CS 582: Distributed Systems

ZooKeeper

Dr. Zafar Ayyub Qazi

Fall 2024

Acknowledgments: Slides use some material from Ryan Stutsman (University of Utah)

Specific learning outcomes

By the end of today's lecture, you should be able to:

- ❑ Explain the core design of ZooKeeper
- ❑ Analyze how ZooKeeper can scale system throughput with increasing replicas
- ❑ Analyze the consistency guarantees that ZooKeeper is providing

We have covered the fundamentals

- System Models
- Failure Detectors
- Remote Procedure Calls
- Physical Time Synchronization
- Logical Clocks: Lamport and Vector Clocks
- Leader Elections
- CAP and FLP
- Different Consistency Models
- 2-PC
- Basic Paxos and Multi Paxos
- Raft
- PBFT

Moving forward: key learning objective

- Understand how different techniques are applied and synthesized in designing large practical distributed systems

ZooKeeper

- General-purpose distributed coordination service
- Many use cases
  - Configuration management, leader election, naming service, data synchronization, etc.
- Used by many companies
  - Yahoo, Yelp, Reddit, Facebook, Twitter, eBay, Rackspace, etc.

Performance Question

- In a replicated system with N replica servers, can we get N times higher performance?

ZooKeeper Setup

ZooKeeper Performance

Servers	100% **Reads**	0% **Reads**
13	460k	8k
9	296k	12k
7	257k	14k
5	165k	18k
3	87k	21k

Table 1: The throughput performance of the extremes of a saturated system.

Recap: Linearizability

- All replicas execute operations in some total order
- That total order preserves the real-time ordering between operations
  - If operation A completes before operation B begins, then A is ordered before B in real-time
  - If neither A nor B completes before the other begins, then there is no real-time order
    - (But there must be *some* total order)

ZooKeeper consistency guarantees?

ZooKeeper consistency guarantees?

- Linerizable writes: all requests that update the ZooKeeper will be in some total order which respects the real-time ordering

- FIFO client order: all requests from a given client are executed in the order that they were sent by the client

Example (FIFO Client Ordering)

How is FIFO client ordering implemented?

ZooKeeper Sync API

- What is it? How is it implemented?

ZooKeeper Sync API

- Sync when applied before a read causes a server to apply all pending write requests before processing the read
  - A way to see do fresh reads, however such reads are slow

- Sync have to go through the leader (just like writes)

ZooKeeper Watch

ZooKeeper Watch Flag

- Clients can issue a read operation with a watch flag set
  - In which case the server promises to notify the client when the information returned has changed

Experiences of using ZooKeeper

Summary: ZooKeeper

- Service scales linearly with number of replicas, but only because reads can be done locally at replicas
- Consistency guarantees
  - Weaker than linearizability but well-defined, providing linearizable writes and FIFO client ordering
  - Also provides sync API which allows fresh reads but causes reads to slow
- Designed for ready-heavy workloads

4

Dynamo: Amazon's Highly Available Key-value Store

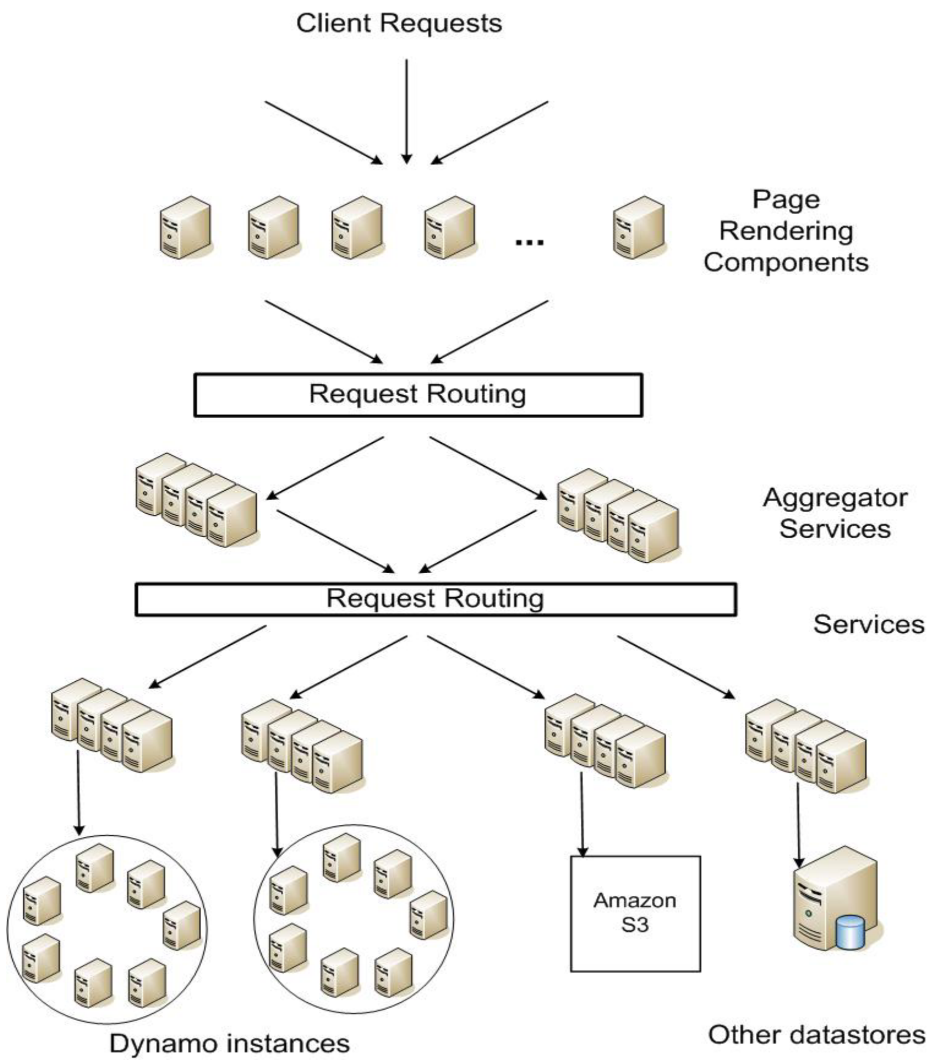
Great Success Story

- Many services at Amazon use it!
  - Shopping cart
  - Customer preferences
  - Session management
  - Sales rank
  - Product Catalog
  - …
- Now available as a cloud service on AWS
- Able to achieve high availability at Amazon scale
  - Applications using Dynamo have received successful responses for 99.9995% of their requests

Amazon Workload (in 2007)

- Peak load: Tens of millions of requests per day
- Tens of thousands of servers in globally distributed data centers

Architecture of Amazon’s Platform



Tiered service-oriented architecture

- Stateless web page rendering servers
- Stateless aggregator servers
- Stateful data stores (e.g., Dynamo)

Dynamo Requirements

- Highly available despite failures being commonplace
  - "Even if a data center is destroyed by tornadoes"
  - "Always writeable"
- Low request-response latency
  - Focus on 99.9^{th} percentile of the distribution
  - Ensure it is less than a threshold, typically 300ms
- Incrementally scalable as servers grow to workloads
  - Adding "nodes" should be seamless
- Comprehensible conflict resolution
  - High availability in the above sense implies conflicts
  - Need a way to resolve conflicts

Operating Enviornment

- Dynamo is only used by Amazon's internal services
- Assumes its operating enviornment is non-hostile
  - No security related requirements such as authentication and authorization
- Each service uses its distinct instance of Dynamo

Some Key Design Questions

- How to partition and replicate data?
- How to route requests and handled them in a replicated system?
- What sort of consistency guarantees to provide?
- How to resolve conflicts?
- How to cope with node failures?
- How to detect failures?

Dynamo: Synthesis of Many Ideas

Consistent Hashing

Vector Clocks

Failure Detection

Virtual Nodes

Object versioning

Gossip-based membership protocol

Sloppy Quorums

Hinted Handoffs

Merkle Trees

Next Lecture

- Deep dive into Dynamo design