David Gonçalves 73891
João Portugal 84731
Nuno Pinhão 84748

Report

Introduction

For this assignment, we were asked to extend the functionality of the first part to support tolerance to Byzantine faults in servers, Byzantine Clients and prevent potential spam attempts by users when trying to transfer a good.

Problem

Every distributed system is prone to crashes and faults and should be able to handle them. There are different types of faults such as crash failures, byzantine failures, network failures. Our system implements and focus on the first two type of failures. Crash failures can happen if a participants shutdown unexpectedly. This type of crashes was already implemented in the first part. Byzantine Faults happen when a participant acts arbitrarily. This may be due to an adversary taking control of a server, after which they may actively attempt to undermine the system. It was a possibility that both clients and servers could become byzantine and it was needed an architecture that could prevent the trading system from being corrupted and abused. It was also objective of our system to implement an anti-spam system, to prevent malicious clients from spamming and overburdening the system, and to find ways to prevent Byzantine clients disrupting the system.

Solution

The solution we were proposed and implemented was an implementation of a Byzantine Atomic Register, that functions on a quorum of N servers where N = 3f+1, f being the maximum number of faults that can occur. In a test basis, we set 5 servers, meaning f can only be 1, however our system is implemented to support N servers. Our system works on a quorum basis, where we wait for (N/2) + 1 answers before making a decision whether it is a read operation or a write operation. For correctness, we assure the properties of a Byzantine Atomic Register: Liveness, Safety and Ordering. Liveness as in, any operation will eventually return a majority of correct processes (a quorum). Safety where In the absence of concurrent or failed operation, a read operation returns the last value written and Ordering where we ensure that if a read returns a value v and a subsequent read returns a value w, then the write of w does not precede the write of v. In practical terms, what happens is the following: a client chooses a read / write operation:

1) All the servers are enquired about the read operation and return the result (whether it's the direct result of the operation, or the last operation they have stored)[*1] plus the tag they have stored (in a persistent way, in the database in case the server crashes). The client waits for a quorum of answers and then outputs it.
2) If a server responds with a tag behind of the quorum, it's performed a write-back operation where the server is informed of all the write operations that the used performed.
3) (Only for write operations) The client then proceeds to write to all servers the write operation and the tag + 1 and waits on a quorum of replies (if the operation was successful or not).

In this way we ensure that for every operation, there is always a quorum of answers preventing byzantine servers from any wrong-doing.

For the anti-spam, we decided to implement (but did not fully implement) a small proof of work algorithm, where a block is hashed many times until the hash contains a fixed predefined number of leading 0's (we have a nonce, part of the message to hash, that increments with each hash), which may take a few seconds, minutes or hours depending on the predefined number (in our case it's seconds). Then, this hash is verified by the server as well as the hash of the previous block (request). This way, spamming is mitigated.

We did not implement any measures to prevent possible byzantine clients. However we do have some measures that can be applied to prevent them. One is open source implementation of clients (and OS, and libraries...) which means security via public scrutiny, reduce attack surface via trusted and tamper-resistant devices, for example use smartcards for crypto operations, protection of cryptographic material. In our case, a citizen card could be used for users for example. However this only prevents the system from generic attacks. If a client sends a proper write with a byzantine value the server will still accept it. One measure we thought of, despite not being efficient at all and very heavy, was if a tag to be written is suspicious (for example if the tag held is very different from the received tag) then the server could eventually enquire other servers of the tag they currently have to ensure that the client isn't trying to write arbitrary values. If every server doesn't respond with tagtobewritten-1 value then the client is byzantine and the server discards.

[*1]Clarification: If a user performs GetStateOfGood the server replies with the result of the state of good and the current tag, if it is a write operation it answers with the last write operation performed on the server and the tag. This is only to obtain the last tag to get the most recent tag.