Distributed Systems Final Exam

Report

Viktor Máni Mønster
https://github.com/Hexfall/DISYSExam

Attribution

I hereby declare that this submission was created in its entirety by me and only me.

My implementation is built on my solution to the mock exam as a base. That implementation in turn, borrows some patterns from earlier mini-projects, but none worth mentioning.
https://github.com/Hexfall/DISYSMockExam

Multiple Choice Answers

Question	1	2	3	4	5	6	7	8	9	10
Answer	iii	ii	iii	ii	ii	i	ii	ii	ii	i

Implementation Discussion

Along with the code turned in with this report, my implementation can also be found at https://github.com/Hexfall/DISYSExam. (for the sake of guaranteeing that my work cannot be plagiarised, this repository will be private until the end of the exam)

System Model

My implementation follows a passive replication architecture, which is inspired by the RAFT model, though not entirely faithful to it.

This means that one replica is designated the leader of the cluster and only it takes requests from clients. It then propagates changes to the rest of the cluster, which serve as back-ups in case of node failure in the leader.

Minimum Node count

The minimum amount of nodes in the system is 2 (both replicas), on account of the fact that the system must tolerate at least one node failure (and I consider a system of zero nodes as failed).

Realistically however, you would want to add one more node to this number, as there is little meaning in a system with no clients.

Crash Recovery

Explain how your system recovers from crash failure.

The case of a client node-failure is trivial. The system is unhurt, and a user can simply restart their client.

The case of a non-leader replica-failure, whilst a little more complex, is still relatively simple. The leader listens to the heartbeat of all it's sub-nodes, and should it fail to detect a heartbeat, it removes the unresponsive node from it's register of sub-nodes, and propagates this failure to the other sub-nodes as well.

In the case of the failure of the leader node, another replica rises to the occasion and assumes the mantle in their fallen brother's place. Rather than through election, my implementation decides the next leader of the cluster through succession. The leader node keeps a register of all sub-replicas (in the order they joined the cluster), and the first element of this list assumes leadership in the case of decapitation.
Just like the leader listens to the heartbeats of sub-nodes, so do sub-nodes listen to the heartbeat of the leader. Should they find that the king is dead, the successor recognizes their promotion, and other sub-nodes rejoin the cluster with the new leader at the helm (though after a small delay, to make sure the new leader has also notice the leaders failure)
Since all nodes store the same data, clients should feel no difference, except perhaps having to connect to another node.

Repeatable Read Guarantee

if a call put(key,val) is made, with no subsequent calls to put, then a new call get(key) to that node will return val.

When my system receives a request to store a value… it does so. Therefore, so long as the value isn't overwritten, it can easily retrieve this value upon request.

Initialization to zero Guarantee

if no call to put has been made for a given key, any get call returns 0 (zero).

When requested for a value, a replica checks whether it contains an entry for the given key. Should it fail to find an entry, it defaults to returning zero.

Consistency Guarantee

given a non-empty sequence of successful (returning true) put operations performed on the hash table, any subsequent get operation at any node should return the same value

This is guaranteed by value propagation from the leader node.

Liveness Guarantee

every call to put and get returns a value. If a call put(k,v) returns false, then some call in a repeated sequence of put(k,v) will return true (eventually, put and get succeed).

Except in the case of untimely node failure, the system never intentionally declines a Put request.

Reliability Guarantee

your system can tolerate a crash-failure of 1 node.

The system can tolerate crashes up until the death of the very last replica, so long as the failures are simultaneous, through the method described in Crash Recovery.