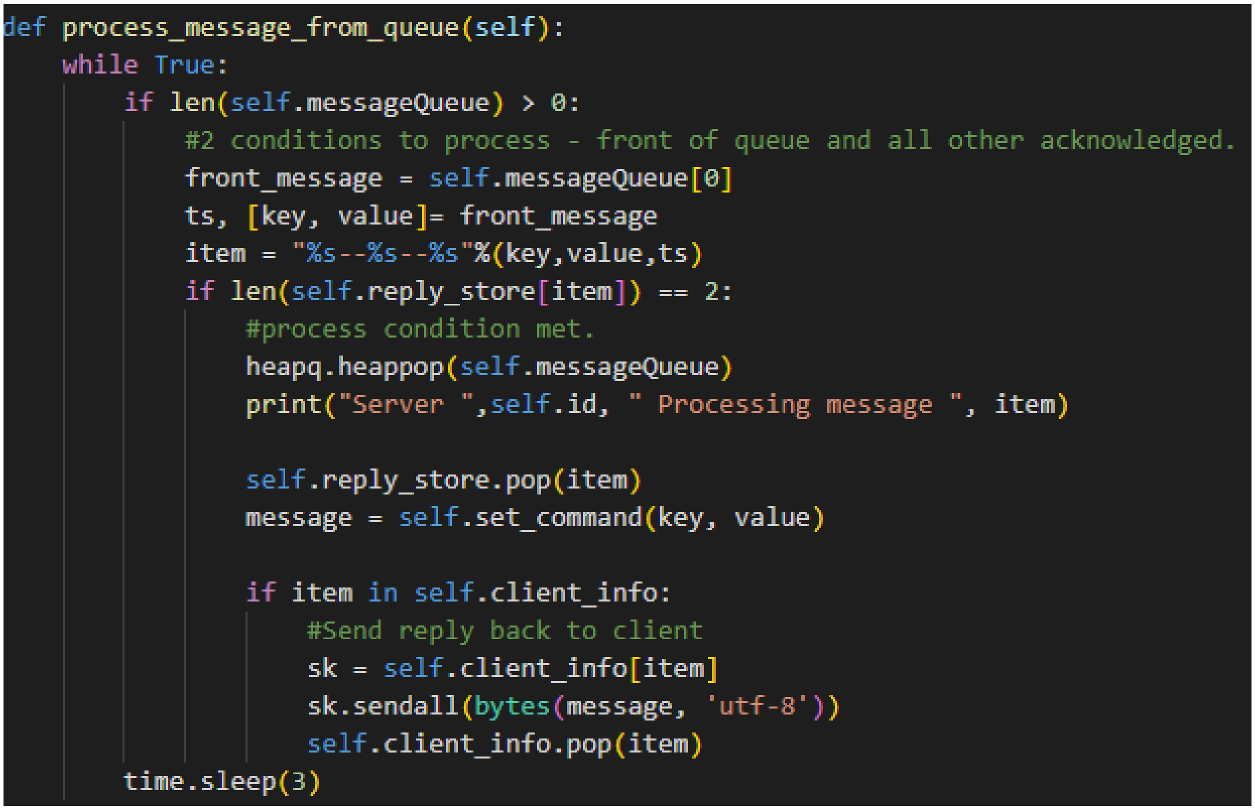


```
#broadcast async
def broadcast(self, key, value):
    '''

    message format- set--{id}--{ts}--{key}--{value}
    '''
```

In the handle client function, for the set command, we mark a recv event, push the message in queue, create the unique message identifier and add it to the reply store. Using this identifier we store the client socket as well so that when this is processed from the queue we can check the client_info dictionary to see if the socket object is there or not. And then we initiate the broadcast.

The get command is exactly the same as eventual, as the reads are instant and local.

Function process_message_from_queue()



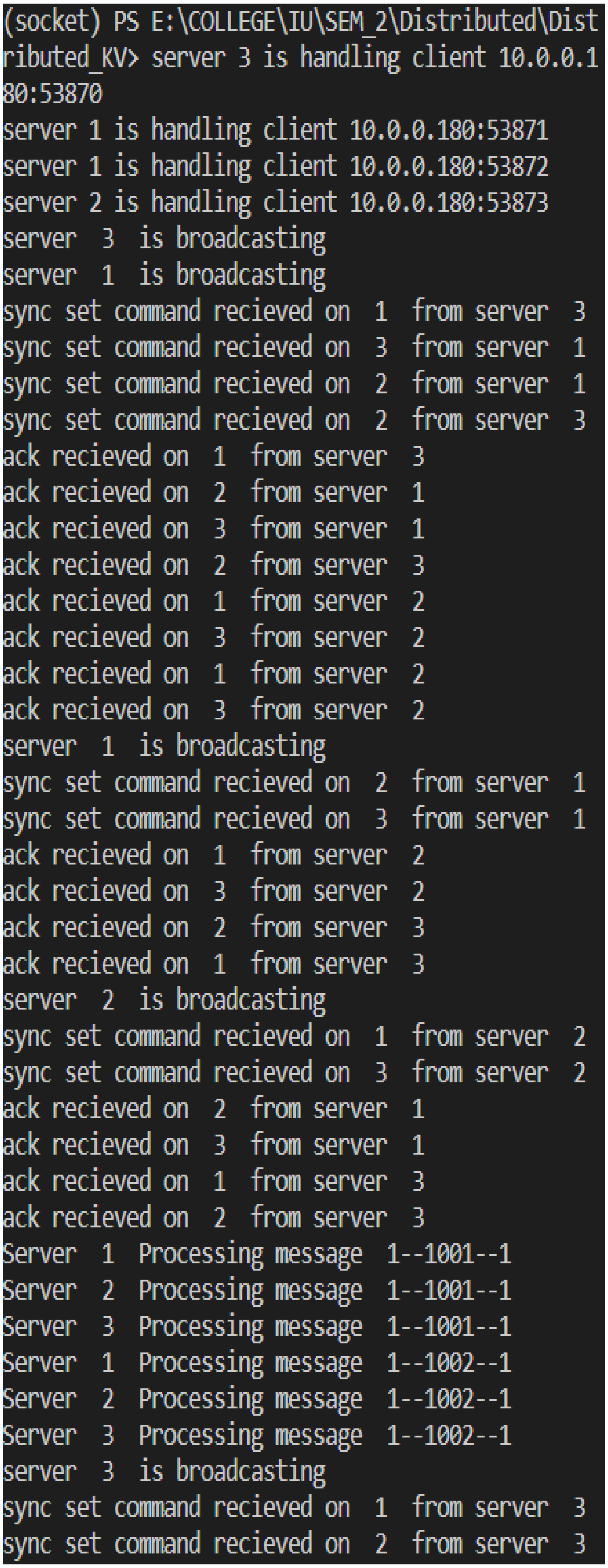
```python
def process_message_from_queue(self):
    while True:
        if len(self.messageQueue) > 0:
            #2 conditions to process - front of queue and all other acknowledged.
            front_message = self.messageQueue[0]
            ts, [key, value]= front_message
            item = "%s--%s--%s"%(key,value,ts)
            if len(self.reply_store[item]) == 2:
                #process condition met.
                heapq.heappop(self.messageQueue)
                print("Server ",self.id, " Processing message ", item)

                self.reply_store.pop(item)
                message = self.set_command(key, value)

                if item in self.client_info:
                    #Send reply back to client
                    sk = self.client_info[item]
                    sk.sendall(bytes(message, 'utf-8'))
                    self.client_info.pop(item)
        time.sleep(3)
```

This function runs in parallel after init. According to the total order multicast there are 2 conditions to process a message - 1) it should be in front of the queue and 2) It should have all acknowledgements from other servers. Those conditions are checked in this function and the client is sent the reply back as well if this is the server that it was connected to. A sleep for 3 seconds introduced - every 3 seconds it checks for the front message in the queue.

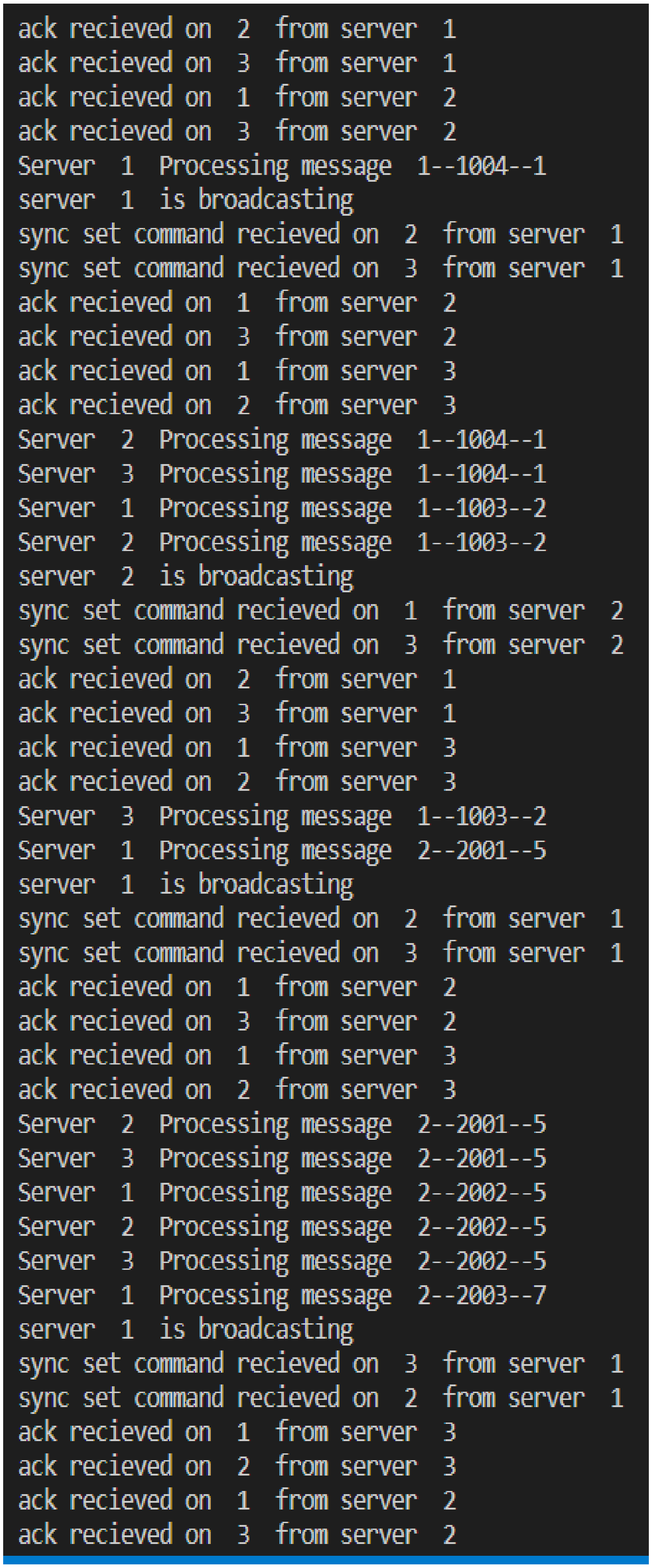
Test Cases - 1 -

File client_starter.py - spawns 4 instances of test_client.py

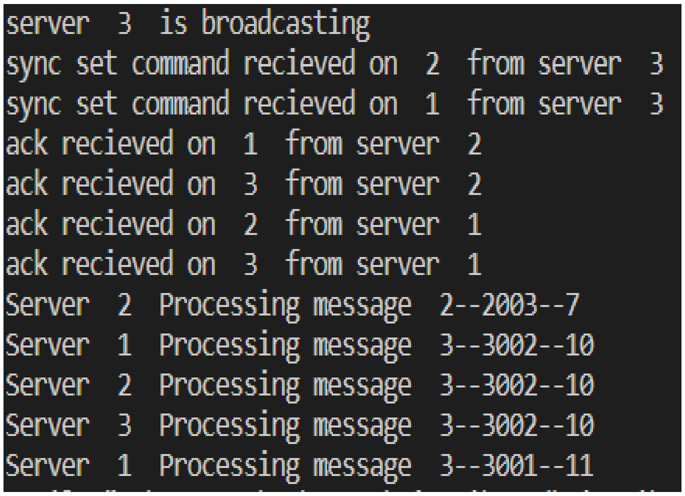
test_client.py has a combination of set get commands with random wait times. On the server side we can see that all the writes are processed in the same order. The server side output when it shows message processed that signifies the execution of writes. Individual servers might process the write in any order(say 1 message is processed by first 1, 2, and then 3rd server and other message might be with 2, 1 ,3) but the overall order of writes is still preserved.



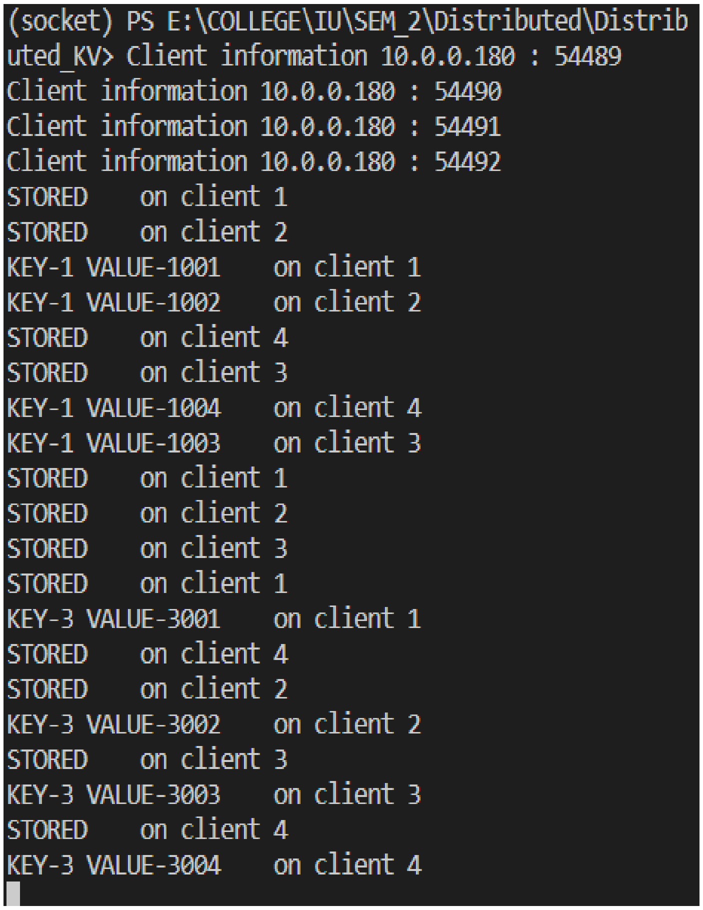
```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distributed_KV> server 3 is handling client 10.0.0.180:53870
server 1 is handling client 10.0.0.180:53871
server 1 is handling client 10.0.0.180:53872
server 2 is handling client 10.0.0.180:53873
server  3  is broadcasting
server  1  is broadcasting
sync set command recieved on  1  from server  3
sync set command recieved on  3  from server  1
sync set command recieved on  2  from server  1
sync set command recieved on  2  from server  3
ack recieved on  1  from server  3
ack recieved on  2  from server  1
ack recieved on  3  from server  1
ack recieved on  2  from server  3
ack recieved on  1  from server  2
ack recieved on  3  from server  2
ack recieved on  1  from server  2
ack recieved on  3  from server  2
server  1  is broadcasting
sync set command recieved on  2  from server  1
sync set command recieved on  3  from server  1
ack recieved on  1  from server  2
ack recieved on  3  from server  2
ack recieved on  2  from server  3
ack recieved on  1  from server  3
server  2  is broadcasting
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
ack recieved on  2  from server  1
ack recieved on  3  from server  1
ack recieved on  1  from server  3
ack recieved on  2  from server  3
Server  1  Processing message  1--1001--1
Server  2  Processing message  1--1001--1
Server  3  Processing message  1--1001--1
Server  1  Processing message  1--1002--1
Server  2  Processing message  1--1002--1
Server  3  Processing message  1--1002--1
server  3  is broadcasting
sync set command recieved on  1  from server  3
sync set command recieved on  2  from server  3
```



```
ack recieved on  2  from server  1
ack recieved on  3  from server  1
ack recieved on  1  from server  2
ack recieved on  3  from server  2
Server  1  Processing message  1--1004--1
server  1  is broadcasting
sync set command recieved on  2  from server  1
sync set command recieved on  3  from server  1
ack recieved on  1  from server  2
ack recieved on  3  from server  2
ack recieved on  1  from server  3
ack recieved on  2  from server  3
Server  2  Processing message  1--1004--1
Server  3  Processing message  1--1004--1
Server  1  Processing message  1--1003--2
Server  2  Processing message  1--1003--2
server  2  is broadcasting
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
ack recieved on  2  from server  1
ack recieved on  3  from server  1
ack recieved on  1  from server  3
ack recieved on  2  from server  3
Server  3  Processing message  1--1003--2
Server  1  Processing message  2--2001--5
server  1  is broadcasting
sync set command recieved on  2  from server  1
sync set command recieved on  3  from server  1
ack recieved on  1  from server  2
ack recieved on  3  from server  2
ack recieved on  1  from server  3
ack recieved on  2  from server  3
Server  2  Processing message  2--2001--5
Server  3  Processing message  2--2001--5
Server  1  Processing message  2--2002--5
Server  2  Processing message  2--2002--5
Server  3  Processing message  2--2002--5
Server  1  Processing message  2--2003--7
server  1  is broadcasting
sync set command recieved on  3  from server  1
sync set command recieved on  2  from server  1
ack recieved on  1  from server  3
ack recieved on  2  from server  3
ack recieved on  1  from server  2
ack recieved on  3  from server  2
```



```
server  3  is broadcasting
sync set command recieved on  2  from server  3
sync set command recieved on  1  from server  3
ack recieved on  1  from server  2
ack recieved on  3  from server  2
ack recieved on  2  from server  1
ack recieved on  3  from server  1
Server  2  Processing message  2--2003--7
Server  1  Processing message  3--3002--10
Server  2  Processing message  3--3002--10
Server  3  Processing message  3--3002--10
Server  1  Processing message  3--3001--11
```



```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distrib
uted_KV> Client information 10.0.0.180 : 54489
Client information 10.0.0.180 : 54490
Client information 10.0.0.180 : 54491
Client information 10.0.0.180 : 54492
STORED    on client 1
STORED    on client 2
KEY-1 VALUE-1001    on client 1
KEY-1 VALUE-1002    on client 2
STORED    on client 4
STORED    on client 3
KEY-1 VALUE-1004    on client 4
KEY-1 VALUE-1003    on client 3
STORED    on client 1
STORED    on client 2
STORED    on client 3
STORED    on client 1
KEY-3 VALUE-3001    on client 1
STORED    on client 4
STORED    on client 2
KEY-3 VALUE-3002    on client 2
STORED    on client 3
KEY-3 VALUE-3003    on client 3
STORED    on client 4
KEY-3 VALUE-3004    on client 4
```

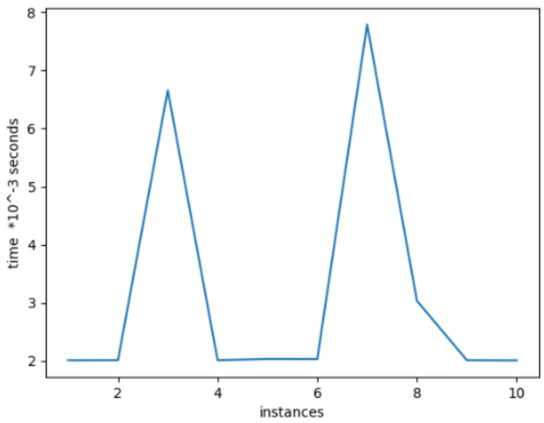
Client outputs

More test Cases -

Run the client.py file in different terminals and give custom set and get commands whil observing the outputs. The reads are returned instantly and the writes are only processed according to the total order multicast.

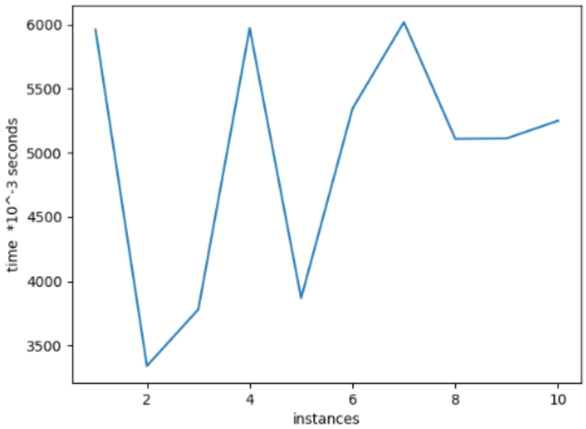
Performance Evaluation -
X axis - number of specific request, 3 => 3rd request
Y axis - time taken to respond to client

Graph for 10 consecutive get requests



Not much change in reads

Graph for 10 consecutive set requests



writes take more time because of total order multicast

LINEARIZABLE CONSISTENCY

The main concept behind linearizable consistency according to this implementation is that all requests are processed using total order multicast.

Required Files in the Directory

- Server_file is linearizable_server.py

Working Details -

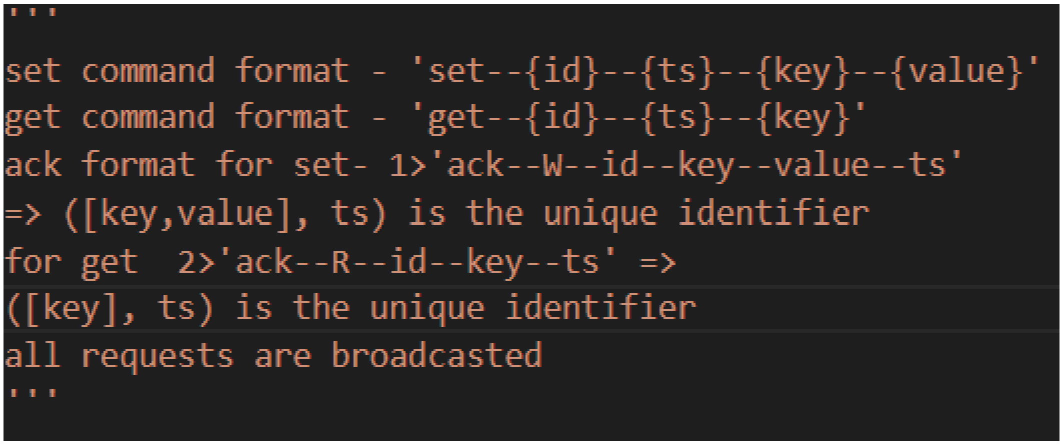
- **Config -** change the model parameter to *linearizable.*
- Run starter.py
- Run client_starter.py to see given test case
- Run client.py on different terminals for custom client commands.

Design details and changes -

Get command is also changed to just return the return message which is then processed by process_get function. Similarly the process_message_from_queue function calls process set for a set command being processed from the queue.

Lamport clock receive events also happen with get requests along with other things required for total order multicast. Message Q tuple is - (ts, [key]) for get and (ts, [key, value]).

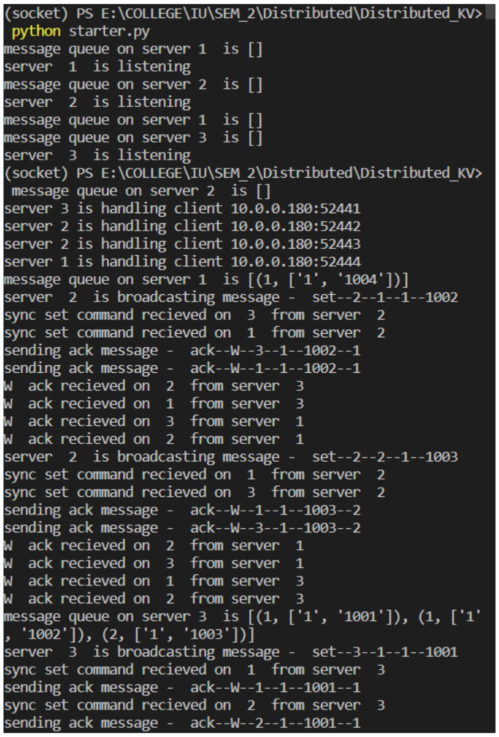
Changes in message formats -



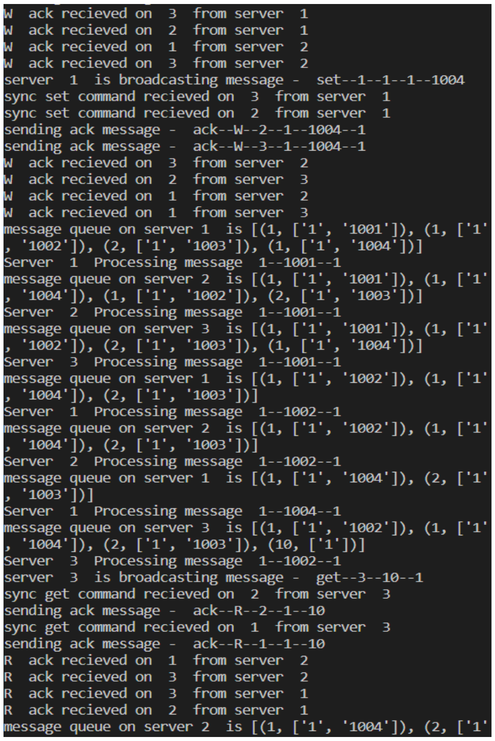
```
'''
set command format - 'set--{id}--{ts}--{key}--{value}'
get command format - 'get--{id}--{ts}--{key}'
ack format for set- 1>'ack--W--id--key--value--ts'
=> ([key,value], ts) is the unique identifier
for get  2>'ack--R--id--key--ts' =>
([key], ts) is the unique identifier
all requests are broadcasted
'''
```

Test Case - 1

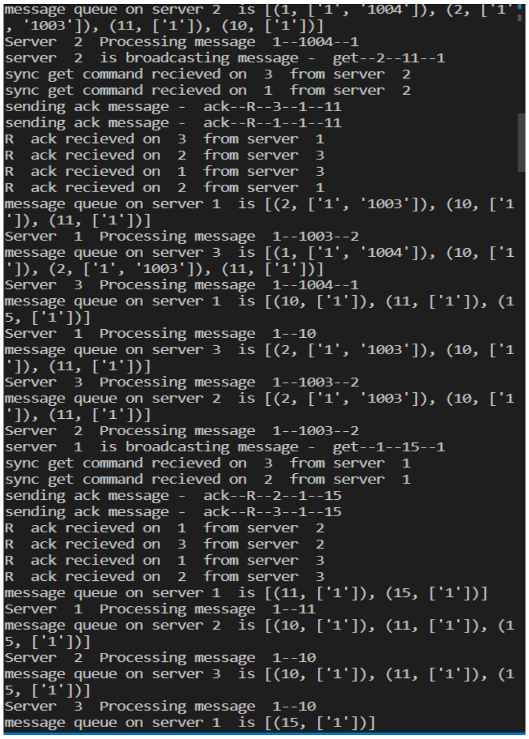
Run the client starter file after servers start running.



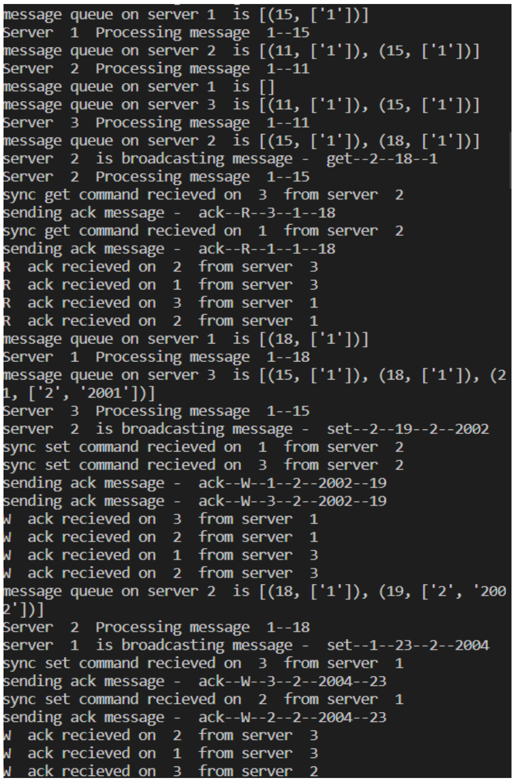
```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distributed_KV>
 python starter.py
message queue on server 1  is []
server  1  is listening
message queue on server 2  is []
server  2  is listening
message queue on server 1  is []
message queue on server 3  is []
server  3  is listening
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distributed_KV>
 message queue on server 2  is []
server 3 is handling client 10.0.0.180:52441
server 2 is handling client 10.0.0.180:52442
server 2 is handling client 10.0.0.180:52443
server 1 is handling client 10.0.0.180:52444
message queue on server 1  is [(1, ['1', '1004'])]
server  2  is broadcasting message -  set--2--1--1--1002
sync set command recieved on  3  from server  2
sync set command recieved on  1  from server  2
sending ack message -  ack--W--3--1--1002--1
sending ack message -  ack--W--1--1--1002--1
W  ack recieved on  2  from server  3
W  ack recieved on  1  from server  3
W  ack recieved on  3  from server  1
W  ack recieved on  2  from server  1
server  2  is broadcasting message -  set--2--2--1--1003
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
sending ack message -  ack--W--1--1--1003--2
sending ack message -  ack--W--3--1--1003--2
W  ack recieved on  2  from server  1
W  ack recieved on  3  from server  1
W  ack recieved on  1  from server  3
W  ack recieved on  2  from server  3
message queue on server 3  is [(1, ['1', '1001']), (1, ['1'
, '1002']), (2, ['1', '1003'])]
server  3  is broadcasting message -  set--3--1--1--1001
sync set command recieved on  1  from server  3
sending ack message -  ack--W--1--1--1001--1
sync set command recieved on  2  from server  3
sending ack message -  ack--W--2--1--1001--1
```



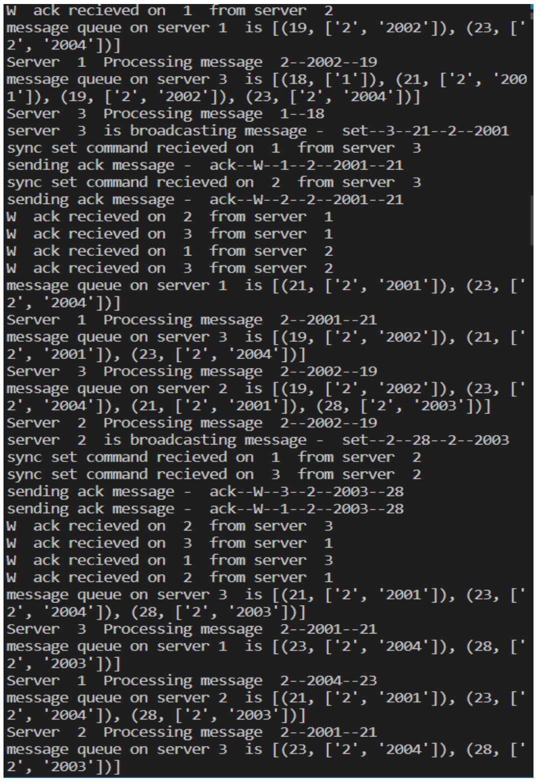
```
W  ack recieved on  3  from server  1
W  ack recieved on  2  from server  1
W  ack recieved on  1  from server  2
W  ack recieved on  3  from server  2
server  1  is broadcasting message -  set--1--1--1--1004
sync set command recieved on  3  from server  1
sync set command recieved on  2  from server  1
sending ack message -  ack--W--2--1--1004--1
sending ack message -  ack--W--3--1--1004--1
W  ack recieved on  3  from server  2
W  ack recieved on  2  from server  3
W  ack recieved on  1  from server  2
W  ack recieved on  1  from server  3
message queue on server 1  is [(1, ['1', '1001']), (1, ['1'
, '1002']), (2, ['1', '1003']), (1, ['1', '1004'])]
Server  1  Processing message  1--1001--1
message queue on server 2  is [(1, ['1', '1001']), (1, ['1'
, '1004']), (1, ['1', '1002']), (2, ['1', '1003'])]
Server  2  Processing message  1--1001--1
message queue on server 3  is [(1, ['1', '1001']), (1, ['1'
, '1002']), (2, ['1', '1003']), (1, ['1', '1004'])]
Server  3  Processing message  1--1001--1
message queue on server 1  is [(1, ['1', '1002']), (1, ['1'
, '1004']), (2, ['1', '1003'])]
Server  1  Processing message  1--1002--1
message queue on server 2  is [(1, ['1', '1002']), (1, ['1'
, '1004']), (2, ['1', '1003'])]
Server  2  Processing message  1--1002--1
message queue on server 1  is [(1, ['1', '1004']), (2, ['1'
, '1003'])]
Server  1  Processing message  1--1004--1
message queue on server 3  is [(1, ['1', '1002']), (1, ['1'
, '1004']), (2, ['1', '1003']), (10, ['1'])]
Server  3  Processing message  1--1002--1
server  3  is broadcasting message -  get--3--10--1
sync get command recieved on  2  from server  3
sending ack message -  ack--R--2--1--10
sync get command recieved on  1  from server  3
sending ack message -  ack--R--1--1--10
R  ack recieved on  1  from server  2
R  ack recieved on  3  from server  2
R  ack recieved on  3  from server  1
R  ack recieved on  2  from server  1
message queue on server 2  is [(1, ['1', '1004']), (2, ['1'
```



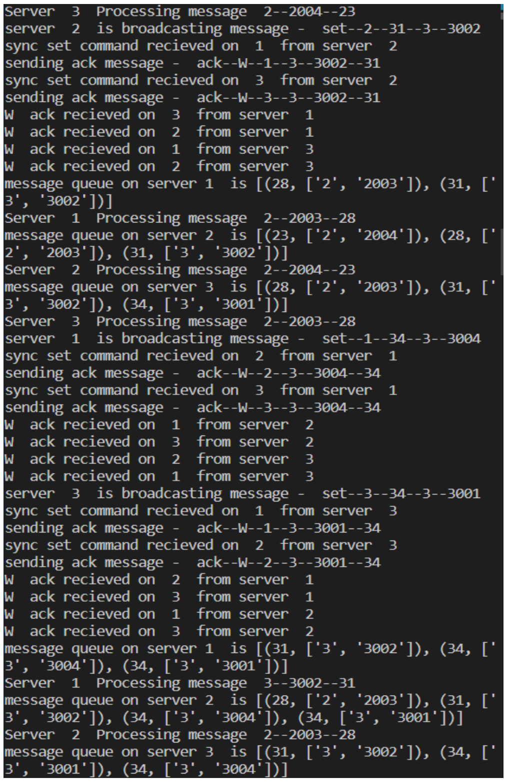
```
message queue on server 2  is [(1, ['1', '1004']), (2, ['1'
, '1003']), (11, ['1']), (10, ['1'])]
Server  2  Processing message  1--1004--1
server  2  is broadcasting message -  get--2--11--1
sync get command recieved on  3  from server  2
sync get command recieved on  1  from server  2
sending ack message -  ack--R--3--1--11
sending ack message -  ack--R--1--1--11
R  ack recieved on  3  from server  1
R  ack recieved on  2  from server  3
R  ack recieved on  1  from server  3
R  ack recieved on  2  from server  1
message queue on server 1  is [(2, ['1', '1003']), (10, ['1
']), (11, ['1'])]
Server  1  Processing message  1--1003--2
message queue on server 3  is [(1, ['1', '1004']), (10, ['1
']), (2, ['1', '1003']), (11, ['1'])]
Server  3  Processing message  1--1004--1
message queue on server 1  is [(10, ['1']), (11, ['1']), (1
5, ['1'])]
Server  1  Processing message  1--10
message queue on server 3  is [(2, ['1', '1003']), (10, ['1
']), (11, ['1'])]
Server  3  Processing message  1--1003--2
message queue on server 2  is [(2, ['1', '1003']), (10, ['1
']), (11, ['1'])]
Server  2  Processing message  1--1003--2
server  1  is broadcasting message -  get--1--15--1
sync get command recieved on  3  from server  1
sync get command recieved on  2  from server  1
sending ack message -  ack--R--2--1--15
sending ack message -  ack--R--3--1--15
R  ack recieved on  1  from server  2
R  ack recieved on  3  from server  2
R  ack recieved on  1  from server  3
R  ack recieved on  2  from server  3
message queue on server 1  is [(11, ['1']), (15, ['1'])]
Server  1  Processing message  1--11
message queue on server 2  is [(10, ['1']), (11, ['1']), (1
5, ['1'])]
Server  2  Processing message  1--10
message queue on server 3  is [(10, ['1']), (11, ['1']), (1
5, ['1'])]
Server  3  Processing message  1--10
message queue on server 1  is [(15, ['1'])]
```



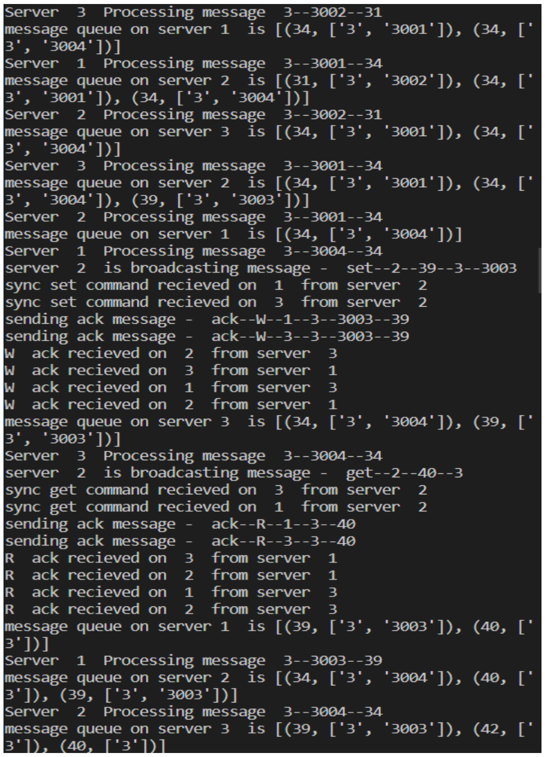
```
message queue on server 1  is [(15, ['1'])]
Server  1  Processing message  1--15
message queue on server 2  is [(11, ['1']), (15, ['1'])]
Server  2  Processing message  1--11
message queue on server 1  is []
message queue on server 3  is [(11, ['1']), (15, ['1'])]
Server  3  Processing message  1--11
message queue on server 2  is [(15, ['1']), (18, ['1'])]
server  2  is broadcasting message -  get--2--18--1
Server  2  Processing message  1--15
sync get command recieved on  3  from server  2
sending ack message -  ack--R--3--1--18
sync get command recieved on  1  from server  2
sending ack message -  ack--R--1--1--18
R  ack recieved on  2  from server  3
R  ack recieved on  1  from server  3
R  ack recieved on  3  from server  1
R  ack recieved on  2  from server  1
message queue on server 1  is [(18, ['1'])]
Server  1  Processing message  1--18
message queue on server 3  is [(15, ['1']), (18, ['1']), (2
1, ['2', '2001'])]
Server  3  Processing message  1--15
server  2  is broadcasting message -  set--2--19--2--2002
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
sending ack message -  ack--W--1--2--2002--19
sending ack message -  ack--W--3--2--2002--19
W  ack recieved on  3  from server  1
W  ack recieved on  2  from server  1
W  ack recieved on  1  from server  3
W  ack recieved on  2  from server  3
message queue on server 2  is [(18, ['1']), (19, ['2', '200
2'])]
Server  2  Processing message  1--18
server  1  is broadcasting message -  set--1--23--2--2004
sync set command recieved on  3  from server  1
sending ack message -  ack--W--3--2--2004--23
sync set command recieved on  2  from server  1
sending ack message -  ack--W--2--2--2004--23
W  ack recieved on  2  from server  3
W  ack recieved on  1  from server  3
W  ack recieved on  3  from server  2
```



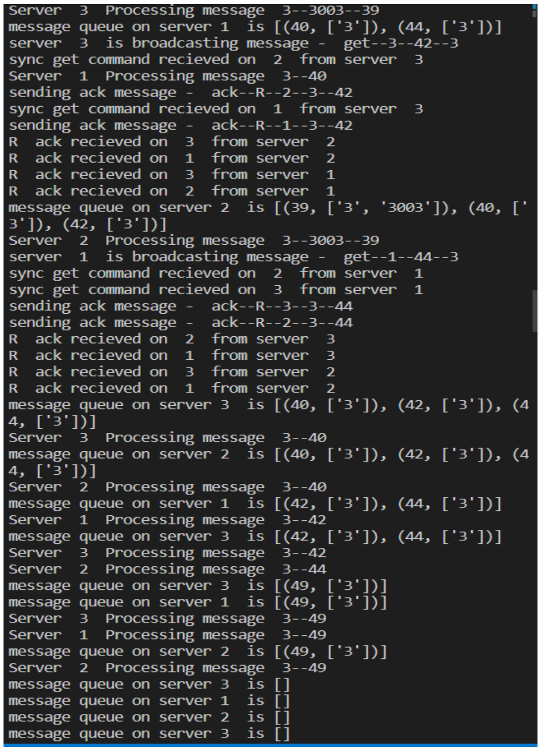
```
W  ack recieved on  1  from server  2
message queue on server 1  is [(19, ['2', '2002']), (23, ['
2', '2004'])]
Server  1  Processing message  2--2002--19
message queue on server 3  is [(18, ['1']), (21, ['2', '200
1']), (19, ['2', '2002']), (23, ['2', '2004'])]
Server  3  Processing message  1--18
server  3  is broadcasting message -  set--3--21--2--2001
sync set command recieved on  1  from server  3
sending ack message -  ack--W--1--2--2001--21
sync set command recieved on  2  from server  3
sending ack message -  ack--W--2--2--2001--21
W  ack recieved on  2  from server  1
W  ack recieved on  3  from server  1
W  ack recieved on  1  from server  2
W  ack recieved on  3  from server  2
message queue on server 1  is [(21, ['2', '2001']), (23, ['
2', '2004'])]
Server  1  Processing message  2--2001--21
message queue on server 3  is [(19, ['2', '2002']), (21, ['
2', '2001']), (23, ['2', '2004'])]
Server  3  Processing message  2--2002--19
message queue on server 2  is [(19, ['2', '2002']), (23, ['
2', '2004']), (21, ['2', '2001']), (28, ['2', '2003'])]
Server  2  Processing message  2--2002--19
server  2  is broadcasting message -  set--2--28--2--2003
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
sending ack message -  ack--W--3--2--2003--28
sending ack message -  ack--W--1--2--2003--28
W  ack recieved on  2  from server  3
W  ack recieved on  3  from server  1
W  ack recieved on  1  from server  3
W  ack recieved on  2  from server  1
message queue on server 3  is [(21, ['2', '2001']), (23, ['
2', '2004']), (28, ['2', '2003'])]
Server  3  Processing message  2--2001--21
message queue on server 1  is [(23, ['2', '2004']), (28, ['
2', '2003'])]
Server  1  Processing message  2--2004--23
message queue on server 2  is [(21, ['2', '2001']), (23, ['
2', '2004']), (28, ['2', '2003'])]
Server  2  Processing message  2--2001--21
message queue on server 3  is [(23, ['2', '2004']), (28, ['
2', '2003'])]
```



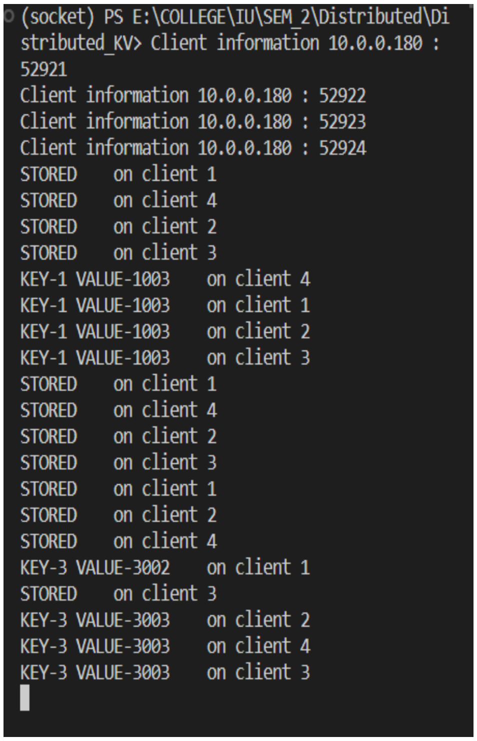
```
Server  3  Processing message  2--2004--23
server  2  is broadcasting message -  set--2--31--3--3002
sync set command recieved on  1  from server  2
sending ack message -  ack--W--1--3--3002--31
sync set command recieved on  3  from server  2
sending ack message -  ack--W--3--3--3002--31
W  ack recieved on  3  from server  1
W  ack recieved on  2  from server  1
W  ack recieved on  1  from server  3
W  ack recieved on  2  from server  3
message queue on server 1  is [(28, ['2', '2003']), (31, ['
3', '3002'])]
Server  1  Processing message  2--2003--28
message queue on server 2  is [(23, ['2', '2004']), (28, ['
2', '2003']), (31, ['3', '3002'])]
Server  2  Processing message  2--2004--23
message queue on server 3  is [(28, ['2', '2003']), (31, ['
3', '3002']), (34, ['3', '3001'])]
Server  3  Processing message  2--2003--28
server  1  is broadcasting message -  set--1--34--3--3004
sync set command recieved on  2  from server  1
sending ack message -  ack--W--2--3--3004--34
sync set command recieved on  3  from server  1
sending ack message -  ack--W--3--3--3004--34
W  ack recieved on  1  from server  2
W  ack recieved on  3  from server  2
W  ack recieved on  2  from server  3
W  ack recieved on  1  from server  3
server  3  is broadcasting message -  set--3--34--3--3001
sync set command recieved on  1  from server  3
sending ack message -  ack--W--1--3--3001--34
sync set command recieved on  2  from server  3
sending ack message -  ack--W--2--3--3001--34
W  ack recieved on  2  from server  1
W  ack recieved on  3  from server  1
W  ack recieved on  1  from server  2
W  ack recieved on  3  from server  2
message queue on server 1  is [(31, ['3', '3002']), (34, ['
3', '3004']), (34, ['3', '3001'])]
Server  1  Processing message  3--3002--31
message queue on server 2  is [(28, ['2', '2003']), (31, ['
3', '3002']), (34, ['3', '3004']), (34, ['3', '3001'])]
Server  2  Processing message  2--2003--28
message queue on server 3  is [(31, ['3', '3002']), (34, ['
3', '3001']), (34, ['3', '3004'])]
```



```
Server  3  Processing message  3--3002--31
message queue on server 1  is [(34, ['3', '3001']), (34, ['
3', '3004'])]
Server  1  Processing message  3--3001--34
message queue on server 2  is [(31, ['3', '3002']), (34, ['
3', '3001']), (34, ['3', '3004'])]
Server  2  Processing message  3--3002--31
message queue on server 3  is [(34, ['3', '3001']), (34, ['
3', '3004'])]
Server  3  Processing message  3--3001--34
message queue on server 2  is [(34, ['3', '3001']), (34, ['
3', '3004']), (39, ['3', '3003'])]
Server  2  Processing message  3--3001--34
message queue on server 1  is [(34, ['3', '3004'])]
Server  1  Processing message  3--3004--34
server  2  is broadcasting message -  set--2--39--3--3003
sync set command recieved on  1  from server  2
sync set command recieved on  3  from server  2
sending ack message -  ack--W--1--3--3003--39
sending ack message -  ack--W--3--3--3003--39
W  ack recieved on  2  from server  3
W  ack recieved on  3  from server  1
W  ack recieved on  1  from server  3
W  ack recieved on  2  from server  1
message queue on server 3  is [(34, ['3', '3004']), (39, ['
3', '3003'])]
Server  3  Processing message  3--3004--34
server  2  is broadcasting message -  get--2--40--3
sync get command recieved on  3  from server  2
sync get command recieved on  1  from server  2
sending ack message -  ack--R--1--3--40
sending ack message -  ack--R--3--3--40
R  ack recieved on  3  from server  1
R  ack recieved on  2  from server  1
R  ack recieved on  1  from server  3
R  ack recieved on  2  from server  3
message queue on server 1  is [(39, ['3', '3003']), (40, ['
3'])]
Server  1  Processing message  3--3003--39
message queue on server 2  is [(34, ['3', '3004']), (40, ['
3']), (39, ['3', '3003'])]
Server  2  Processing message  3--3004--34
message queue on server 3  is [(39, ['3', '3003']), (42, ['
3']), (40, ['3'])]
```



```
Server  3  Processing message  3--3003--39
message queue on server 1  is [(40, ['3']), (44, ['3'])]
server  3  is broadcasting message -  get--3--42--3
sync get command recieved on  2  from server  3
Server  1  Processing message  3--40
sending ack message -  ack--R--2--3--42
sync get command recieved on  1  from server  3
sending ack message -  ack--R--1--3--42
R  ack recieved on  3  from server  2
R  ack recieved on  1  from server  2
R  ack recieved on  3  from server  1
R  ack recieved on  2  from server  1
message queue on server 2  is [(39, ['3', '3003']), (40, ['
3']), (42, ['3'])]
Server  2  Processing message  3--3003--39
server  1  is broadcasting message -  get--1--44--3
sync get command recieved on  2  from server  1
sync get command recieved on  3  from server  1
sending ack message -  ack--R--3--3--44
sending ack message -  ack--R--2--3--44
R  ack recieved on  2  from server  3
R  ack recieved on  1  from server  3
R  ack recieved on  3  from server  2
R  ack recieved on  1  from server  2
message queue on server 3  is [(40, ['3']), (42, ['3']), (4
4, ['3'])]
Server  3  Processing message  3--40
message queue on server 2  is [(40, ['3']), (42, ['3']), (4
4, ['3'])]
Server  2  Processing message  3--40
message queue on server 1  is [(42, ['3']), (44, ['3'])]
Server  1  Processing message  3--42
message queue on server 3  is [(42, ['3']), (44, ['3'])]
Server  3  Processing message  3--42
Server  2  Processing message  3--44
message queue on server 3  is [(49, ['3'])]
message queue on server 1  is [(49, ['3'])]
Server  3  Processing message  3--49
Server  1  Processing message  3--49
message queue on server 2  is [(49, ['3'])]
Server  2  Processing message  3--49
message queue on server 3  is []
message queue on server 1  is []
message queue on server 2  is []
message queue on server 3  is []
```



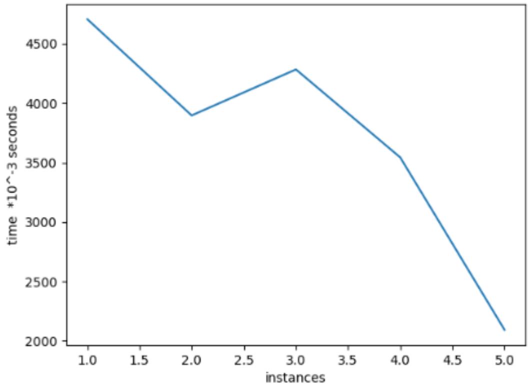
```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Di
stributed_KV> Client information 10.0.0.180 :
52921
Client information 10.0.0.180 : 52922
Client information 10.0.0.180 : 52923
Client information 10.0.0.180 : 52924
STORED    on client 1
STORED    on client 4
STORED    on client 2
STORED    on client 3
KEY-1 VALUE-1003    on client 4
KEY-1 VALUE-1003    on client 1
KEY-1 VALUE-1003    on client 2
KEY-1 VALUE-1003    on client 3
STORED    on client 1
STORED    on client 4
STORED    on client 2
STORED    on client 3
STORED    on client 1
STORED    on client 2
STORED    on client 4
KEY-3 VALUE-3002    on client 1
STORED    on client 3
KEY-3 VALUE-3003    on client 2
KEY-3 VALUE-3003    on client 4
KEY-3 VALUE-3003    on client 3
```

Client output

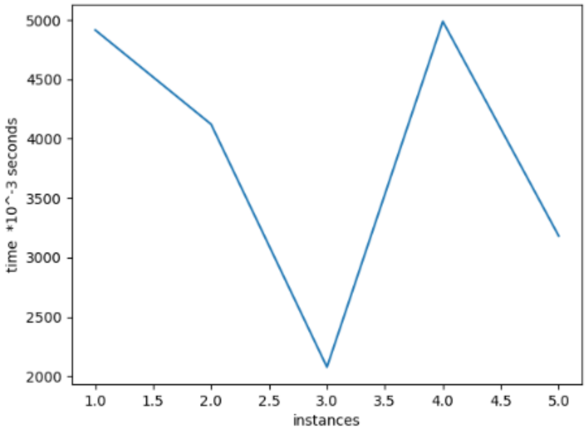
In the above test case we can see that all the processing messages are done in a single order across the 3 servers. *** One identified limitation while testing was that because of the design including broadcasting of acknowledgements when the number of requests increases some messages might get missed because of high network traffic, it is possible for this implementation to get stuck as it waits for acknowledgements before moving forward. To tackle this problem we can implement timeouts with each requests so that if a message was missed, the implementation will not stop functioning. ***

More test cases - Run the client.py file in different terminals and provide custom set and get commands.

5 Consecutive get requests



5 consecutive set requests



Both take more time because of writing and broadcast delays.

CAUSAL CONSISTENCY

The main concept behind causal consistency is to maintain the happens before relation between the events. For this implementation we need instant write reply with an async broadcast to other servers and a read operation that waits till pending writes are processed.

Required Files in the Directory

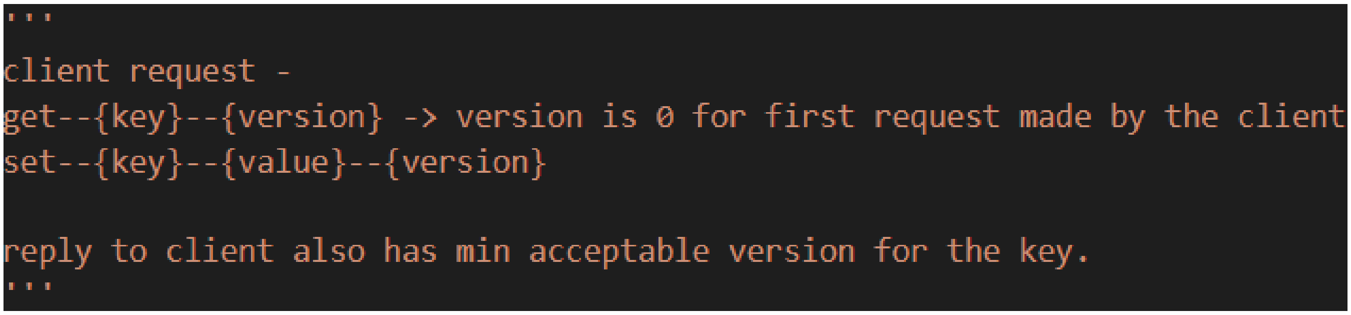
- Server_file is causal_server.py

Working Details -

- **Config -** change the model parameter to *causal.*
- Run starter.py
- Run causal_client.py to see given test case

Design details and changes -

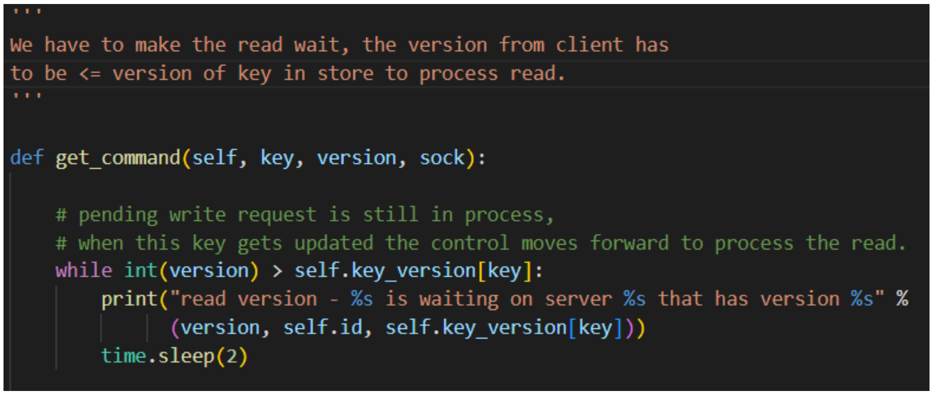
Message format changes -



```
'''
client request -
get--{key}--{version} -> version is 0 for first request made by the client
set--{key}--{value}--{version}

reply to client also has min acceptable version for the key.
'''
```

A version parameter is included which is used to determine if any pending writes are there for that key. This parameter is also included in the asyc broadcast to other servers. On server side when write is performed the version is incremented and sent back to client and broadcasted as well.

The main change is in the get_command function -



```
'''
We have to make the read wait, the version from client has
to be <= version of key in store to process read.
'''

def get_command(self, key, version, sock):

    # pending write request is still in process,
    # when this key gets updated the control moves forward to process the read.
    while int(version) > self.key_version[key]:
        print("read version - %s is waiting on server %s that has version %s" %
                (version, self.id, self.key_version[key]))
        time.sleep(2)
```

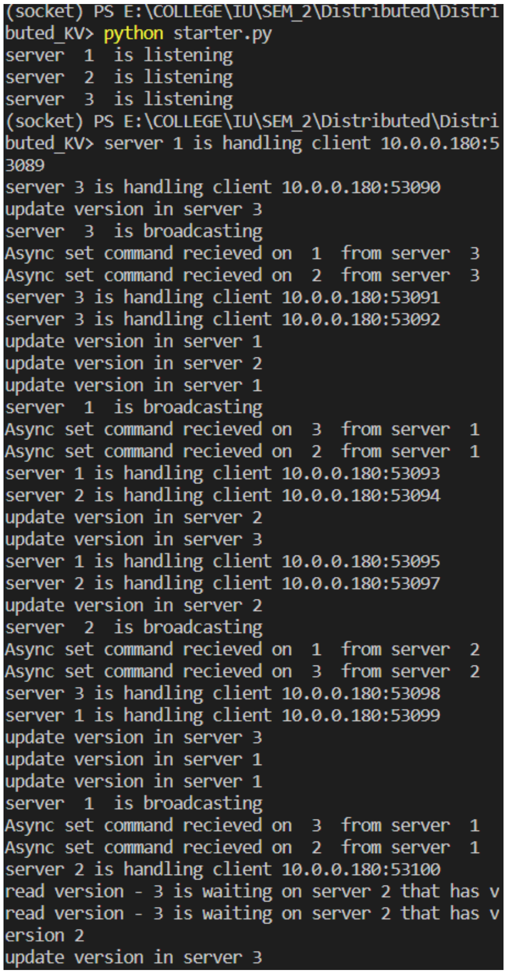
This while loop is designed to wait till the current server updates the version of this key. When this condition is satisfied meaning pending writes have been processed only then it can move to normal get_command execution.

Changes in client side -

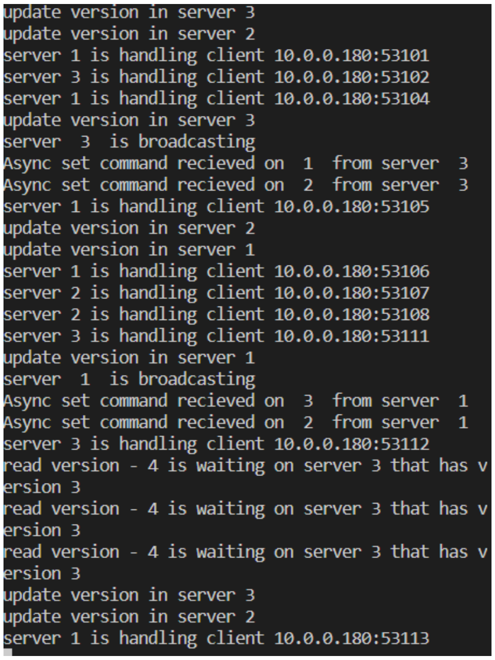
It is required to maintain a version vector storing the version of each key it makes a request on. To get a clear example in test cases I have made this so that for each request the client connects to a random server.

Test Case -

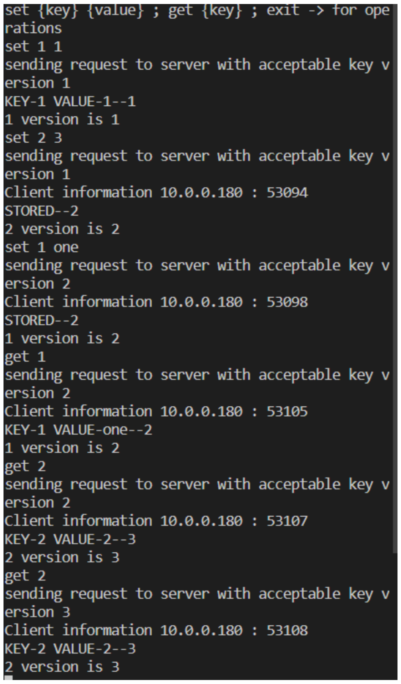
Run causal_client.py file on 3 different terminals to simulate 3 clients.



```
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distri
buted_KV> python starter.py
server  1  is listening
server  2  is listening
server  3  is listening
(socket) PS E:\COLLEGE\IU\SEM_2\Distributed\Distri
buted_KV> server 1 is handling client 10.0.0.180:5
3089
server 3 is handling client 10.0.0.180:53090
update version in server 3
server  3  is broadcasting
Async set command recieved on  1  from server  3
Async set command recieved on  2  from server  3
server 3 is handling client 10.0.0.180:53091
server 3 is handling client 10.0.0.180:53092
update version in server 1
update version in server 2
update version in server 1
server  1  is broadcasting
Async set command recieved on  3  from server  1
Async set command recieved on  2  from server  1
server 1 is handling client 10.0.0.180:53093
server 2 is handling client 10.0.0.180:53094
update version in server 2
update version in server 3
server 1 is handling client 10.0.0.180:53095
server 2 is handling client 10.0.0.180:53097
update version in server 2
server  2  is broadcasting
Async set command recieved on  1  from server  2
Async set command recieved on  3  from server  2
server 3 is handling client 10.0.0.180:53098
server 1 is handling client 10.0.0.180:53099
update version in server 3
update version in server 1
update version in server 1
server  1  is broadcasting
Async set command recieved on  3  from server  1
Async set command recieved on  2  from server  1
server 2 is handling client 10.0.0.180:53100
read version - 3 is waiting on server 2 that has v
read version - 3 is waiting on server 2 that has v
ersion 2
update version in server 3
```

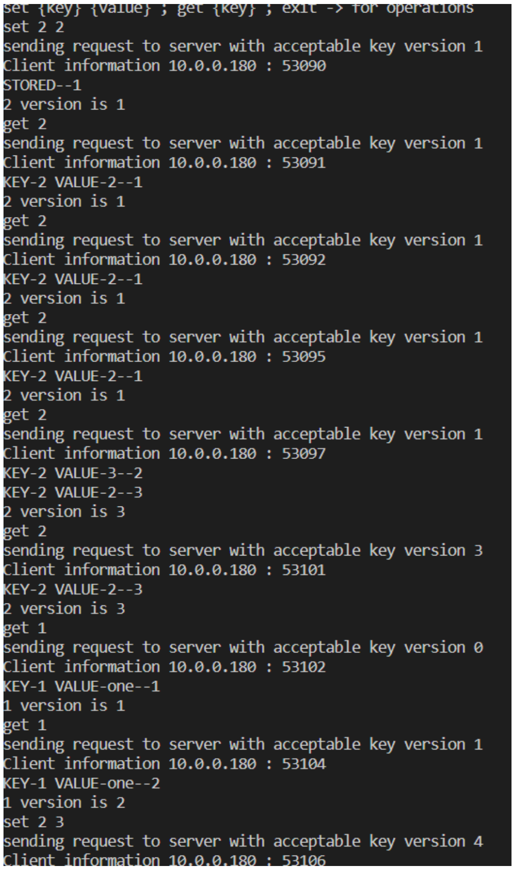


```
update version in server 3
update version in server 2
server 1 is handling client 10.0.0.180:53101
server 3 is handling client 10.0.0.180:53102
server 1 is handling client 10.0.0.180:53104
update version in server 3
server  3  is broadcasting
Async set command recieved on  1  from server  3
Async set command recieved on  2  from server  3
server 1 is handling client 10.0.0.180:53105
update version in server 2
update version in server 1
server 1 is handling client 10.0.0.180:53106
server 2 is handling client 10.0.0.180:53107
server 2 is handling client 10.0.0.180:53108
server 3 is handling client 10.0.0.180:53111
update version in server 1
server  1  is broadcasting
Async set command recieved on  3  from server  1
Async set command recieved on  2  from server  1
server 3 is handling client 10.0.0.180:53112
read version - 4 is waiting on server 3 that has v
ersion 3
read version - 4 is waiting on server 3 that has v
ersion 3
read version - 4 is waiting on server 3 that has v
ersion 3
update version in server 3
update version in server 2
server 1 is handling client 10.0.0.180:53113
```

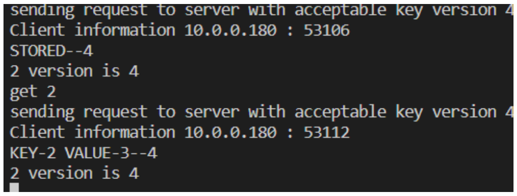


```
set {key} {value} ; get {key} ; exit -> for ope
rations
set 1 1
sending request to server with acceptable key v
ersion 1
KEY-1 VALUE-1--1
1 version is 1
set 2 3
sending request to server with acceptable key v
ersion 1
Client information 10.0.0.180 : 53094
STORED--2
2 version is 2
set 1 one
sending request to server with acceptable key v
ersion 2
Client information 10.0.0.180 : 53098
STORED--2
1 version is 2
get 1
sending request to server with acceptable key v
ersion 2
Client information 10.0.0.180 : 53105
KEY-1 VALUE-one--2
1 version is 2
get 2
sending request to server with acceptable key v
ersion 2
Client information 10.0.0.180 : 53107
KEY-2 VALUE-2--3
2 version is 3
get 2
sending request to server with acceptable key v
ersion 3
Client information 10.0.0.180 : 53108
KEY-2 VALUE-2--3
2 version is 3
```

client 1 output



```
set {key} {value} ; get {key} ; exit -> for operations
set 2 2
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53090
STORED--1
2 version is 1
get 2
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53091
KEY-2 VALUE-2--1
2 version is 1
get 2
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53092
KEY-2 VALUE-2--1
2 version is 1
get 2
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53095
KEY-2 VALUE-2--1
2 version is 1
get 2
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53097
KEY-2 VALUE-3--2
KEY-2 VALUE-2--3
2 version is 3
get 2
sending request to server with acceptable key version 3
Client information 10.0.0.180 : 53101
KEY-2 VALUE-2--3
2 version is 3
get 1
sending request to server with acceptable key version 0
Client information 10.0.0.180 : 53102
KEY-1 VALUE-one--1
1 version is 1
get 1
sending request to server with acceptable key version 1
Client information 10.0.0.180 : 53104
KEY-1 VALUE-one--2
1 version is 2
set 2 3
sending request to server with acceptable key version 4
Client information 10.0.0.180 : 53106
```



```
sending request to server with acceptable key version 4
Client information 10.0.0.180 : 53106
STORED--4
2 version is 4
get 2
sending request to server with acceptable key version 4
Client information 10.0.0.180 : 53112
KEY-2 VALUE-3--4
2 version is 4
```

Client 2 outputs

As you can see in the server side outputs, reads wait for the server version to get updated and then they are returned.