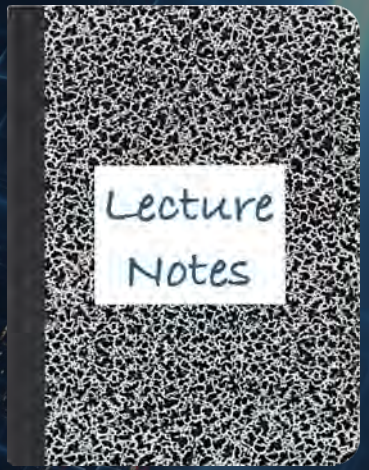


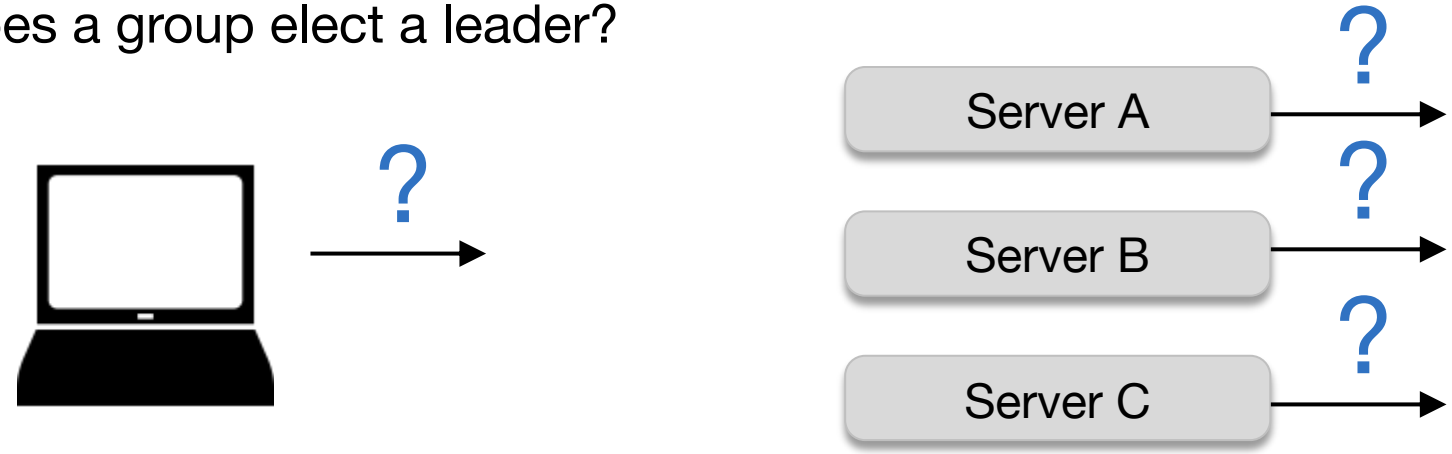
CS 417 – DISTRIBUTED SYSTEMS

Week 6: Part 1
Coordination Services

Paul Krzyzanowski

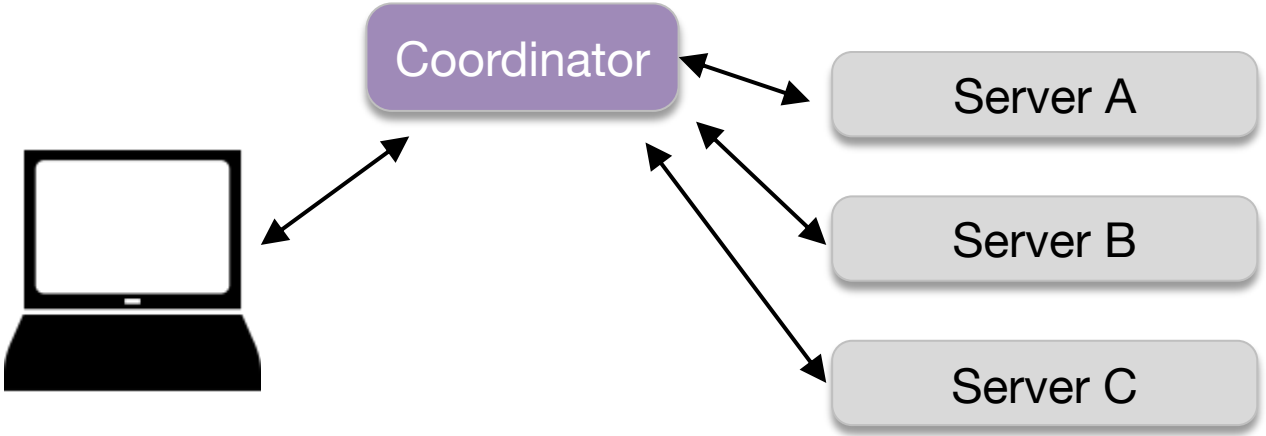
Coordination Problem

- How does a client know who to talk to?
- How does group member discover other group members?
- How does a client or group member grab a lock?
- How does a group elect a leader?



Add a Coordinator!

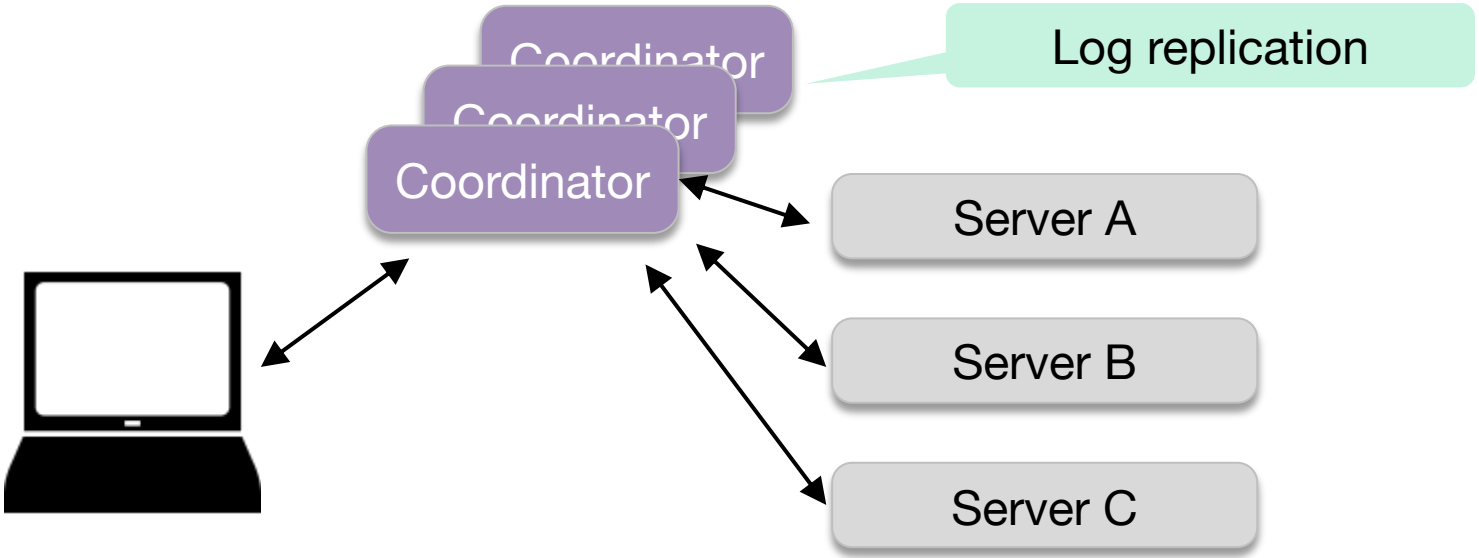
A central service for electing leaders, locating replicas, locking (mutual exclusion), and storing configuration information about various services



But Make It Safe

The coordinator cannot be a single point of failure!

We need to replicate it and keep all replicas up to date

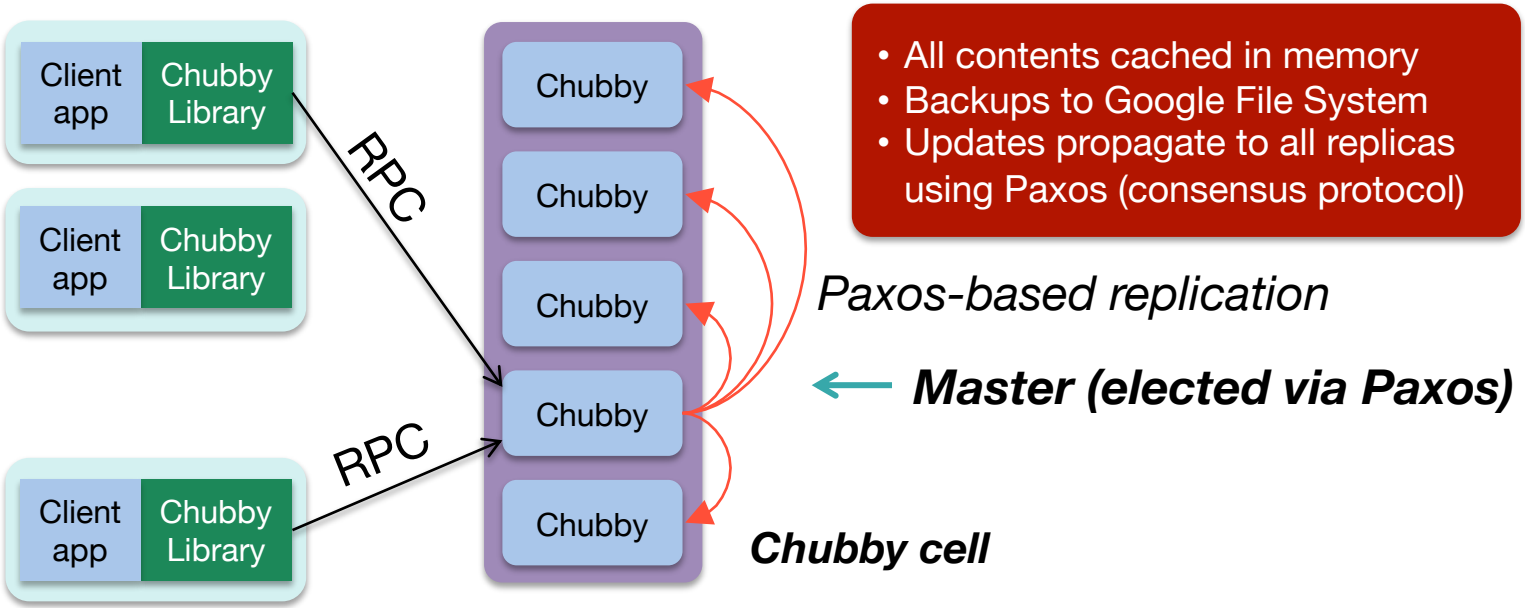


We'll look at three services: **Chubby, ZooKeeper, etcd**

Google Chubby

Chubby Deployment

- Typical deployment: 5 replicas = Chubby cell
- One elected master: all requests go to the master
  - Requests to other replicas fail and identify the current master



Chubby File System

- Every resource (**node**) is a file: `/ls/cell/rest/of/name`
  - `/ls`: lock service (common to all Chubby names)
  - `cell`: resolved to a set of servers in a Chubby cell via DNS lookup
  - `/rest/of/name`: interpreted within the cell

No integration into the OS: Think of Chubby as an object store accessed via a library

- Each node may have
  - Data (optional)
  - Children (optional) – *like a directory name that contains data as well*
- Read/write **entire contents** only
  - No partial reads/write
  - Write-through caching: updates sent to master
  - Master sends invalidations to clients with cached copies

Files as Locks (Leases)

- Every file & directory can be a reader-writer lock
  - One client can hold an **exclusive** (writer) **lock**
  - Or multiple clients can hold **shared** (reader) **locks**

- Locks are advisory
  - Locks are technically **leases**: they expire
    - A client sends a *KeepAlive* RPC during a Chubby session to say it is still alive
    - If a client fails, the lock will be unavailable for a *lock-delay* period (typically 1 minute)
  - If a client releases a lock, the lock is immediately available

- Chubby is designed for **coarse-grained locks**: hours/days
  - Not **fine-grained locks**: milliseconds (like locking a record in a database)

Events

Clients may subscribe to be notified of events:

- File content modifications
- Child node added/removed/modified
- Chubby master failed over
- File handle & its lock became invalid
- Lock acquired
- Conflicting lock request from another client

Avoids polling

Leader Election Made Simple

Suppose a group of Chubby clients needs to elect a leader

- Each client opens the same file (e.g., `"/ls/cs/myservice"`)
  … and tries to get an exclusive lock for that resource

- Whoever gets the lock, wins the election
- The winner can now store its contact info in the file
- When the lease on the file expires, run the election again

No need for user systems to participate in a consensus protocol

… the programmer doesn't need to figure out consensus

– Chubby provides the needed fault tolerance

Apache ZooKeeper

Apache ZooKeeper

Created at Yahoo to support services like Hadoop MapReduce, HBase, HDFS

- Inspired by Google's Chubby paper
- No locks
- Basic set of primitives that allow you to build locks, elections, barriers, …

File System: znodes

Like Chubby: pathname + optional data + optional children

- Types of znodes:
  - **Persistent nodes**: survive disconnection (act like regular files)
  - **Ephemeral nodes:** deleted when the client session ends
    - Used to detect client failures (name disappears = client's dead)
- Either node may be created as a **sequential znode**
  - System appends a monotonically increasing counter
    - 10-digit decimal integer
    - Provides unique naming

"Watches" – Event Notification

Clients can subscribe to events on a znode

- Get notified if
  - Node created or deleted
  - Node data changed
  - A node's immediate children list changes
  - Node is deleted
- Watches are one-time triggers
  - The watch event is cleared after delivered
  - Client will have to do another *read* and set another *watch*

ZooKeeper Consistency

Replicas updated via consensus-based log replication

- Zab (ZooKeeper Atomic Broadcast) – similar to Raft

- All writes go through the leader
  - Applied at all followers in the same order
- Reads: can be read from replicas (unlike Chubby)
  - May result in stale data
  - Optional **sync** operation forces a client to catch up to a leader's state

Building Locks

ZooKeeper does not offer locks, but you can build them

1. Create a sequential ephemeral znode under a lock directory
   - `"/locks/my-lock/lock-0000000042"`
2. List all children of `"/locks/my-lock"`
   - If your znode has the lowest number in the list, you hold the lock
   - Every new client that creates the znode will get a higher #
3. If not, watch the znode with the next-lowest sequence number
   - `watch("/locks/my-lock/lock-0000000041")`
   - When it's deleted (because the client crashed or released the lock), check the list again

etcd

etcd

Initially created for storing information about CentOS clusters

What was wrong with ZooKeeper?

- It was written in Java – required a JVM environment
- They wanted an HTTP/JSON interface for wide interoperability

etcd became the service for tracking the state of a cluster of Kubernetes services

- Kubernetes is a service that automates the deployment and management of containerized applications
- No etcd → no Kubernetes!

Design

- Fault-tolerant log replication via Raft consensus
- All reads & writes go to the leader by default
  - Optionally, you can read from a follower (but risk stale data)

etcd Files

- No directory tree (unlike Chubby and Zookeeper)
  - Flat key-value storage
  - Arbitrary object names for keys – no parent-child relationships
- **Watch API**
  - Monitor an object (name) or a set of objects (matching a prefix)
  - Receive a stream of change events
    - Not just one notification
- **Leases**: separate objects (identified by a lease ID)
  - Create a lease with a time-to-live and associate one or more keys with it
  - Refresh the lease periodically with keepalive messages
  - When the lease expires, the attached keys are deleted

Usage in the real world

- Chubby
  - Google-only solution, but the paper described the design
- ZooKeeper
  - Widely used in Apache projects (Hadoop, Hbase, …)
  - Used by Box to support the file storage service
- etcd
  - Backing store for Kubernetes clusters
  - Used at Google, Alibaba Cloud, Amazon, Boking.com, Spotify, Uber, Visa, …

Example Usage Patterns

Leader Election

Compete for a name in the coordination service

- Maintain the session or lease for it once you have it

- **Chubby**
  - Attempt to open the lock file and get an exclusive lock
  - Only one can succeed – winner writes info to the file and holds the lease

- **Zookeeper**
  - Each candidate creates a sequential ephemeral znode
  - The candidate with the lowest sequence number is the winner

- **etcd**
  - Candidate attempts a transaction that create a key under the condition that the key does not exist
  - The key disappears when the lease expires

Distributed Locks

- Grant exclusive access to a shared resource
- Coordination service provides the serialization
- Acquiring a lock is a write that goes through the consensus algorithm
  - Crashed lock holder's lease expires and the lock is released

Configuration Management & Service Discovery

Store service configuration data as values associated with keys

- Configuration changes go through consensus for consistent replication
- Clients watch the keys they care about for modification events

Discovery of new servers/clients

- A service can register its address with a name & unique prefix
  - Example: `/services/payments/instance-7`
  - Other systems can look for it or get alerts when the parent directory changes
  - The key will be deleted (or lose its lease) when the service dies

Fencing Tokens

What happens when a leader is paused (e.g., heavy system load)?

It loses its lease and another leader takes over … but may try to resume

- Fencing token
  - Monotonically increasing number associated with each lock grant
  - The coordination service increments the number each time a lock is granted
  - The resource (e.g., database) rejects any request with a token lower than the highest number it has seen
  - When the old leader wakes up, it uses a stale token and its writes are rejected

Similar to Raft's use of epochs

The End