**DESIGN DOCUMENT**

**Client**: The client does the following –

1. Reads from config file all the server details.
2. Randomly choose an object from a list of objects. (Equally likely)
3. Generate the server details where the object is available using the hash function.
4. Randomly decide whether to read/write to the object. (Equally likely)
5. The client keeps doing this n number of times given as input (20 default)

If Read –

1. Randomly select a server from the hashed server values and try to connect with it. If it fails keep trying to setup connection with any of the remaining servers, until one succeeds or all fails.
2. Send the Read Request to the chosen server if any.
3. Read the response from the server to fetch the contents of the object in that server, if the client receives an ACK from the server. If it receives an ERR, it prints to the console and lets the user know that the read request failed.
4. Close the connection.

If Write –

1. Client tries to set up a channel between itself and all the servers individually that are needed for that object.
2. If the client is unable to connect to minimum two servers then send a write abort message to the third server if it connects to the third server.
3. Else, send a Write Request with the time stamp, the client id, the replica servers, the write message and the file name to all the servers individually.
4. Receive a write ACK or a write ERR from the server in case the write went successfully at the server or failed.
5. Close all connections.

**Server**: The server does the following –

1. Create a Thread Pool.
2. Have a Priority Blocking Queue to hold request jobs for each object.
3. Have a Map to implement lock for each file and keep track of the job that locked that file.
4. Read all the server details from the config file.
5. Accept incoming connection requests and use a thread from the pool to handle those requests.
6. Receive the incoming message. Split the message and determine the details of the request, if the request comes from a client or a server.
7. If the request is from a Client –
8. If it is a Read Request –

Call Client Read Request Handler.

* Read from the requested file and send the content of the file to the requesting client with an ACK message if the read is successful otherwise send an ERR back to the client.

1. If it a Write Request –

Call Client Write Request Handler.

* Create a job for the incoming write request.
* Add the job in the Priority Queue of the corresponding object.
* Check if the job as reached the front of the queue, if not sleep for sometime until it is at the front of the queue.
* Lock the object and do not allow any server to proceed on the same line unless this server processes it and unlocks it. Also, keep track of the job that has locked the file.
* Send Vote Request to all the other replica servers.
* Receive the Response of the Vote Request from all the servers. If an ERR is received reject the request. If a Vote Accept is received keep track of the count of votes casted by the other servers for the request. If a Vote Reject is received remove the lock on the file and retry the voting procedure again after sometime.
* If the count of votes cast by the other servers is at least 1, then the server has sufficient votes to perform the write operation, it performs the write operation and removes the job for that request from the queue and sends a release message to the replica servers, and receive a release ACK from them. It releases the lock on the object after that.
* Sends an ACK or ERR to the client.

1. If it is a Write Abort request –

* Print the message was received and send an ACK back to the client.

1. If the request is from a Server –
2. If it is a Vote Request

Call Server Vote Request Handler.

* Create a job for the vote request.
* Check the job that is currently locking the file, if there is no job currently locking the file wait for some time, if the job that has locked the file is the same as the job for the vote request (same logical time stamp and same requesting client id), then send a Vote Accept. If this is not the case check the job at the front of the queue for the file, and if they are not related to this job then send a Vote Reject and then make the thread sleep for some time unless the next request is received.

1. If it is a Release Request

Call Server Release Request Handler.

* Create a new job for the release request
* Check if the queue contains the job with the same details as the release job (same logical time stamp and same requesting client id), if so wait until it is processed. When it is processed and removed from the queue send a release ack.