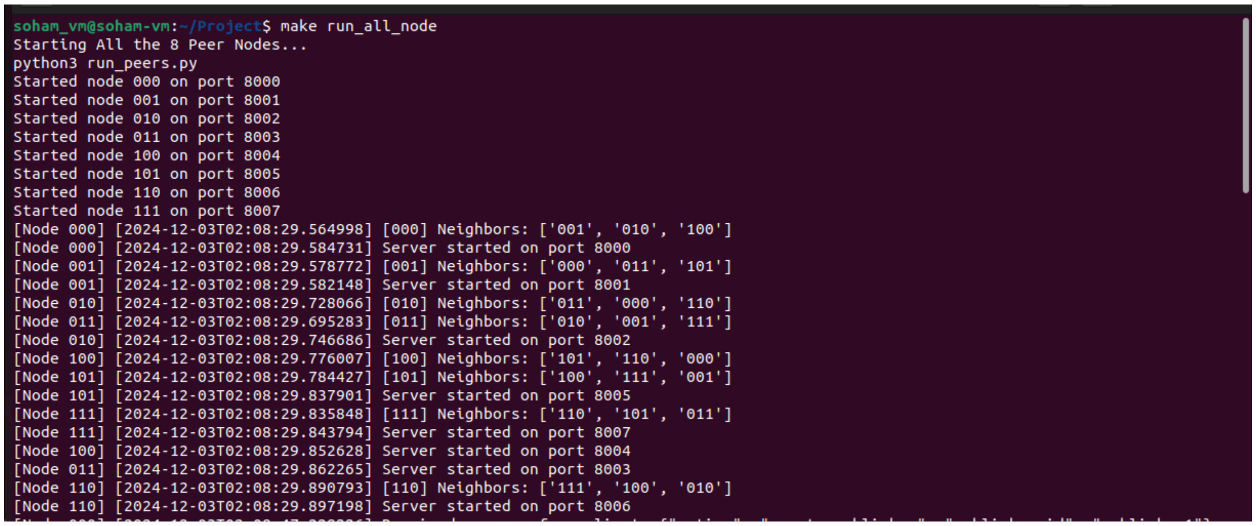
End of semester project

Name:Soham Sonar
Report

Evaluation

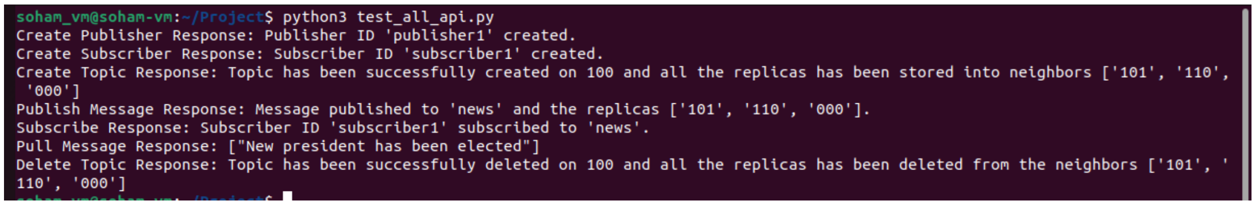
1. Deploying 8 peers. They can be set up on the same machine or different machines.
    a. Ensure all APIs are working properly.

1. Started All the nodes



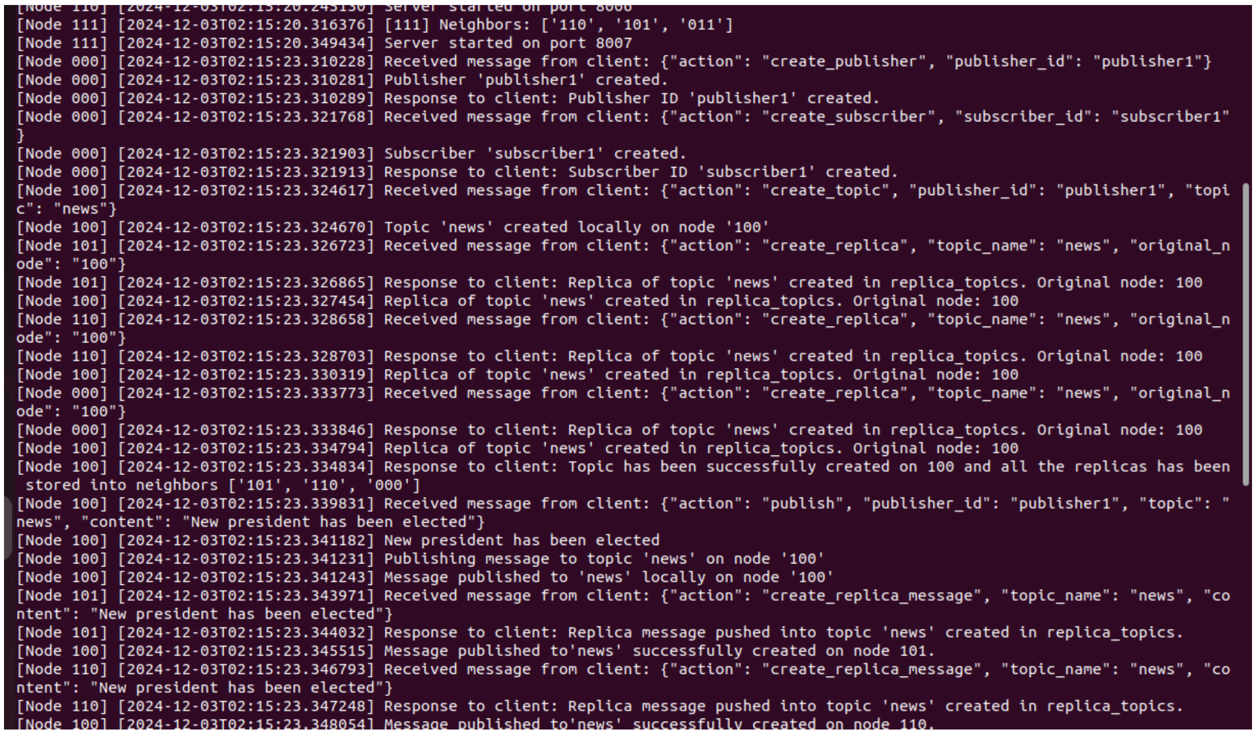
```
soham_vm@soham-vm:~/Project$ make run_all_node
Starting All the 8 Peer Nodes...
python3 run_peers.py
Started node 000 on port 8000
Started node 001 on port 8001
Started node 010 on port 8002
Started node 011 on port 8003
Started node 100 on port 8004
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T02:08:29.564998] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T02:08:29.584731] Server started on port 8000
[Node 001] [2024-12-03T02:08:29.578772] [001] Neighbors: ['000', '011', '101']
[Node 001] [2024-12-03T02:08:29.582148] Server started on port 8001
[Node 010] [2024-12-03T02:08:29.728066] [010] Neighbors: ['011', '000', '110']
[Node 011] [2024-12-03T02:08:29.695283] [011] Neighbors: ['010', '001', '111']
[Node 010] [2024-12-03T02:08:29.746686] Server started on port 8002
[Node 100] [2024-12-03T02:08:29.776007] [100] Neighbors: ['101', '110', '000']
[Node 101] [2024-12-03T02:08:29.784427] [101] Neighbors: ['100', '111', '001']
[Node 101] [2024-12-03T02:08:29.837901] Server started on port 8005
[Node 111] [2024-12-03T02:08:29.835848] [111] Neighbors: ['110', '101', '011']
[Node 111] [2024-12-03T02:08:29.843794] Server started on port 8007
[Node 100] [2024-12-03T02:08:29.852628] Server started on port 8004
[Node 011] [2024-12-03T02:08:29.862265] Server started on port 8003
[Node 110] [2024-12-03T02:08:29.890793] [110] Neighbors: ['111', '100', '010']
[Node 110] [2024-12-03T02:08:29.897198] Server started on port 8006
```

2. All APIs working properly



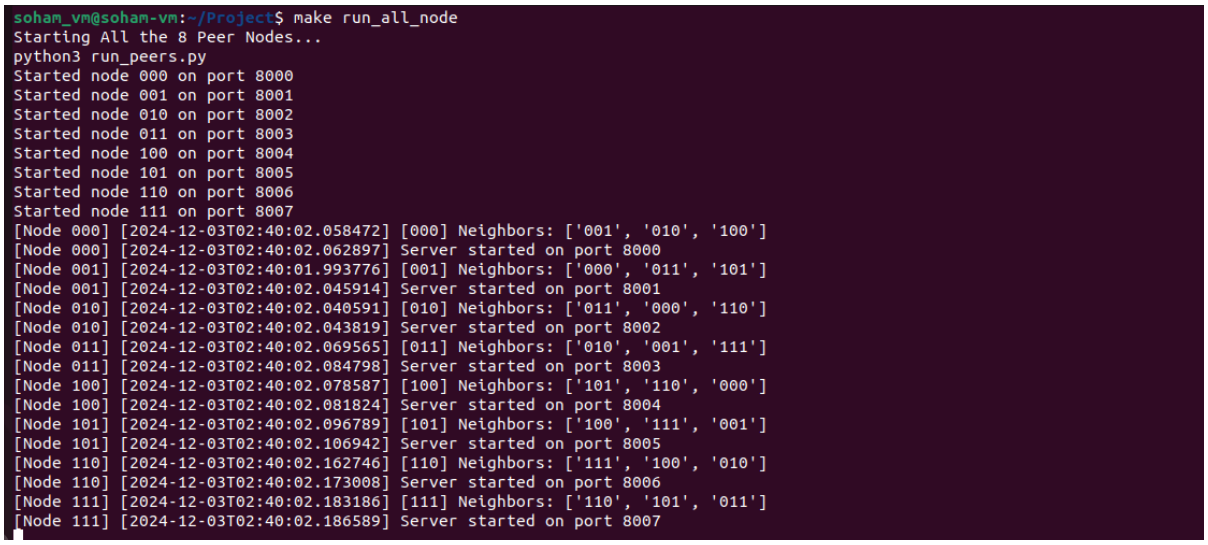
```
soham_vm@soham-vm:~/Project$ python3 test_all_api.py
Create Publisher Response: Publisher ID 'publisher1' created.
Create Subscriber Response: Subscriber ID 'subscriber1' created.
Create Topic Response: Topic has been successfully created on 100 and all the replicas has been stored into neighbors ['101', '110',
 '000']
Publish Message Response: Message published to 'news' and the replicas ['101', '110', '000'].
Subscribe Response: Subscriber ID 'subscriber1' subscribed to 'news'.
Pull Message Response: ["New president has been elected"]
Delete Topic Response: Topic has been successfully deleted on 100 and all the replicas has been deleted from the neighbors ['101', '
110', '000']
```

    b. Ensure multiple peer nodes can simultaneously publish and subscribe to a topic.

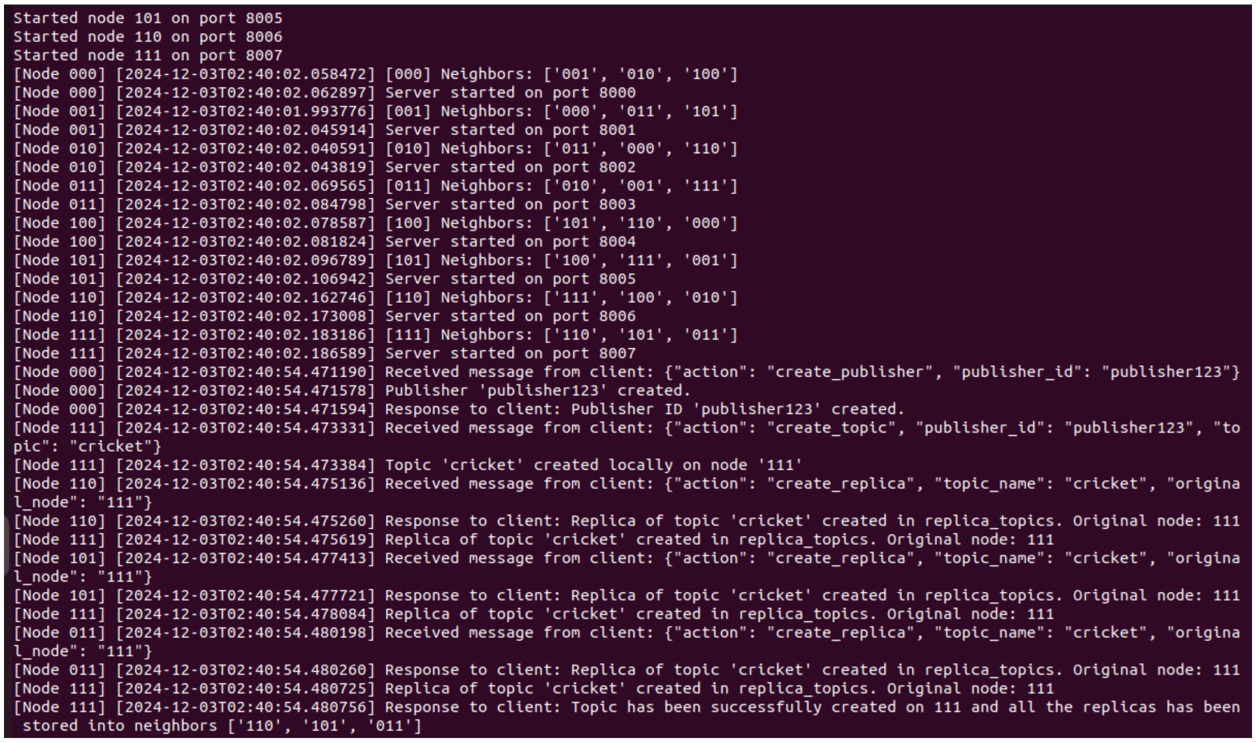


```
[Node 110] [2024-12-03T02:15:20.245130] Server started on port 8006
[Node 111] [2024-12-03T02:15:20.316376] [111] Neighbors: ['110', '101', '011']
[Node 111] [2024-12-03T02:15:20.349434] Server started on port 8007
[Node 000] [2024-12-03T02:15:23.310228] Received message from client: {"action": "create_publisher", "publisher_id": "publisher1"}
[Node 000] [2024-12-03T02:15:23.310281] Publisher 'publisher1' created.
[Node 000] [2024-12-03T02:15:23.310289] Response to client: Publisher ID 'publisher1' created.
[Node 000] [2024-12-03T02:15:23.321768] Received message from client: {"action": "create_subscriber", "subscriber_id": "subscriber1"
}
[Node 000] [2024-12-03T02:15:23.321903] Subscriber 'subscriber1' created.
[Node 000] [2024-12-03T02:15:23.321913] Response to client: Subscriber ID 'subscriber1' created.
[Node 100] [2024-12-03T02:15:23.324617] Received message from client: {"action": "create_topic", "publisher_id": "publisher1", "topi
c": "news"}
[Node 100] [2024-12-03T02:15:23.324670] Topic 'news' created locally on node '100'
[Node 101] [2024-12-03T02:15:23.326723] Received message from client: {"action": "create_replica", "topic_name": "news", "original_n
ode": "100"}
[Node 101] [2024-12-03T02:15:23.326865] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T02:15:23.327454] Replica of topic 'news' created in replica_topics. Original node: 100
[Node 110] [2024-12-03T02:15:23.328658] Received message from client: {"action": "create_replica", "topic_name": "news", "original_n
ode": "100"}
[Node 110] [2024-12-03T02:15:23.328703] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T02:15:23.330319] Replica of topic 'news' created in replica_topics. Original node: 100
[Node 000] [2024-12-03T02:15:23.333773] Received message from client: {"action": "create_replica", "topic_name": "news", "original_n
ode": "100"}
[Node 000] [2024-12-03T02:15:23.333846] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T02:15:23.334794] Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T02:15:23.334834] Response to client: Topic has been successfully created on 100 and all the replicas has been
 stored into neighbors ['101', '110', '000']
[Node 100] [2024-12-03T02:15:23.339831] Received message from client: {"action": "publish", "publisher_id": "publisher1", "topic": "
news", "content": "New president has been elected"}
[Node 100] [2024-12-03T02:15:23.341182] New president has been elected
[Node 100] [2024-12-03T02:15:23.341231] Publishing message to topic 'news' on node '100'
[Node 100] [2024-12-03T02:15:23.341243] Message published to 'news' locally on node '100'
[Node 101] [2024-12-03T02:15:23.343971] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 101] [2024-12-03T02:15:23.344032] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T02:15:23.345515] Message published to'news' successfully created on node 101.
[Node 110] [2024-12-03T02:15:23.346793] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 110] [2024-12-03T02:15:23.347248] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T02:15:23.348054] Message published to'news' successfully created on node 110.
```

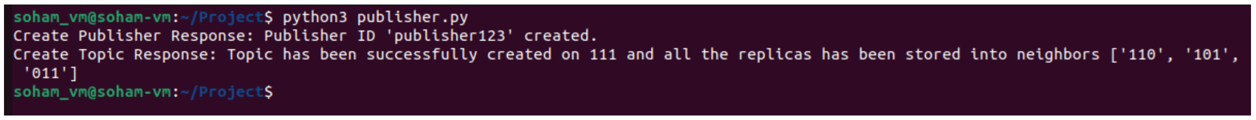
2. Deploy enough peers. Prove topics can be correctly replicated on the right node.



```
soham_vm@soham-vm:~/Project$ make run_all_node
Starting All the 8 Peer Nodes...
python3 run_peers.py
Started node 000 on port 8000
Started node 001 on port 8001
Started node 010 on port 8002
Started node 011 on port 8003
Started node 100 on port 8004
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T02:40:02.058472] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T02:40:02.062897] Server started on port 8000
[Node 001] [2024-12-03T02:40:01.993776] [001] Neighbors: ['000', '011', '101']
[Node 001] [2024-12-03T02:40:02.045914] Server started on port 8001
[Node 010] [2024-12-03T02:40:02.040591] [010] Neighbors: ['011', '000', '110']
[Node 010] [2024-12-03T02:40:02.043819] Server started on port 8002
[Node 011] [2024-12-03T02:40:02.069565] [011] Neighbors: ['010', '001', '111']
[Node 011] [2024-12-03T02:40:02.084798] Server started on port 8003
[Node 100] [2024-12-03T02:40:02.078587] [100] Neighbors: ['101', '110', '000']
[Node 100] [2024-12-03T02:40:02.081824] Server started on port 8004
[Node 101] [2024-12-03T02:40:02.096789] [101] Neighbors: ['100', '111', '001']
[Node 101] [2024-12-03T02:40:02.106942] Server started on port 8005
[Node 110] [2024-12-03T02:40:02.162746] [110] Neighbors: ['111', '100', '010']
[Node 110] [2024-12-03T02:40:02.173008] Server started on port 8006
[Node 111] [2024-12-03T02:40:02.183186] [111] Neighbors: ['110', '101', '011']
[Node 111] [2024-12-03T02:40:02.186589] Server started on port 8007
```

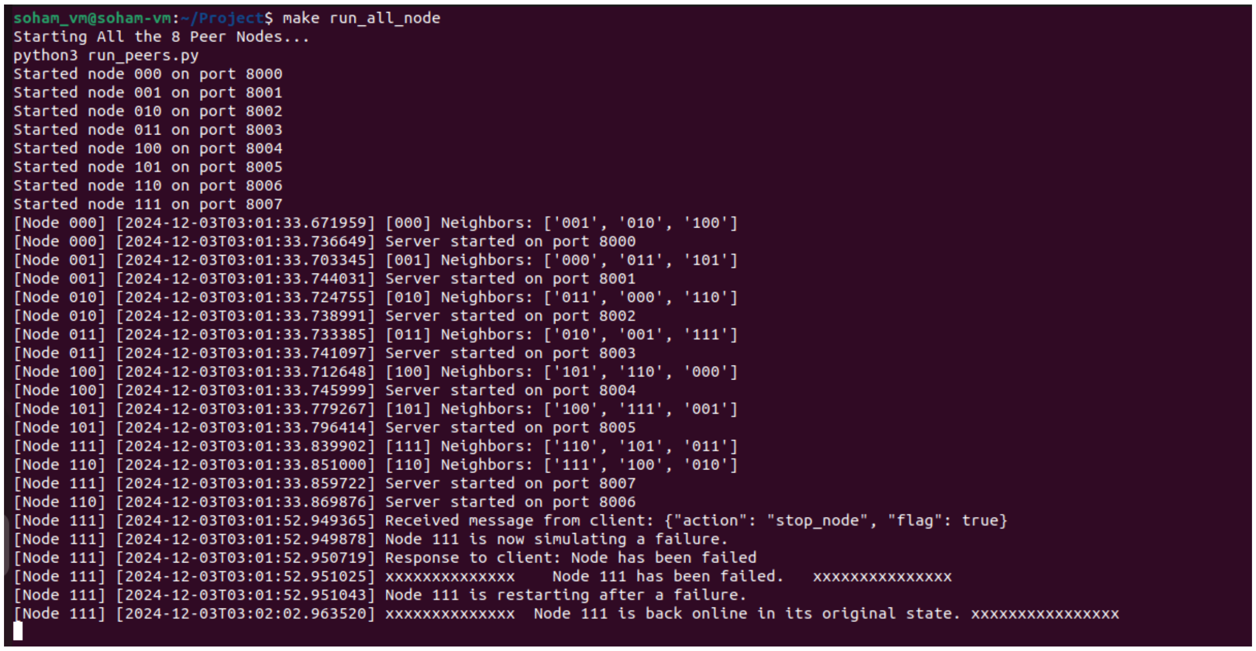


```
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T02:40:02.058472] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T02:40:02.062897] Server started on port 8000
[Node 001] [2024-12-03T02:40:01.993776] [001] Neighbors: ['000', '011', '101']
[Node 001] [2024-12-03T02:40:02.045914] Server started on port 8001
[Node 010] [2024-12-03T02:40:02.040591] [010] Neighbors: ['011', '000', '110']
[Node 010] [2024-12-03T02:40:02.043819] Server started on port 8002
[Node 011] [2024-12-03T02:40:02.069565] [011] Neighbors: ['010', '001', '111']
[Node 011] [2024-12-03T02:40:02.084798] Server started on port 8003
[Node 100] [2024-12-03T02:40:02.078587] [100] Neighbors: ['101', '110', '000']
[Node 100] [2024-12-03T02:40:02.081824] Server started on port 8004
[Node 101] [2024-12-03T02:40:02.096789] [101] Neighbors: ['100', '111', '001']
[Node 101] [2024-12-03T02:40:02.106942] Server started on port 8005
[Node 110] [2024-12-03T02:40:02.162746] [110] Neighbors: ['111', '100', '010']
[Node 110] [2024-12-03T02:40:02.173008] Server started on port 8006
[Node 111] [2024-12-03T02:40:02.183186] [111] Neighbors: ['110', '101', '011']
[Node 111] [2024-12-03T02:40:02.186589] Server started on port 8007
[Node 000] [2024-12-03T02:40:54.471190] Received message from client: {"action": "create_publisher", "publisher_id": "publisher123"}
[Node 000] [2024-12-03T02:40:54.471578] Publisher 'publisher123' created.
[Node 000] [2024-12-03T02:40:54.471594] Response to client: Publisher ID 'publisher123' created.
[Node 111] [2024-12-03T02:40:54.473331] Received message from client: {"action": "create_topic", "publisher_id": "publisher123", "to
pic": "cricket"}
[Node 111] [2024-12-03T02:40:54.473384] Topic 'cricket' created locally on node '111'
[Node 110] [2024-12-03T02:40:54.475136] Received message from client: {"action": "create_replica", "topic_name": "cricket", "origina
l_node": "111"}
[Node 110] [2024-12-03T02:40:54.475260] Response to client: Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 111] [2024-12-03T02:40:54.475619] Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 101] [2024-12-03T02:40:54.477413] Received message from client: {"action": "create_replica", "topic_name": "cricket", "origina
l_node": "111"}
[Node 101] [2024-12-03T02:40:54.477721] Response to client: Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 111] [2024-12-03T02:40:54.478084] Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 011] [2024-12-03T02:40:54.480198] Received message from client: {"action": "create_replica", "topic_name": "cricket", "origina
l_node": "111"}
[Node 011] [2024-12-03T02:40:54.480260] Response to client: Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 111] [2024-12-03T02:40:54.480725] Replica of topic 'cricket' created in replica_topics. Original node: 111
[Node 111] [2024-12-03T02:40:54.480756] Response to client: Topic has been successfully created on 111 and all the replicas has been
 stored into neighbors ['110', '101', '011']
```



```
soham_vm@soham-vm:~/Project$ python3 publisher.py
Create Publisher Response: Publisher ID 'publisher123' created.
Create Topic Response: Topic has been successfully created on 111 and all the replicas has been stored into neighbors ['110', '101',
 '011']
soham_vm@soham-vm:~/Project$
```

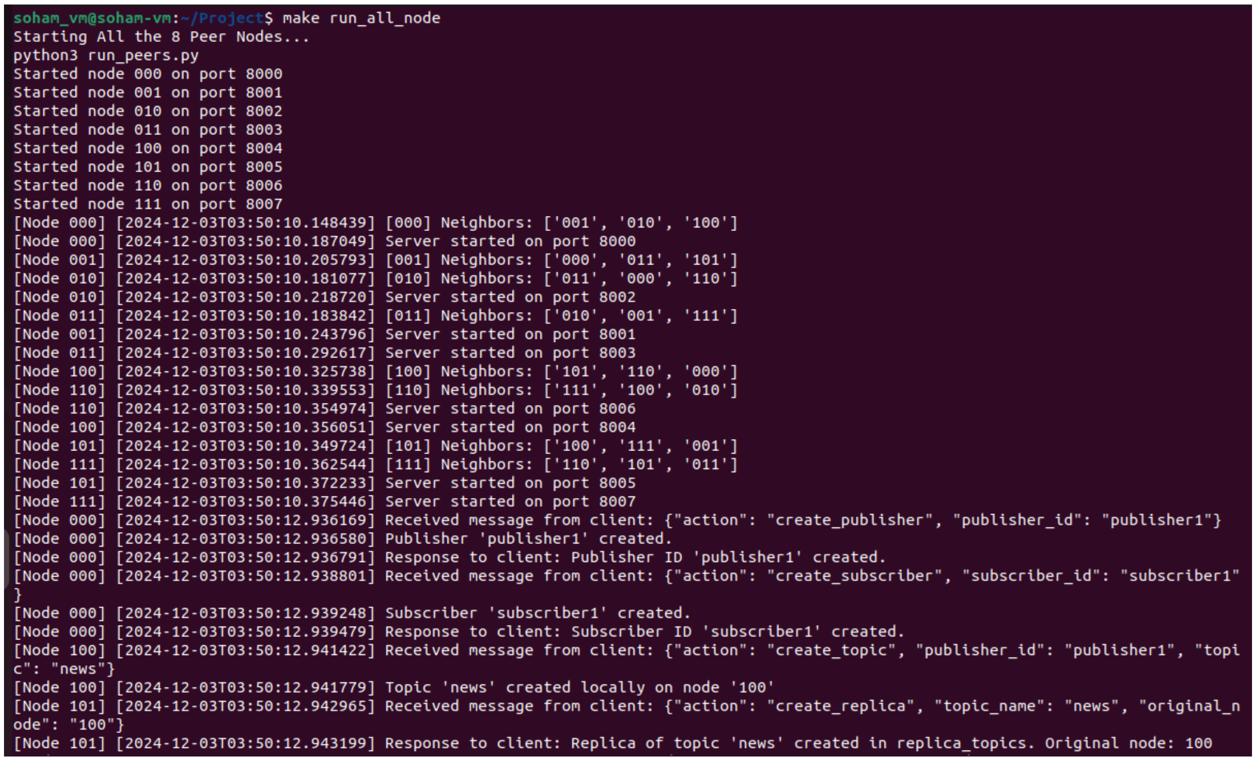
3. Deploy enough peers. Prove a failed node can recover to the state before the failure.



```
soham_vm@soham-vm:~/Project$ make run_all_node
Starting All the 8 Peer Nodes...
python3 run_peers.py
Started node 000 on port 8000
Started node 001 on port 8001
Started node 010 on port 8002
Started node 011 on port 8003
Started node 100 on port 8004
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T03:01:33.671959] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T03:01:33.736649] Server started on port 8000
[Node 001] [2024-12-03T03:01:33.703345] [001] Neighbors: ['000', '011', '101']
[Node 001] [2024-12-03T03:01:33.744031] Server started on port 8001
[Node 010] [2024-12-03T03:01:33.724755] [010] Neighbors: ['011', '000', '110']
[Node 010] [2024-12-03T03:01:33.738991] Server started on port 8002
[Node 011] [2024-12-03T03:01:33.733385] [011] Neighbors: ['010', '001', '111']
[Node 011] [2024-12-03T03:01:33.741097] Server started on port 8003
[Node 100] [2024-12-03T03:01:33.712648] [100] Neighbors: ['101', '110', '000']
[Node 100] [2024-12-03T03:01:33.745999] Server started on port 8004
[Node 101] [2024-12-03T03:01:33.779267] [101] Neighbors: ['100', '111', '001']
[Node 101] [2024-12-03T03:01:33.796414] Server started on port 8005
[Node 111] [2024-12-03T03:01:33.839902] [111] Neighbors: ['110', '101', '011']
[Node 110] [2024-12-03T03:01:33.851000] [110] Neighbors: ['111', '100', '010']
[Node 111] [2024-12-03T03:01:33.859722] Server started on port 8007
[Node 110] [2024-12-03T03:01:33.869876] Server started on port 8006
[Node 111] [2024-12-03T03:01:52.949365] Received message from client: {"action": "stop_node", "flag": true}
[Node 111] [2024-12-03T03:01:52.949878] Node 111 is now simulating a failure.
[Node 111] [2024-12-03T03:01:52.950719] Response to client: Node has been failed
[Node 111] [2024-12-03T03:01:52.951025] xxxxxxxxxxxxxx    Node 111 has been failed.   xxxxxxxxxxxxxxx
[Node 111] [2024-12-03T03:01:52.951043] Node 111 is restarting after a failure.
[Node 111] [2024-12-03T03:02:02.963520] xxxxxxxxxxxxxx  Node 111 is back online in its original state. xxxxxxxxxxxxxxxx
```

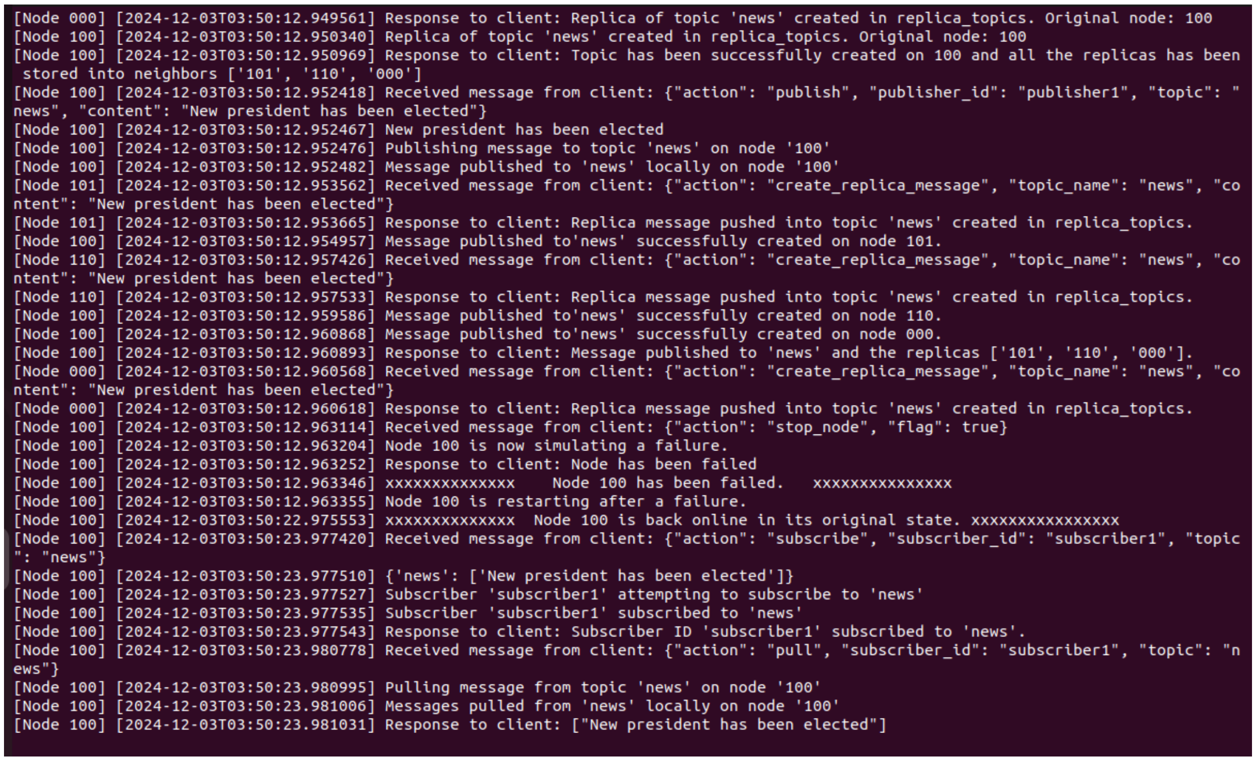
4. Deploy enough peers. Prove when a node comes online, it can recover to the state before failure (if that’s the case) and fetch all topics belonging to it from other (if exists).

Running all nodes -



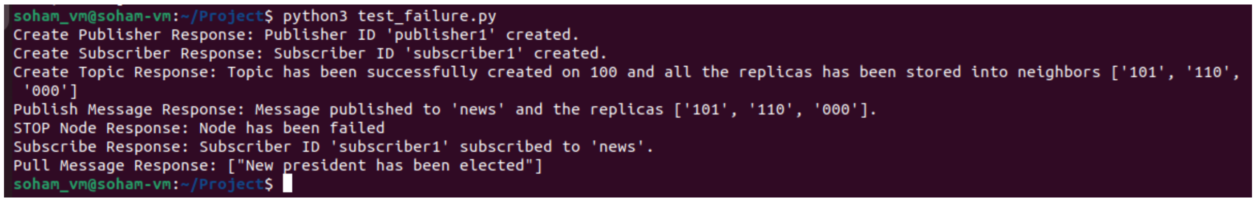
```
soham_vm@soham-vm:~/Project$ make run_all_node
Starting All the 8 Peer Nodes...
python3 run_peers.py
Started node 000 on port 8000
Started node 001 on port 8001
Started node 010 on port 8002
Started node 011 on port 8003
Started node 100 on port 8004
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T03:50:10.148439] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T03:50:10.187049] Server started on port 8000
[Node 001] [2024-12-03T03:50:10.205793] [001] Neighbors: ['000', '011', '101']
[Node 010] [2024-12-03T03:50:10.181077] [010] Neighbors: ['011', '000', '110']
[Node 010] [2024-12-03T03:50:10.218720] Server started on port 8002
[Node 011] [2024-12-03T03:50:10.183842] [011] Neighbors: ['010', '001', '111']
[Node 001] [2024-12-03T03:50:10.243796] Server started on port 8001
[Node 011] [2024-12-03T03:50:10.292617] Server started on port 8003
[Node 100] [2024-12-03T03:50:10.325738] [100] Neighbors: ['101', '110', '000']
[Node 110] [2024-12-03T03:50:10.339553] [110] Neighbors: ['111', '100', '010']
[Node 110] [2024-12-03T03:50:10.354974] Server started on port 8006
[Node 100] [2024-12-03T03:50:10.356051] Server started on port 8004
[Node 101] [2024-12-03T03:50:10.349724] [101] Neighbors: ['100', '111', '001']
[Node 111] [2024-12-03T03:50:10.362544] [111] Neighbors: ['110', '101', '011']
[Node 101] [2024-12-03T03:50:10.372233] Server started on port 8005
[Node 111] [2024-12-03T03:50:10.375446] Server started on port 8007
[Node 000] [2024-12-03T03:50:12.936169] Received message from client: {"action": "create_publisher", "publisher_id": "publisher1"}
[Node 000] [2024-12-03T03:50:12.936580] Publisher 'publisher1' created.
[Node 000] [2024-12-03T03:50:12.936791] Response to client: Publisher ID 'publisher1' created.
[Node 000] [2024-12-03T03:50:12.938801] Received message from client: {"action": "create_subscriber", "subscriber_id": "subscriber1"
}
[Node 000] [2024-12-03T03:50:12.939248] Subscriber 'subscriber1' created.
[Node 000] [2024-12-03T03:50:12.939479] Response to client: Subscriber ID 'subscriber1' created.
[Node 100] [2024-12-03T03:50:12.941422] Received message from client: {"action": "create_topic", "publisher_id": "publisher1", "topi
c": "news"}
[Node 100] [2024-12-03T03:50:12.941779] Topic 'news' created locally on node '100'
[Node 101] [2024-12-03T03:50:12.942965] Received message from client: {"action": "create_replica", "topic_name": "news", "original_n
ode": "100"}
[Node 101] [2024-12-03T03:50:12.943199] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
```

initially creating topics and publish message, then server is failed and after server is restored subscribing and pulling the original topic and message.



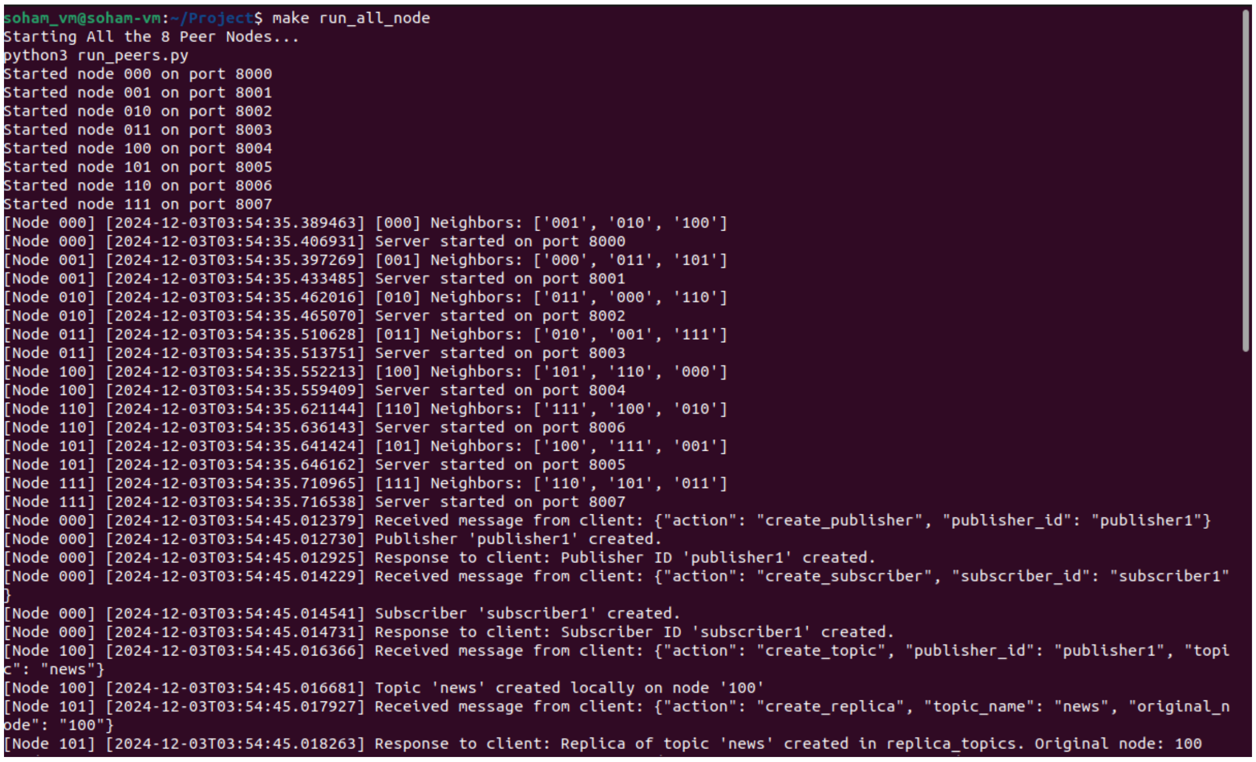
```
[Node 000] [2024-12-03T03:50:12.949561] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T03:50:12.950340] Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T03:50:12.950969] Response to client: Topic has been successfully created on 100 and all the replicas has been
 stored into neighbors ['101', '110', '000']
[Node 100] [2024-12-03T03:50:12.952418] Received message from client: {"action": "publish", "publisher_id": "publisher1", "topic": "
news", "content": "New president has been elected"}
[Node 100] [2024-12-03T03:50:12.952467] New president has been elected
[Node 100] [2024-12-03T03:50:12.952476] Publishing message to topic 'news' on node '100'
[Node 100] [2024-12-03T03:50:12.952482] Message published to 'news' locally on node '100'
[Node 101] [2024-12-03T03:50:12.953562] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 101] [2024-12-03T03:50:12.953665] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:50:12.954957] Message published to'news' successfully created on node 101.
[Node 110] [2024-12-03T03:50:12.957426] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 110] [2024-12-03T03:50:12.957533] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:50:12.959586] Message published to'news' successfully created on node 110.
[Node 100] [2024-12-03T03:50:12.960868] Message published to'news' successfully created on node 000.
[Node 100] [2024-12-03T03:50:12.960893] Response to client: Message published to 'news' and the replicas ['101', '110', '000'].
[Node 000] [2024-12-03T03:50:12.960568] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 000] [2024-12-03T03:50:12.960618] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:50:12.963114] Received message from client: {"action": "stop_node", "flag": true}
[Node 100] [2024-12-03T03:50:12.963204] Node 100 is now simulating a failure.
[Node 100] [2024-12-03T03:50:12.963252] Response to client: Node has been failed
[Node 100] [2024-12-03T03:50:12.963346] xxxxxxxxxxxxxx    Node 100 has been failed.   xxxxxxxxxxxxxxx
[Node 100] [2024-12-03T03:50:12.963355] Node 100 is restarting after a failure.
[Node 100] [2024-12-03T03:50:22.975553] xxxxxxxxxxxxxx  Node 100 is back online in its original state. xxxxxxxxxxxxxxxx
[Node 100] [2024-12-03T03:50:23.977420] Received message from client: {"action": "subscribe", "subscriber_id": "subscriber1", "topic
": "news"}
[Node 100] [2024-12-03T03:50:23.977510] {'news': ['New president has been elected']}
[Node 100] [2024-12-03T03:50:23.977527] Subscriber 'subscriber1' attempting to subscribe to 'news'
[Node 100] [2024-12-03T03:50:23.977535] Subscriber 'subscriber1' subscribed to 'news'
[Node 100] [2024-12-03T03:50:23.977543] Response to client: Subscriber ID 'subscriber1' subscribed to 'news'.
[Node 100] [2024-12-03T03:50:23.980778] Received message from client: {"action": "pull", "subscriber_id": "subscriber1", "topic": "n
ews"}
[Node 100] [2024-12-03T03:50:23.980995] Pulling message from topic 'news' on node '100'
[Node 100] [2024-12-03T03:50:23.981006] Messages pulled from 'news' locally on node '100'
[Node 100] [2024-12-03T03:50:23.981031] Response to client: ["New president has been elected"]
```

On client:



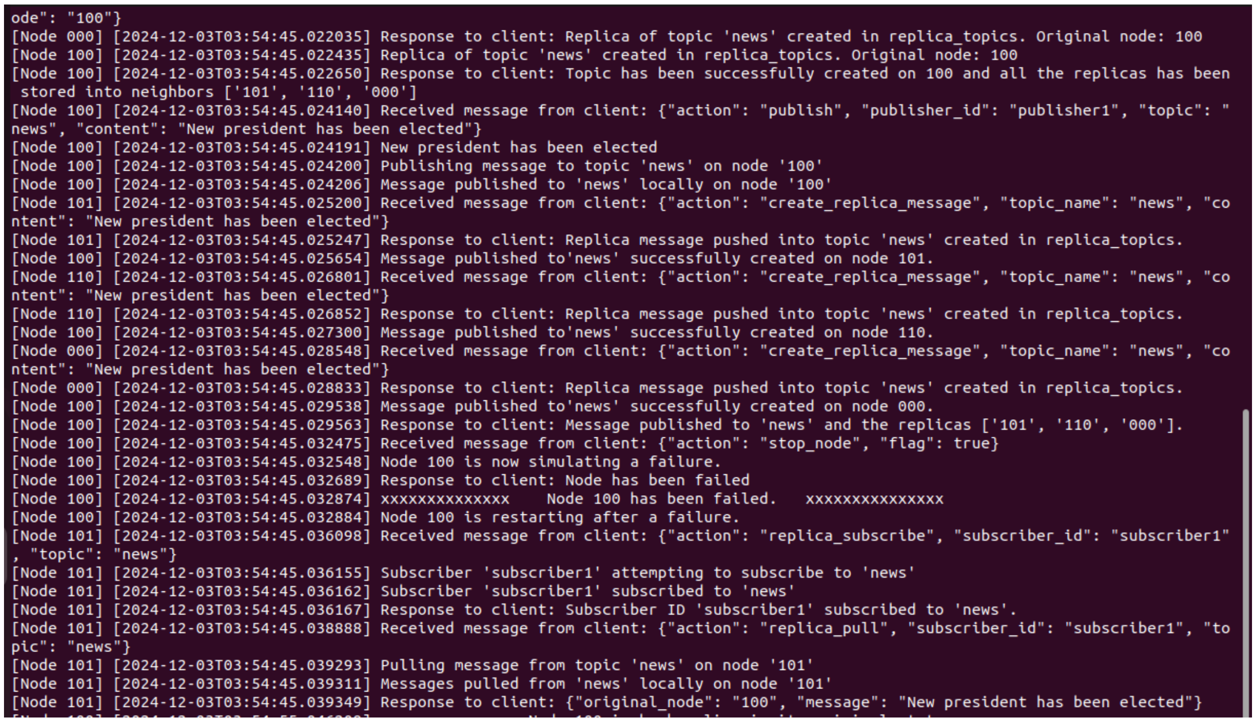
```
soham_vm@soham-vm:~/Project$ python3 test_failure.py
Create Publisher Response: Publisher ID 'publisher1' created.
Create Subscriber Response: Subscriber ID 'subscriber1' created.
Create Topic Response: Topic has been successfully created on 100 and all the replicas has been stored into neighbors ['101', '110',
 '000']
Publish Message Response: Message published to 'news' and the replicas ['101', '110', '000'].
STOP Node Response: Node has been failed
Subscribe Response: Subscriber ID 'subscriber1' subscribed to 'news'.
Pull Message Response: ["New president has been elected"]
soham_vm@soham-vm:~/Project$
```

5. Deploy enough peers. Prove if the “right” node for a given topic is offline, you can correctly find its replica and talk to the host node.
   Peers deployed and topics are created and message is published.



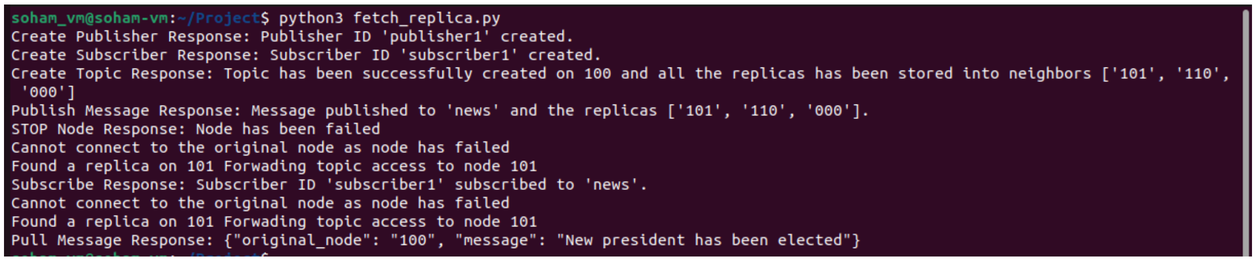
```
soham_vm@soham-vm:~/Project$ make run_all_node
Starting All the 8 Peer Nodes...
python3 run_peers.py
Started node 000 on port 8000
Started node 001 on port 8001
Started node 010 on port 8002
Started node 011 on port 8003
Started node 100 on port 8004
Started node 101 on port 8005
Started node 110 on port 8006
Started node 111 on port 8007
[Node 000] [2024-12-03T03:54:35.389463] [000] Neighbors: ['001', '010', '100']
[Node 000] [2024-12-03T03:54:35.406931] Server started on port 8000
[Node 001] [2024-12-03T03:54:35.397269] [001] Neighbors: ['000', '011', '101']
[Node 001] [2024-12-03T03:54:35.433485] Server started on port 8001
[Node 010] [2024-12-03T03:54:35.462016] [010] Neighbors: ['011', '000', '110']
[Node 010] [2024-12-03T03:54:35.465070] Server started on port 8002
[Node 011] [2024-12-03T03:54:35.510628] [011] Neighbors: ['010', '001', '111']
[Node 011] [2024-12-03T03:54:35.513751] Server started on port 8003
[Node 100] [2024-12-03T03:54:35.552213] [100] Neighbors: ['101', '110', '000']
[Node 100] [2024-12-03T03:54:35.559409] Server started on port 8004
[Node 110] [2024-12-03T03:54:35.621144] [110] Neighbors: ['111', '100', '010']
[Node 110] [2024-12-03T03:54:35.636143] Server started on port 8006
[Node 101] [2024-12-03T03:54:35.641424] [101] Neighbors: ['100', '111', '001']
[Node 101] [2024-12-03T03:54:35.646162] Server started on port 8005
[Node 111] [2024-12-03T03:54:35.710965] [111] Neighbors: ['110', '101', '011']
[Node 111] [2024-12-03T03:54:35.716538] Server started on port 8007
[Node 000] [2024-12-03T03:54:45.012379] Received message from client: {"action": "create_publisher", "publisher_id": "publisher1"}
[Node 000] [2024-12-03T03:54:45.012730] Publisher 'publisher1' created.
[Node 000] [2024-12-03T03:54:45.012925] Response to client: Publisher ID 'publisher1' created.
[Node 000] [2024-12-03T03:54:45.014229] Received message from client: {"action": "create_subscriber", "subscriber_id": "subscriber1"
}
[Node 000] [2024-12-03T03:54:45.014541] Subscriber 'subscriber1' created.
[Node 000] [2024-12-03T03:54:45.014731] Response to client: Subscriber ID 'subscriber1' created.
[Node 100] [2024-12-03T03:54:45.016366] Received message from client: {"action": "create_topic", "publisher_id": "publisher1", "topi
c": "news"}
[Node 100] [2024-12-03T03:54:45.016681] Topic 'news' created locally on node '100'
[Node 101] [2024-12-03T03:54:45.017927] Received message from client: {"action": "create_replica", "topic_name": "news", "original_n
ode": "100"}
[Node 101] [2024-12-03T03:54:45.018263] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
```

Node is Failed, And when subscribed it redirects to neighbor node and displays the replica node.



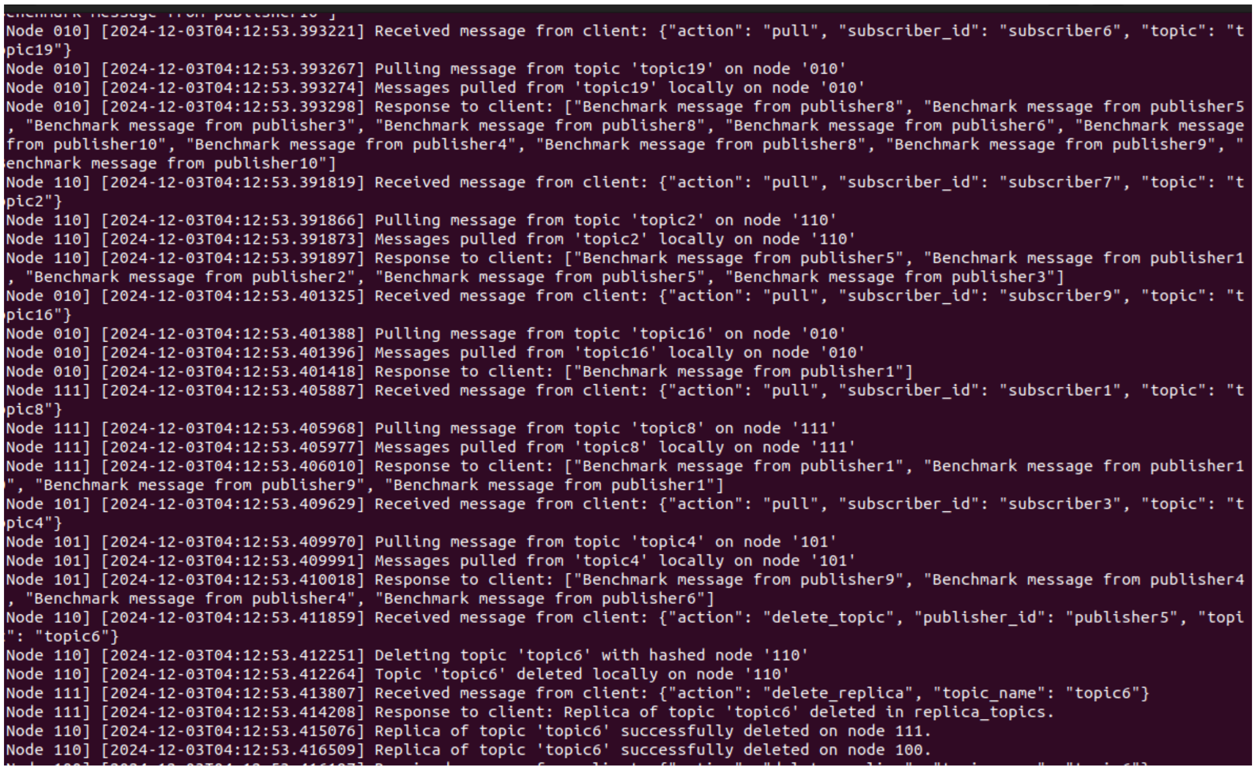
```
ode": "100"}
[Node 000] [2024-12-03T03:54:45.022035] Response to client: Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T03:54:45.022435] Replica of topic 'news' created in replica_topics. Original node: 100
[Node 100] [2024-12-03T03:54:45.022650] Response to client: Topic has been successfully created on 100 and all the replicas has been
 stored into neighbors ['101', '110', '000']
[Node 100] [2024-12-03T03:54:45.024140] Received message from client: {"action": "publish", "publisher_id": "publisher1", "topic": "
news", "content": "New president has been elected"}
[Node 100] [2024-12-03T03:54:45.024191] New president has been elected
[Node 100] [2024-12-03T03:54:45.024200] Publishing message to topic 'news' on node '100'
[Node 100] [2024-12-03T03:54:45.024206] Message published to 'news' locally on node '100'
[Node 101] [2024-12-03T03:54:45.025200] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 101] [2024-12-03T03:54:45.025247] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:54:45.025654] Message published to'news' successfully created on node 101.
[Node 110] [2024-12-03T03:54:45.026801] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 110] [2024-12-03T03:54:45.026852] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:54:45.027300] Message published to'news' successfully created on node 110.
[Node 000] [2024-12-03T03:54:45.028548] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "New president has been elected"}
[Node 000] [2024-12-03T03:54:45.028833] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T03:54:45.029538] Message published to'news' successfully created on node 000.
[Node 100] [2024-12-03T03:54:45.029563] Response to client: Message published to 'news' and the replicas ['101', '110', '000'].
[Node 100] [2024-12-03T03:54:45.032475] Received message from client: {"action": "stop_node", "flag": true}
[Node 100] [2024-12-03T03:54:45.032548] Node 100 is now simulating a failure.
[Node 100] [2024-12-03T03:54:45.032689] Response to client: Node has been failed
[Node 100] [2024-12-03T03:54:45.032874] xxxxxxxxxxxxxx    Node 100 has been failed.   xxxxxxxxxxxxxxx
[Node 100] [2024-12-03T03:54:45.032884] Node 100 is restarting after a failure.
[Node 101] [2024-12-03T03:54:45.036098] Received message from client: {"action": "replica_subscribe", "subscriber_id": "subscriber1"
, "topic": "news"}
[Node 101] [2024-12-03T03:54:45.036155] Subscriber 'subscriber1' attempting to subscribe to 'news'
[Node 101] [2024-12-03T03:54:45.036162] Subscriber 'subscriber1' subscribed to 'news'
[Node 101] [2024-12-03T03:54:45.036167] Response to client: Subscriber ID 'subscriber1' subscribed to 'news'.
[Node 101] [2024-12-03T03:54:45.038888] Received message from client: {"action": "replica_pull", "subscriber_id": "subscriber1", "to
pic": "news"}
[Node 101] [2024-12-03T03:54:45.039293] Pulling message from topic 'news' on node '101'
[Node 101] [2024-12-03T03:54:45.039311] Messages pulled from 'news' locally on node '101'
[Node 101] [2024-12-03T03:54:45.039349] Response to client: {"original_node": "100", "message": "New president has been elected"}
```

Client Response:

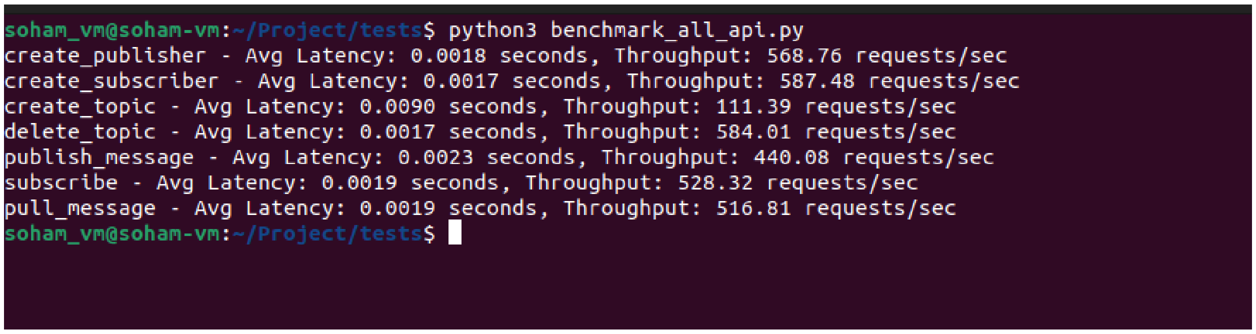


```
soham_vm@soham-vm:~/Project$ python3 fetch_replica.py
Create Publisher Response: Publisher ID 'publisher1' created.
Create Subscriber Response: Subscriber ID 'subscriber1' created.
Create Topic Response: Topic has been successfully created on 100 and all the replicas has been stored into neighbors ['101', '110',
 '000']
Publish Message Response: Message published to 'news' and the replicas ['101', '110', '000'].
STOP Node Response: Node has been failed
Cannot connect to the original node as node has failed
Found a replica on 101 Forwading topic access to node 101
Subscribe Response: Subscriber ID 'subscriber1' subscribed to 'news'.
Cannot connect to the original node as node has failed
Found a replica on 101 Forwading topic access to node 101
Pull Message Response: {"original_node": "100", "message": "New president has been elected"}
```

6. Similar to PA2, you need to benchmark the latency and throughput of each API.
    a. Deploy 8 peers. Benchmark each API on each node using randomly generated workload.

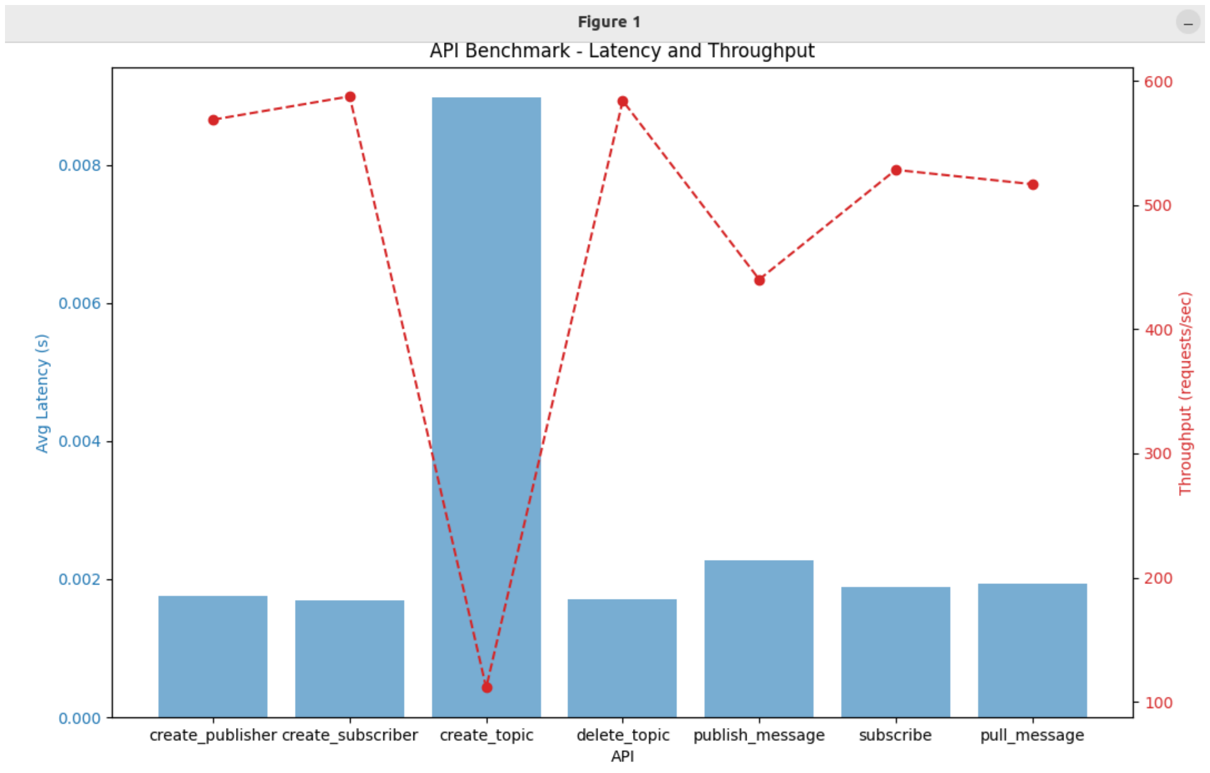


```
Node 010] [2024-12-03T04:12:53.393221] Received message from client: {"action": "pull", "subscriber_id": "subscriber6", "topic": "t
pic19"}
Node 010] [2024-12-03T04:12:53.393267] Pulling message from topic 'topic19' on node '010'
Node 010] [2024-12-03T04:12:53.393274] Messages pulled from 'topic19' locally on node '010'
Node 010] [2024-12-03T04:12:53.393298] Response to client: ["Benchmark message from publisher8", "Benchmark message from publisher5
, "Benchmark message from publisher3", "Benchmark message from publisher8", "Benchmark message from publisher6", "Benchmark message
from publisher10", "Benchmark message from publisher4", "Benchmark message from publisher8", "Benchmark message from publisher9", "
enchmark message from publisher10"]
Node 110] [2024-12-03T04:12:53.391819] Received message from client: {"action": "pull", "subscriber_id": "subscriber7", "topic": "t
pic2"}
Node 110] [2024-12-03T04:12:53.391866] Pulling message from topic 'topic2' on node '110'
Node 110] [2024-12-03T04:12:53.391873] Messages pulled from 'topic2' locally on node '110'
Node 110] [2024-12-03T04:12:53.391897] Response to client: ["Benchmark message from publisher5", "Benchmark message from publisher1
, "Benchmark message from publisher2", "Benchmark message from publisher5", "Benchmark message from publisher3"]
Node 010] [2024-12-03T04:12:53.401325] Received message from client: {"action": "pull", "subscriber_id": "subscriber9", "topic": "t
pic16"}
Node 010] [2024-12-03T04:12:53.401388] Pulling message from topic 'topic16' on node '010'
Node 010] [2024-12-03T04:12:53.401396] Messages pulled from 'topic16' locally on node '010'
Node 010] [2024-12-03T04:12:53.401418] Response to client: ["Benchmark message from publisher1"]
Node 111] [2024-12-03T04:12:53.405887] Received message from client: {"action": "pull", "subscriber_id": "subscriber1", "topic": "t
pic8"}
Node 111] [2024-12-03T04:12:53.405968] Pulling message from topic 'topic8' on node '111'
Node 111] [2024-12-03T04:12:53.405977] Messages pulled from 'topic8' locally on node '111'
Node 111] [2024-12-03T04:12:53.406010] Response to client: ["Benchmark message from publisher1", "Benchmark message from publisher1
", "Benchmark message from publisher9", "Benchmark message from publisher1"]
Node 101] [2024-12-03T04:12:53.409629] Received message from client: {"action": "pull", "subscriber_id": "subscriber3", "topic": "t
pic4"}
Node 101] [2024-12-03T04:12:53.409970] Pulling message from topic 'topic4' on node '101'
Node 101] [2024-12-03T04:12:53.409991] Messages pulled from 'topic4' locally on node '101'
Node 101] [2024-12-03T04:12:53.410018] Response to client: ["Benchmark message from publisher9", "Benchmark message from publisher4
, "Benchmark message from publisher4", "Benchmark message from publisher6"]
Node 110] [2024-12-03T04:12:53.411859] Received message from client: {"action": "delete_topic", "publisher_id": "publisher5", "topi
": "topic6"}
Node 110] [2024-12-03T04:12:53.412251] Deleting topic 'topic6' with hashed node '110'
Node 110] [2024-12-03T04:12:53.412264] Topic 'topic6' deleted locally on node '110'
Node 111] [2024-12-03T04:12:53.413807] Received message from client: {"action": "delete_replica", "topic_name": "topic6"}
Node 111] [2024-12-03T04:12:53.414208] Response to client: Replica of topic 'topic6' deleted in replica_topics.
Node 110] [2024-12-03T04:12:53.415076] Replica of topic 'topic6' successfully deleted on node 111.
Node 110] [2024-12-03T04:12:53.416509] Replica of topic 'topic6' successfully deleted on node 100.
```



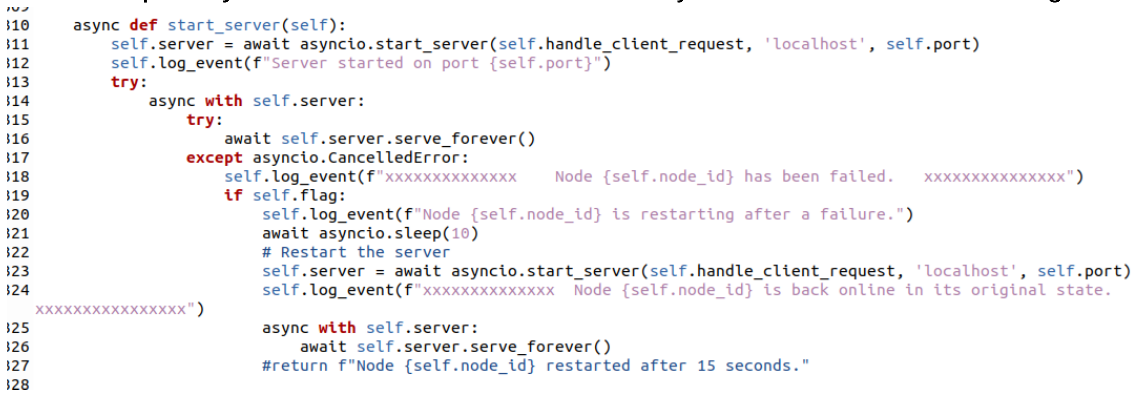
```
soham_vm@soham-vm:~/Project/tests$ python3 benchmark_all_api.py
create_publisher - Avg Latency: 0.0018 seconds, Throughput: 568.76 requests/sec
create_subscriber - Avg Latency: 0.0017 seconds, Throughput: 587.48 requests/sec
create_topic - Avg Latency: 0.0090 seconds, Throughput: 111.39 requests/sec
delete_topic - Avg Latency: 0.0017 seconds, Throughput: 584.01 requests/sec
publish_message - Avg Latency: 0.0023 seconds, Throughput: 440.08 requests/sec
subscribe - Avg Latency: 0.0019 seconds, Throughput: 528.32 requests/sec
pull_message - Avg Latency: 0.0019 seconds, Throughput: 516.81 requests/sec
soham_vm@soham-vm:~/Project/tests$
```

    b. Graph your results



7. Use sleep() in your code to manually control the communication latency between nodes. Explore in what scenarios your system is faster than your PA3 code.

Used Sleep in my code to restart server after a delay of 10 seconds after server gets failed.



```
    async def start_server(self):
        self.server = await asyncio.start_server(self.handle_client_request, 'localhost', self.port)
        self.log_event(f"Server started on port {self.port}")
        try:
            async with self.server:
                try:
                    await self.server.serve_forever()
                except asyncio.CancelledError:
                    self.log_event(f"xxxxxxxxxxxxxx    Node {self.node_id} has been failed.   xxxxxxxxxxxxxxx")
                    if self.flag:
                        self.log_event(f"Node {self.node_id} is restarting after a failure.")
                        await asyncio.sleep(10)
                        # Restart the server
                        self.server = await asyncio.start_server(self.handle_client_request, 'localhost', self.port)
                        self.log_event(f"xxxxxxxxxxxxxx  Node {self.node_id} is back online in its original state.
xxxxxxxxxxxxxxxx")
                        async with self.server:
                            await self.server.serve_forever()
                        #return f"Node {self.node_id} restarted after 15 seconds."
```

1. Reduced Latency in Failure Scenarios

- **Older System:**
  When a node fails, the system might lose access to the topics stored on that node, leading to timeouts or errors when subscribers or publishers try to interact with those topics. This results in retries or system stalls while waiting for recovery.

- **New System:**
  With fault tolerance (replica storage), neighbors hold replicas of the topics. If a node fails, the system can immediately redirect requests to a neighbor, reducing latency

compared to waiting for the failed node to recover.
Faster in Scenarios:

- When one or more nodes fail during high-load operations like topic subscriptions or message publishing.

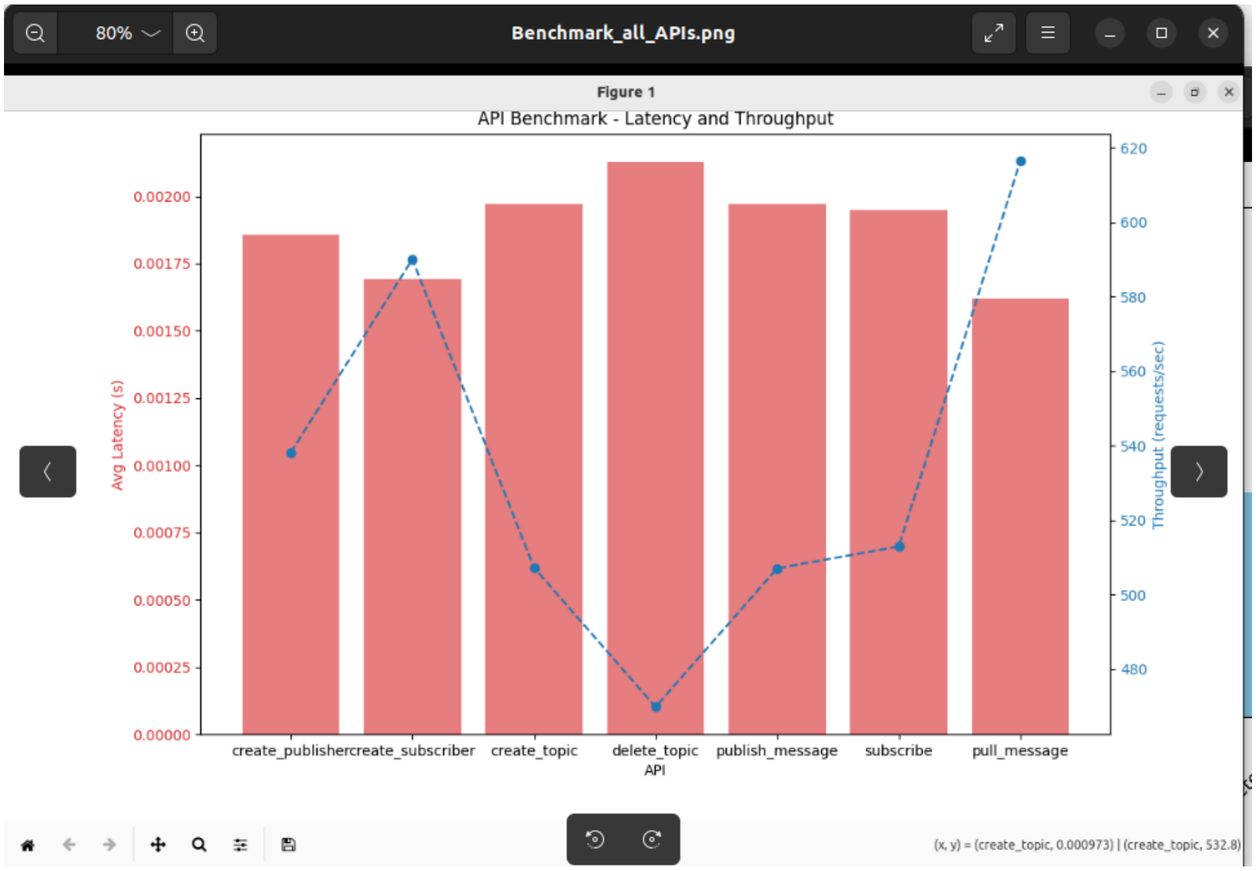
2. Improved Parallelism During Node Recovery

- **Older System:**
  Recovery from node failure might require re-distributing topics to other nodes or waiting for the failed node to come back online. During this process, operations related to the failed node can block or queue up.
- **New System:**
  With replicas already distributed across neighboring nodes, the system doesn't need to halt operations. Neighboring nodes can handle requests seamlessly, allowing faster recovery of the system's operational state. **Faster in Scenarios:**
  - When nodes need to recover from failure under high request concurrency.
  - Systems requiring high availability with minimal downtime.

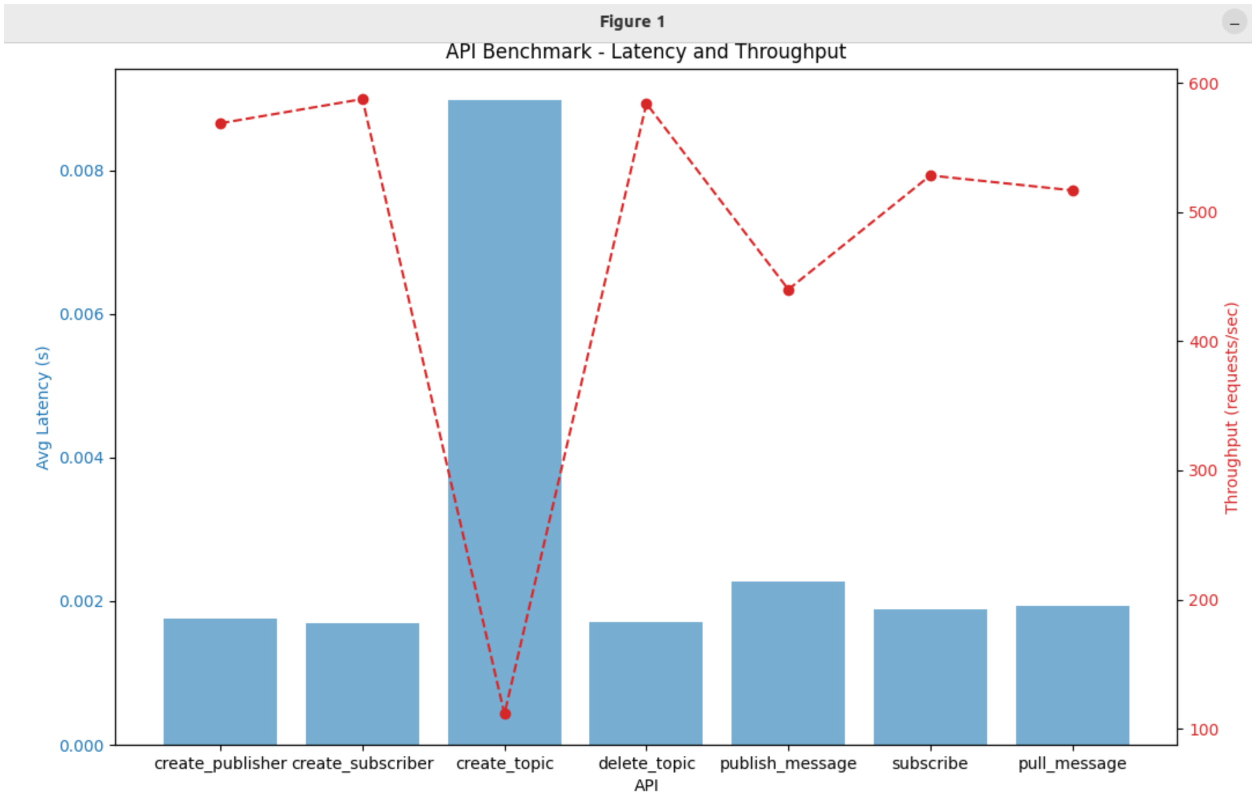
3. Faster Benchmarking with High Concurrency

- **Older System:**
  Under high concurrency, failure of a single node could stall requests, as there are no fallback mechanisms. Each stalled request contributes to higher overall latency and lower throughput.
- **New System:**
  Fault tolerance ensures that concurrent requests can be redirected to neighboring nodes holding replicas, maintaining throughput and reducing average response times.
  Faster in Scenarios:
  - Benchmarks where multiple peer nodes concurrently query or publish messages with N = [4, 8].
  - Stress tests simulating random node failures during peak operations.

PA3 Benchmarking Graph -



Project Benchmarking Graph (New System):



8.

Extra credits

Up to 10 points. Describe what you are curious about consistency model in distributed system, conduct experiments to solve/verify your questions on your own.

Eventual Consistency Model

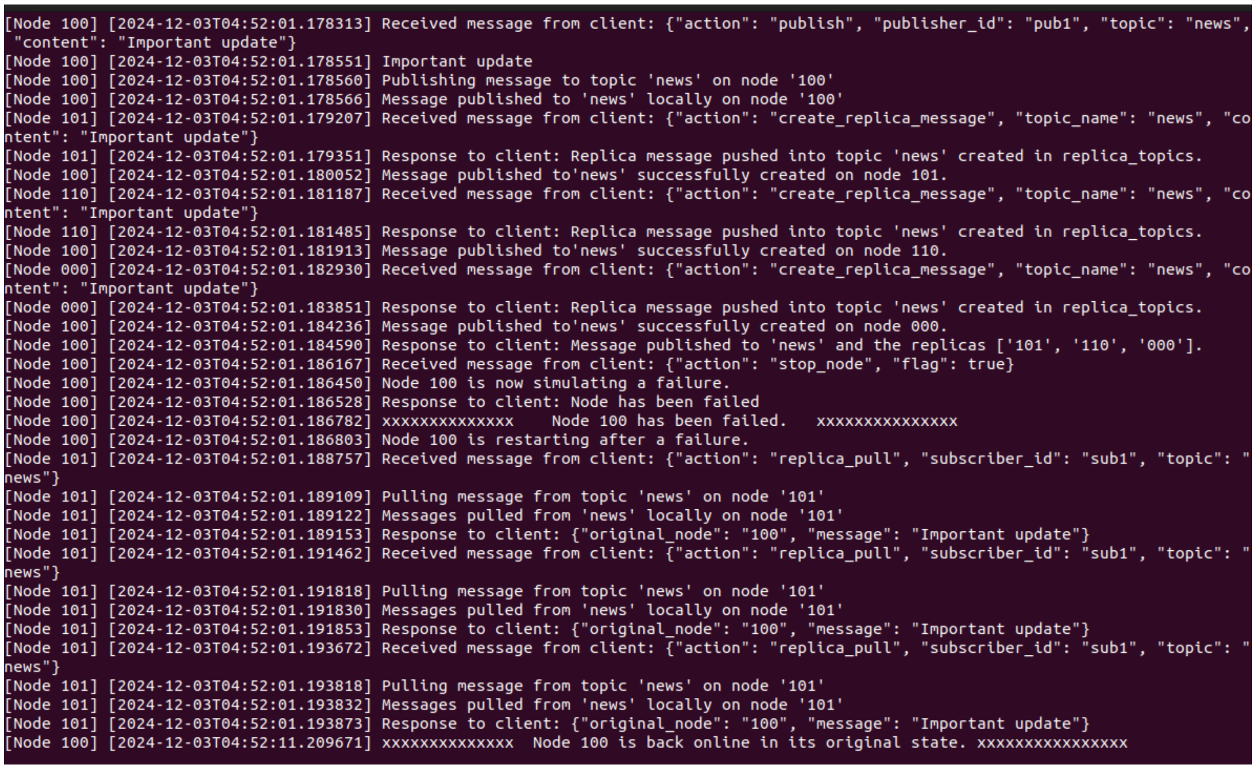
The system implements an eventual consistency model, where replicas are updated asynchronously. This is evident from the following observations:

1. When a topic is created or a message is published, the operation is first performed on the local node, and then replicas are updated on neighboring nodes.
2. There's no explicit mechanism for immediate synchronization across all nodes.

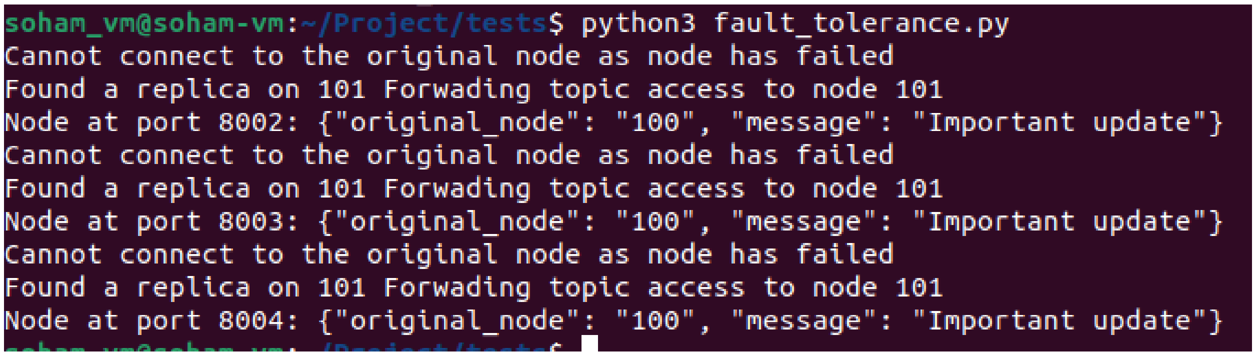
To verify this, we can conduct the following experiment:

Replication Strategy

The system uses a neighbor-based replication strategy, where each node replicates its topics to its immediate neighbors in the hypercube. This approach provides fault tolerance.To test the fault tolerance, we can conduct the following experiment:



```
[Node 100] [2024-12-03T04:52:01.178313] Received message from client: {"action": "publish", "publisher_id": "pub1", "topic": "news",
 "content": "Important update"}
[Node 100] [2024-12-03T04:52:01.178551] Important update
[Node 100] [2024-12-03T04:52:01.178560] Publishing message to topic 'news' on node '100'
[Node 100] [2024-12-03T04:52:01.178566] Message published to 'news' locally on node '100'
[Node 101] [2024-12-03T04:52:01.179207] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "Important update"}
[Node 101] [2024-12-03T04:52:01.179351] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T04:52:01.180052] Message published to'news' successfully created on node 101.
[Node 110] [2024-12-03T04:52:01.181187] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "Important update"}
[Node 110] [2024-12-03T04:52:01.181485] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T04:52:01.181913] Message published to'news' successfully created on node 110.
[Node 000] [2024-12-03T04:52:01.182930] Received message from client: {"action": "create_replica_message", "topic_name": "news", "co
ntent": "Important update"}
[Node 000] [2024-12-03T04:52:01.183851] Response to client: Replica message pushed into topic 'news' created in replica_topics.
[Node 100] [2024-12-03T04:52:01.184236] Message published to'news' successfully created on node 000.
[Node 100] [2024-12-03T04:52:01.184590] Response to client: Message published to 'news' and the replicas ['101', '110', '000'].
[Node 100] [2024-12-03T04:52:01.186167] Received message from client: {"action": "stop_node", "flag": true}
[Node 100] [2024-12-03T04:52:01.186450] Node 100 is now simulating a failure.
[Node 100] [2024-12-03T04:52:01.186528] Response to client: Node has been failed
[Node 100] [2024-12-03T04:52:01.186782] xxxxxxxxxxxxxx    Node 100 has been failed.   xxxxxxxxxxxxxxx
[Node 100] [2024-12-03T04:52:01.186803] Node 100 is restarting after a failure.
[Node 101] [2024-12-03T04:52:01.188757] Received message from client: {"action": "replica_pull", "subscriber_id": "sub1", "topic": "
news"}
[Node 101] [2024-12-03T04:52:01.189109] Pulling message from topic 'news' on node '101'
[Node 101] [2024-12-03T04:52:01.189122] Messages pulled from 'news' locally on node '101'
[Node 101] [2024-12-03T04:52:01.189153] Response to client: {"original_node": "100", "message": "Important update"}
[Node 101] [2024-12-03T04:52:01.191462] Received message from client: {"action": "replica_pull", "subscriber_id": "sub1", "topic": "
news"}
[Node 101] [2024-12-03T04:52:01.191818] Pulling message from topic 'news' on node '101'
[Node 101] [2024-12-03T04:52:01.191830] Messages pulled from 'news' locally on node '101'
[Node 101] [2024-12-03T04:52:01.191853] Response to client: {"original_node": "100", "message": "Important update"}
[Node 101] [2024-12-03T04:52:01.193672] Received message from client: {"action": "replica_pull", "subscriber_id": "sub1", "topic": "
news"}
[Node 101] [2024-12-03T04:52:01.193818] Pulling message from topic 'news' on node '101'
[Node 101] [2024-12-03T04:52:01.193832] Messages pulled from 'news' locally on node '101'
[Node 101] [2024-12-03T04:52:01.193873] Response to client: {"original_node": "100", "message": "Important update"}
[Node 100] [2024-12-03T04:52:11.209671] xxxxxxxxxxxxxx  Node 100 is back online in its original state. xxxxxxxxxxxxxxxx
```



```
soham_vm@soham-vm:~/Project/tests$ python3 fault_tolerance.py
Cannot connect to the original node as node has failed
Found a replica on 101 Forwading topic access to node 101
Node at port 8002: {"original_node": "100", "message": "Important update"}
Cannot connect to the original node as node has failed
Found a replica on 101 Forwading topic access to node 101
Node at port 8003: {"original_node": "100", "message": "Important update"}
Cannot connect to the original node as node has failed
Found a replica on 101 Forwading topic access to node 101
Node at port 8004: {"original_node": "100", "message": "Important update"}
```

- Discussion: You should also discuss the design of your code. E.g. Do you think the APIs are sufficient? Where are the bottlenecks? You don't have to answer exactly these questions. We look for any opinions/ideas/discussion that show you are actively thinking about this project is fine.

Discussion on Code Design

API Design

The current API set, as outlined in the design document, is functional but has room for improvement. Here's a detailed evaluation:

- **Sufficiency of Current APIs:**
  The existing APIs for creating topics, publishing messages, subscribing to topics, and retrieving messages provide a solid foundation. However:
  - The lack of APIs for **replica management** (e.g., querying replica locations or manually synchronizing replicas) might limit debugging and operational flexibility.
  - APIs for **node management**, such as dynamically adding or removing nodes, are not explicitly defined, which could make integration with automation systems difficult.
- **Potential Additions:**
  - **Replica Query API:** Enables clients to fetch the list of nodes hosting replicas for a topic.
  - **Node Status API:** Reports the health and availability of nodes to facilitate monitoring.
  - **Consistency Control API:** Allows clients to specify their desired consistency level during operations.

Bottlenecks

The design introduces complexity and potential bottlenecks in the following areas:

1. **Topic Replication Synchronization:**
   - **Issue:** Synchronizing replicas across nodes introduces latency and overhead, especially under high write loads.
   - **Mitigation:** Use asynchronous replication for non-critical topics and prioritize critical data with synchronous replication.
2. **Dynamic Topology Management:**
   - **Issue:** Adding or removing nodes dynamically may cause delays in routing table updates and topic reassignment.
   - **Mitigation:** Employ a lazy synchronization strategy to update routing tables incrementally while ensuring immediate consistency for critical paths.
3. **Hash Function Hotspots:**
   - **Issue:** The hashing mechanism may lead to uneven distribution of topics, causing some nodes to become overloaded.
   - **Mitigation:** Implement a **consistent hashing with virtual nodes** approach to evenly distribute load across physical nodes.
4. **Fault Detection and Recovery Latency:**
   - **Issue:** Periodic heartbeat mechanisms can lead to delays in detecting node failures.
   - **Mitigation:** Use an event-driven failure detection system with immediate neighbor notifications.

Tradeoffs in the Current Design

1. **Consistency vs. Performance:**
   - Stronger consistency models increase synchronization overhead, potentially degrading performance during high write activity.
   - Current choice of a primary-secondary model balances these tradeoffs, but exploring hybrid models (e.g., quorum-based) may be worthwhile.
2. **Complexity vs. Simplicity:**
   - The hypercube topology simplifies routing but adds complexity to fault detection and topology updates.
   - Simpler topologies, like a ring, could reduce operational complexity but increase hop counts.
3. **Asynchronous Communication vs. Debugging Complexity:**
   - Asynchronous operations improve throughput but complicate debugging and traceability.
   - Including **end-to-end transaction tracing** in the API design could mitigate this issue.

Areas for Future Refinement

1. **Logging and Monitoring:**
   Comprehensive logging and a centralized dashboard would allow developers to trace bottlenecks and optimize performance.
2. **Testing for Edge Cases:**
   The system should be stress-tested under:
   - High node churn (frequent addition/removal of nodes).
   - Network partition scenarios to assess how the system behaves when parts of the network are isolated.