CSCI 565: Distributed Computing Systems (Fall 2024)

Bulletin Board Consistency

1. Overview

In this project, you will implement a simple Bulletin Board system (BBS) (like the Ed Discussion forum) in which clients can post, reply, and read articles stored in the BBS. The BBS is maintained by a group of replicated servers that offer **sequential consistency, Quorum consistency, or Read-your-Write consistency**. You may reuse any of the code developed in your own previous programming assignments or you can start from scratch. You can use any communications (UDP or TCP)as you want. In this project, you will learn about how to implement various forms of consistency and their tradeoffs. The desired consistency mechanism is supplied as a parameter at runtime.

2. Project Details

The project will be comprised of: **clients** and **servers** (one of the servers is designed as the **primary or coordinator**). The client does not know or care who the coordinator is. All of the other servers are configured to know who the coordinator is. Clients can connect to any servers. Thus, the client knows the address of every server (you may deploy servers on the same machine with different ports if you like). To keep things simple, there is only one bulletin board with a single topic stream. The operations that the clients should be able to perform are below:

1. `Post` an article
2. `Read` a list of articles and display the list each item on one line (maybe just the first few words or the title)
3. `Choose` one of articles and display its contents
4. `Reply` to an existing article (also posts a new article)

Internally, the client should be able to connect or disconnect to any server (not just the coordinator) to carry out any operation. However, the article IDs are generated by the coordinator. So the contacted server (on a `post` or `reply`) will ask the coordinator for the next ID. The server will order the articles in increasing order using an ID (1, 2 …) that it generates and provides, and the client should print the articles in this order using indentation for replies, e.g.

```
1   Article 1
2   Article 2
        3   A reply to Article 2
        4   A reply to Article 2
                5 A reply to Article 4
```

Since there can be many articles in the bulletin board, you should consider how many articles should be shown in a page and how to explore these pages.

The article IDs are returned with the article and can be used both in formatting and in calls to `choose` or `reply`. You can decide on the structure and format of an article. There are multiple clients on this system and each client will perform operations concurrently. Your clients will have a user interface to manipulate the operations on the bulletin board (the look of the interface is up to you). Each article will have a unique ID defined by the server. You will need to decide how to represent "reply" articles so that the client may format things properly. To emulate the propagation delay one might see in a wide-area network, you can delay the sending of a message (from client to server or server to server) by sleeping a random amount of seconds.

a) Implement sequential consistency

This means that all clients should see the same order of articles on a `read` from **any** server even if they were posted by concurrent clients to **any** servers. You can use the primary-backup protocol. Measure the cost of client operations.

b) Implement quorum consistency

Given N replicas, you will need to assemble a read quorum (N_R) which is an **arbitrary** collection of servers in order to `read/choose`, and a write quorum (N_W), an **arbitrary** collection of servers in order to `post/reply` for the client. The values of N_R and N_W are subject to the following two constraints:

1. $N_R + N_W > N$

2. $N_W > N/2$

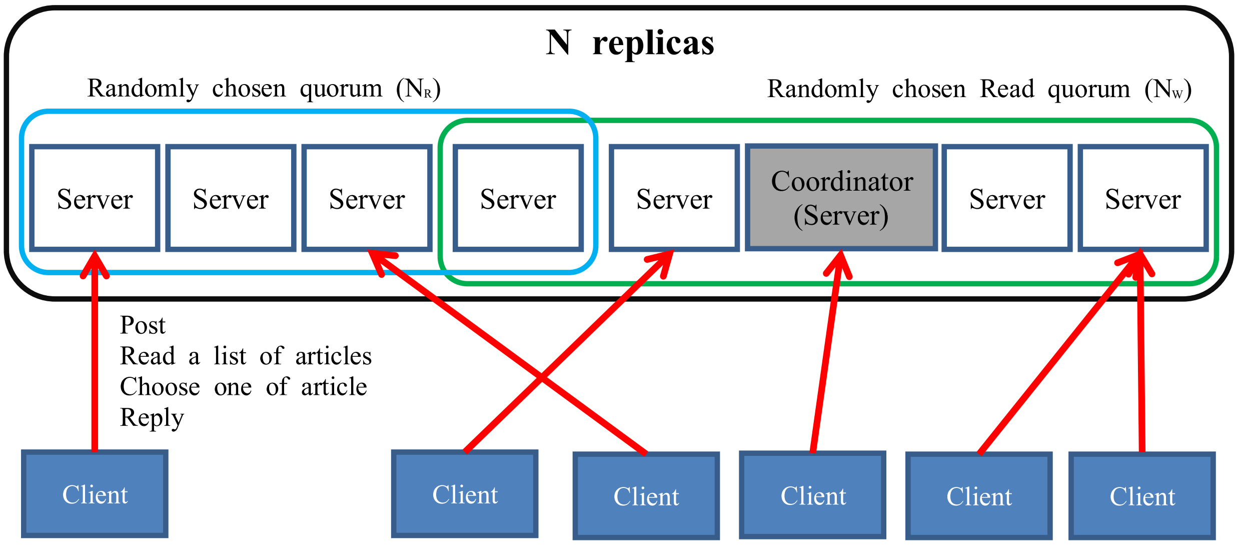
You may use the coordinator as a control point for your quorum. That is, the client contacts any server, which in turn, contacts the coordinator to do the operation by contacting the other **randomly chosen** servers needed for the quorum. Now, vary the values of N_R and N_W and measure the cost (as seen from the client) to do a write or read operation. Present data graphs and provide simple analysis. How does this compare with sequential consistency?

c) Implement Read-your-Write consistency

For this, suppose a client C `posts` an article or `reply` to a specific server S1. Later, if the client C connects to a different server S2 and does a `read` or `choose`, they are guaranteed to see the prior updates. You can use the local-write protocol.

d) Allow a server to fail and hold a leader election to determine a new server.

To realize your project goals, you have to define an API for server-server communication to propagate updates, request new article IDs, and so on. This is up to you to define.



3. Implementation Details

You may borrow code that you like from your own previous programming assignments. Do not use any code found online. To make multiple servers run easily, your servers may run in a single machine with different port numbers. Note that your servers are also expected to work well when deployed across different machines. In the quorum protocol, replicas can get out of synch. That is, a reader is always guaranteed to get the most recent article (i.e. the latest ID) from one of the replicas, but there is no guarantee that the history of updates will be preserved at all replicas. To fix this problem, implement a `synch` operation that brings all replicas up to date with each other and can be called periodically in the background.

4. Project Group

You may do this project individually or in a group of two members.

5. Test Cases

You should also develop your own test cases for all the components of the architecture, and provide documentation that explains how to run each of your test cases, including the expected results. Also tell us about any possible deadlocks or race conditions.

There are many failure modes relating to the content of messages, parameters, and system calls. Try to catch as many of these as possible.

6. Deliverables

a. Design document describing each component, including performance graphs and simple analysis. Not to exceed 3 pages.

b. Instructions explaining how to run each component and how to use the service as a whole, including command line syntax, configuration file particulars, and user input interface.

c. Testing description, including a list of cases attempted (which must include negative cases) and the results.
d. Source code, makefiles and/or a script to start the system, and executables.

Note: a, b, and c should be combined in a single document file.

7. Grading

The grade for this assignment will include the following components:

a. 40% - The document you submit – This includes a detailed description of the system design and operation along with the test cases used for the system (must include negative cases)
b. 50% - The functionality and correctness of your own server and clients
c. 10% - The quality of the source code, in terms of style and in line documentation

8. Resources

a. D. K. GIFFORD, *Weighted voting for replicated data*, in Proc. 7th Annual ACM Symp. Oper. Sys. Principles (SIGOPS), ACM, New York, 1979.

b. S. B. DAVIDSON, H. GARCIA-MOLINA, AND D. SKEEN, *Consistency in partitioned networks*, ACM Computing Surveys, 17 (1985).