Term Project: A Replicated Distributed File System

Objective

- Get hands on experience in implementing distributed file systems.
- Better understand transactions, fault tolerance, concurrency control and consistency.

Overview

It is required to implement a replicated file system. There will be one main server (master) and, data will be replicated on multiple replicaServers. This file system allows its concurrent users to perform transactions, while guaranteeing ACID properties. This means that the following need to be ensured:

- The master server maintains metadata about the replicas and their locations.
- The user can submit multiple operations that are performed atomically on the shared files stored in the distributed file system.
- Files are not partitioned.
- Assumption: each transaction access only one file.
- Each file stored on the distributed file system has a primary replica. This means that you will need to implement sequential consistency through a protocol similar to the passive (primary-backup) replication protocol.
- After performing a set of operations, it is guaranteed that the file system will remain in consistent state.
- A lock manager is maintained at each replicaServer.
- Once any transaction is committed, its mutations to files stored in the file system are required to be durable.

Characteristics of the Distributed File System

1. **Reads from the file system.** Files are read entirely. When a client sends a file name to the primary replicaServer, the entire file is returned to the client. The client initially contacts the master node to get the IP of the replicaServer acting as primary for the file that it wants to read.

2. **Writes to files stored on the file system.** This procedure is followed:
   (a) The client requests a new transaction ID from the server. The request includes the name of the file to be mutated during the transaction.
   (b) The server generates a unique transaction ID and returns it, a timestamp, and the location of the primary replica of that file to the client in an acknowledgment to the clients file update request.
   (c) If the file specified by the client does not exist, the master creates metadata for the file and chooses where its replicas can be located and which one of them will be the primary. The client will then communicate these information to the primary along with the write request. Selecting the replicaServers to place the replicas of the file and selecting of the primary replica can be random.
   (d) All subsequent client messages corresponding to a transaction will be directed to the replicaServer with the primary replica contain the ID of that transaction.
   (e) The client sends to the replicaServer a series of write requests to the file specified in the transaction. Each request has a unique serial number. The server appends all writes sent by the client to the file. Updates are also propagated in the same order to other replicaServers.
   (f) The replicaServer with the primary replica must keep track of all messages received from the client as part of each transaction. The server must also apply file mutations based on the correct order of the transactions.
   (g) At the end of the transaction, the client issues a commit request. This request guarantees that the file is written on all the replicaServer disks. Therefore, each replicaServer flushes the file data to disk and sends an acknowledgement of the committed transaction to the primary replicaServer for that file. Once the primary replicaServer receives acknowledgements from all replicas, it sends an acknowledgement to the client.
   (h) The new file must not be seen on the file system until the transaction commits. That is a read request to a file that is being updated by an uncommitted transaction must generate an error.
3. **Client specification.**
   (a) Clients read and write data to the distributed file system.
   (b) Each client has a file in its local directory that specifies the main server IP address.
4. **Master Server specification.**
   (a) The master server should communicate with clients through the given RMI interface.
   (b) The master server need to be invoked using the following command.
   server -ip [ip_address_string] -port [port_number] -dir <directory path>
   where:

- **ip_address_string** is the ip address of the server. The default value is 127.0.0.1.
- **port_number** is the port number at which the server will be listening to messages. The default is 8080.

(c) Note: you can start the replicaServers similar to the master server, or the master server can start them while it is starting

5. **ReplicaServer specification.**

   (a) Each file is replicated on three replicaServers. The IP addresses of all replicaServers are specified in a file called repServers.txt. The master keeps heartbeats with these servers.

   (b) Acknowledgements are sent to the client when data is written to all the replicaServers that contain replicas of a certain file.

   (c) You will need to implement a similar protocol to the **passive (primary-backup) replication protocol** studied in class.

   (d) When the client asks the primary replicaServer to commit a transaction, the primary replica must ensure that the transaction data is flushed to disk on the all the replicaServers local file systems before committing the transaction. Using the write is not enough to accomplish this because it puts the data into file system buffer cache, but does not force it to disk. Therefore, you must explicitly flush the data to disk when committing client transactions or when writing important housekeeping messages that must survive crashes. To flush the data to disk, use flush() and close() methods of the FileWriter in Java.

6. **Concurrency.** Multiple clients may be executing transactions against the same file. Isolation and concurrency control properties should be guaranteed by the server.

Testing

You will need to follow the above specifications to allow us to test your file system using our client. Below some test scenarios that may guide you to evaluate the correctness of your systems.

- Basic client read and write operations to the file system. This includes: (1) the file updated by the client contains the correct written data after committing the transaction, (2) partial updates (before transaction commits) are not seen by other clients or other transactions by the same client, and (3) a new file created by a transaction is not visible in the file system until the transaction commits.

- Test multiple concurrent clients accessing the same file and check if the mutations to the file are done as if the clients transactions were performed sequentially.

Bonus

1. **Handling multiple reads and writes per transaction**. Clients can send read and write requests for the same file in one transaction.

2. **Handling Transaction Failure**. For example:

   - A client can decide to abort the transaction after it has started.
   - A client crashes before committing the transaction.
   - Some messages that are a part of a transaction are lost.

Deliverables

- Complete source code, commented thoroughly and clearly.
- A report that describes the following: (1) how your code is organized, (2) its main functions, (3) how to compile and run your code, (4) details of the components of your file system, (5) any assumptions that you have made, and (6) the role of each of the team members.

Grading Policies

- You should work in groups of 4 or 5 students.

Remark: This assignment is loosely based on lab assignments given at Simon Fraser University, Canada

Good Luck