Distributed Computing

A-04. Apache Zookeper

Apache Zookeeper

- Open source project
  - https://zookeeper.apache.org/
- Originally developed at Yahoo!
- Paper presented at the USENIX ATC 2010 conference

A Coordination Service

- Group Membership
  - Add/remove workers/machines
- Leader election
- Dynamic Configuration
- Status Monitoring
- Queueing, barriers, critical sections & locks…

How ZooKeeper Is Used

Design Goals

- Multiple outstanding requests
- Read-intensive workload
  - For every *write*, 100s-1000s of *reads*
- General
- Reliable
- Easy to use

Building Blocks

1) Builds on top of a (Raft-like) consensus algorithm
2) Wait-Free Architecture
3) Ordering
4) Change Events

Wait-Free

- No locks or other primitives that **stop** a server
- No blocking in the implementation
- Simplifies the implementation
- No deadlocks
- Needs an alternative solution to wait for conditions

Ordering

- Writes are **linearizable** (i.e., they are executed in the order they are performed)
- Reads are **serializable** (i.e., you might read stale data)
  - Weaker than linearizable, it's ACID's isolation
- All operations on the same client will be serialized in FIFO order

Change Events

- Clients can **request to be notified** of changes
- When a change happens, the client gets notified
- They get notification of a change before seeing the result

Inspiration

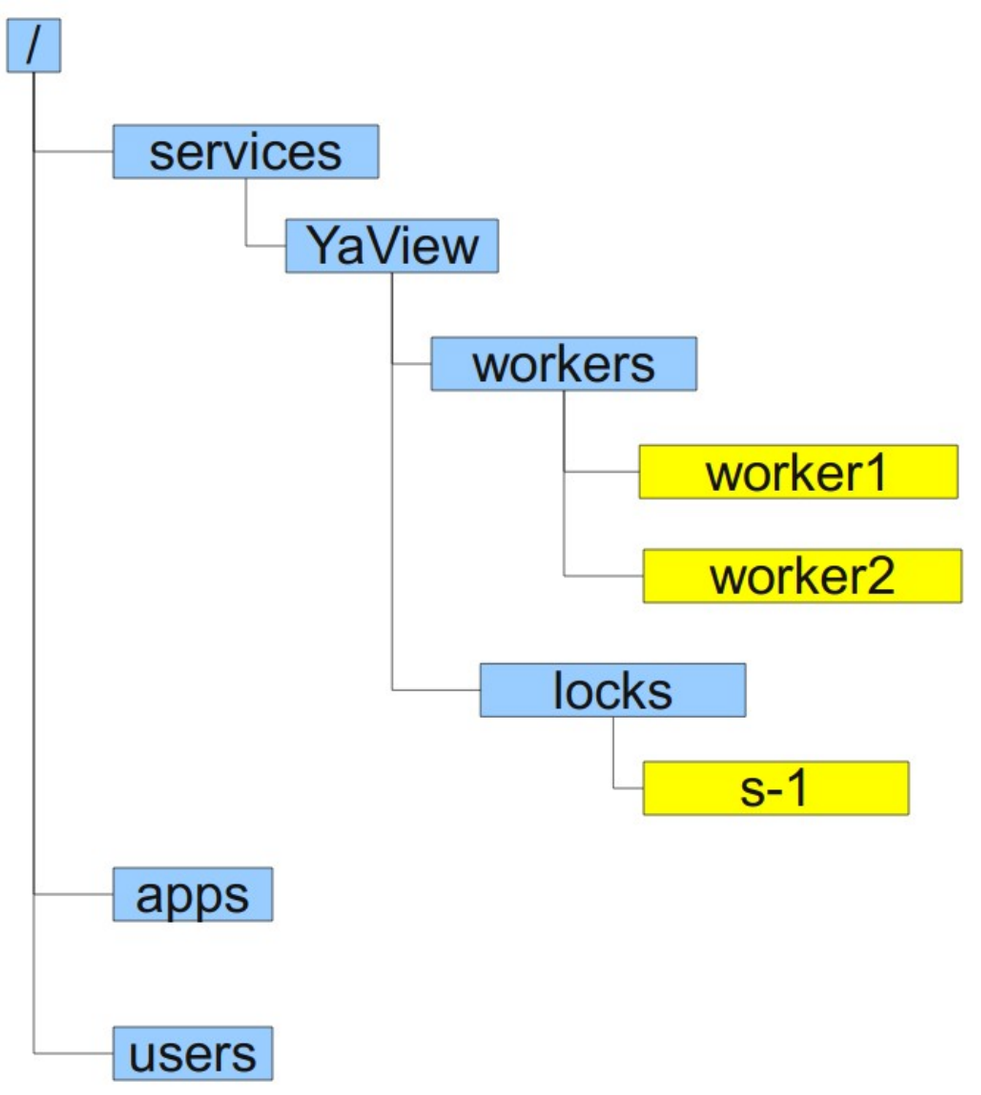
- A **distributed filesystem**
  - Zookeper designers have seen that their engineers were using the NFS filesystem to coordinate their applications
  - It “almost works”--when it fails, it’s ugly though :)
- Works like a filesystem for small pieces of data
  - Plus notification of changes
  - Minus partial read/writes

API

- `create(path, data, acl, flags)`
- `delete(path, expectedVersion)`
- `setData(path, data, expectedVersion)`
- `getData(path, watch)`
- `exists(path, watch)`
- `getChildren(path, watch)`
- `void sync()`
- `setACL(path, acl, expectedVersion)`
- `getACL(path)`

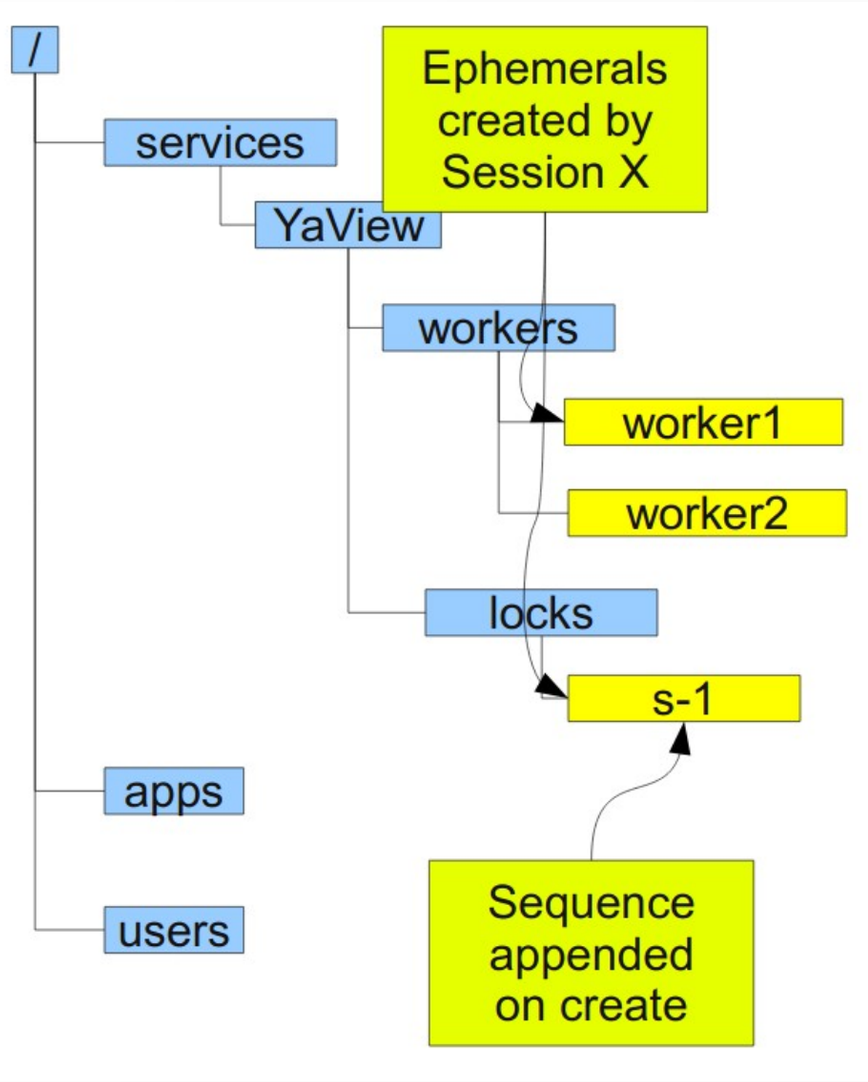
Data Model

- Hierarchical namespace
- “Znodes” are like both files and directories: they have data and children
- Data is read and written in its entirety



Create Flags

- **Ephemeral**: znode is deleted when creator fails
- **Sequence**: append a monotonically-increasing counter



Configuration

- A worker starts and gets the configuration
  - `GetData("/myApp/config", watch=True)`
- Admins change the configuration
  - `setData("/myApp/config", newConf, expectedVersion=-1)`
- Workers get notified of the change and re-run `getData`

Group Membership

- A worker starts and gets registers itself in the group
  - `create("/myApp/workers/" + my_name, my_info, ephemeral=True)`
- List members
  - `getChildren("myApp/workers", watch=True)`

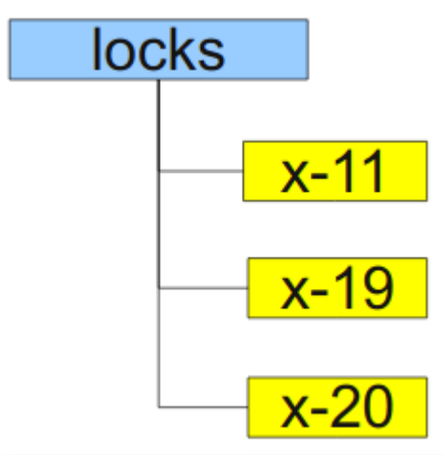
```
worker1
worker2
```

Leader Election

- Check who's the current leader
  - `getData("/myApp/workers/leader", watch=True)`
- If successful, follow the leader
- Else, propose yourself as candidate
  - `create("/myApp/workers/leader", my_name, ephemeral=True)`
  - If successful, you're the leader; otherwise restart
- Since workers are watching changes for the leader, they'll know if the leader fails or changes

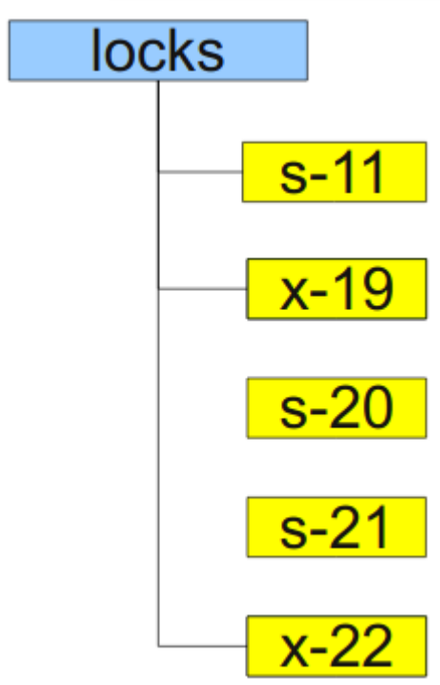
(Exclusive) Locks

- `id = create("myApp/locks/x-", sequence=True, ephemeral=True)`
- `getChildren("myApp/locks/")`
- If `id` is the first child, lock is mine
- Otherwise:
  - `if exists(`last child before `id, watch=True)`
    - Wait for the event (when the previous owner releases the lock)
  - Else, return to `getChildren`
- Note that every node only watches the one in their front in the queue :)



Shared Locks

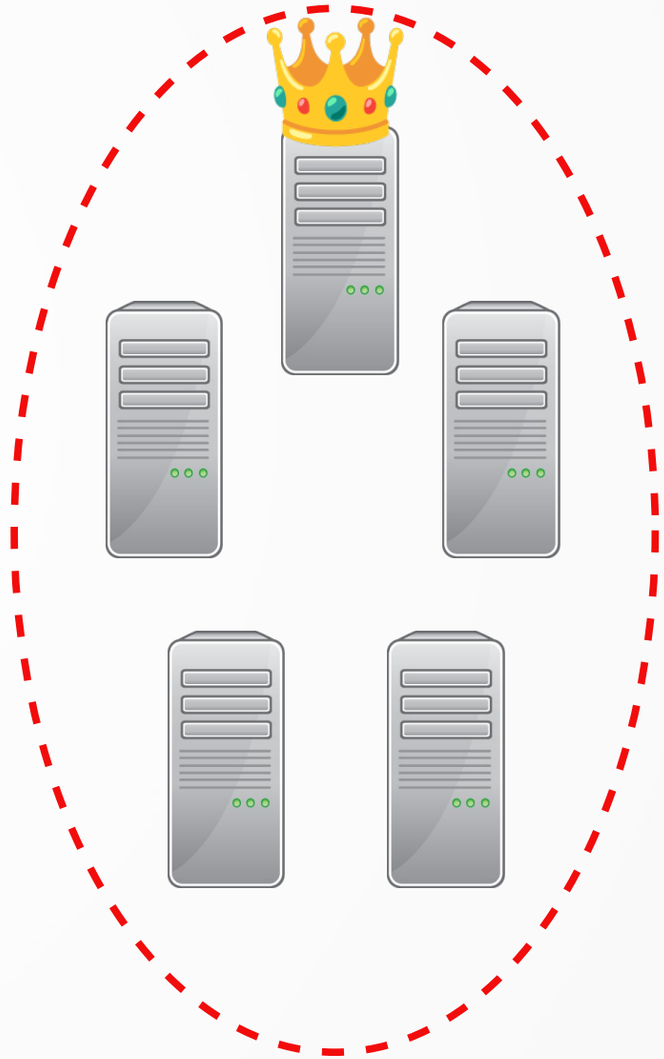
- `id = create("myApp/locks/s-", sequence=True, ephemeral=True)`
- `getChildren("myApp/locks/")`
- If no exclusive "x-" lock before `id`, go ahead
- Otherwise:
  - `if exists(last "x-" before id, watch=True)`
    - Wait for the event
  - Else, return to getChildren



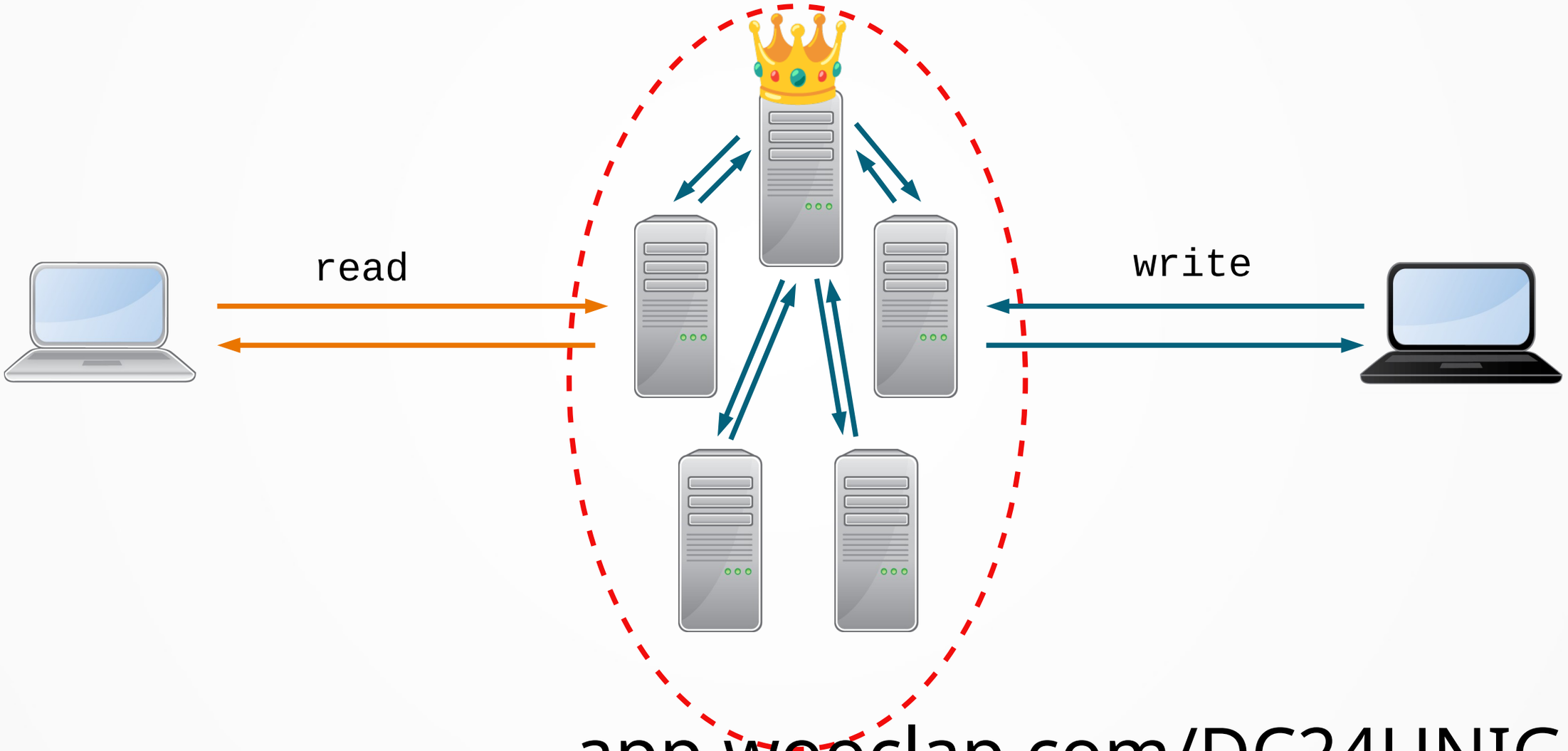
How Zookeeper Is Implemented

Architecture

- All servers have a full copy of the state in memory
- Uses a consensus protocol that's similar to Raft
  - Actually, developed before it
- There's a leader
- Update is committed when a majority of servers saved the change
  - As we're used to, we need *2m+1* servers to tolerate *m* failures



Reads and Writes



app.wooclap.com/DC24UNIGE

Operations Per Second