Project 2 : Bulletin Board

Mayura Nene (Student id: 5589174, email: nene0002@umn.edu)
Roopana Chenchu (Student id: 5595508, email: vuppa007@umn.edu)
Ranja Bati Sen (Student id: 5586108, email: sen00016@umn.edu) Date : April 06, 2020

Objective of the project:

To design a bulletin board where multiple clients can post articles, read a list of articles, choose an article to see its content and reply to an existing article. There exist multiple servers with a coordinator server. 3 kinds of consistencies are implemented: Sequential consistency, Quorum consistency and Read-your-Write consistency.

Design: Main components involved:

1. Client
   - Client UI (class Name: ClientUIConnectorServlet)
   - Client Server (class Name: ClientExecutor)
2. Server
   - Coordinator Server (class Name: CoordinatorServer)
   - Server (class Name: ServerExecutor)

Client:

For the Client, we have created a dynamic web project. It has two parts, code related to HTML User Interface and code for interaction of client and server.

Client UI:

1. index.html :- The HTML page which displays the form for selecting type of consistency and type of operation to be performed (post/read/reply/choose).
2. ClientUIConnectorServlet :- Java Servlet code for receiving form data from html page and sending it to the Client Server via UDP and then receiving output from Client Server via UDP and displaying the output.
3. On User Interface, user can select which type of consistency he wants to test as well as specific function to test.
4. On User Interface, we implemented pagination, i.e only 5 articles will be displayed on the interface, if more than 5 articles are present, Next button is present which will display next 5 articles
5. We also display time taken for each operation initiated by the user.

Client-Server Interaction code:

1. ClientExecutor.java :- This is the main class of the client-server interaction. When we run the jar, a client creates TCP socket with server and UDP socket with the User Interface servlet.
2. Once connection with server is established, client requests server to give list of all the up and running servers.
3. Then client waits for user to provide the input regarding type of consistency and the method (Read, Choose, Post, Reply)
4. Once user specifies the consistency type and method along with the necessary input, client sends this information to the server via TCP socket and waits for the reply.
5. Once reply from server is available, client passes it to the UI code via UDP socket and it is displayed to the user.

Coordinator Server:

1. In the coordinator server, *CoordinatorServer* is the entry point. At the coordinator server following things are created in separate threads.
   a. Socket for coordinator to which other servers will connect
   b. Socket for running synchronization operation

c. Socket for coordinator as a server to which other clients will connect

2. Coordinator server does following tasks upon different messages from server (ServerConnectionsHandler.java) :
   a. POST and REPLY for Sequential consistency & Read-your-write consistency
      i. It broadcasts the article to other servers and waits for their acknowledgements. (BroadcastHandler.java)
      ii. After receiving acknowledgements from all the servers, coordinator writes response to originator server telling POST/REPLY is successful.
      iii. In case of error, error is communicated to the server and then to the client.
   b. CHOOSE, READ, POST, REPLY for Quorum Consistency
      i. It first checks if the provided quorum (nr, nw) values are valid as per the number of servers currently running. In case of error, appropriate error message is returned.
      ii. If valid quorum is present, it randomly chooses the servers needed for the quorum and broadcasts the operation to them.
      iii. After receiving acknowledgements from all the servers in the quorum, coordinator writes response to originator server telling operation is successful.
      iv. In case of error, error is communicated to the server.
3. Coordinator assigns unique article ID to each of the article. After each update operation (POST/REPLY), coordinator server updates the articles map and in case the article is reply to another article, it is added to the replies list of the parent article.
4. Every 90 seconds, SYNCH operation is initiated by the coordinator. In quorum consistency, replicas can get out of synch, SYNCH operation makes all the replicas consistent periodically.
5. Additionally, Clients can also connect to the coordinator server and do all the normal operations. This is handled in *ServerExecutorAtCoord.java*

Server:

1. For the server program, entry point is *ServerExecutor.java*. It creates following things in standalone threads
   a. Socket for connection to coordinator
   b. Socket for accepting connections from clients
2. Server keeps on listening to client requests on the socket. Once a request is received, it does below tasks based on the incoming message *(ServerClientConnectionThread.java, ServerServerConnection.java)*
   a. READ/CHOOSE for Sequential consistency & Read-your-write consistency
      i. The server will send the response using the article map it has.
   b. POST/REPLY for Sequential consistency
      i. The server will send the request to coordinator and wait for coordinator's acknowledgement
      ii. Once it receives acknowledgement from coordinator, server updates its articles map and writes response to clients.
      iii. In case of error, error is communicated to the client.
   c. POST/REPLY for Sequential consistency
      i. The server will send the acknowledgement to the client.
      ii. It will send request to coordinator and wait for coordinator's acknowledgement
      iii. Once it receives acknowledgement from coordinator, server updates its articles map and
      iv. In case of error, error is communicated to the client.
   d. READ/CHOOSE/REPLY/POST for Quorum Consistency
      i. For all the operations in quorum consistency, server will send request to coordinator to get majority from quorum.

ii. Once response form coordinator is received, server sends it to the client.

3. Server maintains map of all the clients connected to it.
4. Upon receiving SYNCH request, the server sends its current article list to coordinator. The coordinator computes the latest versioned articles and communicates back to the server.

Procedure to execute:

1) Following executable files are created.
    a. bbCoordServer.jar - executable for coordinator server
    b. bbServer.jar - executable for Server
    c. bbClient.jar - executable for client
    d. bb_clientside.war - UI web application
2) Do the following steps to run the code.
    a. Install tomcat on the machine. (If linux machine, do the following)
        i. wget https://www.apache.org/dist/tomcat/tomcat-9/v9.0.33/bin/apache-tomcat-9.0.33.tar.gz
        ii. tar xzf apache-tomcat-9.0.33.tar.gz
        iii. mv apache-tomcat-9.0.33 /home/your_user_home/tomcat9
        iv. echo “export CATALINA_HOME=”/home/your_user_home/tomcat9” >> ~/ .bashrc
        v. source ~/ .bashrc
        vi. cd tomcat9/
        vii. chmod +x ./bin/startup.sh
        viii. ./bin/startup.sh
    b. Run bbCoordServer.jar by passing 5 arguments – coordinator port, synchronizer port, port for server at coordinator, N_R, N_w
        i. Example: java -jar bbCoordServer.jar 5057 5058 5075 1 2 (for 2 servers - coordinator server, participant server)
    c. Run bbServer.jar by passing 6 arguments – coordinator IP, coordinator port, synchronizer port, port for server, N_R, N_w
        i. Example: java -jar bbServer.jar 127.0.0.1 5057 5058 5060 1 2
    d. Run bbClient.jar by passing 3 arguments – server IP, server port, port for UI
        i. Example: java -jar bbClient.jar 127.0.0.1 5060 1234
    e. Put bb_clientside.war in webapps folder inside tomcat9
    f. Put the following four jars in lib folder inside tomcat9 - javax.servlet-api-4.0.1.jar, jackson-annotations-2.11.0.rc1.jar, jackson-core-2.11.0.rc1.jar, jackson-databind-2.11.0.rc1.jar
    g. Stop the tomcat if running currently,
        i. ./bin/shutdown.sh
    h. Start the tomcat
        i. ./bin/startup.sh
    i. Go to the browser and type localhost:8080/bb_clientside
    j. Note – port for UI (3rd argument given while running bbClient.jar) and Port which is asked on the HTML UI should be same.
    k. To bring up multiple clients, run bbClient.jar with different uiport than already running client and open a new tab for localhost:8080/bb_clientside with the same uiport used for running bbClient.jar.

Test cases:

Run all the steps as mentioned above. Following tests are performed for the combinations:

1. 2 servers with 1 client and 1 coordinator

2. 2 servers with 2 clients and 1 coordinator
3. 4 servers with 3 clients and 1 coordinator

Description	Response	Result
Concurrent Clients post article to multiple servers and then perform read operation (Sequential consistency)	All clients see same ordering of the articles	Pass
Client posts article on 1 server and then reads from server 2 (Read your write consistency)	Client sees prior updates	Pass
Client can connect to any server including coordinator server	Connection is accepted by the server	Pass
Client can post article to the server	Client receives acknowledgement from the server	Pass
Client can read article from the server	Client receives articles and their list of replies	Pass
Client can choose particular article	Client receives content of the article	Pass
Client can reply to the particular article	Client receives acknowledgement from the server	Pass
Error Handling:		
Invalid value of read/ write quorum i.e Following conditions are violated – 1. $N_R + N_w > N$ 2. $N_w > \frac{N}{2}$	Client receives response that invalid quorum	Pass
In CHOOSE, client inputs invalid article ID	Client receives response that article does not exist	Pass
If there are no articles on server and client does READ	Client receives response that there are no articles to display	Pass
Indentation & Pagination	Read function displays properly indented articles and nested replies with 5 articles per page	Pass

Profiling of consistency operations:

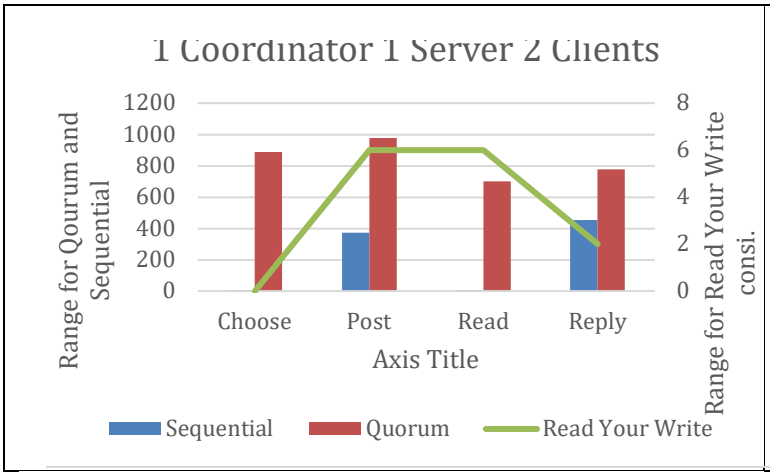
Trends in Total Time taken for each operation are represented in graphs and is calculated as below:

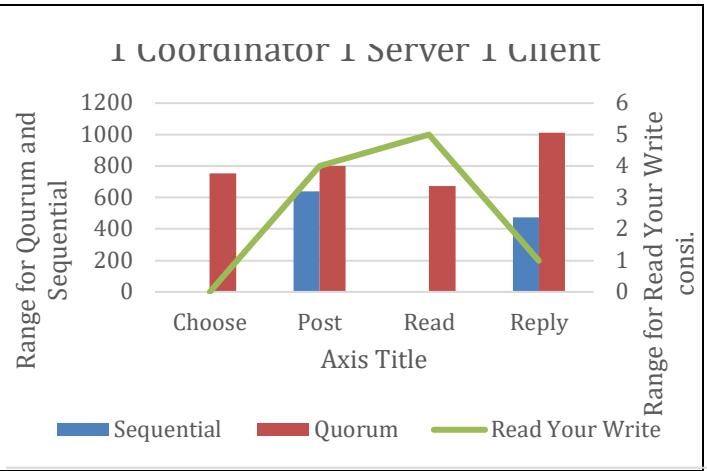
a. Start time - time at which operation is initiated by client
b. End time - time at which client receives response
c. Total time taken = End time – start time

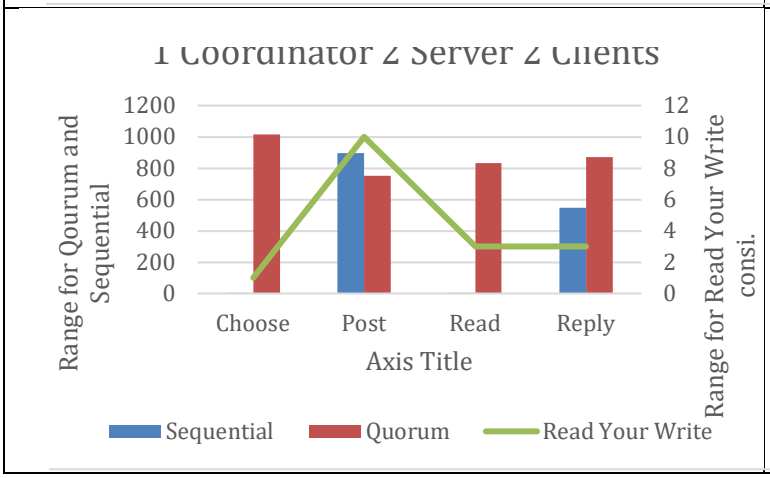
Comparative Analysis of all operations with all consistencies:

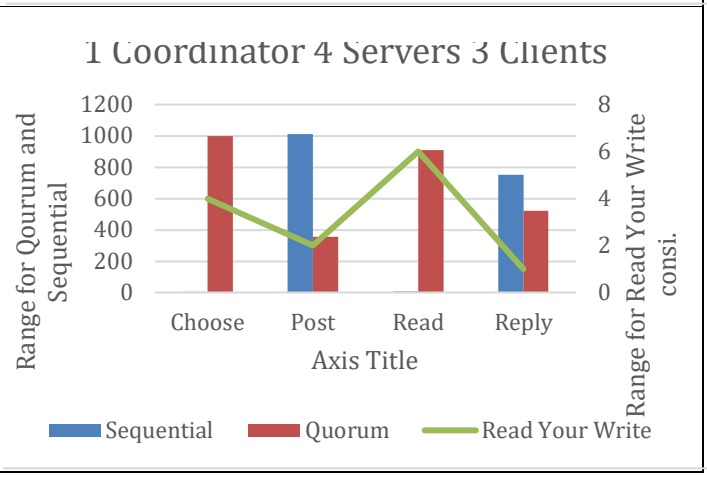
1) We present total time of execution for different number of servers, clients and all type of operations in all consistencies. (Note: Some values are really less, to fit in the scale, hence not visible. Also, green line shows values for RW consistency for which scale on the right side should be referred)

2) Observations and analysis are as follows:
   a. Lowest time is taken when client performs all the operations in Read-your-write consistency. (Because primary migrates to the server where data item is being edited)
   b. As number of servers increases, time taken for POST and REPLY in sequential consistency increases than quorum consistency (because in sequential, we broadcast to every server)
   c. The execution time for reply is in general greater than CHOOSE/POST since it involves both adding a new article (POST)and updating an existing article (CHOOSE).





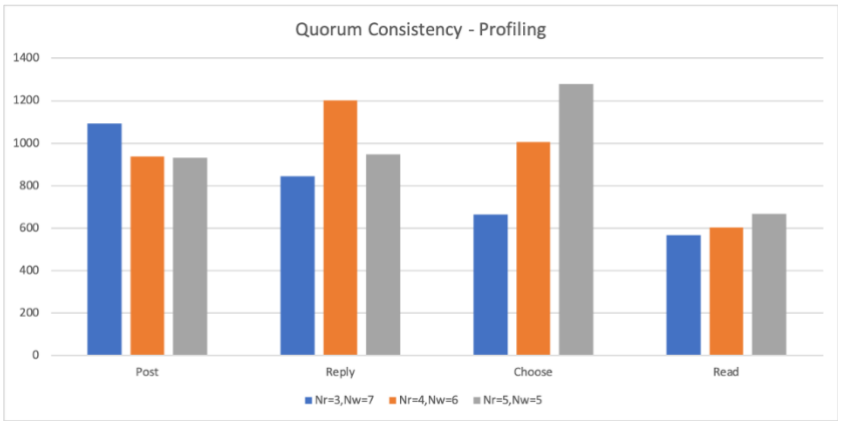




Analysis of all operation with varying values of quorum N_R, N_w:-

We have analyzed the quorum consistency profile for 9 servers with N_R, N_w as (3,7), (4,6) & (5,5).

1. From the graph, it can be noted that as N_R increases, time required to perform READ and CHOOSE operation increases which can be accounted for increase in the communication between servers.
2. From the graph, it can be seen that as N_w increases, time required to perform POST and REPLY operation increases due to more communication between servers.



Limitations

- On the HTML UI, we did not get time to do validation checks. So please provide all the necessary input and then press submit button.
- Coordinator server needs to be started before any participant servers

- Sometimes, we were getting time out errors when trying to run on multiple CSE lab machines. However, it worked on multiple local machines and also single CSE lab machine

Resources

Source code and executables are provided in the zip file.

Additionally, Please find the source code and screenshots of the output for different test cases :-

Client & UI – https://github.com/Roopana/distributedsystems_bulletinboard_client

Server and Coordinator - https://github.com/Roopana/distributedsystems_bulletinboard_server

Race conditions

In quorum consistency, while checking acknowledgements from servers to identify the latest version of article in choose/ read operations, multiple threads can respond to the coordinator server at the same time, hence the number of acknowledgements although satisfy the quorum might not be incremented correctly. We tried to implement ReentrantReadWriteLock and synchronization to mitigate this. This is a generic issue observed whenever a message is broadcasted to participant servers and multiple servers send their responses to the coordinator.