Paper Review 7: Chain Replication for Supporting High Throughput and Availability

Yibo Zhao
2019533235
Shanghaitech University
zhaoyb1@shanghaitech.edu.cn

I. Introduction

To build a better performance replicated fail-stop storage system, the authors proposed chain replication, a large scale storage service approach that maintains high throughput and availability while ensuring strong consistency.

II. Details

A. Target Systems and Features

Two main current storage systems are file system and database system, providing QUERY and UPDATE operations that client can get and modify the storage data. The paper concerns about storage service, which is an appropriate solution when database system is too expensive and file system lacks rich semantics.

Currently system solutions, such as GFS, do not support the strong consistency guarantees based on consideration of not sacrificing performance and availability. However, chain replication can simultaneously support these three features.

The storage system provides two operations: query*(objId, opts)* and update*(objId, newVal, opts)*. In this system, three transitions are allowed in client's view: T1: client request arrived, T2: client request ignored, and T3: client request processed, and any of the state transition in any of server must equivalent to no-op or one of these three transactions. Another character is storage system cannot distinguish between the lost of client's request and the transient outages of storage server.

B. Protocol Design

For object replicated among t servers, chain replication connect all t servers as a chain, with a head server and a tail server. The client's queries are directly redirected to the tail server to process, while the update requests are sent to the head server to apply. The updates are then propagate along the chain until it arrives tail server. Strong consistency are therefore guaranteed since the operations are serially performed on tail server.

Each server maintains two lists: $Hist_{objID}$ for the updated being performed on *objID*, and $Pending_{objID}$ for the unprocessed requests. From the client's view, $Hist_{objID} == Hist^{tail}_{objID}$, $Pending_{objID} = Pending^{i}_{objID}$, which means the two allowed state transitions are client requests arrives at chain and request processed by tail, and the two state transitions are equivalent to T1 and T3, respectively.

When coping with server failures, the system needs a master services to detect failure and adjust servers, and each server maintains a $sent_i$ queue storing updates that forwarded downwards but have not received the ack from tail. So when

- head H fails, we just delete H, which is consistent with T2.
- tail T fails, just deleting T is also consistent with repeating T3.
- other server S fails, just forwards $Send_{s^-}$ to S^+.
- extending chain, just adding server at the tail of chain.

When comparing with primary-backup protocol, both the worst and best cases of chain replication are better.

III. Strong Point

- Ruling the state transition of servers from the view of client, which absolutely make the protocol design clearer.
- It provide high availability while ensuring strong consistency.
- The protocol dealing server failure are simple and efficient.

IV. Weak Point and future work

- Chain replication disseminate updates serially along the chain, which induce higher latency in update operations. So chain replication is not suitable for frequent updates.
- Maybe an optimization that alter the chain topology but also provides strong consistency guarantee can be the future work.