CSCI - 5105 Introduction to distributed systems
Project 2: Bulletin Board Consistency

Rohit Sindhu [**sindh010**], Aravind Alagiri Ramkumar [**alagi005**], Dhruv Kumar [**kumar434**]

*[See **README.md** in the code folder for instructions on running the code and test cases.]*

Note: Every consistency protocol implementation share the following features:

1. Every server stores a local article database in memory.
2. We can fetch any article from the local article database on any server using the article id.
3. We can also read the entire article database in one request.
4. Each consistency protocol's implementation involves having a primary server or co-ordinator.
5. A sequencer is maintained in the co-ordinator which keeps track of the last generated article Id. Whenever any client sends a POST/REPLY request to any server, the server asks the primary server for the next article id. The primary server increments the last generated article Id by 1 and sends this updated Id back to the requesting server. This is done in a mutually exclusive manner so that no two requesting servers have the same article Id returned to them.
6. All the read/write operations are performed with delays to mimic wide area network.

Sequential Consistency:

1. **POST/REPLY Operation:** Whenever a client sends a POST/REPLY request to the server,
    a. The server first gets the new article ID from the co-ordinator and then sends the actual article content to the primary server along with the new article Id.
    b. Once the primary server has received the article id along with the actual article content, it first updates its local article database and broadcasts this update to all the other servers.
    c. When any server receives a write request from the primary server, it writes the article into its local database and sends the acknowledgement to the primary server.
    d. The primary server waits for acknowledgement from all the servers and then sends the "Write Complete" acknowledgement to the server from which it received the write request initially i.e. the server to which the client actually sent the write request.
    e. This server then sends the "Write Complete" acknowledgement back to the client.
2. **CHOOSE Operation:** Whenever a client requests for the contents of any article Id,
    a. The server retrieves the requested article ID from its own database and returns the content of the article.
3. **READ Operation:** Whenever a client requests for the entire article database,
    a. The server reads its own database, formats the data in the expected bulletin board format and returns it to the client.

Quorum consistency:

1. **POST/REPLY operation:** Whenever a clients sends a POST/REPLY request to the server,
    a. The server first gets the new article ID from the co-ordinator.
    b. The server then assembles the write quorum to perform the write operation.
    c. The write quorum is established by choosing N_W servers randomly and making these N_W chosen servers consistent using a *sync* function across those Nw servers.
    d. The *sync* function fetches the entire database from all the chosen servers, identifies the latest updated database and then sends the delta updates (only those updates which are not present in each server) to each of the chosen servers.
    e. Once these servers are consistent, the server (on which the client request came) broadcasts the write request to all the chosen servers including itself. The server waits for acknowledgement from all the Nw servers and then sends the "Write Complete" acknowledgement back to the client.
    f. When any server receives a write request from any other server, it just writes the article into its local database.
2. **CHOOSE Operation:** Whenever a client requests for the contents of any article Id,
    a. The server first assembles the read quorum by choosing N_R servers randomly.
    b. The server then requests the maximum article id from these N_R chosen servers.
    c. Once the requested article Id is found in any one of the N_R servers, the server finally returns the content corresponding to that article Id.
3. **READ Operation:** Whenever a client requests for the entire article database,
    a. The server first assembles the read quorum by choosing N_R servers randomly.

b. The server then requests the maximum article id from each of the N_R chosen servers and chooses the server which returned the maximum Id. Then the server fetches the database from the selected server, formats the data in the expected bulletin board format and returns it to the client.

4. **SYNC thread:** A separate thread runs at the co-ordinator which keeps executing the *sync* function (described in 1d above) at a regular interval. This thread synchronizes the article database across all the threads.

Read your write Consistency:

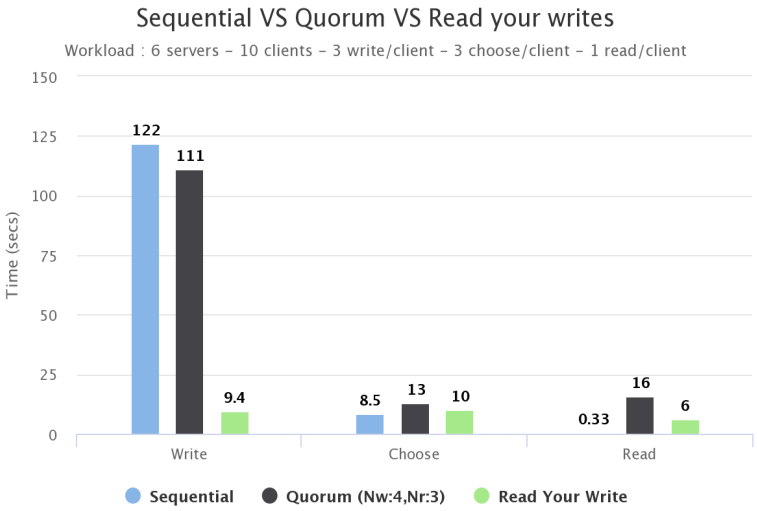
4. **POST/REPLY Operation:** Whenever a client sends a POST/REPLY request to the server,
   a. The server first gets the new article ID from the co-ordinator and updates its local database with the new article id and the new article's content.
   b. It then broadcasts this update to all the other servers. This is a non-blocking broadcast i.e. without waiting for the acknowledge from all the servers, the server sends the "Write Complete" acknowledgement back to the client.
   c. When any server receives a write request from any server, it just writes the article into its local database.
5. **CHOOSE Operation:** Whenever a client requests for the contents of any article Id,
   a. The server checks if the requested article ID exists in its own database.
   b. If the id does not exist, the server requests the maximum article id (last generated id at the server) from the co-ordinator. If the maximum article id is greater than or equal to the requested article id, the server waits for its local database to receive the update corresponding to the requested article id. As soon as the update becomes available in the local database, the server returns the content corresponding to that article Id to the client.
6. **READ Operation:** Whenever a client requests for the entire article database,
   a. The server first of all, requests the maximum article id (last generated id at the server) from the co-ordinator.
   b. If the max article Id at the co-ordinator is greater than the max article id in the local database in the server, the server waits for its local database to receive all the updates.
   c. As soon as all the updates have been written into the local database, the server reads its own database, formats the data in the expected bulletin board format and returns it to the client.

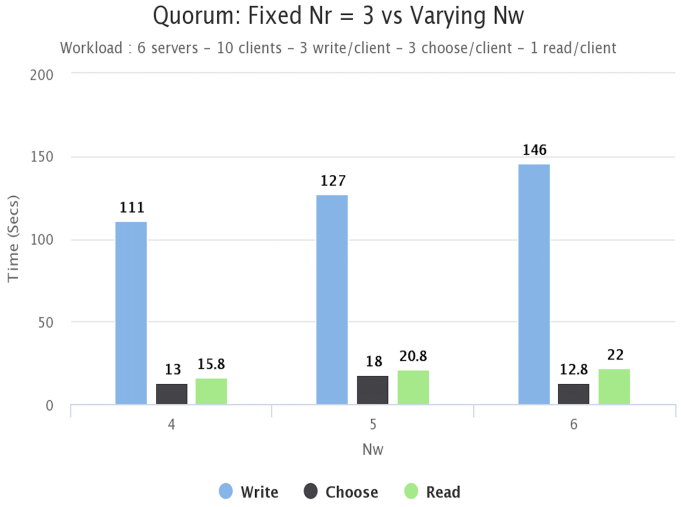
Test Cases attempted:

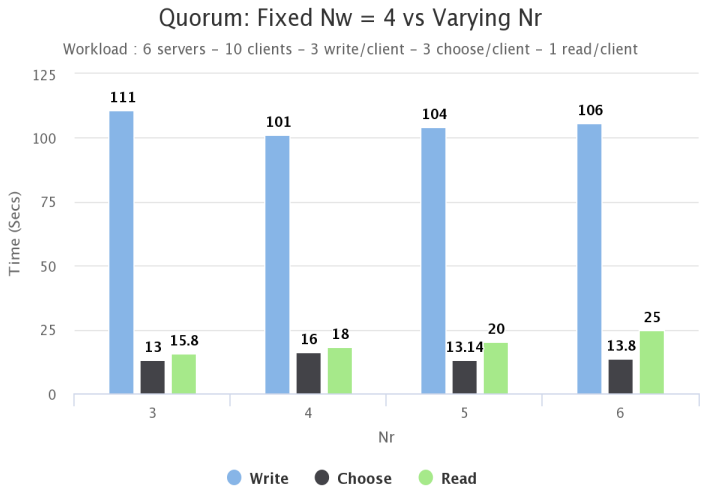
(All the test cases described below are present in separate Java files. Comments are included in the code)

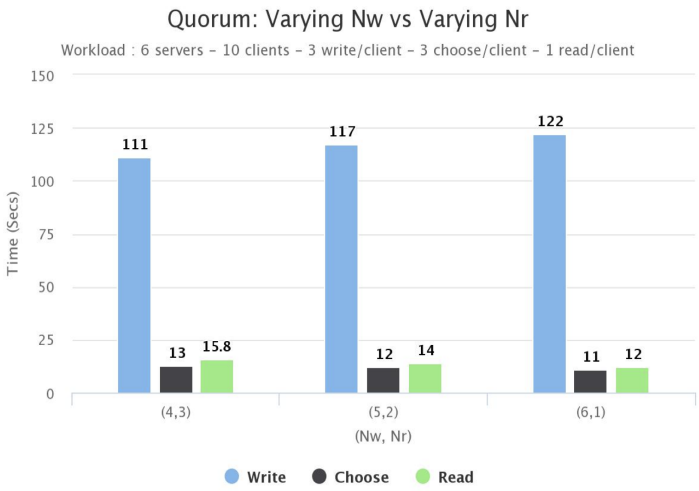
1. **ClientTestDriver:** Multiple clients perform operations on multiple machines concurrently.
   a. First, multiple clients do multiple POST/REPLY concurrently.
   b. Then, multiple clients do multiple CHOOSE concurrently.
   c. Finally, multiple clients do multiple READ concurrently.
   d. For each write/read, every client connects to a randomly chosen server and performs a write/read request.
   e. This just makes sure that the service allows concurrent read/write from multiple clients on multiple servers.
2. **TestSequentialConsistency:** Testing Sequential consistency:
   a. First perform a bunch of writes.
   b. Store the write log (the order in which the articles were written into the database) of the local database at each server.
   c. In the end, compare the write log of all servers. It should be exactly the same.
3. **TestQuorumConsistency:** Testing Quorum consistency:
   a. First perform a bunch of writes.
   b. Compare the article databases of all the servers. At least, Nw servers should have the latest database.
4. **TestReadYourWriteConsistency:** Testing Read your write consistency:
   a. First perform a write and then immediately a read from any randomly selected server. The read should have the last written value.
   b. Perform the previous step multiple times from multiple clients concurrently.
5. **Sync function:** Testing the sync function used in Quorum consistency:
   a. This is being tested in TestQuorumConsistency implicitly. The sync function works correctly if the TestQuorumConsistency Test case passes.

Charts : ***[All the readings below are average of multiple runs]***









Analysis of results and charts:

1. **Sequential vs Quorum vs Read Your Write:** As can be seen from the graph, Write operations for Read your write consistency are very fast as compared to the other two consistencies. This is because Read Your write consistency does lazy or non-blocking writes whereas the other two consistencies perform blocking writes. They send write acknowledgement to the client only when the write has been performed on the required number of servers. The read operations are slowest in Quorum consistency because we read from multiple servers where as in other consistencies, we only read from a single server. The reads are faster in sequential consistency as compared to Read Your write consistency because a read request may have to wait for the lazy updates in read your write.
2. **Quorum: Fixed Nr = 3 vs varying Nw:** As can be seen from the graph, the time taken for write operations increases with an increase in Nw. This is because larger the number of servers to write, larger would be the delay. The read operations don't show much variation since the Nr is same for all the values of Nw.
3. **Quorum: Fixed Nw = 4 vs varying Nr:** As can be seen from the graph, the time taken for write operations does not vary significantly since the Nw is same for all the values of Nr. The time taken for read operations increases gradually as the Nr increases.
4. **Quorum: varying Nw vs varying Nr:** As can be seen from the graph, the time taken for write operations increases with an increase in Nw. This is because larger the number of servers to write, larger would be the delay. The time taken for read operations decreases gradually as Nr decreases.

Description of Individual Components/Java files:

1. **Article:** Each object of this class stores details related to a particular article - its id, its parent id if its a reply, list of replies and its actual content.
2. **DataService:** Each object of this class is a database storing (id, Article) pairs. Each server instantiates one such object as its local database.
3. **ConsistencyProtocol:** An abstract class which is extended by individual consistency protocols. Server uses this class for deciding the consistency policy for making updates to the local database and sending updates to other servers.
4. **SequentialConsistency:** This class extends the ConsistencyProtocol and implements the sequential consistency.
5. **QuorumConsistency:** This class extends the ConsistencyProtocol and implements the Quorum consistency.
6. **QuorumSyncThread:** This class implements the logic used by the background thread which syncs the server replicas using the sync function.
7. **ReadYourWriteConsistency:** This class extends the ConsistencyProtocol and implements the ReadYourWrite consistency.
8. **Bulletin_Board:** Each object of this class stores all the details related to a particular server. Every server has one such object. It includes reference to the local database, the type of consistency implemented by the server, the list of other servers in the group, server Id, primary Server Id, write log, max article id, listener port etc.
9. **TCPConnection:** A helper class for managing socket connections.
10. **RequestHandler:** A helper class used for parsing the incoming request at the server and deciding what to do next based on the actual message in the request.
11. **RequestListener:** A helper class used by the server to listen to incoming connections and start a new thread (RequestHandler) for each incoming request. The thread exits when the request has been fully executed.
12. **ServerInfo:** This class stores the ip address and port number of any server.
13. **UpdateHandler:** A helper class used by the server for broadcasting updates to other servers.
14. **Utils:** A utility class which has methods used by various classes across the project.
15. **TestRead:** A test helper class which performs multiple reads on multiple servers.
16. **TestWrite:** A test helper class which performs multiple writes on multiple servers.
17. **Server:** Initiates multiple servers listening at multiple ports on the same machine.
18. **IP.properties:** stores the ip address and port number for each server. If you want to run the server at a different port, edit in this file.
19. **CONSTANTS:** this class has the constants used by the entire project.
20. **ConsistencyTypes:** This just stores the enum for different types of consistency protocols.
21. **Server:** Driver for server. Once you have stored the list of servers in IP.properties, run this script as instructed in README.md.
22. **ClientTestDriver:** Test driver for the client. This performs multiple reads, writes at multiple servers concurrently.
23. **TestSequentialConsistency:** Test driver for sequential consistency.
24. **TestQuorumConsistency:** Test driver for quorum consistency.
25. **TestReadYourWriteConsistency:** Test driver for Read your write consistency.