Chapter 2
Replication Techniques for Availability

Robbert van Renesse and Rachid Guerraoui

Abstract The chapter studies how to provide clients with access to a replicated object that is logically indistinguishable from accessing a single yet highly available object. We study this problem under two different models. In the first, we assume that failures can be detected accurately. In the second we drop this assumption, making the model more realistic but also significantly more challenging. Under the first model, we present the primary-backup and chain replication techniques. Under the second model, we present techniques based on voting. We conclude with a discussion on reconfiguration.

2.1 Introduction

Replication is creating multiple copies of a possibly mutating object (file, file system, database, and so on) with the objective to provide high availability, high integrity, high performance, or any combination thereof. For high availability and integrity, the replicas need to be diverse, so failures are sufficiently independent. For high performance, there just needs to be a sufficient number of replicas in order to meet the load imposed on the replicated object.

In this chapter, we will focus on replication techniques that ensure high availability. In particular, we will study techniques that provide clients with access to a replicated object that is logically indistinguishable from accessing a single (non-replicated), yet highly available, object. This “indistinguishable from a single object” property is sometimes called linearizability, one-copy semantics, or simply consistency, and is ensured by enforcing a total order on client operations. Of course, such a strategy can only work under a restricted failure model. For example, if failed communication links partition the replicas, then it may be impossible to provide both availability and consistency for an object.
While a number of different replication techniques exists, two different approaches have become particularly well-known: active replication and passive replication. In active replication, also known as state machine replication, client operations are ordered by an ordering protocol and directly forwarded to a collection of replicas. Each replica individually executes the operation. Keeping the replicas consistent requires that processing be deterministic: given the same client operation and the same state, the same state update is produced by each replica.

In passive replication, also known as primary-backup replication, one of the replicas is designated primary. It executes the operations and sends the resulting state updates to each of the replicas (including itself), which, passively, apply the state updates in the order received. Note that in passive replication it is not necessary that operations be deterministic—typically, the primary will resolve non-determinism and produce state updates, which are deterministic.

These approaches have various advantages and drawbacks when compared with one another. If operations are compute intensive, then active replication can waste computational resources, but if state updates are large, passive replication can waste network bandwidth. Active replication cannot deal with non-deterministic processing but can mask failures without performance degradation, while passive replication may involve a detection and recovery delay in case the primary crashes.

Various hybrid solutions that combine both approaches are common. Some processing is executed on just one replica, while other processing is performed by all replicas. They are neither purely active nor passive approaches, and face different trade-offs.

This chapter avoids discussion of how operations are processed. Instead, it models an object’s state by the sequence of operations. For example, if the object represents a bank account, we keep track of the history of deposit and withdraw operations, rather than of the running total. Doing so makes it easier to talk about consistency, as we can compare histories stored at different replicas and determine if one is a prefix of the other, or not. If all we had is a running total, then such a comparison would be impossible.

The chapter is organized as follows. In Section 2.2, we will present a convenient model of an unreplicated object. Then, using this model, we will describe two replication techniques that assume a simple failure model in Section 2.3. In Section 2.4, we will make the failure model more realistic (and more challenging) while discussing how to adapt the replication techniques accordingly. Section 2.5 discusses approaches for reconfiguring a replicated object. Finally, Section 2.6 concludes with a brief comparison of the techniques discussed in this chapter.

### 2.2 Model

For simplicity, we will assume that there is only one object. We find it convenient to model an object as a finite sequence of uniquely identified deltas, \( H = \langle d_1, d_2, \ldots, d_b \rangle \), encoding a history of \( b \) updates applied to the object. A delta is a tuple \( (\text{update identifier}, \text{operation}) \). A particular update identifier can only appear once in the history, although two different deltas may well contain the