CSCI 5105: Introduction to Distributed Computing
Spring 2018
Instructor: Jon Weissman
Project 2: Bulletin Board Consistency
Due: Mar 23 12pm (noon)

1. Overview

In this project, you will implement a simple Bulletin Board (BB) system (like our Moodle forum) in which clients can post, reply, and read articles stored in the BB. The BB 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 Project 1 or you can start from scratch. However, unlike the PubSub system, your server(s) will store and remember all articles and not communicate with other student's servers. You can use any communication protocol/system (e.g. UDP, TCP, and RPC) as you want. In this lab, 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 server(s). Thus, the client knows the address of every server (you may deploy servers on the same machine with different ports if you like or within virtual machines). To keep things simple, there is only one BB 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 one item per line (maybe just the first few words or the title of the article)
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. Each article has an internal unique ID that is generated by the coordinator. So the contacted server (on a `post` or `reply`) will ask the coordinator for the next unique ID to use. The server will order the articles in increasing order using the ID (1, 2 …) as this captures time order. For the `Read` comment, 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 the number of articles in the BB can be large, you should consider how many articles should be shown in a page and how to enable subsequent pages to be shown.

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 in this system and each client will perform operations concurrently. Your clients will have a simple UI to manipulate the operations on the BB (the look of the interface is up to you). 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 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.

b) Implement *quorum consistency*

Given N replicas, you will 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 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.

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. Measure the cost of client operations and compare.

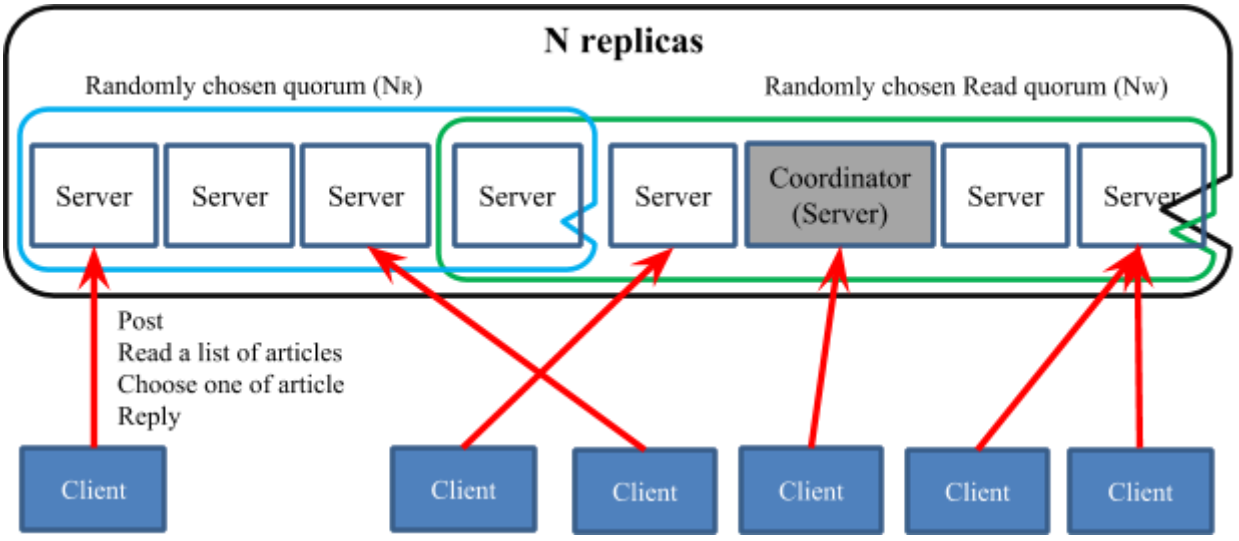
For all consistency policies, measure the cost of client operations, and compare across the policies.

Done Early? Try this for no extra credit:

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

e) Pick another consistency policy to implement.

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 Project 1. **Do not use** any code found on-line. 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 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

All students should work in groups of size 2. There are some students who have done the Project 1 without a partner. So if you were doing the project alone but want to find a partner, we encourage you to use the forum to find your partner.

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. **Finally, you should run your clients and servers within the CSE firewall – as outside access particularly through UDP and TCP may be blocked.**

6. Deliverables

a. Design document briefly describing each component. 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 in a single document file.

7. Grading

The grade for this assignment will include the following components:

a. 20% - 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. 70% - 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).