Abstract

Prior to the digital era, distance learning appeared as corresponding courses, broadcast messages, and closed circuit video access and earlier form of e learning. Brought as a result of open education resources (OER) movement, Massive Open Online Course (MOOC) is an interactive online course aimed at unlimited participation and open access via web. The goal to this project is to design an approach that provides the fastest possible communication within a changing, unbounded, network topology. The ability for clusters of groups to join (discovery) other groups and participate in cooperative problem solving is necessary. This problem solving and resolution is a feature that is found in many cooperative-based systems where scheduling of resources, distributing data, or ensemble-based solving is desired.

Our project aims to focus on inter cluster communication, intra cluster communication, communication performance, overlay network flexibility, leader election and voting. We have created a MOOC cluster, where each team's PC represents virtual groups that compete within the larger community. Each cluster provides functionality like Sign Up for an account, Sign In, Search for available courses, Search for a particular course and Enrollment for a course. Any cluster can communicate with any other cluster and even start a voting competition. Additionally, nodes within a cluster can compete for incoming job requests and the job is assigned to the winning node based on the cluster's voting strategy. The network topology for our cluster is a ring topology. We try to minimize the single point of failure through leader election, where a leader node's failure will start an election algorithm to choose the next leader for the cluster. We have also implemented a failover strategy to pass a message to the next nearest node if the immediate nearest node is not available.

The technologies that we explored in this project are Java and Python for designing client and storage servers, Google Protobuf to define the data structure and protocol to be used for communication, JBoss Netty for asynchronous communication, RabbitMQ for intra-cluster queuing and MongoDB for storage purposes.

Contents

1. Introduction

About MOOC System:

MOOCs or Massive Online Open Courses based on Open Educational Resources (OER) might be one of the most versatile ways to offer access to quality education, especially for those residing in far or disadvantaged areas. And this project analyzes the state of the art on MOOCs, exploring open research questions and setting interesting topics and goals for further research. Finally, it proposes a framework that includes the use of software agents with the aim to improve and personalize management, delivery, efficiency and evaluation of massive online courses on an individual level basis.

Design of MOOC System: MOOC models are evolving quickly; according to Siemens [2] they can be currently classified as xMOOCs, cMOOCs and quasi-MOOCs. xMOOCs replicate online the traditional model of an expert tutor and learners as knowledge consumers, with saved video tutorials and graded assignments. cMOOCs are based on a connectives' pedagogical model that views knowledge as a networked state and learning as the process of generating those networks, in this case using online and social tools. Lastly, the category of quasi-MOOCs encompasses a myriad of web-based tutorials as OER that are technically not courses but are intended to support learning-specific tasks and consist of asynchronous learning resources that do not offer the social interaction of cMOOCs or the automated grading and tutorial-driven format of xMOOCs.

About the project:

The aim of this project is to ensure a stable communication across multiple servers in a dynamic overlay network. In a decentralized, loosely coupled, service oriented architecture; a significant amount of time is spent on reliability and scalability aspects of the system. The major challenges that a distributed network has to deal with include storage and retrieval of data, establishing connection across the servers, promptly responding to the requests and graceful handling of errors. There are several design concerns as mentioned below that should be addressed in the implementation:

- Inspecting the request and response type.
- Understanding DNS Resolution and IP Identification between the servers and requesting nodes i.e. client
- Heterogeneous development and training environments
- Testing options are limited because assessments have to be conducted using automated tools and allowing limited peer interaction

Even though this problem looks straightforward, the solution is non-trivial when the network is loosely coupled and dynamic in nature. Our approach to solving these problems involves multiple technologies that not only help in solving network communication problems, storage and retrieval but also set global standards to make this communication system interoperable.

2. Tools/Technologies

2.1 Netty

Netty is a non-blocking I/O (NIO) client-server framework for the development of Java network applications such as protocol servers and clients. The asynchronous event-driven network application framework and tools are used to simplify network programming such as TCP and UDP socket servers. Netty includes an implementation of the reactor pattern of programming. Besides being an asynchronous network application framework, Netty also includes built-in HTTP protocol support, including the ability to run inside a servlet container, support for Web Sockets, and integration with Google Protocol Buffers, SSL/TLS support, support for SPDY protocol and support for message compression. Netty has been around since before 2004. [3]

Netty architecture comprises of three main components:

- Buffer: Netty uses its own Channel Buffer to represent sequence of bytes. Use of Channel Buffer improves communication performance and allows expansion of buffer on demand.
- Channel: Channel is the asynchronous I/O that provides abstraction to the operations used by Netty for communication. Each request is provided a channel that has a unique channel id and each channel can be reused after serving response for that request. The responses are handed by identifying the corresponding channel ids.
- Event model: Netty is an extensible architecture that allows even based communication that serves as an advantage over other frameworks. Channel Handlers in the Channel Pipeline handles channel events.

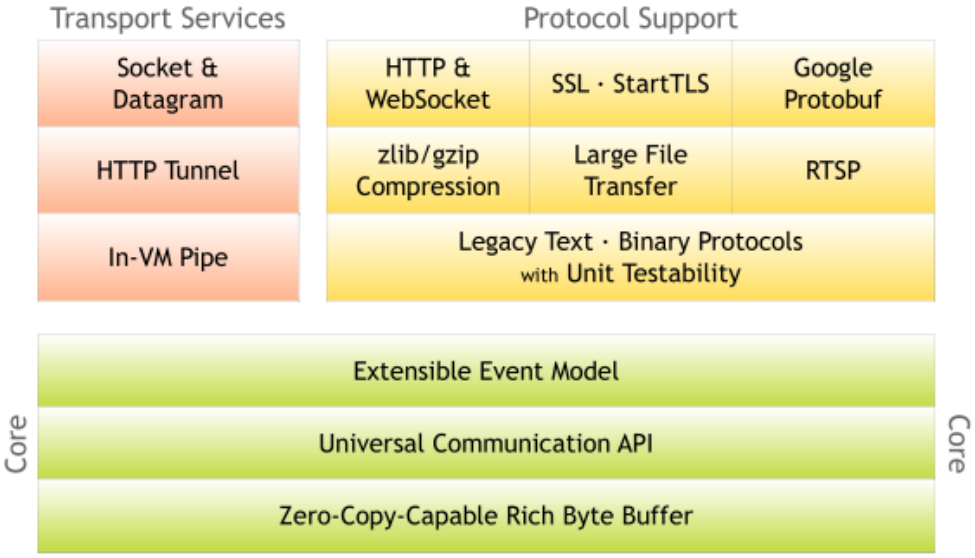


Figure1: Netty communication architecture

Some of the Netty features that describe why Netty is useful for building communication system are:

1. Design
    a. Unified API for various transport types - blocking and non-blocking socket
    b. Based on a flexible and extensible event model which allows clear separation of concerns
    c. Highly customizable thread model - single thread, one or more thread pools such as SEDA
    d. 1.4. True connectionless datagram socket support (since 3.1)
2. Ease of use
    a. Well-documented Javadoc, user guide and examples
    b. No additional dependencies but JDK 1.5 (or above)
3. Performance
    a. Better throughput, lower latency
    b. Less resource consumption
    c. Minimized unnecessary memory copy
4. Security
    a. Complete SSL/TLS and Start TLS support
5. Community
    a. Release early, release often
6. Unified API

    a. Netty provides a unified API that does not add restrictions in running the applications on different underlying java versions.
7. Ease of use
    a. Netty allows a very quick and comparatively easy development of network applications without compromising with flexibility, performance and stability.
8. Integration with Google Protobuf
    a. Netty enables integration with Google Protobuf that helps in defining protocol standards for an efficient communication.

2.2 MongoDB:

MongoDB (from "hu**mongo**us") is a cross-platform document-oriented database system. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software.
Some of the main features include: [7]

1. Ad hoc queries: MongoDB supports search by field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions.
2. Indexing: Any field in a MongoDB document can be indexed (indices in MongoDB are conceptually similar to those in RDBMSes). Secondary indices are also available.
3. Replication: MongoDB provides high availability and increased throughput with replica sets. [8] A replica set consists of two or more copies of the data. Each replica may act in the role of primary or secondary replica at any time. The primary replica performs all writes and reads by default. Secondary replicas maintain a copy of the data on the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can also perform read operations, but the data is eventually consistent by default.
4. Load balancing: MongoDB scales horizontally using sharding. [9] The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.)
5. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy, and new machines can be added to a running database.
6. File storage: MongoDB can be used as a file system, taking advantage of load balancing and data replication features over multiple machines for storing files. This function, called GridFS, [10] is included with MongoDB drivers and available with no difficulty for development languages (see "Language Support" for a list of supported languages). MongoDB exposes functions for file manipulation and content to developers. GridFS is used, for example, in plugins for NGINX[11] and lighttpd.[12] Instead of storing a file in a single document, GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.[13]
7. In a multi-machine MongoDB system, files can be distributed and copied multiple times between machines transparently, thus effectively creating a load-balanced and fault-tolerant system.
8. Aggregation: MapReduce can be used for batch processing of data and aggregation operations. The aggregation framework enables users to obtain the kind of results for which the SQL GROUP BY clause is used.
9. Server-side JavaScript execution: JavaScript can be used in queries, aggregation functions (such as MapReduce), and sent directly to the database to be executed.
10. Capped collections: MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

2.3 Google Protobuf:

Google Protobuf allows serialization of structured data in a client/server communication. The advantages of using it include *platform and language independence*, simple and faster to use since the file size is very small.
The first step to use protocol buffer is to create a .proto file that contains serialized messages. The messages are structured in a name-value pair and are identified by unique number tags. Once the message definition is complete, .proto file is compiled to produce data access classes that provide methods to serialize and parse data.

2.4 ANT:

Our project uses Apache Ant for the building target.

2.5 RabbitMQ

RabbitMQ is a messaging broker that provides a common platform for applications to send and receive messages. Messaging is asynchronous which helps decouple applications from one another. Some of the main features that make RabbitMQ popular are:

1. Reliability: RabbitMQ offers persistence and message acknowledgements and high availability.
2. Ease of Use: RabbitMQ ships with a web Management UI that allows you to monitor the messaging queue.
3. Many clients: RabbitMQ offers clients for almost any language and runs on all major operating systems.
4. Support: RabbitMQ is open source and commercially supported.

3. System architecture

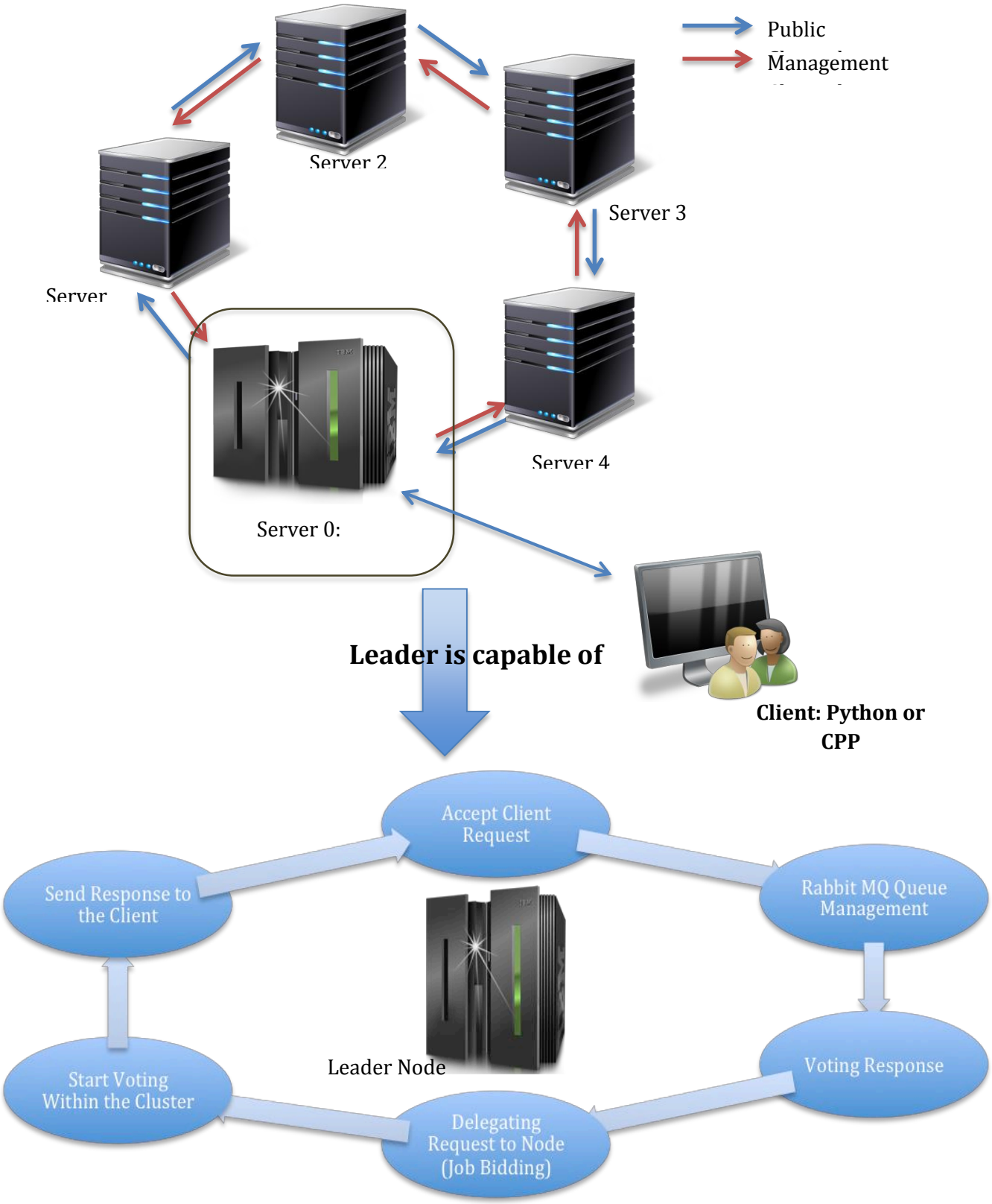


Figure 2: System design – Overlay MOOC Network

In our design, we have used JBoss Netty to create an overlay network. This framework allows asynchronous communication across the different server nodes where each server node has a shared pair of Master-Slave MongoDB database that stores and processes the file related operations such as listing the course details, getting course by ID, Sign In, Sign Up and Enrollment into a course. Each server connects to a shared MongoDB database.

3.1 Overlay Network

An overlay network is a computer network, which is built on the top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, through many physical links, in the underlying network. For example, distributed systems such as cloud computing, peer-to-peer networks, and client-server applications are overlay networks because their nodes run on top of the Internet. We have created an overlay network with two channels *Public channel (for all client and inter-server communication)* and *Management Channel (for all intra cluster communication in the network)* as shown in the figure.

We have used ring topology for request/response handling. Servers are connected to each other in the form of ring. Each server has two neighbors. The routing takes place on the basis of the request sent by the client. Request processing is handled as:

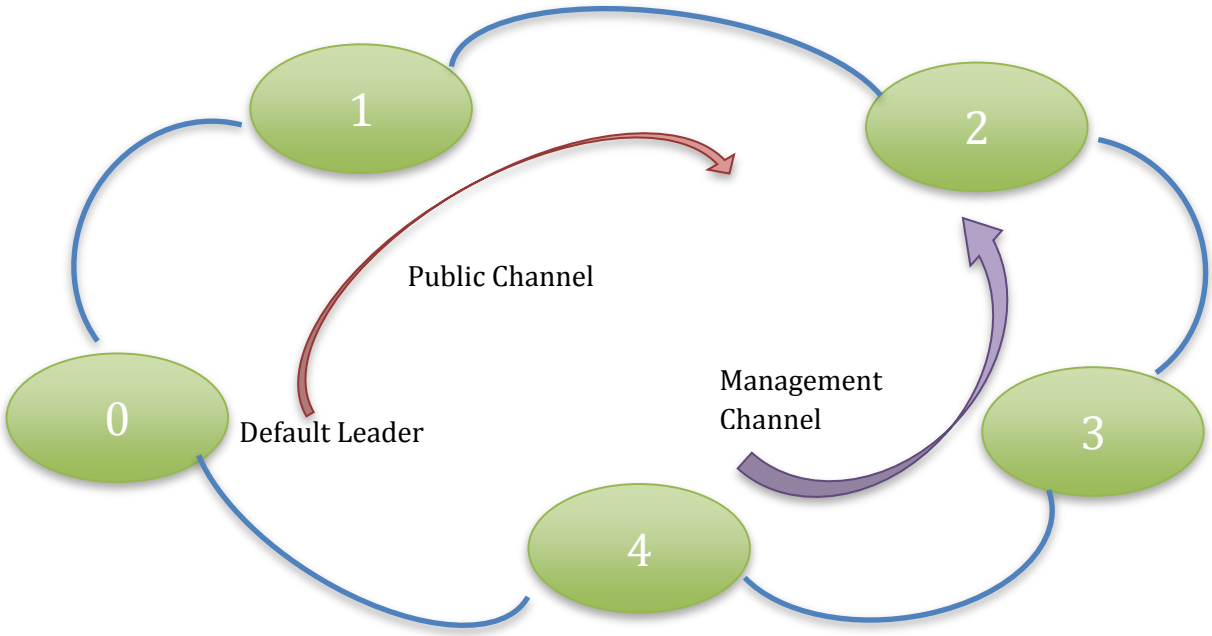


Figure 3: Network Topology

In our design, we have implemented an asynchronous messaging system. Each server uses an inbound and outbound queue for processing requests.

4. Design features:

4.1 Job proposal and Job Bidding

Job proposal is a feature to determine which server node in a cluster will process the incoming request. This is determined by the outcome of the Job Bidding process. When the leader of the cluster receives a request from the client, the leader prepares a job proposal which includes the job weight and the type of job. The leader forwards this to its nearest node via its outbound channel. When a node receives a job proposal message, it creates a job bid request with the same job ID and assigns the bid value as the total weight of all the jobs it has processed till the current time. It then forwards this job bid to its nearest node. Each node that gets a job bid will set a bid value as its own total weight of all processed jobs and compare the incoming bid value with its own bid value. A node will forward the job bid if the incoming bid value is less than its own bid value. Eventually, the leader receives a job bid message that has the node ID and bid value of the node which has the lowest bid value. This ensures that the node which has the least job weight processed gets the next request.

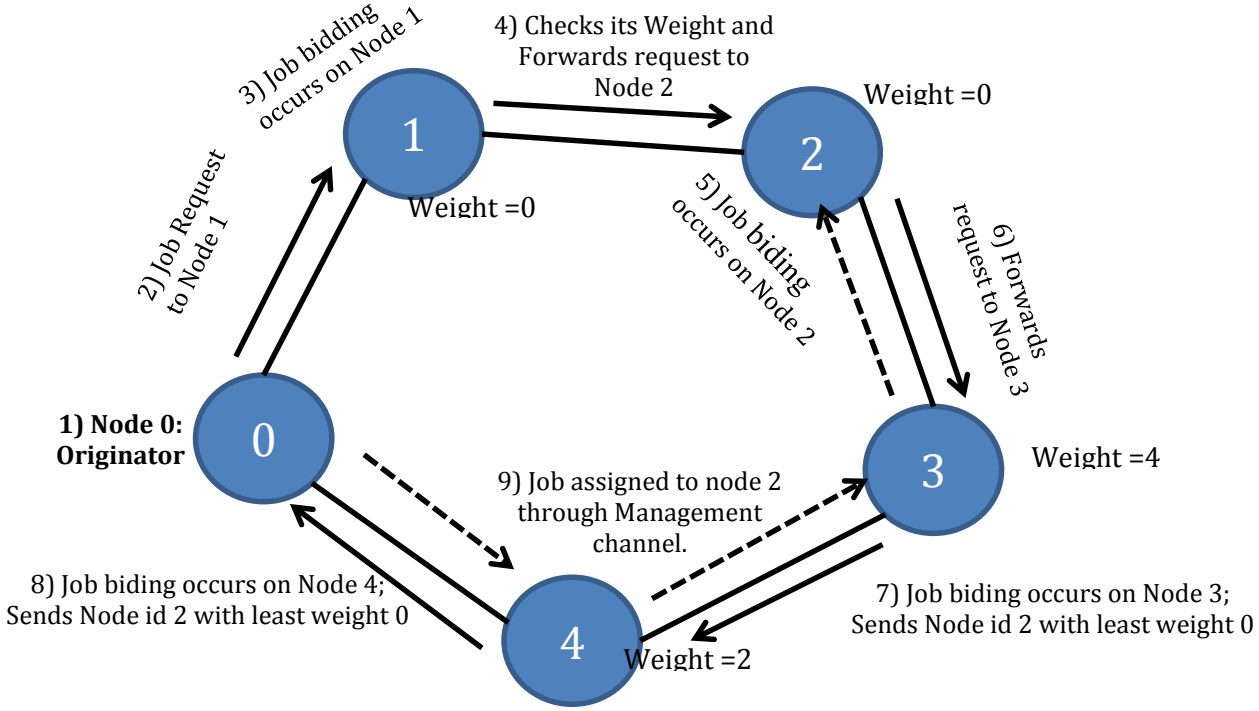


Figure 4: Forward Request and Job Bidding strategy

4.2 Forward request

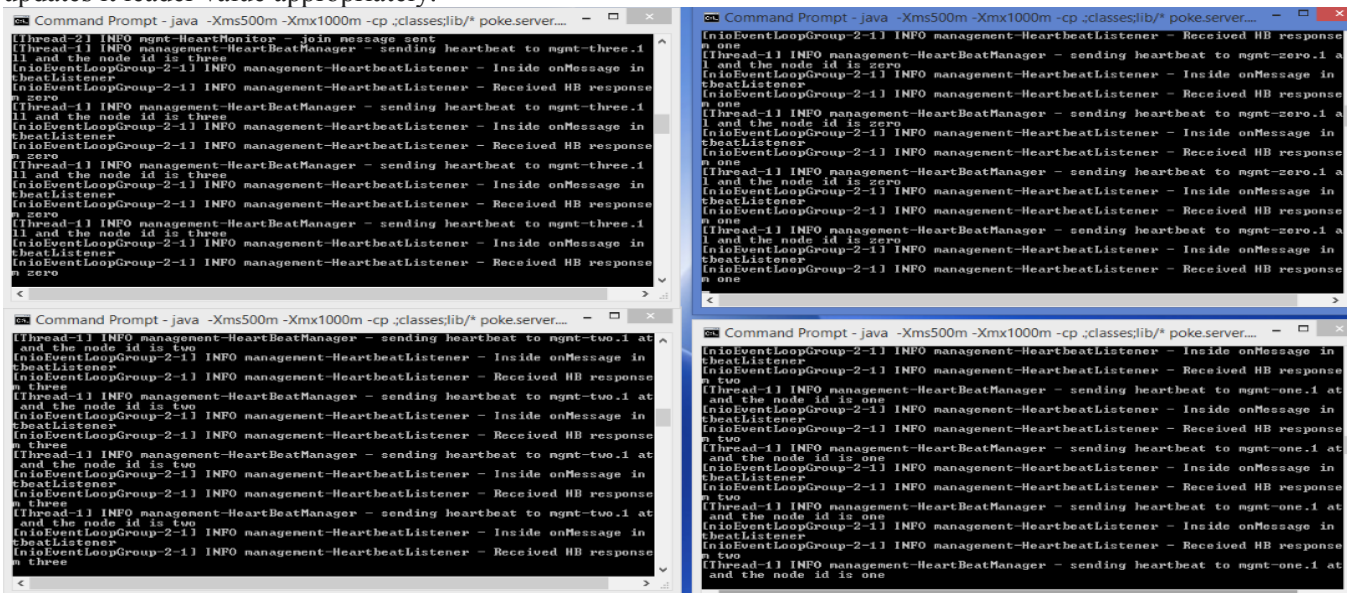
Once the leader of the cluster determines which server node can process the request (as a result of the job bid process), the leader adds the node id to the client request and publishes the request onto a Publisher-Subscribe RabbitMQ queue. Each of the other server nodes acts as a subscriber to this same queue. A node listens to a topic which is the same as its node id. For example, node 1 listens to topic "one" on the queue, node 2 listens to topic "two" on the queue, etc. Once a server node (i.e. subscriber) gets a message from the queue, it checks the job

weight of the request, adds the value to the node's total processed job weight and then processes the request. It reads the client request, accesses the database for fetching required values and builds a response to be sent back.

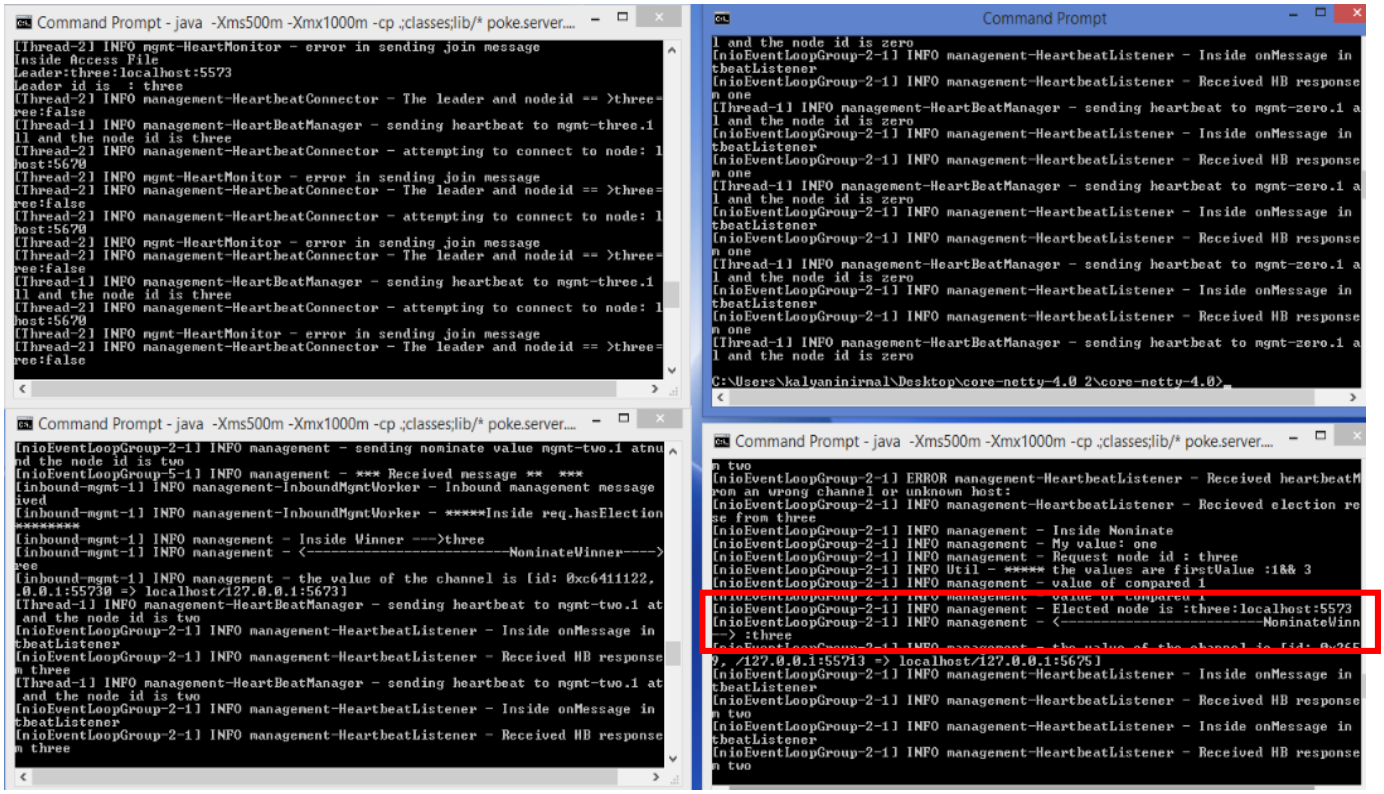
4.3 Leader Election

Leader in a MOOC improves the performance of all the nodes and improves the efficiency of each node. In a MOOC, leader's functions are to assign work to the nodes and to coordinate with other MOOCs in the Voting Process.

In this project we used a modified version of LCR algorithm. In the original LCR algorithm, the nodes are connected through a Unidirectional channel. In this project we implemented Leader Election using 2 channels, one management channel and a public channel. Management channel is used to Nominate the Leader and send the nomination message through the network and Public channel is used for declaring the winner to the other nodes. Consider a MOOC with four nodes named zero, one, two, three: Zero would be sending heart beat to Four, One would send heart beat to Zero, Two would send heartbeat to One, Three would send heart beat to Two. If node zero is the leader and it fails, node four would stop getting heartbeats from node zero. Node Four will wait for five missed beats and then start the election process. When Four starts the election, it nominates itself as the leader and sends that message to the next node through the management channel. When Three receives the nomination election message from four, it compares the nominated node value (node ID) with itself and forwards the message if the node value is greater than its own value, else (if own node ID is greater than the nominated value), a new nominated value is created and then sent to node Two. Similarly, node Two does the above and forwards the message to node One. When node One receives the message and tries to forward it to Zero and fails, as zero is down, it uses the public channel to declare the nominated value to all the other nodes. That is done by passing the declared value through the ring network in the public channel. When all the nodes receive the declared values, it updates it leader value appropriately.



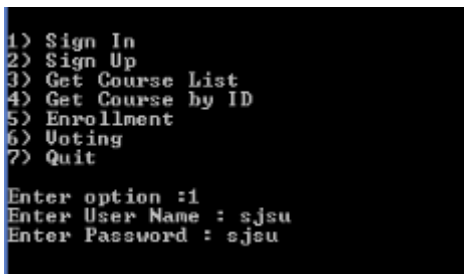
Node Three is nominated as "Leader Node" :



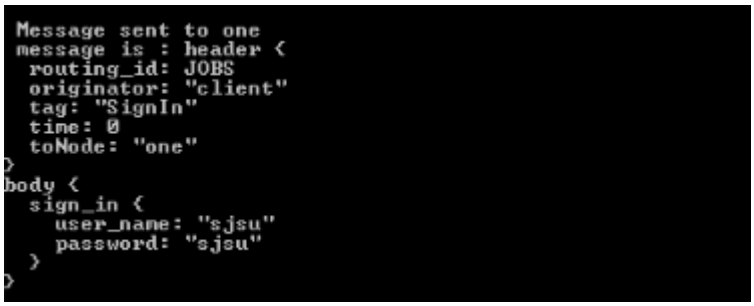
4.4 Request mechanism

Client builds the request based on user input using protobuf. Python client and Leader node both have the same port. In Python client, the given request is serialized using the SerializedtoString() and sendAll() method is used to send the request to the Leader node and only Leader node can process the request. Request goes to the inbound queue. Leader node will take this request and send it to RabbitMQ. Worker nodes are subscribed to the RabbitMQ queue. Every request has the header tag associated with it. Based on the header tag, request is processed by resource which will perform all database operations. After completing the process, request is built in the protobuf format and sent as a response to the outbound queue. Outbound queue sends to it to RabbitMQ. Leader will take it from queue and post it to its outbound queue. Client is always listening at leader port so it will receive that request using recv() method.

e.g. Sign In request sent from client.



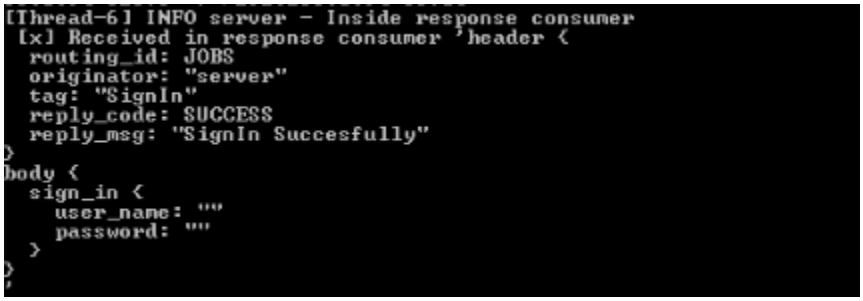
e.g. This is the SignIn request sent from python client to leader. Its header tag is set as “SignIn”.



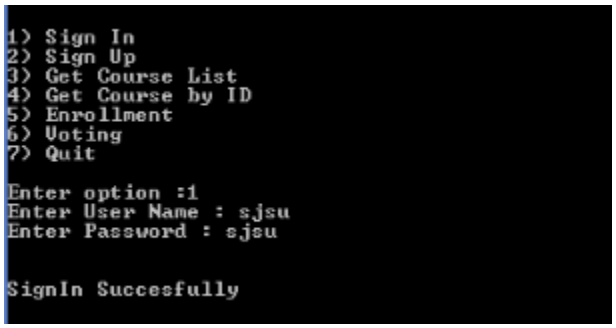
4.5 Response mechanism

After performing all database queries, worker node builds the response using protobuf builder and sends it to the RabbitMQ. Leader receives response from RabbitMQ and put it in its outbound queue. Client is always listening to leader node (both ports are same). Client receives the response from the outbound queue using the socket connection. This response is decoded using ParseFromString() function to print it in human readable format.

e.g. This is the response obtained from RabbitMQ queue to the leader node. Reply message is set as “SignIn Successfully”.



e.g. Response printed on Python Client



4.6 Voting Mechanism

The voting mechanism consists of three sections:

1. Client->Server:
   When a client chooses to initiate a voting competition among the servers, it sends a request to a server. This starts the inter-cluster voting process.

2. Inter-cluster voting:
   When a server receives a request for a voting competition, it sends out a message to all the nodes in the network and tries to find the leaders of each cluster. Once it gets the leaders information, it sends a message to all the leaders to initiate voting and waits until any leader responds. Once the first response is received from any leader, it sends this information as the winner of the competition to the client.

3. Intra-cluster voting:
   When a leader receives a message from another cluster to initiate voting, it starts a voting proposal inside its own cluster. It creates a job proposal for voting and sends it to its nearest node. The nearest node then creates a job bid with a bid value as either zero or one and forwards the message. Subsequently, each node randomly chooses either zero or one and adds it to the bid value it received from the previous node. Eventually when the leader receives the final job bid message, it now knows how many nodes have voted as Yes (i.e. one). A counter field helps determine the number of nodes that have participated in this process. If this shows that a majority of nodes in the cluster have voted as Yes, the leader sends back a response to the cluster that initiated the voting. If the majority of nodes have not voted as Yes, the leader does not send any response back.

4.7 Failover strategy

We have implemented a failover strategy in our design so that communication between nodes does not halt if any node in the cluster gets disconnected. Each server maintains the information about its nearest node and the node that is next to the nearest node. For example, in a cluster with nodes 0, 1, 2 and 3, node 1 maintains information about node 2 and node 3. Node 2 maintains information about node 3 and node 0. During the job proposal, job bidding or voting process, if any node is unable to connect to its nearest node, it then tries to connect to its next nearest node and forwards the message to that node. In the meanwhile, it also continuously tries to connect to the disconnected node and sets up a reconnection when the node is running again.

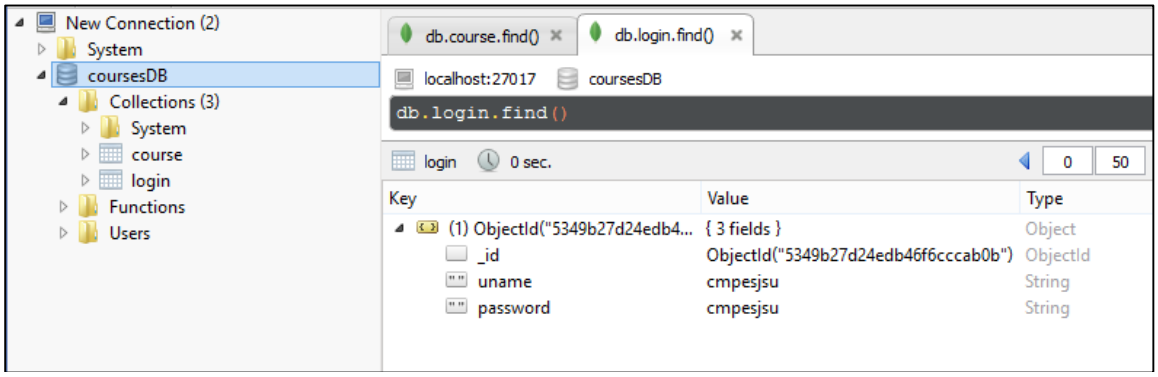
5. Storage

We have implemented MongoDB on two nodes. If a node fails to connect to with the database on the first node then it will fetch data from the database replicated on the other node. While performing write operations both the databases will get updated to maintain consistency.

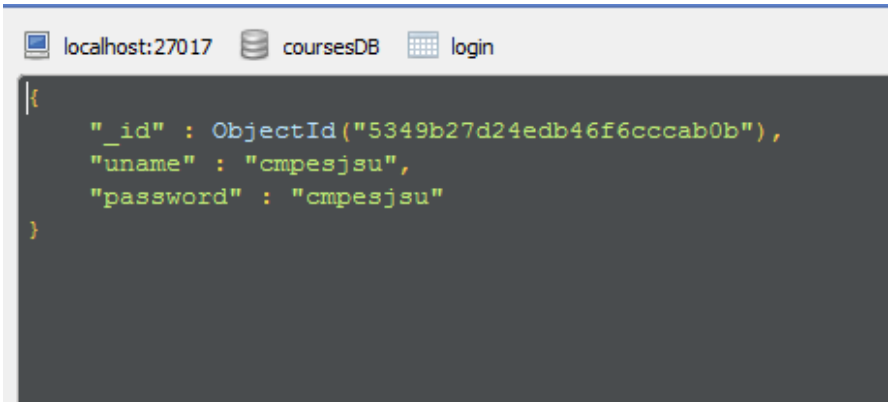
5. 1 How data is stored in MongoDB

Data is stored in the form of documents in MongoDB. For this MOOC, we have created a database called **coursesDB** that contains two collections:

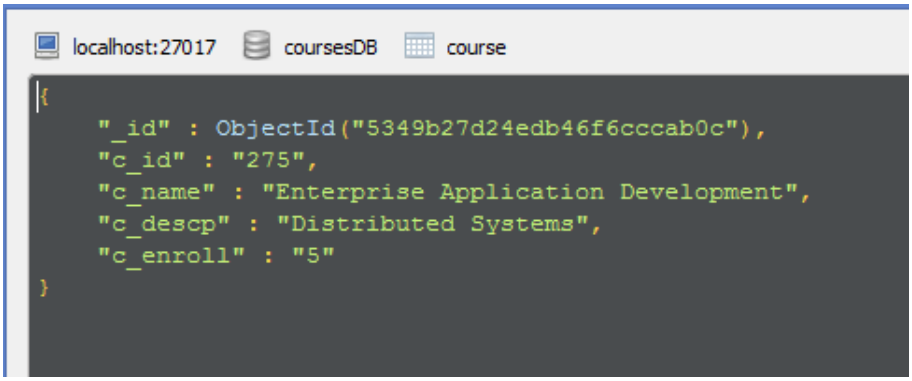
1. **login** – login collection stores details entered by user during Sign Up request. This collection is used for performing authentication of username password.
2. **course** – course collection stores the details of all available courses. This collection is used to search all available courses, search course by ID and enroll in the course.



5.2 Document Stored in login collection



5.3 Document Stored in course collection

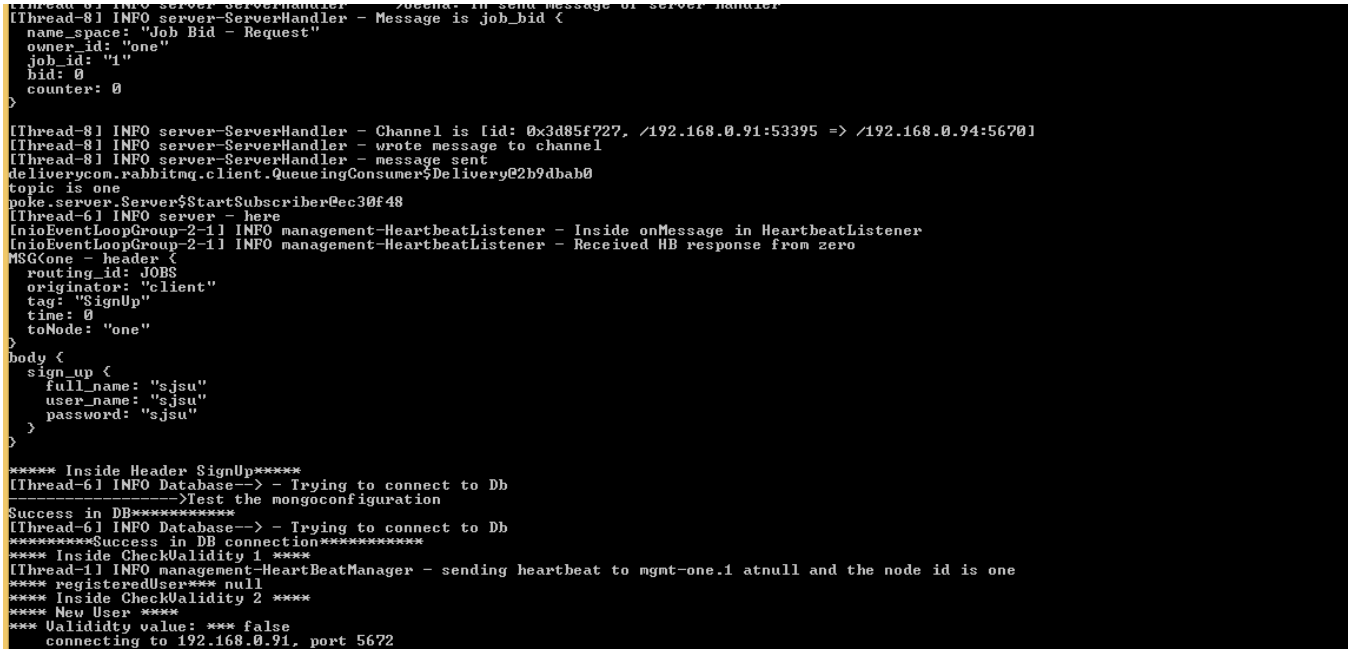


6. Scenarios implemented

These are the tasks that a user can perform using the Python client:

6.1 Sign Up operation:

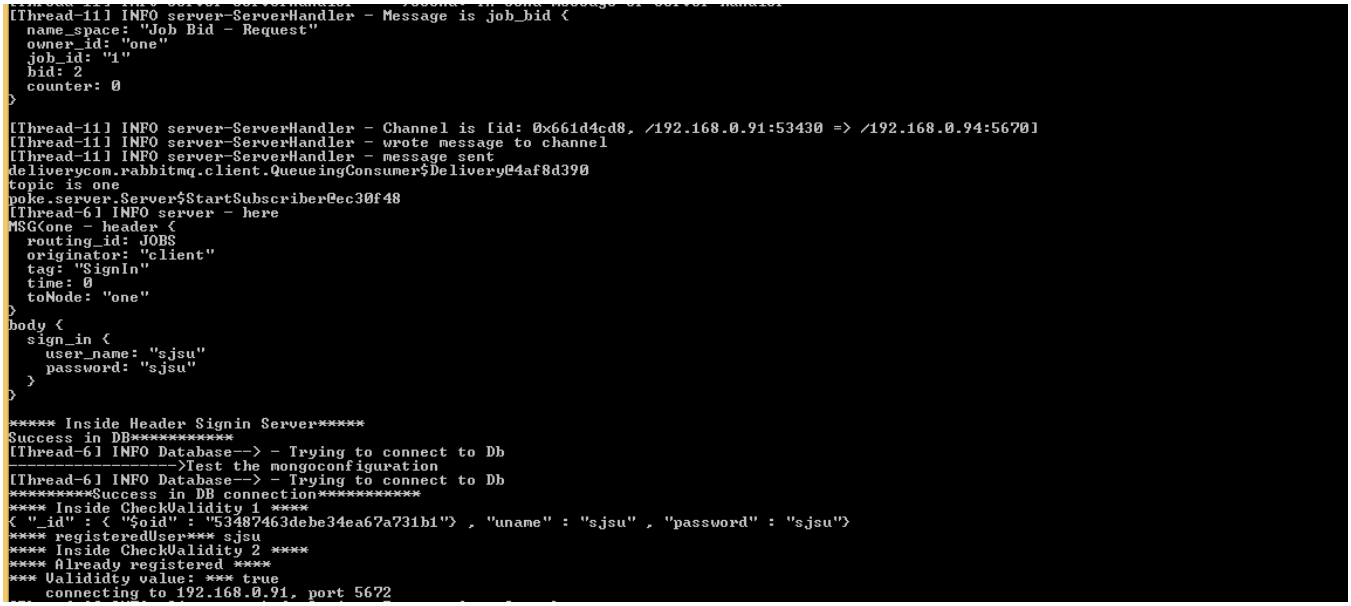
A user can enter his full name, username and password to register with our MOOC. The client gets a success message on successful registration.



Response on Client side:

6.2 Sign In operation

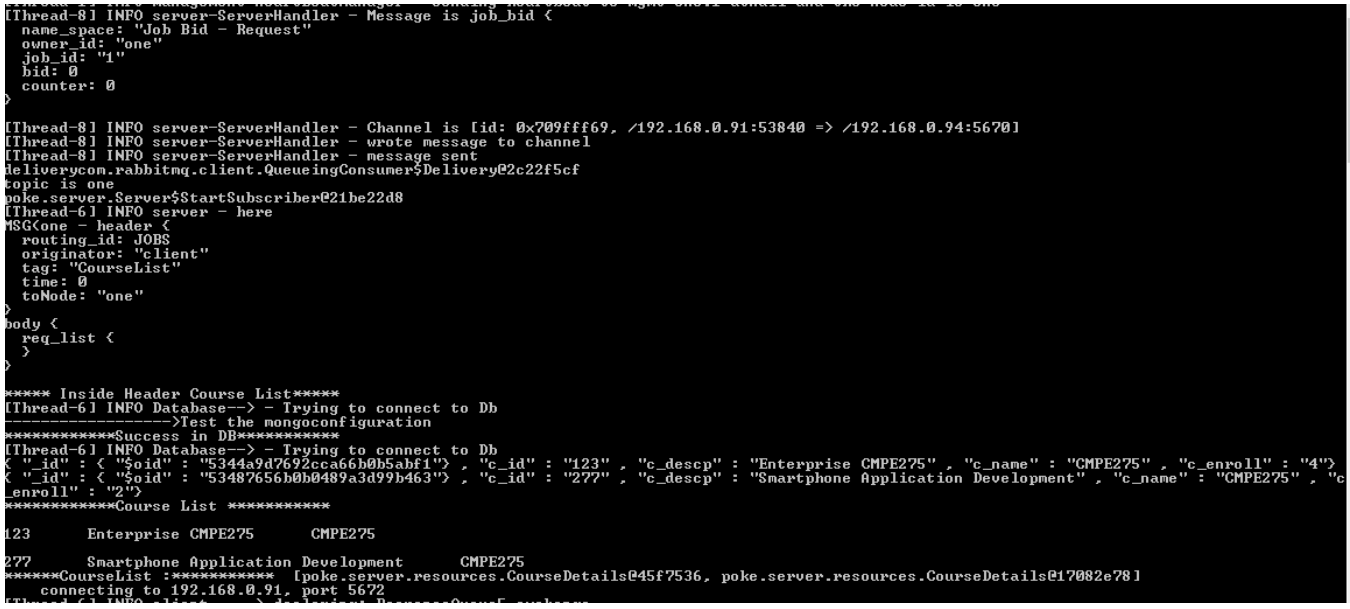
A user can enter using his username and password to sign in to the MOOC. The server validates the users' credentials and responds with a success message. Validations for existing username and correct password are checked in the functionality.



Response on Client side:

6.3 List All Courses operation

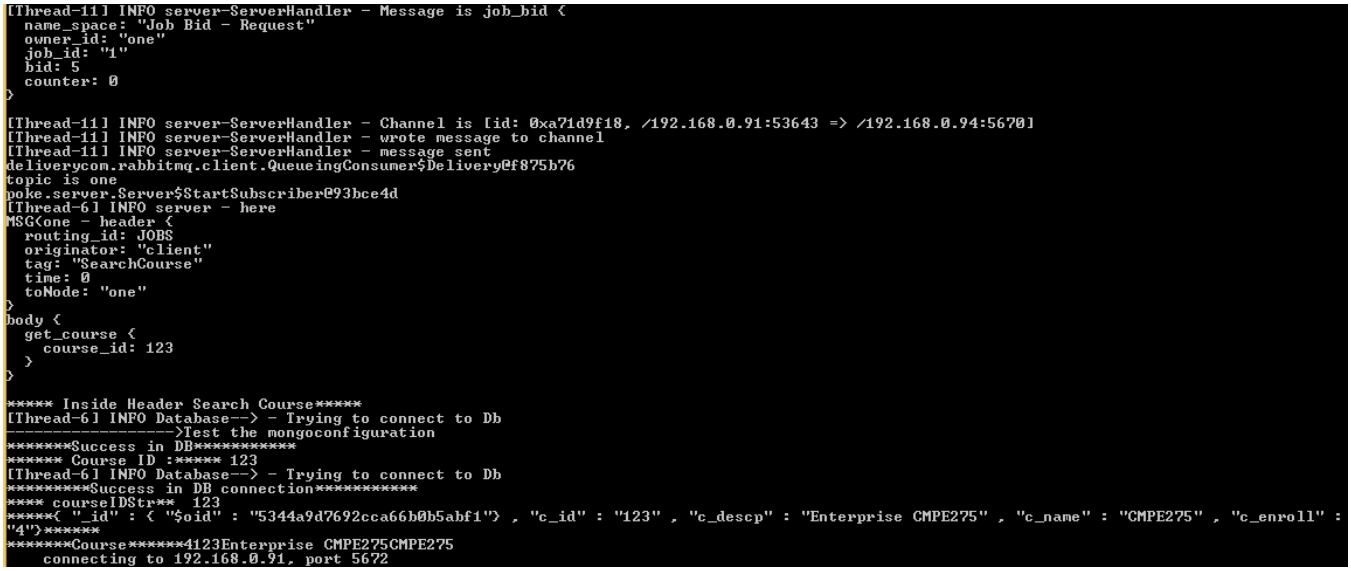
A user can choose this option to see a list of all the courses that are offered in this MOOC. The information includes course ID, name and description.



Response on Client side:

6.4 List By Course ID operation

A user can choose this option to enter a course ID and get details about the course, such as course name and description.



Response on Client side:

6.5 Enrollment:

A user can choose to enroll in a course. The courses database is updated with the number of enrollments.

```
[Thread-11] INFO server-ServerHandler - -->Jeena: In send message of server handler
[Thread-11] INFO server-ServerHandler - Message is job_bid {
  name_space: "Job Bid - Request"
  owner_id: "one"
  job_id: "1"
  bid: 5
  counter: 0
}

[Thread-11] INFO server-ServerHandler - Channel is [id: 0xb419362e, /192.168.0.91:53873 => /192.168.0.94:5670]
[Thread-11] INFO server-ServerHandler - wrote message to channel
[Thread-11] INFO server-ServerHandler - message sent
deliverycom.rabbitmq.client.QueueingConsumer$Delivery@5169d1a5
topic is one
poke.server.Server$StartSubscriber@21be22d8
[Thread-6] INFO server - here
MSG{one - header {
  routing_id: JOBS
  originator: "client"
  tag: "Enroll"
  time: 0
  toNode: "one"
}
body {
  get_course {
    course_id: 123
  }
}

***** Inside Header Search Course*****
[Thread-6] INFO Database--> - Trying to connect to Db
------------------>Test the mongoconfiguration
*******Success in DB**********
****** Course ID :***** 123
[Thread-6] INFO Database--> - Trying to connect to Db
*********Success in DB connection**********
**** courseIDStr**  123
*****{ "_id" : { "$oid" : "5344a9d7692cca66b0b5abf1"} , "c_id" : "123" , "c_descp" : "Enterprise CMPE275" , "c_name" : "CMPE275" , "c_enroll" :
"4"}******
*******Course******4123Enterprise CMPE275CMPE275
Enroll count: 5
[Thread-1] INFO management-HeartBeatManager - sending heartbeat to mgmt-one.1 atnull and the node id is one
    connecting to 192.168.0.91, port 5672
```

Response on Client side:

```
1) Sign In
2) Sign Up
3) Get Course List
4) Get Course by ID
5) Enrollment
6) Voting
7) Quit

Enter option :5

Enter Course ID to Enroll : 123

Course ID : 123
Course Name : CMPE275
Course Description : Enterprise CMPE275
Enrollment Status : Enrolled Succesfully
```

7. Design Limitations

1. Ring architecture invokes the classic disadvantage of communication latency, where communication delay is directly proportional to the number of nodes. We have tried to minimize this latency in Leader Election by making the communication bi-directional.
2. Moving or adding new nodes to the network can affect the network.
3. If the leader goes down, there is no stand by node to take over until the leader election is complete.

8. Future Enhancements

1. We could implement security mechanisms such as encryption of messages before forwarding messages to the server nodes and encryption of user information before storing in the database.
2. If the data requested by the client is unavailable within our cluster, we could forward it to other clusters.
3. We could design a graphical user interface to enhance the user experience.

9. Testing

9.1 Test Cases

S.No	Test Case	Expected Outcome	Test Case Status	Test Case Pass/Fail
1	Leader node gets disconnected	Leader election starts and new cluster leader is elected.	Test complete	Test Passed
2	Leader node gets disconnected.	Client requests are still maintained in RabbitMQ.	Test complete	Test Passed
3	Node gets disconnected	Ring network does not break. New connections are formed between nodes.	Test complete	Test Passed
4	Disconnected node is available.	Connection is re-established.	Test complete	Test Passed
5	Database connection lost	Maintain data consistency and continue processing client requests.	Test complete	Test Passed
6	Request Response	Get the correct response based on the database and header tag.	Test complete	Test Passed

9.2 JUnit Testing

JUnit Test cases have been implemented to test simple functionalities for database connection, enqueueing and dequeueing from the public channel and for testing the publisher-subscriber network of the RabbitMQ. All test cases have passed.

10. References:

1. Retrieved from: http://ieeexplore.ieee.org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=6681230
2. G. Siemens "Massive Open Online Courses: Innovation in Education?" eds. Commonwealth of learning, Perspectives on Open and Distance Learning: Open Educational Resources: Innovation, Research and Practice, p. 5. , 2013, retrieved from http://www.col.org/resources/publications/Pages/detail.aspx?PID=44
3. Retrieved from: http://searchdisasterrecovery.techtarget.com/feature/Disaster-recovery-replication-guide-A-look-at-data-replication-tools
4. Retrieved from: https://developers.google.com/protocol-buffers/docs/overview
5. Retrieved from: http://docs.jboss.org/netty/3.1/guide/html/architecture.html
6. Retrieved from: http://ieeexplore.ieee.org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=6654451&tag=1
7. Retrieved from https://www.rabbitmq.com/features.html