Project 2

User Experience

In our system we have divided all kind of articles into 2 group (Article and Reply). When user asks for a list of articles, he will only see articles excluding replies. Then user can choose one articleid and see all the associated replyids. Similarly user has both the options to create a new article or reply to an existing article. Our implementation is close to the user experience that we see in moodle.

API exposed to Client

Our client is serverless and is free to call any server endpoint using below API definition.

```
List<Article> readArticle(int articleid, int maxidseenyet) throws RemoteException;
Article[] readArticles(int id,  int maxidseenyet) throws RemoteException;
int postArticle(String content, int parentReplyId, int parentArticleId) throws RemoteException;
ArrayList<ServerInfo> getListOfServers() throws RemoteException;
```

Client will only know the leader's IP and Port number. Client has to call getListOfServers() to get all the available servers that client can connect to. readArticles() function is used to get all the articles (excluding replies). The parameters id and maxidseenyet are used for pagination and as a token for ReadYourWrite protocol respectively. readArticle takes an articleid and returns all the replies and multilevel replies(if exists), which clients uses to print in a well formatted way with indentations. postArticle() is used to post an article. parentReplyId and parentArticleId parameters are used to track if it is a reply to an existing article or it is a new article.

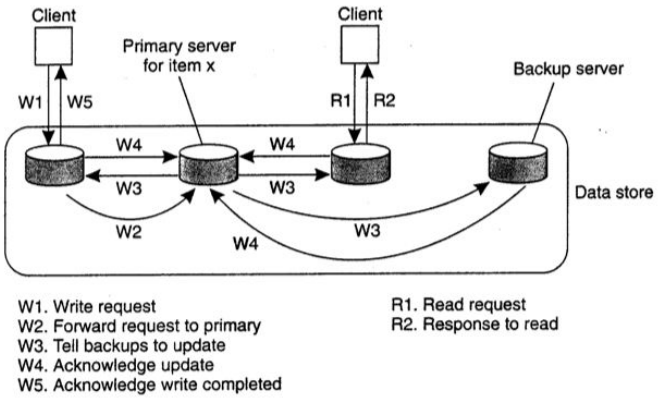
Server Protocols:

1)Sequential Consistency

We have used Remote Write Protocol for implementing Sequential consistency protocol. For managing database we have used local sqlite database. Multiple servers can be created within the same machine by specifying different port numbers. So different copies of sqlite databases will be created based on that port number.

As we are using remote write protocol, our leader server will act as a sequencer. All the writes are being done at the leader and leader updates all other servers. Client can connect to any server and perform read or write by calling appropriate function as specified by above API. During write, the server which gets the write request calls the leader server using RMI synchronously. Leader will get the request from any other server and inserts the article into the sqlite database and generates a new id. Then it calls all other servers with the new id for insert using multiple threads and once all the servers are done with writes, RMI is completed and control is returned to the initial server where write request was initiated and finally control goes back to the client. For read, the server uses own sqlite database and completes the read request. With this protocol, write is slower, but it ensures sequential consistency. For concurrency issues we have used sqlite feature of transaction and hence database will never go to any inconsistent state. However, in our implementation we have assumed that none of the servers will go down completely and have not implemented any recovery protocols (i.e. not detecting if a server is down by using heartbeat messages).

If a server goes down and rejoins, it will however first check with other available servers for missed articles and does a synch.



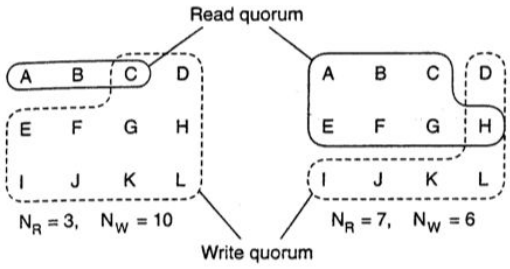
[Image from text book. In our code, we aren't sending acknowledgement update(W4)]

Quorum Consistency

For quorum protocol, we require N_w quorum nodes for write and N_r quorum nodes for read. If there are total N nodes then the following two conditions should be satisfied while choosing N_w and N_r :

$$N_w > N/2 \quad \& \quad N_w + N_r > N$$

First condition will ensure that there are no write conflicts and the second condition will ensure that there are no read-write conflicts. Since $N_w > N/2$, any write quorum selected will have at least one up-to-date quorum member. $N_w + N_r > N$ will ensure that at least one up-to-date replica will be part of any read quorum chosen.



[Image from text book]

For reading the article(s), client contacts one of the server and that server then creates a read quorum of size(N_r) by locking N_r number of server nodes as well as finds the maximum article ID available at each of those locked server so that it can find out the up-to-date server(server with max article ID as in our case we don't update any created articles). The up-to-date server returns article(s) to the server to which client is connected which in turn forwards the returned article(s) to the client. Once the up-to-date server returns the article(s), lock on each of the read quorum member is released.

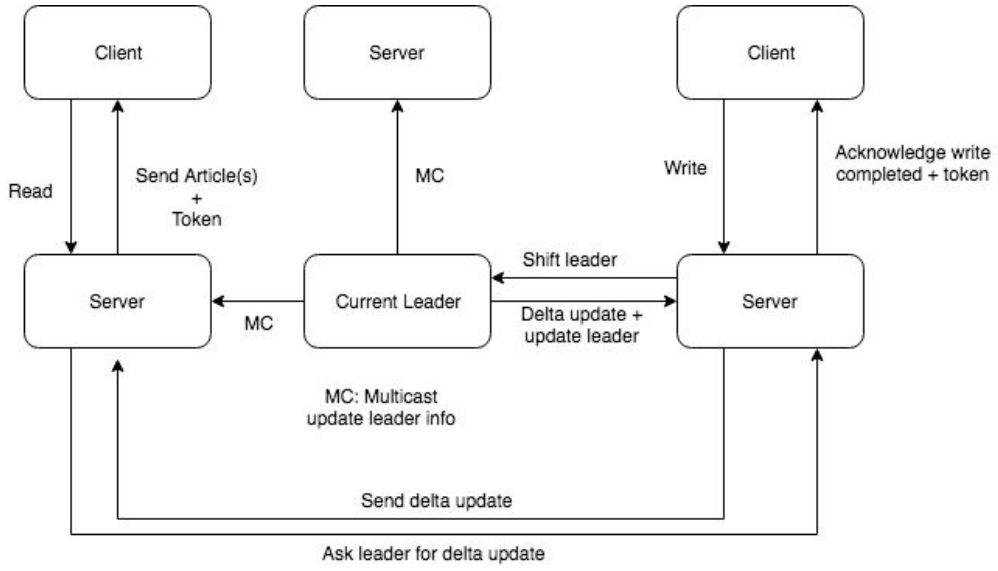
For writing article, client contacts one of the server and that server then creates a write quorum of size(N_w) by locking N_w number of server nodes as well as finds the maximum article ID available at each of those locked server so that it can find out the the up-to-date server(server with max article ID as in our

case we don't update any article). The up-to-date server inserts the new article into its database and then updates all the other members in the write quorum. Once it is done with these updates, lock on all the quorum members are released.

Read Your Write Consistency

We have used local write protocol to implement read your write consistency. In read your write, server needs to know what the client has already seen and we are reusing the article ids as a token for client. In client side we are maintaining the maximum article id the client has seen after any read or write operation. In each request the client has to pass that maximum id as a token to the server and server uses that token to decide how to serve information to the client.

On each read request, server compares the maximum id in its database with the client's token to decide if it has to sync with current leader before it can serve the client. For example, if client has posted an article with 28 as article id at server 6 and did a read request at server 10. Server 10 has to check if it has at least article id 28 in its local database. If not, server 10 would do a sync request at current leader and get the delta update and then complete the client's request. For a write request at any server, we need to make the current server as the leader. Every server knows about the current leader server and during a write request it makes a request to the current leader for a leader change using a RMI function. The current leader will broadcast the message saying we have a change in leader, send delta updates to the server who made the call, update new leader information in its system and then returns the call. After leader change, the server where write request was initiated would be able to write into its database and then completes the client's request. Meanwhile, due to the multicast message every other servers will be aware of the new leader. Now during this process, we may face concurrency issue and we are solving concurrency issue by locking the existing leader during leader change phase. Whole protocols runs with an assumption of a reliable multicast.



Running server and client:

First we have to run the leader(with the IP address and port number mentioned in server_config.properties). If the leader has to be changed, update the config file and restart all the servers and clients. Always the leader server(server mentioned in the config file) has to be running(irrespective of the protocol) as it maintains the list of servers running. For each protocol, this server would be called from other servers to get the list of joined servers. Whenever the protocol is changed in the config file, please restart all the servers and client

Tar/zip file contains data folder in which database files will be created. If it doesn't exist, folder named "data" has to be created in the same folder where the jar files and config file are placed before starting server for the first time. If dbpath(./data/article) mentioned in the config file is changed then name the folder accordingly

To start the primary server(say on port no (ex 5005) and IP address mentioned in the config file). Please update the IP address of primary server in the config file to the IP address of the machine on which the primary will be run:

java -Djava.net.preferIPv4Stack=true -jar Server.jar 5005

To start next server on port 5006

java -Djava.net.preferIPv4Stack=true -jar Server.jar 5006

Same commands can be used to start servers on another machine(Please create the "data" folder on another machine also when run for the first time). Code takes care of creating/updating db file if necessary.

To start client:

java -jar Client.jar

Options available for the client is explained in the test cases section

Assumption:

Servers and clients are fault tolerant.

Test cases:

Following options are available for a client :

"**Choose Server**" option sets the server to which client will make requests.

"**Read Article**": option is used to read a particular article. It will show the content and replies of the selected article. Replies to an article will be showed only when we select an article(Similar to moodle. We can see replies to an article only once you select the article)

"**Post an Article**" option will allow the user to post an article

"**Display Articles**" option will display all the articles(excluding the replies)

"**Reply to Article**" option will allow users to reply to an article. This option should be selected only after choosing "Read Article" option as we post replies to an article after opening that article(similar to moodle).

We are printing the whole stack trace some places for debugging process and during some negative test cases. So a stack trace in console may be due to an exception during a negative case.

Protocol: Sequential

- Run leader server on the port number(5005 in our case) and ip address mentioned in the config file. **Please update the config file with the ip address and port number on which the leader server will be run.**
- Run another 2 more servers on the same machine(with different port numbers) and another 3 servers on another machine.
- Create client on the same machine as the leader server and do the following operations:
    - ➔ Select option 1 : "Choose Server". It will display the list of available servers.
    - ➔ Choose one of the servers.
    - ➔ Choose "Read Articles" option. Server will show "No more Articles to Display" as there aren't any articles.
    - ➔ Choose "Post an Article". Server asks for article content. Enter "Article 1"
    - ➔ Choose "Post an Article". Server asks for article content. Enter "Article 2"
    - ➔ Choose "Read Articles" option. It displays both the articles created
    - ➔ Choose "Read Article option". It will ask for the article ID of the article to be displayed. Enter 1(Article ID is displayed besides each post). It will display the contents of the article("Article 1") as well as the replies associated with the article. As the article doesn't have any replies yet, it will only show the content of the article
    - ➔ Choose "Reply to Article". **This option should be selected only after choosing "Read Article" option as user should reply to an article only after opening the article in the same way as moodle.** It will ask the article number to which you want to reply. Enter the ID of the article if you want to create reply to the article. If you want to reply to a reply,

then enter the ID of the reply to which you want to reply. Since we haven't created any reply yet, we will reply to the article first. After that choose the option "Reply to Article" and give the ID of the reply just created and then enter the message you want to give as reply to the reply. Again choose "Read an Article" option and enter the ID of the main article(in this case 1). It will display all the replies to the article with proper indentation

- ➔ Create another 7 more articles and then choose "Display Articles" option so that we can see pagination.
- ➔ Each write operation makes update to all the replicas. Drawback of this protocol is that it updates all the replicas for each write which increases the time required for each successful write. It can be verified by opening the db file(in data folder) associated with each server using sqlite browser(https://sqliteonline.com/#/)
- ➔ Try creating more articles, reply to articles, multi level replies as well as multiple clients.

Negative case:
If the leader server goes down, then none of the write operations can take place. Read operation can still take place.

Protocol: Quorum
Stop all the servers and clients. Update "protocol" value in the config file(in the same folder as jar file) to "quorum"(don't put quotes in the config file). If you are using the class files instead of the jar file, then build the project so that the updated config file would be used while running. When new server is created it checks whether database exists for that server. If it exists, it then updates to the most updated version of replica available(leader server mentioned in the config file should be up as it would be called by other servers to get the list of joined servers). If it doesn't exist, it creates db file which contains all the rows of the most updated replica.

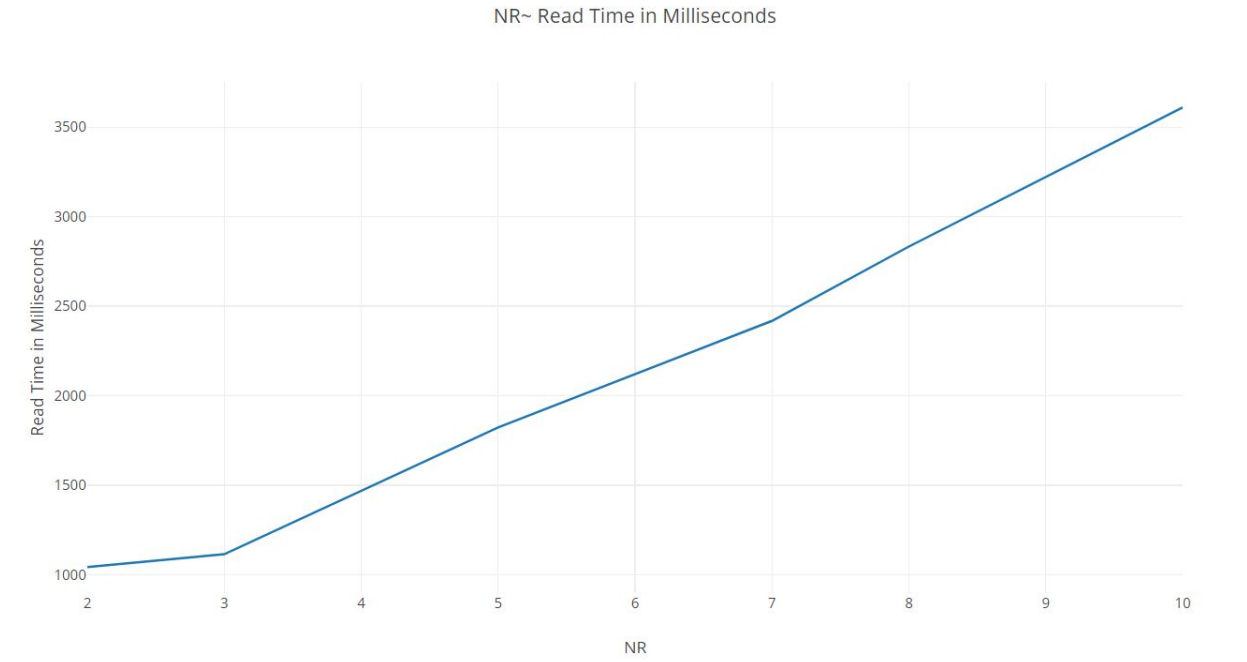
- Run leader server on the port number(5005 in our case) and ip address mentioned in the config file. **Please update the config file with the ip address and port number on which the leader server will be run.**
- Run another 2 more servers on the same machine(with different port numbers) and another 3 servers on another machine.
- **N_r is the required read quorum size and N_w is the required write quorum size. N_r(quorumreadmembercount) and N_w(quorumwritemembercount) is specified in the config file**
- Create client on the machine that doesn't host the leader server and do the following operations:
  - ➔ Select option 1 : "Choose Server". It will display the list of available servers.
  - ➔ Choose one of the servers.
  - ➔ Choose "Read Articles" option. Server shows all the articles that were created during the testing of previous protocol.
  - ➔ Choose "Post an Article". Server asks for article content. Enter "Article x"
  - ➔ Choose "Post an Article". Server asks for article content. Enter "Article y"
  - ➔ Choose "Read Articles" option. It will display all the articles created until now with pagination

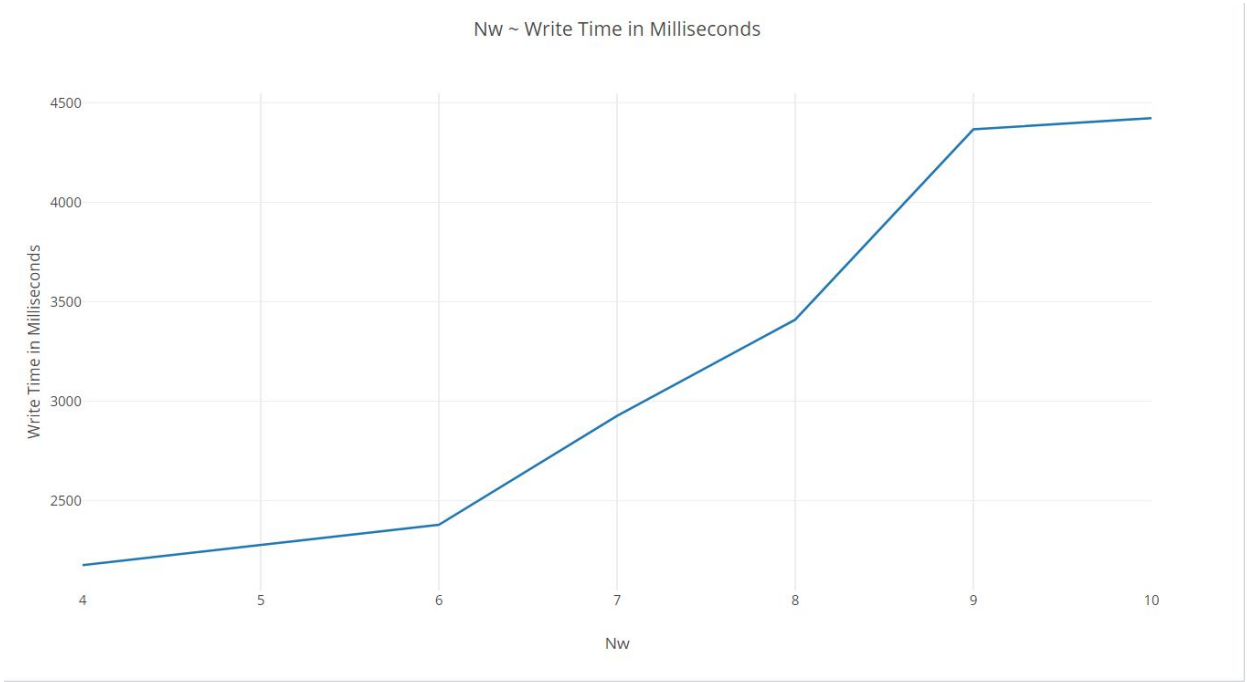
- ➔ Choose “Read Article option”. It will ask for the article ID of the article to be displayed. Enter 1(Article ID is displayed besides each post). It will display the contents of the article(“Article 1”) as well as the replies associated with the article.
- ➔ Choose “Reply to Article”. **This option should be selected only after choosing “Read Article” option as user should reply to an article only after opening the article in the same way as moodle.** It will ask the article number to which you want to reply. Enter the ID of the article if you want to create reply to the article. If you want to reply to a reply, then enter the ID of the reply to which you want to reply and following that enter the reply. Choose “Read an Article” option and enter the ID of the article which has to be read. It will display all the replies to the article with proper indentation
- ➔ Each write operation makes update to N_w quorum nodes. After each update, all the quorum nodes used for write will have the latest version of the bulletin board. For each read, N_r nodes will be locked(in-order to prevent read-write conflict) and data would be returned by the most updated replica in the quorum. Since we are ensuring $N_w > N/2$ and $N_r + N_w > N$, at least one node in the read quorum as well as write quorum will have the most updated version of the database. Based on the requirement of the application(more frequent reads/more frequent writes), we can choose appropriate values of N_w and N_r in such a way that $N_w > N/2$ and $N_r + N_w > N$ are still satisfied. If we need very frequent reads then we can set N_r as low as possible and N_w as high as possible. But in that case write will take longer time. We can do similar adjustments in case of frequent writes
- ➔ Database file associated with each server can be viewed using sqlite browser(https://sqliteonline.com/#/)
- ➔ Try creating more articles, reply to articles, multi level replies as well as multiple clients.

We tried following possible values of N_r and N_w:

N	**N_w**	**N_r**	**Write time(in milliseconds)**	**Read time(in milliseconds)**
6	4	3	2176	1115
10	9	2	4425	1043
10	6	5	2379	1823
10	10	10	4368	3610
10	8	7	3411	2418
10	7	8	2927	2833

Graph:





Conclusion : When N_w/N_r increases, it takes more time for write/read.

Negative case:

For N=10, N_r =2 and N_w = 9, if we try to write from clients concurrently, then none of the clients will be able to lock N_w nodes at the same time. As a result both the clients won't be getting response as it won't be able get lock on required number of quorum nodes.

For N = 6, N_w = 4, N_r =3, if three of the joined servers shuts down, then clients won't be able to do any write as it won't be able to acquire lock on N_w quorum nodes.

Protocol: Read Your Write

Stop all the servers and clients. Update "protocol" value in the config file(in the same folder as jar file) to "readYourWrite"(don't put quotes in the config file). If you are using the class files instead of the jar file, then build the project so that the updated config file would be used while running. When new server is created it checks whether database exists for that server. If it exists, it then updates to the most updated version of replica available(leader server mentioned in the config file should be up as it would be called by other servers to get the list of joined servers). If it doesn't exist, it creates db file which contains all the rows of the most updated replica.

- Run leader server on the port number(5005 in our case) and ip address mentioned in the config file. **Please update the config file with the ip address and port number on which the leader server(server which will maintain the list of connected servers) will be run.**
- Run another 2 more servers on the same machine(with different port numbers) and another 3 servers on another machine.
- Create client on the machine that doesn't host the leader server and do the following operations:
    - ➔ Selected option 1 : "Choose Server". It displays the list of available servers.
    - ➔ Choose one of the servers.
    - ➔ Choose "Read Articles" option. Server shows all the articles that were created during the testing of previous protocol.
    - ➔ Choose "Post an Article". Server asks for article content. Enter "Article a"
    - ➔ Choose "Post an Article". Server asks for article content. Enter "Article b"
    - ➔ Choose "Read Articles" option. It displays all the articles created until now with pagination(if needed)
    - ➔ Choose "Read Article option". It will ask for the article ID of the article to be displayed. Enter 1(Article ID is displayed besides each post). It will display the contents of the article("Article 1") as well as the replies associated with the article.
    - ➔ Choose "Reply to Article". **This option should be selected only after choosing "Read Article" option as user should reply to an article only after opening the article in the same way as moodle.** It will ask the article number to which you want to reply. Enter the ID of the article if you want to create reply to the article. If you want to reply to a reply, then enter the ID of the reply to which you want to reply and following that enter the reply. Choose "Read an Article" option and enter the ID of the article which has to be read. It will display all the replies to the article with proper indentation
    - ➔ Each write operation will shift the primary(Can be different from the leader server. This server has the up-to-date version of database) to the server to which the client is

connected. Server has to become primary before it can execute write operation. Other servers will be notified using a multicast. Once multicast reaches other servers, they will update the address of the primary they have. Only current primary server will have all the updates until the sync process is run.

- ➔ Database file associated with each server can be viewed using sqlite browser(https://sqliteonline.com/#/)
- ➔ Try creating more articles, reply to articles, multi level replies as well as multiple clients.

Negative case:

If the current primary server shuts down, then it won't be possible to make any writes as the primary server can't be shifted.

Sync Background Process

Synchronizer is a different process which runs in the background and tries to sync all the available servers. For syncing, this process asks the leader for all available servers and then asks each server for maximum articleid in their respective database. After getting these ids, synchronizer choses one with maximum id and asks that server to update others by doing a RMI call. The server with most updated data then updates other server by sending the delta that is needed to bring these server in a consistent state. Sync process uses *syncsleeptimeinseconds* flag from the config file as sync period and should be updated as needed.

To run Synchronizer:

```
java  -jar Synchronizer.jar
```