Rubric-GRPC DSCD Assignment 1

GRPC - 20/60 marks

Demo (10/20 marks)

For every point for which there are marks in red, deduct 0.5 marks, if status messages aren't printed on either side of the communication, especially success and failure messages.

1) Deploy registry server
2) Deploy 2 Servers
    a) Register both to the registry server. 1 mark
3) Deploy 1 Client
4) Check for GetServerList returns details of 2 servers 1 mark
5) Join this client to server 1
6) Publish Article to server 1 SPORTS, Jack Dorsey, <blank>, Ronaldo loses too soon. 1 mark
7) Publish Article to server 1 FASHION, Asher Jones, <blank>, Pharell Williams joins Louis Vuitton.
8) Deploy 2nd client
    a) Request article from server 1 - should Fail because the client hasn't joined the server yet 1 mark
    b) Publish an article to Server 1 - should fail because the client hasn't joined the server yet 1 mark
9) Join 2nd client to server 1
    a) Request article with MOVIES - should gracefully fail because of illegal type 1 mark
    b) Request article with POLITICS - should fail/return empty because of no matches 1 mark
    c) Request article with SPORTS - should return only one article of Ronaldo 1 mark
       Article must have the date filled by server 1 mark
    d) Request article with Asher Jones and 01/02/2023- should return only one article of Pharell 1 mark
10) **BONUS**
11) Deploy another server (3 in total)
12) Make client 1 publish articles to all 3 (keep type to be POLITICS for all)
13) Server 1 joins Server 2
14) Server 2 joins Server 3
15) Server 3 joins Server 1
16) Client 2 requests POLITICS messages from Server 1
    a) Should receive exactly 3 articles. Ensure that none of the servers/clients are stuck at this point. 6 marks

Code (5/20)

1) Check proto files written by the students
    a) Correct article format -
       1) Type - one of [SPORTS, FASHION, POLITICS] use oneof datatype (for gRPC) to check fields automatically. 1 mark
       2) Author - a string of characters for the author's name
       3) Time - Time when the message was received at the server
       4) Content - a string of a maximum of 200 characters. 1 mark for points 2, 3, 4
    b) Correct article request format 1 mark overall
       1) Type - one of [SPORTS, FASHION, POLITICS] use oneof datatype (for gRPC) to check fields automatically.
       2) Author - a string of characters for the author's name
       3) Time - Time when the message was received at the server
2) Check for RPC specific code similar to the following. Note: Each implementation may be different, specifically check for stub on client side.
   SERVER SIDE 1 mark

```
return helloworld_pb2.HelloReply(message=f'Hello, {request.name}!')
```

   CLIENT SIDE 1 mark

```
with grpc.insecure_channel('localhost:50051') as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response =
stub.SayHello(helloworld_pb2.HelloRequest(name='you'))
print("Greeter client received: " + response.message)
        response=stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
```

Viva (5/20) - separate for each student

Preferably ask questions individually.

1) Ask how the proto files were compiled
   Answer : `protoc -I=$SRC_DIR --python_out=$DST_DIR $SRC_DIR/{nameof file].proto`

2) Ask each what design choices were made
3) Ask how filtering was performed

RabbitMQ - 20/60 marks

Demo (10/20 marks)[same as gRPC]

For every point for which there are marks in red, deduct 0.5 marks, if status messages aren't printed on either side of the communication, especially success and failure messages.

1) Deploy registry server
2) Deploy 2 Servers
    a) Register both to the registry server. 1 mark
3) Deploy 1 Client
4) Check for GetServerList returns details of 2 servers 1 mark
5) Join this client to server 1
6) Publish Article to server 1 SPORTS, Jack Dorsey, <blank>, Ronaldo loses too soon. 1 mark
7) Publish Article to server 1 FASHION, Asher Jones, <blank>, Pharell Williams joins Louis Vuitton.
8) Deploy 2nd client
    a) Request article from server 1 - should Fail because the client hasn't joined the server yet 1 mark
    b) Publish an article to Server 1 - should fail because the client hasn't joined the server yet 1 mark
9) Join 2nd client to server 1
    a) Request article with MOVIES - should gracefully fail because of illegal type 1 mark
    b) Request article with POLITICS - should fail/return empty because of no matches 1 mark
    c) Request article with SPORTS - should return only one article of Ronaldo 1 mark
       Article must have the date filled by server 1 mark
    d) Request article with Asher Jones and 01/02/2023- should return only one article of Pharell 1 mark
10) **BONUS**
11) Deploy another server (3 in total)
12) Make client 1 publish articles to all 3 (keep type to be POLITICS for all)
13) Server 1 joins Server 2
14) Server 2 joins Server 3
15) Server 3 joins Server 1
16) Client 2 requests POLITICS messages from Server 1
    a) Should receive exactly 3 articles. Ensure that none of the servers/clients are stuck at this point. 6 marks

Code (5/20)

3) Check how queues have been used by the students
    a) Correct article format - using some serialized format (a string) of proto
       1) Type - one of [SPORTS, FASHION, POLITICS] use one of datatype (for gRPC) to check fields automatically. 1 mark
       2) Author - a string of characters for the author's name
       3) Time - Time when the message was received at the server
       4) Content - a string of a maximum of 200 characters. 1 mark for points 2, 3, 4
    b) Correct article request format 1 mark overall
       1) Type - one of [SPORTS, FASHION, POLITICS] use oneof datatype (for gRPC) to check fields automatically.
       2) Author - a string of characters for the author's name
       3) Time - Time when the message was received at the server
4) Check for RPC pattern code similar to the following. Note: Each implementation may be different, specifically check for stub on client side.
   SERVER SIDE 1 mark

```
channel.basic_consume(queue='rpc_queue',on_message_callback=on_reque
st)
```

   CLIENT SIDE 1 mark
   The code on the client side should have some way of handling the RPC pattern below is an example.

```
    def call(self, n):
        self.response = None
        self.corr_id = str(uuid.uuid4())
        self.channel.basic_publish(
            exchange='',
            routing_key='rpc_queue',
            properties=pika.BasicProperties(
                reply_to=self.callback_queue,
                correlation_id=self.corr_id,
            ),
            body=str(n))
        self.connection.process_data_events(time_limit=None)
        return int(self.response)
```

Viva (5/20) - separate for each student

Preferably ask questions individually.

1) Ask if they can implement the same thing w/o RPC pattern and whether the following design with rabbitMQ is optimal (further ideas to improve can also be asked)?- Yes possible and RPC is redundant in this design
2) What is a message Broker? What is AMQP?
3) What is the principle on which message queues work in RabbitMQ?

ZeroMQ - 20/60 marks

Demo (10/20 marks) [same as gRPC and RabbitMQ]

For every point for which there are marks in red, deduct 0.5 marks, if status messages aren't printed on either side of the communication, especially success and failure messages.

17) Deploy registry server
18) Deploy 2 Servers
    a) Register both to the registry server. 1 mark
19) Deploy 1st Client
20) Check that GetServerList returns details of the above 2 servers 1 mark
21) Join 1st client to server 1
22) Publish Article to server 1 <SPORTS, Jack Dorsey, <blank>, Ronaldo loses too soon>. 1 mark
23) Publish Article to server 1 <FASHION, Asher Jones, <blank>, Pharell Williams joins Louis Vuitton>.
24) Deploy 2nd client
    a) Request article from server 1 - should Fail because the client hasn't joined the server yet 1 mark
    b) Publish an article to Server 1 - should fail because the client hasn't joined the server yet 1 mark
25) Join 2nd client to server 1
    a) Request article with MOVIES - should gracefully fail because of illegal type 1 mark
    b) Request article with POLITICS - should fail/return empty because of no matches 1 mark
    c) Request article with SPORTS - should return only one article of Ronaldo 1 mark
       Article must have the date filled by server 1 mark
    d) Request article with Asher Jones and 01/02/2023- should return only one article of Pharell 1 mark
26) **BONUS**

27) Deploy another server (3 in total)
28) Make client 1 publish articles to all 3 (keep type to be POLITICS for all)
29) Server 1 joins Server 2
30) Server 2 joins Server 3
31) Server 3 joins Server 1
32) Client 2 requests POLITICS messages from Server 1
    a) Should receive exactly 3 articles. Ensure that none of the servers/clients are stuck at this point. 6 marks

Code (5/20)

5) Check how queues have been used by the students
    a) Correct article format - using a format like JSON, or as done in gRPC.
    1) Type - one of [SPORTS, FASHION, POLITICS]. Should have done validation so that Type ∈ {SPORTS, FASHION, POLITICS}. 1 mark
    2) Author - a string of characters for the author's name
    3) Time - Time when the message was received at the server
    4) Content - a string of a maximum of 200 characters. 1 mark for points 2, 3, 4
    b) Correct article **request** format 1 mark overall
    1) Type - one of [SPORTS, FASHION, POLITICS]. Should have done validation so that Type ∈ {SPORTS, FASHION, POLITICS}.
    2) Author - a string of characters for the author's name
    3) Time - Time when the message was received at the server
6) Check for RPC pattern code similar to the following. Note: Each implementation may be different. Here the Python like code is for ZeroMQ request-reply pattern using JSON encoded messages. The student may also use Pub-Sub or any other pattern with any other encoding.
SERVER SIDE 1 mark
Setting up a server at some port :

```
import zmq
import socket

context = zmq.Context()
socket = context.socket(zmq.REP) # this is REP from the
request-reply pattern
port = ...
socket.bind(<port>)

# for server to receive messages from the client
socket.recv_json(...)
# for server to send messages to the client
socket.send_json(...)
```

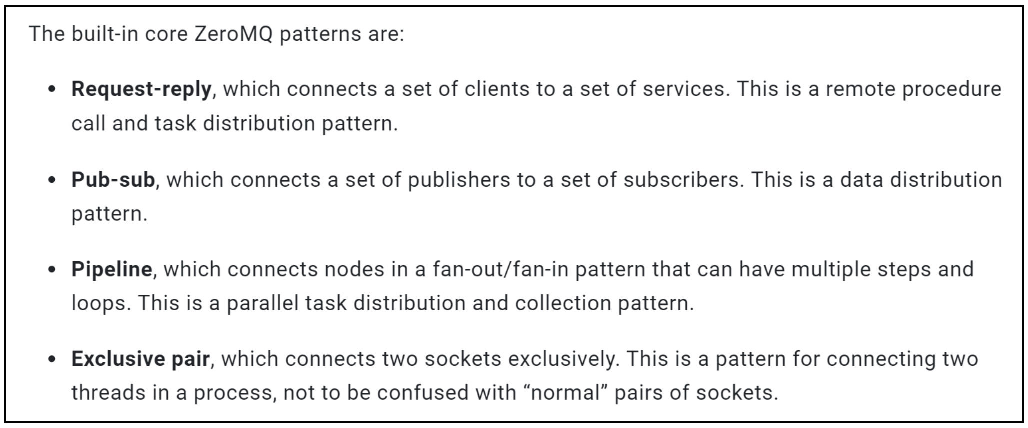
CLIENT SIDE 1 mark

The code on the client side should have some way of handling the RPC pattern below is an example.

```
# to connect to server

addr = <address of server the client wants to connect, like
localhost:9000>

context = zmq.Context()
socket = context.socket(zmq.REQ) # this is REQ from the
request-reply pattern
socket.connect(<addr>)

# to send a request to server
socket.send_json(...)

# to receive reply from server
resp = socket.recv_json()
```

Viva (5/20) - separate for each student

- what is the difference between synchronous and asynchronous
- what are some patterns offered by ZeroMQ ?



The built-in core ZeroMQ patterns are:

- **Request-reply**, which connects a set of clients to a set of services. This is a remote procedure call and task distribution pattern.
- **Pub-sub**, which connects a set of publishers to a set of subscribers. This is a data distribution pattern.
- **Pipeline**, which connects nodes in a fan-out/fan-in pattern that can have multiple steps and loops. This is a parallel task distribution and collection pattern.
- **Exclusive pair**, which connects two sockets exclusively. This is a pattern for connecting two threads in a process, not to be confused with “normal” pairs of sockets.

2 of the above patterns (Request-Reply and Pub-Sub) were discussed in the GC comments. So the student should tell atleast these 2. And student should also tell the difference between them, as mentioned in the SS above.

- how is zeromq different from rabbitmq ? give some examples of message brokers / message queues apart from these, like kafka ...