Cloud
Map Reduce
Map Reduce
Hadoop
Hadoop
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
User Features

- Manage **configuration** across nodes.
- Implement reliable **messaging**.
- Implement **redundant** services.
- **Synchronize** process execution.
Architecture

▪ **SetUp**
  ▪ Data Center (Clusters of Computers)
  ▪ Each computer runs **ZooKeeper**.
  ▪ Communication relies on TCP/IP.
  ▪ Initial “wiring” by system admin.

▪ **Joining**
  ▪ Connection to the network
    ▪ Sending “I’m Alive!”
Architecture

- File System Organization
  - Structure
  - Node Double Role
  - Storage Size & Location
Architecture

- **ZNode**
  - Atomicity
  - **Access Control List (ACL)**
  - Type
    - Ephemeral / Consistent
    - Sequential / Not Sequential
Architecture

- ZNode
  - Watches & Listeners
Engineering Guarantees

- Sequential Consistency
  - Of one client
  - Among clients
- Atomicity
- Single System Image
- Reliability
- Timeliness

(CAP)
Implementation in Depth

- Data on RAM, Serialized and logged on disk.
- Client **connects to one** server and **served by all**.
- Leader & Followers
- Leader Election
- Consensus
Leader & Followers

- **Leader**
  - Centralized point of update
  - Monarch

- **Follower**
  - Non leader Server.
  - Logically, connected to the leader.
  - Physically connected in a mesh.
Leader Election – Communication

- All Server connected with a **full mesh**
  - At least physically, over a TCP/IP Network.

- Upon awakens, each Server assumes **he is the leader**
  - And broadcasts that to everybody

- Upon acceptance of a lower Server, **forfeit the leadership.**

- The last Server standing is the new Leader!
Leader Election – Algorithm

- Assuming there are $N$ ZNodes in a cluster.
- Performed by each ZNode:
  - Create a Sequential ephemeral Znode with the same path.
    - For example: /app/leader_election/myNo_
    - ZooKeeper will append a 10-digit seq. no. to the path (sequential).
    - So it will be: /app/leader_election/myNo_000000000i
  - (For a given instance) the lowest ZNode becomes the leader.
    - The rest automatically become followers.
  - Create a watch to the 000000000[i – 1] ZNode.
    - If the leader fails, the “next-to-crown” will become the leader.
Consensus

- Every **write** is carried by the leader iff the majority (Quorum) accepts it.
  - Write fails if such majority isn’t reached.

The Algorithm:

- **Follower** $x \rightarrow \text{Leader}$: Write($p,v$)
- **Leader** $\rightarrow$ **Followers**: Get Ready!
- **Each Follower** $i \rightarrow \text{Leader}$: Promise, I’m Ready!
- if $(\#(\text{Promises}) > \frac{N}{2})$:
  - **Leader** $\rightarrow$ **Followers**: Accept: $p \rightarrow v$
  - **Each Follower** $i \rightarrow \text{Leader}$: Accepted! (Ack)
  - **Leader** $\rightarrow$ **Followers**: Commit!
- else:
  - Restart Procedure
Consensus
Consensus
Consensus

Leader

Followers

Promise!
Promise!
Promise!
Promise!
Promise!
Promise!
Promise!
Consensus
Consensus
Consensus
Consensus

- Every **read** is carried if **the majority (Quorum)** accepts it.
  - Write fails if such majority isn’t reached.
- Due to the write procedure:

```
<table>
<thead>
<tr>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
</table>
```

At least one node holds the newest data.
Low-Level API

- Create
- Delete
- Exists
- Get data
- Set data
- Get children
- Sync
Live Demo
Performance

- Better when \((Reads) \gg (Writes)\)
Reliability

- Rapid Failure & Recovery
- Leader Election
Questions?