CSCI 5105: Introduction to Distributed Systems

Spring 2016

Instructor: Abhishek Chandra

Programming Assignment 2: Simple Distributed File System

(*Due: Mar/30/2016 – 11:59pm)*

1. Overview

In this programming assignment, you will implement a *simple distributed file system* in which multiple clients can share files together. You may reuse any of the code developed in PA1 or you can start from scratch. In this file system, the files will be replicated to several servers for increased performance and availability.

2. Project Details

In this project, multiple clients will send requests at the same time to the file system. The operations that the client should be able to perform are:

a. Write (update) files in the file system.
b. Read files. If there is no corresponding filename on file system, the client will see appropriate error message.

The files are replicated across multiple file servers. For consistency across the replicated files, the ***quorum based protocol*** should be implemented. The basic idea of this protocol is that the clients need to obtain the permission from multiple servers before either reading or writing a file to the server.

2.1. Quorum based protocol

A quorum is a subgroup of replicas whose size gives it a right to perform operations. In a quorum consensus scheme, update operation may be performed by only a subset of replicas, while other replicas can have out-of-date copies. Version numbers are used to determine which copies are up-to-date and operations are applied to only copies with the current version number. Each replica is assigned an ability to vote which is used during the formation of quorums. In this protocol, in order for the client to be able to read a file which has N replicas, it needs to assemble a read quorum (N_R) which is an **arbitrary** collection of **any** replicated file servers. Similarly, to write (update) the file, the

client needs to assemble a write quorum (N_W), which is also an **arbitrary** collection of **any** replicated file servers. The values of N_R and N_W are subject to following two constraints:

1. $N_R + N_W > N$
2. $N_W > N/2$

The components of the system to be built are:

1. **Clients**: There are multiple clients performing read and write operations on the shared files. The client can **contact any server**. The system should consist of at least 3 clients.

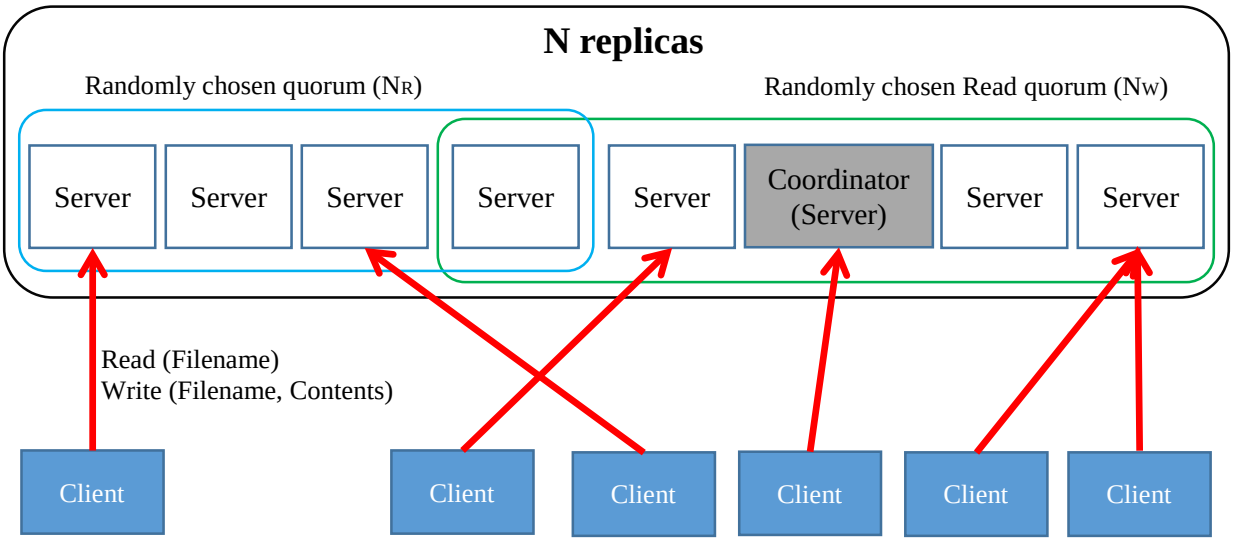
2. **File Servers**: There are servers containing replicas of the files. The system should consist of at least 7 servers containing the file replicas.

3. **Coordinator**: One of file server will act as a control point for your quorum. That is, a server which gets a request from the client contacts the coordinator to do the operation contacting the other **randomly chosen servers** needed for the quorum. Coordinator will be well known to all file server and will be chosen by you.

With this overview, we now look at the steps involved for performing the system operations:

1. Read:
    a. A read quorum should be assembled by the Coordinator and version number enquiries should be made.
    b. The most recent version number should be selected from N_R. As each read quorum overlaps with every write quorum, every read quorum is certain to contain at least one current copy.
    c. The read operation may be applied to **any** up-to-date copy.
2. Write:
    a. A write quorum should be assembled by the Coordinator and version number enquiries should be made.
    b. If there are insufficient up-to-date copies within the quorum, the write operation should be applied to the ones with latest version numbers and the outdated copies should be replaced with this latest copy.
    c. After performing the write operation the version numbers should be incremented and the completion should be notified back to the requesting client. The remaining replicas (outside of the quorum) can then be updated in the background.

3. Concurrency Control:
    a. The Coordinator should be responsible for implementing the concurrency control.
    b. It should implement a requests queue and queue up the requests coming from the servers. By doing this, all the write operations are made sequentially consistent by the Coordinator.



2.2 Performance Evaluation

The performance of the system should be evaluated for different combinations of the N_R and N_W and the results **should be plotted**. That is, measure the cost of client "Write" and "Read" operations in terms of time. The system should also be evaluated for varying work-loads. For e.g.: all the clients in the system are writing (write-heavy workload) or reading (read-heavy workload) at the same time. You should try out various values and observe which combinations of N_R and N_W perform well under different work-loads and why.

3. Implementation Details

Your system may be implemented in C++ or Java using Thrift (or TCP).

You may borrow code from PA1. 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 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 version) 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. This operation will be done eventually (with eventual consistency). You may want to print some log messages to see when synch works in background.

For testing (grading) purpose, you should provide some UI to list the files (with version) on the file system.

Assumptions and Hints:

- The number of files you need to handle will be small (< 10).
- Servers know other servers' and coordinator's information (IP and port).
- The files contain will be very simple (e.g., file name with version number).
- **Please read this document more carefully before you ask.**

4. Project Group

All students should work in groups of size no more than 2 members.

5. Testcases

Basic test cases include testing the system for **read-heavy** and **write-heavy** workloads. You must also develop your own test cases, and provide documentation that explains how to run each of your test cases, including the expected results.

6. Deliverables

- Design document describing each component.
- User document 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
- Testing description, including a list of cases attempted (which must include negative cases) and the results.
- Source code, Makefiles and/or a script to start the system. (Not any object (class) file).
- **Only one submission** for each group. (Don't forget put all names in a group)

7. Grading

The grade for this assignment will include the following components:

- 40% - The document you submit
    - Detailed description of the system design and operation -10%
    - Description of test cases and the performance evaluation result for the system – 30%
- 50% - The functionality and correctness of your program
    - Read, Write operations
- 10% - The quality of the source code, in terms of style and in line documentation

You will lose points if there is any exception, crash or freezing on your programs.

8. References

- D. K. GIFFORD, Weighted voting for replicated data, in Proc. 7th Annual ACM Symp. Oper. Sys. Principles (SIGOPS), ACM, New York, 1979.
- S. B. DAVIDSON, H. GARCIA-MOLINA, AND D. SKEEN, Consistency in partitioned networks, ACM Computing Surveys, 17 (1985).