Course Project: Quorum-based Total Order Broadcast

Distributed Systems 1, 2019 – 2020

Team members: Davide Tessarolo, Devis Dal Moro

# Introduction

In this report, we’ll explain the architectural and implementation details regarding the development of a protocol for the coordination of a group of replicas, sharing the same data, that tolerates multiple node failures thanks to a quorum-based approach. Details on how the protocol works are not dealt within this document and are taken for granted, since they are well explained in the requirements document produced by the teaching instructors.

# Client

The client actor has just two simple needs which consist of reading and writing the value stored by the replicas. In both cases, there is a random selection of the replica that it’s going to query. Indeed, it knows from the beginning all the replicas that should be available, but it doesn’t know if they actually are. Thus, whenever it sends a read or a write request, it cannot be sure its request is going to be fulfilled.   
Specifically, the write request happens in a totally asynchronous mode, in the sense that it’s completely unaware of the fact that the request has actually been received and then processed by the distributed storage system. While, in the case of a failed read (which occurs because the random selection picked up a crashed replica) there is a timeout set up specifically for communicating the failure. Read requests and corresponding timeouts are associated to an internal (wrt client) incrementing id which is passed along in the request itself. If the response arrived, it’d be clear to which read request it’s related to, because the replica would have put the read id in the very same response.

# Replica

All replicas hold an integer value and a flag which to denote the fact that a specific replica should act as a coordinator. At the start, the coordinator is predetermined (simply the one with the greater id), but along the different epochs it changes as a result of the coordinator crash in the previous epoch. However, as result of the architecture and processes detailed below, it’s impossible to have simultaneously more than one coordinator. Furthermore, each non-faulty replica holds the reference to the current coordinator.

Since any replica could be in a future epoch a coordinator and will need to multicast/broadcast messages, when the system boots up a “JoinGroup” message is delivered to every member, notifying about all the replicas and the first coordinator.

Whether the replica is the coordinator or not, it does need an internal clock, which consists of a data structure created ad-hoc containing two integers: one for the epoch and the other for the sequence number. Its value corresponds to the number of delivered updates by the replica. Specifically, the coordinator will also need an hashmap in order to know the amount of UPDATEACK received for each UPDATE message. In the latter scenario, we could use the corresponding clock value as key. Further, all replicas retain the FIFO list of the pending updates, waiting for the WRITEOK broadcasted from the coordinator.

Specific flags are embedded in their structure, so they can leverage them to know if an election or a synchronization phase is in progress, as well as if they are crashed, because the state a replica finds itself in influences its reaction to the receipt of a message. Other data structures worth to mention are the hashmaps needed in order to handle the timeouts related to the waiting of WRITEOK messages (using the update as key) or of the ACK of election messages (using an incrementing counter valid for the replica during that election session as key). Finally, two other individual timeouts are needed to cover the following two cases: the non-arrival of the coordinator’s heartbeat or the complete failure of an election session.

### Dealing with requests

Read, write -> che porta all’update

As per project’s requirements, all non-faulty replicas respond to read requests with their current stored value, without the need of further interaction. Write requests are instead more complex and require the forwarding of the request itself to the coordinator

### Update

Upd, upack, writeok… timer

### Heartbeat

Heartbeat send / listen….

### Crash and election

Sda

# Conclusion