Distributed Systems

Christian J. Rudder

January 2025

Contents

Contents **1**

1 Introduction **5**

2 Working with Distributed Systems **6**

2.1 Saving System State: Snapshots 6

Bibliography **7**

This page is left intentionally blank.

Big thanks to **Professor 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

— 1 —

Introduction

Working with Distributed Systems

2.1 Saving System State: Snapshots

This section discusses saving state. This is useful for fault-tolerance and system migrations.

Definition 1.1: Snapshot

A **snapshot** is a consistent global state of a distributed system at a specific point in time.

Definition 1.2: Consistent vs. Inconsistent Snapshots

To evaluate a snapshot's consistency, we compare events in the system **pre-snapshot** (events before the snapshot) and **post-snapshot** (events after the snapshot). The snapshot itself is instantaneous, like a photograph. Given an event ordering r:

- **Consistent Snapshots**: Respect causal dependencies. Let there be events e_1 and e_2; If $e_1 \rightarrow_r e_2$, then e_1 must be included in the snapshot if e_2 is present.
- **Inconsistent Snapshots**: Violate causal dependencies. If e_2 is included without the causally preceding event e_1, then the snapshot is inconsistent.

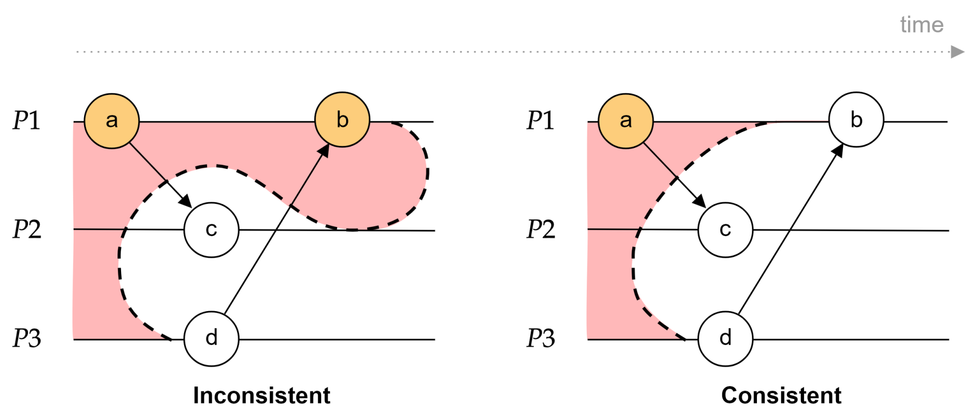

Figure 2.1: Inconsistent vs. Consistent Snapshots (pre-snapshot highlighted in red)

Here, in the inconsistent snapshot, b is included without d, its causally preceding event. In the consistent snapshot, only a is included. In this snapshot c and d could be added without violating causality.

Bibliography

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