Distributed Systems

Christian J. Rudder

January 2025

Contents

This page is left intentionally blank.

Big thanks to **Prof. Anna Arpaci-Dusseau**
and **Prof. Ioannis Liagouris** for teaching CS351: Distributed Systems
at Boston University [1].

All illustration contain original assets.

Disclaimer: These notes are my personal understanding and interpretation of the course material. They are not officially endorsed by the instructor or the university. Please use them as a supplementary resource and refer to the official course materials for accurate information.

Prerequisites

This text assumes the reader has a basic understanding of computer science and programming. It will also assume they are somewhat familiar with computer architecture and operating systems at a high level. The text will review these concepts briefly for completeness, but it will not try to teach them from scratch or provide a full understanding of these topics.

The main focus will be on distributed systems, and will touch on:

- **Concurrency and Parallelism**
  - Concurrency, Parallelism, Threads
- **Consistency and Fault Tolerance**
  - Consistency, Fault-tolerance, Atomicity
- **Distributed Systems and Coordination**
  - Asynchrony, Coordination, Logical Time, Snapshots
- **Consensus Algorithms**
  - Raft, Paxos, Consensus
- **Replication and Data Management**
  - Replication, Sharding, Cluster
- **Protocols and Computing Models**
  - RPC, 2PC, Broadcast
- **Technologies and Tools**
  - MapReduce, Spanner, Dynamo, GFS, TLA+, Golang

0.1 Dynamo: Amazon's Highly Available Key-Value Store

It's 2007 and Amazon's global e-commerce platform must remain "always-on" despite continual component failures—outages cost revenue and customer trust. To achieve this, Dynamo sacrifices strict consistency for low-latency availability:

Definition 1.1: Design Goals of Dynamo

Dynamo is a decentralized, highly available key-value store designed to:

- **Always-Writeable:** Never reject writes under partitions or node failures, deferring conflict resolution to the read path.
- **Incremental Scalability:** Scale out by adding or removing nodes without downtime or manual repartitioning.
- **Decentralized Symmetry:** No central coordinator, based on Consistent Hashing (**??**).
- **Low-Latency Performance:** Dynamo provides its clients an SLA (Service Level Agreement) that under any load, it provides a 300ms response 99.9% of the time.
- **Eventual Consistency:** Allow temporary inconsistencies under failures, ensuring all updates reach replicas eventually.

Consistency Model: Weak Consistency, favoring availability over strict consistency.

Next we discuss how it achieves this decentralized, highly available key-value store:

Definition 1.2: Quorum System – Gossip (Part 1)

A user's keys is hashed to a point on a 128-bit ring and mapped to an ordered list of N nodes (the **preference list**). The first clockwise node, N_i, becomes the **coordinator**:

- **Writes**: Coordinator N_i receives and sends `put(key,value)` in parallel to the next top $N-1$ replicas, waits for acknowledgments from any W distinct replicas (**including itself**), then returns success to the client.
- **Reads**: N_i receives and sends `get(key)` to all $N-1$, returning success after R nodes acknowledge the read.
- **Conflicts**: Divergent data is reconciled via a vector-clock versioning history. If the vector clocks cannot be merged, it is left to the **client** to resolve.
- **Quorum Condition**: Choosing parameters $R+W>N$, ensures R and W overlap, diminishing staleness.

Concretely, if key k hashes to A's segment, A is the owner of k's data. Any other server N holds copies of k's data, but is **not the owner** (replica).

We'll call the below a turtle-back diagram; it illustrates the quorum system in a basic configuration:

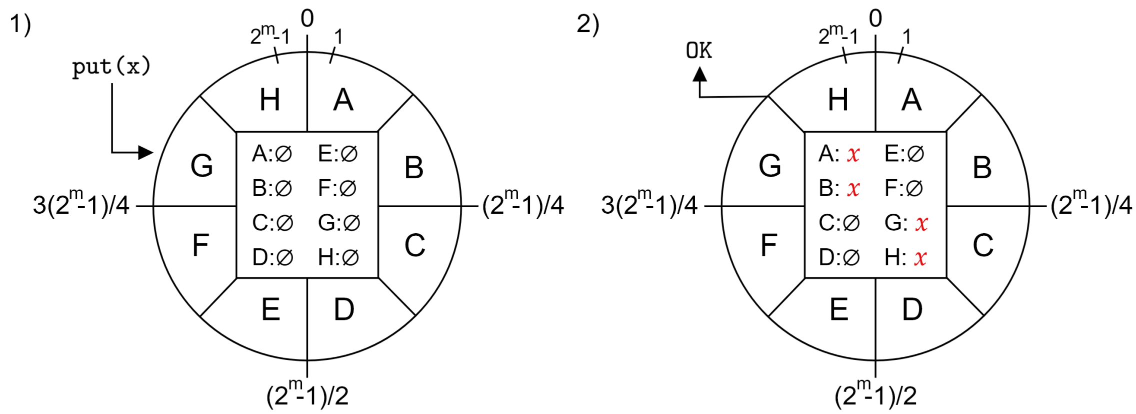

Figure 0.1: A basic quorum system. **To be clear**: virtual nodes are positioned at the spokes. For instance, *H* starts at 0, and ends at the next left-spoke G (the top-left corner of the center box). Here, $R = 4, W = 4, N = 7$, and servers are an ordered list of virtual nodes A–H. Also, here we say `put(x)` for brevity, while it's actually `put(key,value)`. 1) A client's put request falls within G's range. 2) The next top $W - 1$ nodes acknowledge, with G sending back an `OK` (success).

Moving on, we deal with the liveness of nodes, and how to reconcile data:

Definition 1.3: Quorum System — Gossip (Part 2)

Dynamo's monitors liveness with the following mechanisms:

Gossip Frequency & Peer Selection: Every node, once per second, picks a peer uniformly at random and exchanges its local membership-change log. A vector clock represents the change log, where each cell is a (node, version) tuple. This random peer exchange ensures that knowledge of joins, leaves, and failures propagates exponentially fast without a central coordinator.

Failure Detection & Sloppy Quorum: During a write, if a coordinator cannot reach a preferred replica (due to a failure or partition), it writes to the next healthy node in the preference list. That node stores the update alongside a **"hint"** tagging the intended replica.

In particular, each node maintains hints stored in a separate log unbeknownst to the client. The original paper makes no mention of any further constraints. This is called a **sloppy quorum**: it allows writes to proceed even if some replicas are unreachable.

Hinted Handoff: When the failed node rejoins, any node holding a hint for it will detect its revived liveness via gossip and then forward ("handoff") the stored updates—restoring the full set of N replicas **without** blocking client operations.

Below illustrates the gossip protocol and hinted handoff:

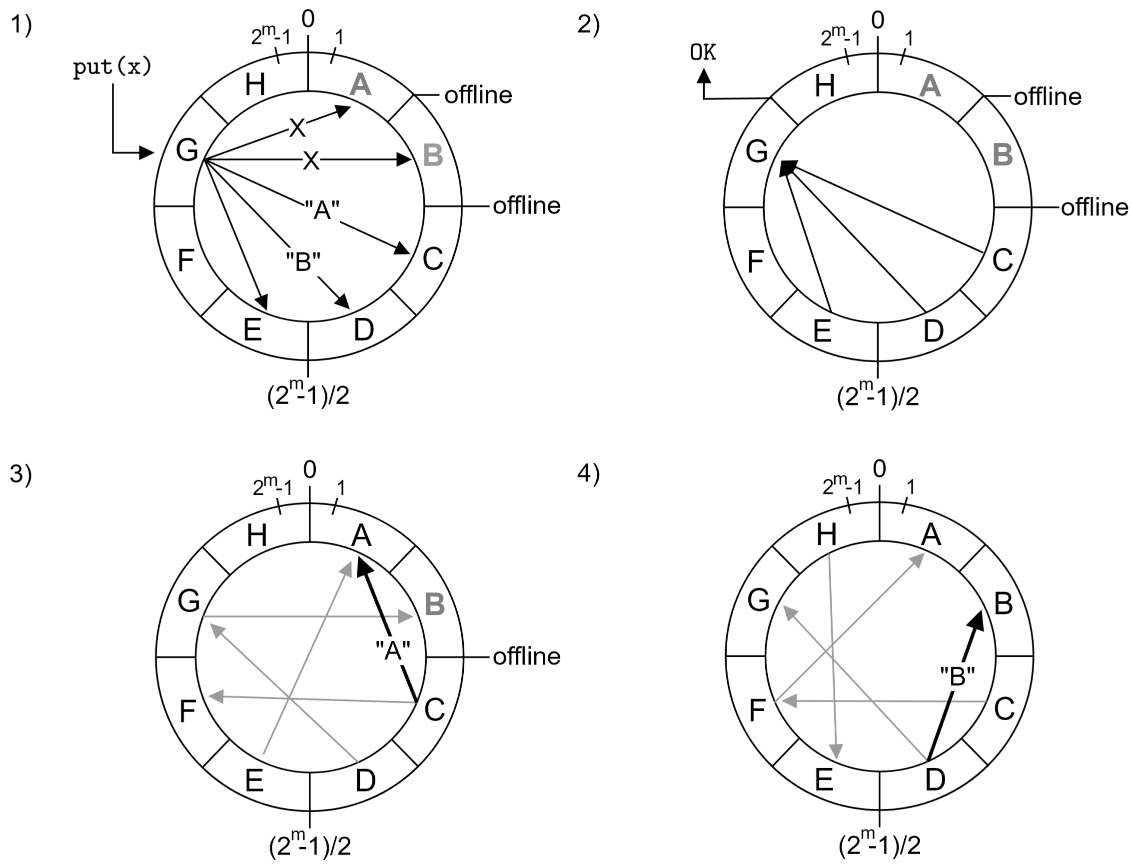

Figure 0.2: Gossip protocol and hinted handoff. 1) A put operation is sent to G, from which is then propagated. Though, A and B appear to be down, G resolves this by giving hints to C and D. 2) Nodes C, D and E `ACK`, allowing G to return `OK`. 3) During the gossip protocol A comes back online; C notices this and sends the hint to A (hinted-handoff). 4) Later in the gossip protocol, B comes back online; D notices and hands off the hint to B. **Note**: The original paper doesn't specify whether a replica may only hold one hint or multiple hints. In this figure we assume one hint per replica.

Bibliography

[1] Ioannis Liagouris. Cs351: Distributed systems. Lecture notes, Boston University, Spring Semester, 2025. Boston University, CS Department.