**CORRECTNESS**

**Hash Function**

The connections between and client and a server is decided by a hash function represented by H(Ok). So to perform read/write the client connects to 3 other servers using the hash function H(Ok) % 7, H(Ok)+1 % 7, H(Ok)+2 % 7. So the hash function produces 3 consecutive values like (2,3,4) or (7,1,2) within the range 1 – 7 which is the number of servers we have. So based on the value of the hash function the client makes connections to the respective servers. The hash function is determined by the object value and each object is assigned a unique value assigned so that it’s used to find out in which sites the object is present. Once the client connects to the 3 servers, it sends request message to perform read/ write operation on the object available in that server. For example files (f1to f7) are assigned initial values based on which it’s replicated across the other two servers using the hash function. So if f1 is selected and it holds the value 3 then it’s available in server 3 and its successive replicated servers are server 4 and server 5.

Selected object – f1 (value assigned =3)

Server 1 = H(f1) % 7 = 3

Server 2 = H(f1+1) % 7 = 4

Server 3 = H(f1+2) % 7 = 5

So the selected server are Server 3, Server 4, and Server 5.

**Disrupted Connections**

In this project, to simulate the disruption of channels I have enabled a timeout where a particular server will not be able to accept any request from any of the sockets. In other means this happens even if any one of the servers is not running thus causing the channel to be disrupted and no messages can be communicated across it. Another disruption that can happen is when the messages in a channel get lost and to implement this I have made sure that a message is not sent through a socket randomly so that the server will not receive any request from the client but still be connected to the client. So as to making sure in case of any disruption how it is handled efficiently in the channel. Thus I have considered the two scenarios and implemented the two types of disruptions possible in this project.

**Voting Protocols**

This project mainly aims to implement data replication across different data servers using a voting protocol. So as part of the voting protocol I have decided to use a simple quorum based voting to decide on performing the write operation on that server file. To ensure serializability, the quorum does not allow two transactions to write a data item concurrently. Every site is assigned a vote, and to perform any operation on a site it needs to acquire the votes from the list of servers in a quorum. Only if the site acquires maximum number of votes, it’s eligible to perform the operation. That is among the 3 servers each server much receive the votes from one other server to get maximum votes (2 out of 3) to perform read/write on those servers.

Let’s say server 1 receives a request from the client and it waits for the votes from the servers in the quorum i.e., server 2 and server 3. If server 1 receives votes from both the server then write operation is performed in the file present in all the 3 servers. If server 1 receives vote from only one other site then write is performed on those two sites only. If it doesn’t receive any acknowledgement from other 2 sites then write is not performed across any of the sites. In the below table I have listed the files and the servers they are replicated in.

|  |  |  |  |
| --- | --- | --- | --- |
| File Name(Ok) | Server 1 | Server 2 | Server 3 |
| F1 | S3 | S4 | S5 |
| F2 | S4 | S5 | S6 |
| F3 | S5 | S6 | S7 |
| F4 | S6 | S7 | S1 |
| F5 | S7 | S1 | S2 |
| F6 | S1 | S2 | S3 |
| F7 | S2 | S3 | S4 |

So the following files are available at the following sites available for write / read operations. If the file is searched in other servers then it will not perform the function not it will send out the vote to the other servers in the quorums. So to access F4 the server 6 getting the request from the server will have to get the other server 7 and server 1 vote to perform the write. For read it’s enough if the object is present at any one of the three sites.

**Safety Conditions**

**Read**

When a read request is sent from a client to one of the servers of the object, the server checks if the file is available to process the request and reads the data from the file. This process of read happens consistently as long as there is no server disruption happens. In case the server crashes or disruption happens in the channel then it is not possible to process the read requests consistently. This situation can persist even after the server recovers because the files in majority servers would not have been in sync. So as a result an error message is displayed and “No acknowledgment” message is sent to the client.

Also another situation a read request might fail is during the first run of the program because the file would be created at that time and no data would be present inside it. Only after few writes on that file it will be ready to process the read requests successfully.

**Write**

When a write request is sent from a client to a server to perform writing a message onto the replicated file in the server, it’s processed in a FIFO channel. To perform the write operation on the site, the client calculates the hash function H(Ok) and the other servers numbers to which it needs to connect. So the client sends write request to all the 3 servers in the quorum to perform the write operation on the objects located at those sites. As this is a FIFO channel, there is a priority queue maintained at each server which takes care of the write process to happen sequentially. This queue will store the client requests in the order of increasing timestamp when the client has initiated the request. This is because there can be many concurrent client request reaching the same server and to maintain consistency in writing to the file the request at the top of the queue is processed first.

Processing the write request takes place after the voting protocol is completed. So the server which got the client request will communicate with other two servers in the quorum list to get enough votes to complete the write request. So this server will send “Vote Request” message to the other two servers and until that it puts a lock on the file and waits till it receives acknowledgment messages from those servers. If the server receives acknowledgement from any one out of the two servers, the number of votes will be 2/3 and as per the voting protocol the write operation happen onto the file. Then it sends release message after exiting the critical section after performing the write and remove the job from the priority queue. Now the lock will be released and acquired by the next process in the queue to enter the critical section to process the next job. Even if the third server in the quorum fails to send the acknowledgement there is maximum votes and it will not affect the write process.

If the server receives “No acknowledgement” from both the servers then according to the voting protocol the write should not happen across any of the sites to maintain consistency of the files. Also when a server crashes or disrupted and is unable to accept any requests or connections fails then in such scenarios the server or client receives the “No acknowledgement” message and thus not contributing to the vote count of the jobs. Thus consistency is maintained in write process across all the files in different servers.

**Liveness**

In this project there will be many request and acknowledgements received and sent from and to the servers and the clients. So when processes happens concurrently there are chances of deadlock happening which can cause inconsistency and also lead to site failure. So one of the liveness conditions maintained is the FIFO priority queue which is maintained to queue up all the request received at a site based on their timestamp. This helps the server to process each request individually and also one at a time. The other one is deadlock prevention which is implemented by locking the sites when processing each request so that no other jobs can perform write/ read on the same site concurrently. So when a job is processed from the top of a queue, a lock is put on the object until that process is complete and is removed from the queue. If a write process needs to be performed, then the object is locked until the process receives majority votes from the other servers to perform the write operation or abort the process. Once the process is completed, the lock is removed from the object at that site and I used for the next process. So no shared resources are locked by two process concurrently. Therefore there will be no possible situation for a deadlock and thus it’s avoided.