CSCI 5105 Project 2: Bulletin Board Consistency

Group members:
Shengping Wu wu000376
Fushan Wang wang9550

A. Instructions explaining

Client:

1) Compile: go into Client/ director and run `make all`
2) Run `./Client.out [host]` ,where host is the registry of the coordinator's address (This will allow client start with contacting the coordinator server and get the full server list back for further interaction)
3) User can type the command as prompted: `read` will return the article content from a random selected server; `post` will post the content user that would like to upload, followed the format `[Author;Title;Content]`; `reply` will ask user for the replied article's id, and accept the replied content; `choose` will ask user for the prompted article's id and then display the content; `test` will enable client to send test cases automatically with current opened servers.

Server:

1) Compile: go into Server/ director and run `make all`
2) Run `./GroupServer.out`, first to open a server as the coordinator. Then input 1,2,3 for policy chosen: 1 for sequential consistency, 2 for quorum consistency,3 for Read-your-Write consistency.
3) After that you can run `./GroupServer.out [host]` ,where host is the coordinator's address you just opened. This will allow all other servers to get IP of all registered servers. Then input the same number to choose the policy as the coordinator.

B. Design details

1. Article format: `[Version;Title;Author;Time;content]`
2. Communication format:
   We use UDP for communication between the clients and servers by sending a string as the request.
   Post: `[POST;Client IP;Client port;Version;Title;author;Time;content]`
   Reply: `[REPLY;Client IP;Client port;Article ID;Version;Title;author;Time;content]`
   Read: `[READ;Client IP;Client port]`
3. One the client side:

I. *Read* a list of articles and display the list one item per line.We assign each article a `hierarchy` for the print formatting. The indentation decided by the value of hierarchy.

II. *Choose* one of the articles and display its contents. We should do this after a read. We just store all the articles by *read* operation and choose that article from client memory to show the detail.

4. On the server side:

We add an abstract class `Server` in *BB_system.h* and add three derived classes from it in *Server.h* for three different policies:

I. Sequential consistency(class `seq_server`).

We use the primary-backup protocol to achieve this consistency. The primary-backup protocol works as shown in Figure 1.

W1: A process wants to perform a write operation on data item x(here is the whole bulletin board).

W2: Local server forwards that operation to the primary server for x.

W3: The primary store the new request to the back of the request queue(`requestQ`) and pop out the head of the queue to perform that request. It performs the update on its local copy of x, and subsequently forwards the update to the backup servers.

W4: Each backup server performs the update as well, and sends an acknowledgment to the primary.

W5: When all backups have updated their local copy, the primary sends an acknowledgment to the initial process, which, in turn, informs the client.

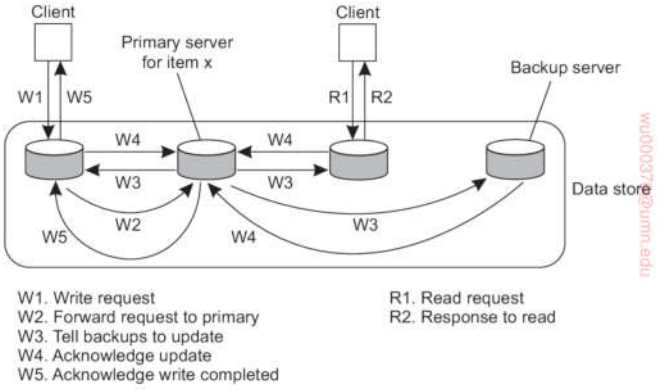


Fig 1. The principle of a primary-backup protocol

II. Read-your-Write consistency(class `LocalWrite_server`)

We implement Read-your-Write consistency by local-write protocol. The local-write protocol works as shown in Figure 2.

W1: A process wants to perform a write operation on data item x(here is the whole bulletin board).

W2: Local server locates the primary copy of x, and subsequently moves it to its own location.

W3:The new primary (just the local server for this process) performs the update on its local copy of x, and then sends an acknowledgment to the client.
W4: The new primary subsequently forwards the update to the backup servers.
W5: Each backup server performs the update as well, and sends an acknowledgment to the primary.

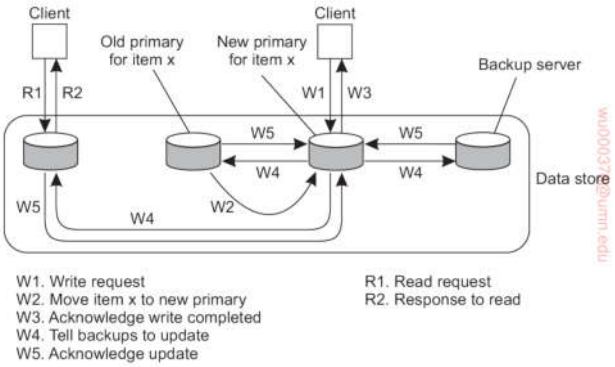


Fig 2. Primary-backup protocol in which the primary migrates to the process wanting to perform an update(Local-write protocols)

III. Quorum consistency(class `qum_server`)

Here we implement the quorum-based protocols. We use the voting algorithm to achieve this consistency as shown in figure 3.

1) To read a replicated article, the client contacts the local server, which in turn, contacts the coordinator to do the operation. The coordinator contacts the other N_R randomly chosen servers needed for the quorum and ask them to send the version numbers associated with the item(`qum_server:VER`). Then the coordinator chooses the largest number of versions(which means the newest) and sends that to the client.
2) To post a new article, the client contacts the local server, which in turn, contacts the coordinator to do the operation. The coordinator contacts the other N_w randomly chosen servers and ask them to update the bulletin board posting the new article.
3) To reply to an article(i.e ID=x), the whole system does the read operation first to find the newest version of that article(ID = x) and then does the post operation to post the reply as a new article.

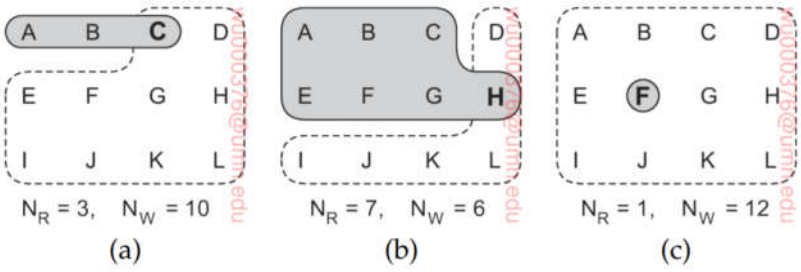


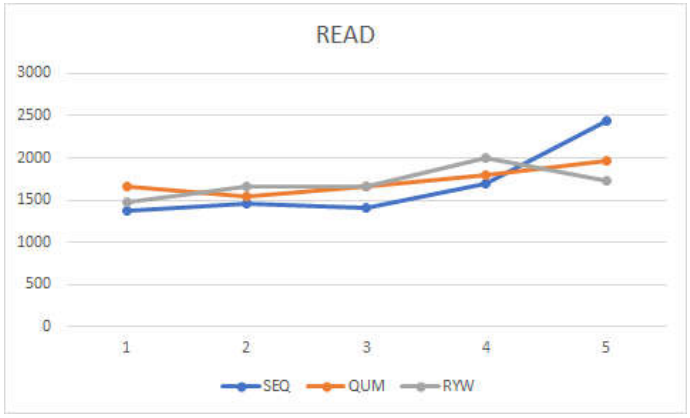
Fig 3. Three examples of the voting algorithm. The gray areas denote a read quorum; the white ones a write quorum. Servers in the intersection are denoted in boldface.

C. Test case

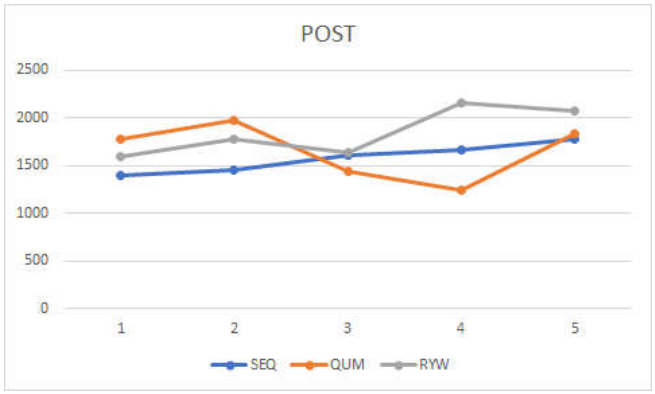
1. Test:

We write a function at the client side that records each response time from the client side for different types of server, and calculate their average response time. The server types and server number is varying during the test. The result will be stored in a file (test.txt) and we make an analysis graph based on this graph. Here are some graphs from the test result. For the following result, the vertical axis represents average response time (ms), and the horizontal line represents the number of available servers for sequential and read-your-write servers.

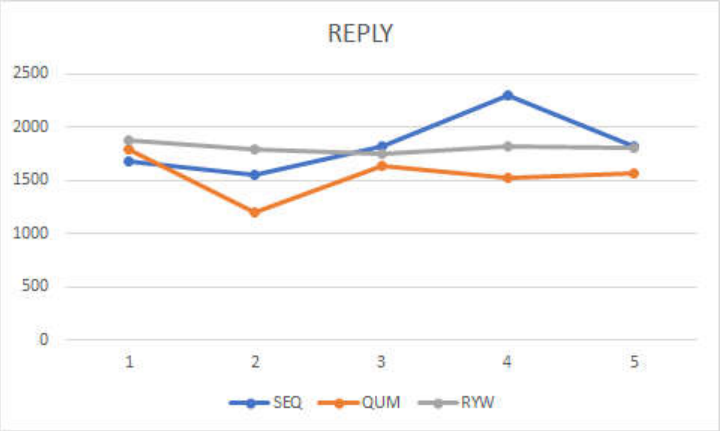
For READ operation, All the servers represent an increase of time with respect to number of available servers, and the sequential server has best performance in read operation. The Qr is increasing from 1 to 5 with Qw = 5.



The Quorum setting is (Qr, Qw)= {(3,3),(3,4),(2,4),(2,5),(1,5)} for POST and REPLY. For POST operation, local-write server and sequential server maintain an increasing tendency with respect to available server, while quorum server is not stable during the change of Qr and Qw. This might due to the auto-synch operation varying because the time of finding the newest server changes with different Qw.



For REPLY Operation, the graphs look different from the POST Operative graph, which might because REPLY operation needs to find the replied article first, which contributes to a different time consumption in total. The quorum has a very different line when compared with previous graph, which might because finding the replied being replied id gets easier when the Qw is increasing, and therefore the finding time becomes solid.



From all above, we could conclude that the sequential server might has the most stable performance in total; Quorum server might be proficient with some special Qr and Qw setting; Read-your-Write server has a similar operation time with Sequential Server, but the latte maintains a better performance.

2. Known limitations:
    1) When opening the servers, all servers should choose the same number for policy. If you input the different number, it will collapse.
    2) We hard code the port for UDP communication, which means each computer can only open one terminal for a certain server.

D. Source code
Github: https://github.com/Inutto/CSCi5105.git. In Project2/ file.
Also available in the .zip file submitted.