CSCI 5105 - Project 2 Bullet Board

Tiannan Zhou <zhou0745@umn.edu>, Xuan Bi <bixxx078@umn.edu>

Mar. 25, 2018

1. Project Description

There are three head files and eight source files implemented by us. They are *client_comm.h, comm.h, server_comm.h* and *boardClient.cpp, boardServer.cpp, client_comm.cpp, comm.cpp, server_comm.cpp, server_seq.cpp, server_quorum.cpp, server_rnw.cpp* and a makefile to compile and run the project program.

1.1 comm.h/comm.cpp

All standard libraries used are all included here. Here also have the definition for common methods both the server side and the client side will use, such as UDP packet sending method, common packets forming and parsing method, etc.

1.2 client_comm.h/client_comm.cpp

The head file and the source file defined all methods for client side, especially for the client process.

1.3 server_comm.h

This head file defined all server side methods.

1.4 server_comm.cpp

This source file implemented the common method for server side's usage, such as registering server to the coordinator server.

1.5 server_seq.cpp, server_quorum.cpp, server_rnw.cpp

This source file contained the three different server's implements. **server_seq.cpp** is for sequential consistency, **server_quorum.cpp** is for quorum consistency and **server_rnw.cpp** is for Read-your-Write consistency.

1.6 boardClient.cpp, boardServer.cpp

This is the source file contained the main body for the Server and the Client. They would create socket, instruct users to set-up the server/client and call the proper API built in the headfiles to start the functionality the user would like.

2. Functionality for Components

2.1 BoardServer

2.1.1 General Implementation

For all regular servers, they would get information about which policy will be used when they register themselves to the Coordinator Server. In our implementation, Coordinator Server has no normal functionality as a regular server but only the coordinate functionality (assigning unique id, forwarding packets, handling data and mutual exclusion lock sometimes). For all of three consistencies' implements, the workflow when a server received a new posted article or reply, it will cache the article/reply into to-be-assigned article queue and send a unique id request to the coordinator server. After the coordinator server sent the assigned unique ID back, the server will pop out the first element in the queue and continue to do the corresponding operations under the specific consistency policy. For a Read/View request, the server will queue it after received the request. When the server has the response for this request (from itself, an external server or coordinator server, which depends on the consistency policy), it will send the response to the client. We used incremental update method in our server's implementations for network resource saving purpose.

2.1.2 Sequential Consistency

For sequential consistency, the server who communicated with the client directly won't do any READ/WRITE operations by itself (except the current server is the primary back-up server). All READ/WRITE requests will be forwarded to the primary back-up server via the coordinator. The communicated servers won't cache anything and will only forward requests and responses to make sure sequential consistency – all the data will only came from the primary back-up server and all clients will see the same result regardless of which server they currently connect to.

2.1.3 Quorum Consistency

The coordinator will have arguments N_W, N_R which are calculated based on the total number of servers. When a server received any READ/WRITE request, it will never do it locally. It will send this READ/WRITE request to the coordinator server. For WRITE

request, the coordinator will send WRITE request with the new article/reply to N_w randomly picked servers and the N_W servers will store the update. For READ request, the coordinator will send READ request to N_R randomly picked servers and the N_R servers will response the READ request with its local storage. The coordinator will merge all responses from the N_R servers and send the result to the original requester server then let it deliver the result to the client.

2.1.4 Read-your-Write consistency

When received a new WRITE request, the server will try to become the primary server to do the back-up operations. If success, it will update its self's storage with the previous primary back-up server and hold the primary server's position to do all the WRITE requests.

2.2 BoardClient

2.2.1 Communicating with The Server

The client will communicate with the server by our self-designed packet system. And it will cache all the articles' first 50 characters' abstract it previously received for doing incremental update to save network resource.

2.2.2 Printing out READ Results

The client will print out the result for Read-All operation with indentation. Every reply will have ONE more space's indentation followed with the article/reply it replied to. We use Depth-First Search to implement this functionality.

3. How to Compile and Run the Project

3.1 Making

We wrote a makefile to simplify the compilation and execution processes. To successfully compile and run the code, simply run the following command in the directory where the *makefile* is located.

$ make

After making, you will see two expectative file named “BoardServer” and “BoardClient” in your directory

3.2 Run BoardServer

$ ./ BoardServer

to start the server. Remember you must enter the port number you want to set up the server. After set up the socket, you should follow the instruction to choose whether this is a coordinator server, consistency policy number if it’s a coordinator and the coordinator server’s network information when this is a regular server.

3.3 Run Client

$ ./BoardClient

to start the client. You will be asked for the server’s address and port number you want to connect (PLEASE NOTE: coordinator server has no functionality to handle client’s request). The client support four different operations: Post-New, Read-All, View-Full, Reply-To. You must Read-All first to cache the article’s abstract you would like to do further operations before you try to do View-Full or Reply-To (this is for safety issues)

4. Testing Cases and Results

Here are mainly four functionalities to test. We begin with testing *post* to verify if it works as expected. We first connected a primary back-up server and a ordinary server to an coordinator. Then set up a client to perform.

To ***Post-New***, the client UI has the following interaction,

```
Please choose the operation you want:
 1.Post an article
 2.Read the list for all articles & replies
 3.Choose an article/reply to view full content
 4.Reply to an article or a reply
Please enter your choice: 1
Please enter your new article's content (not longer than 4000 characters and end with Enter):
01234
Sent the post to the server <128.101.37.30:3333>

Please enter your new article's content (not longer than 4000 characters and end with Enter):
56789
Sent the post to the server <128.101.37.30:3333>

Please choose the operation you want:
 1.Post an article
 2.Read the list for all articles & replies
 3.Choose an article/reply to view full content
 4.Reply to an article or a reply
Please enter your choice: 2
Read request has been sent to the server <128.101.37.30:3333>
Received 2 new articles/replies, updating...
  1.01234
  2.56789
```

The functionality of ***Post-New*** well behaves.

Now to verify if ***Read-All*** is good in use. We mainly need to check the length of the article output from the server and also if a reply thread appends to the designated article. The following is the output from the client UI,

```
Read request has been sent to the server <128.101.37.30:3333>
Received 7 new articles/replies, updating...
  1.01234
   3.43210
  2.56789
  4.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   5.qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
    7.wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
   6.zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
```

It successfully shows that article 3 replies to article 1. articles 5 and 6 reply to article 4 while article 7 replies to article 5. article 4,5,6,7 have the same abstract length. This output also validates the performance of *reply* and indentation of reply thread is as good as expected.

It also validates the ordering of article IDs is increasing among all parent articles, i.e. even though article 7 shows before article 6, it replies to article 5 so it precedes article 6 anyways.

```
Read request has been sent to the server <128.101.37.30:3333>
Local cache is already updated, no need to update again
```

Read-All function successfully found there's no need for update.

Then we see if ***View-Full(Choose)*** function can catch negative cases while displaying the full content of articles.

```
For viewing new articles/replies not in cache, you must Read the list for all articles & replies first
Please enter the article/reply you want to view full content (Range from 1 to 6): 5
View request has been sent to the server <128.101.37.30:3333>
Article / Reply received.
Full content for Article / Reply No.5:
5.(Reply for Article/Reply No.4)
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
```

The full content of article 5 is displayed and is longer than shown in ***Read-All*** output above. Now to check if it warns client when no such article has been posted.

```
For viewing new articles/replies not in cache, you must Read the list for all articles & replies first
Please enter the article/reply you want to view full content (Range from 1 to 7): 9
View request has been sent to the server <128.101.37.30:3333>
Article / Reply received.
Full content for Article / Reply No.9:
Error: out of range input number
```

View-Full displays the full content of designated article as well as catch any invalid article ID. Similarly, ***Reply-To*** function catches invalid cases, too.

```
For replying new articles/replies not in cache, you must Read the list for all articles & replies first
Please enter the article/reply you want to reply (Range from 1 to 1): 5
invalid reply to number
```

5. Analysis about Quorum

When Quorum consistency policy is applied, its performance is significantly affected by the N_w and N_r's values. If we assume READ operations for a single server's cost is C_R and WRITE

operation for a single server's cost is C_W. We can see that the total cost system-wide for a READ request is $N_R * C_R$ and total cost for a WRITE request is $N_W * C_W$. Obviously, the optimized setting for N_W and N_R to minimize the total cost is that :

$$N_W = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

$$N_R = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

There can make sure N_W and N_R is good for the constraints and minimizing the total cost system wide.