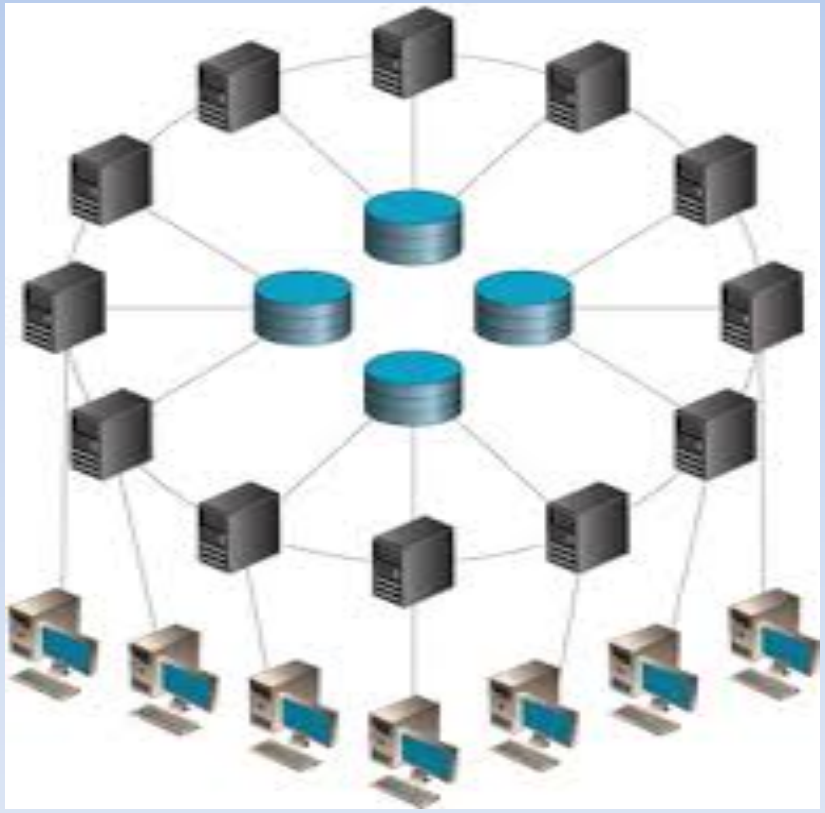
Apache Zookeeper

http://zookeeper.apache.org

What is a Distributed System?

A distributed system consists of multiple computers that communicate through a computer network and interact with each other to achieve a common goal.



Automated Coordination

Coordination in a Distributed System

- Coordination: An act that multiple nodes must perform together.
- Examples:
    - Group membership
    - Locking
    - Leader Election
    - Synchronization.
    - Publisher/Subscriber

Getting node coordination correct is very hard!

Introducing ZooKeeper

ZooKeeper is a Distributed Coordination Service for Distributed Applications. ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers.

- ZooKeeper Wiki

What is ZooKeeper ?

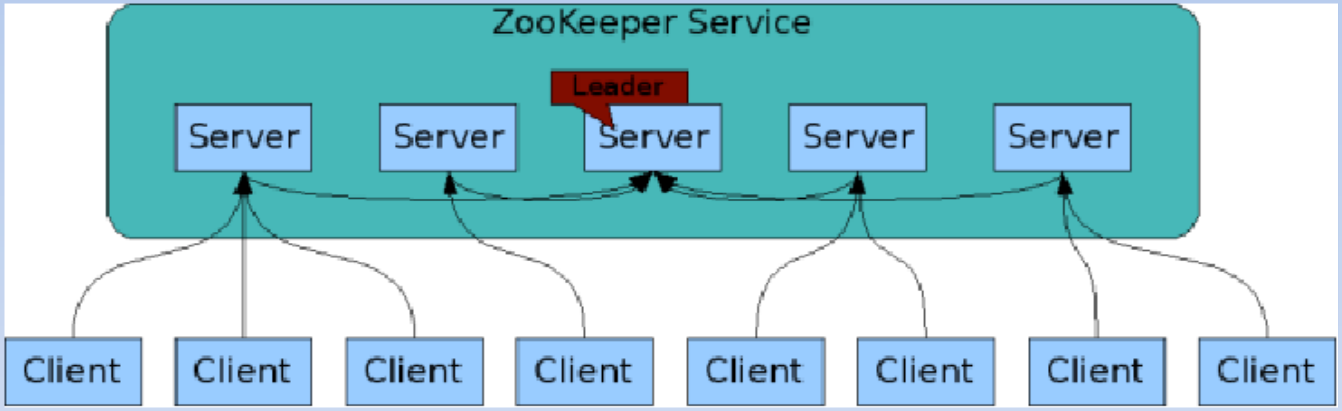
- An open source, high-performance coordination service for distributed applications.
- Exposes common services in simple interface:
  - naming
  - configuration management
  - locks & synchronization
  - group services

  … *developers don't have to write them from scratch*
- Build according to your requirement for specific needs.

ZooKeeper Use Cases

- Configuration Management
  - Cluster member nodes bootstrapping configuration from a centralized source in unattended way
  - Easier, simpler deployment/provisioning
- Distributed Cluster Management
  - Node join / leave
  - Node status in real time
- Naming service – e.g. DNS
- Distributed synchronization - locks, barriers, queues.
- Leader election in a distributed system.
- Centralized and highly reliable (simple) data registry.

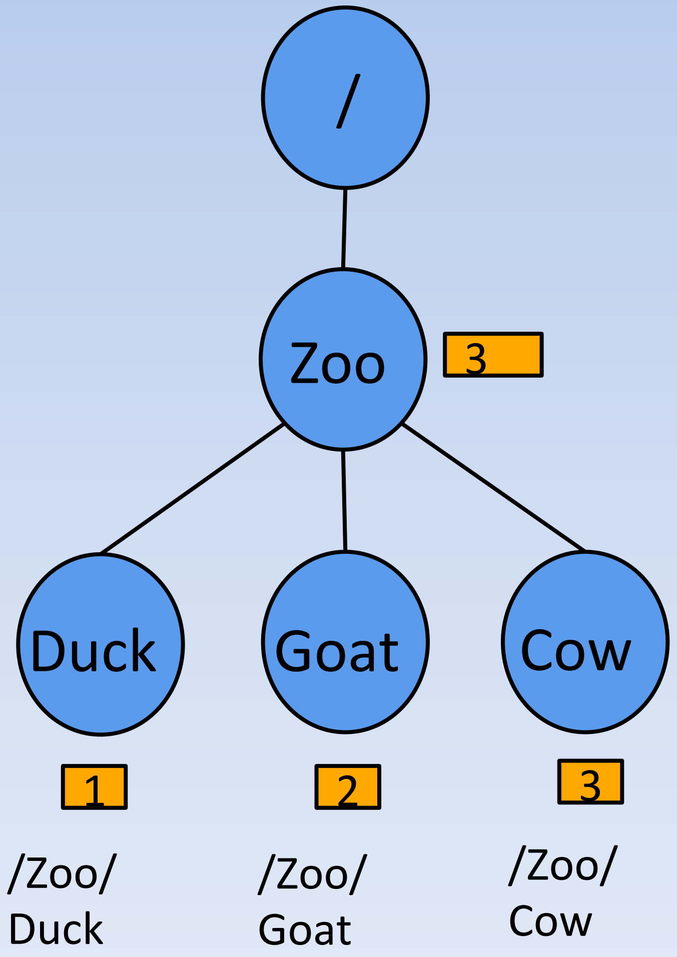
The ZooKeeper Service



- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data (in memory)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server, writes go through the leader & needs majority consensus.

The ZooKeeper Data Model

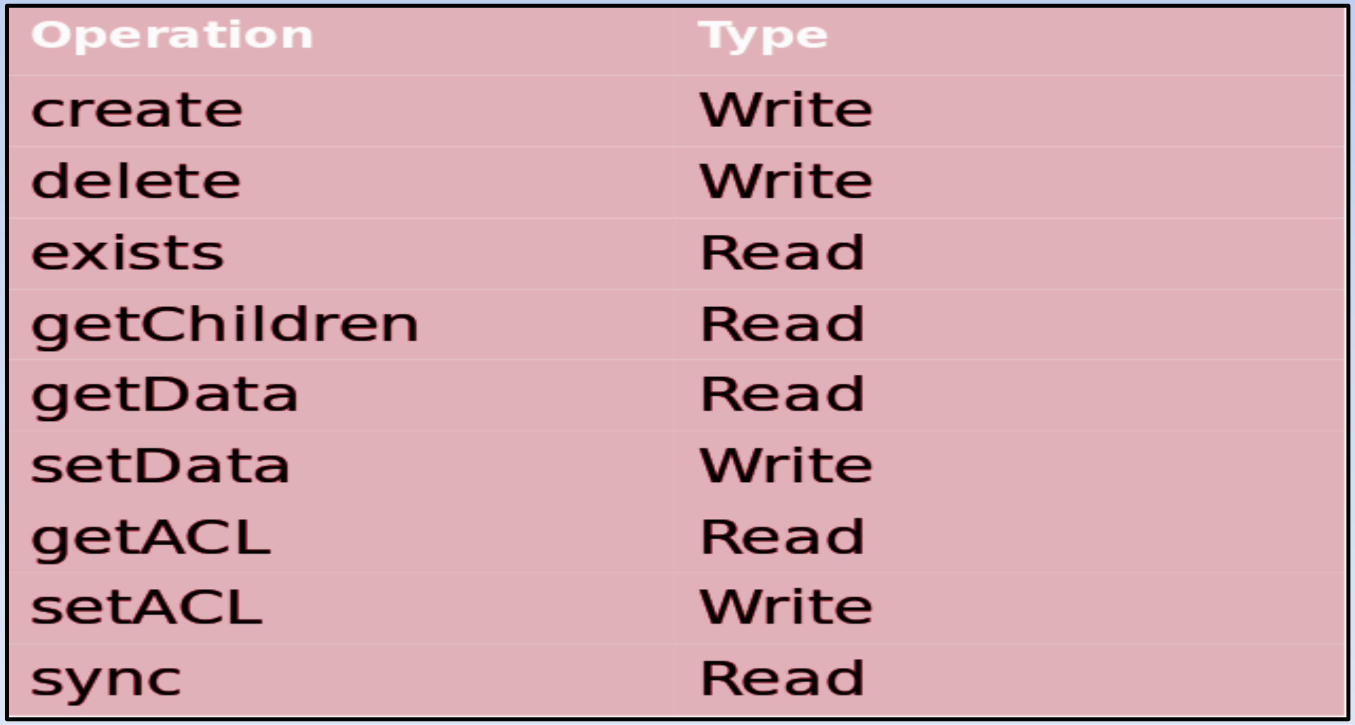
- ZooKeeper has a hierarchical name space.
- Each node in the namespace is called as a ZNode.
- Every ZNode has data (given as byte[]) and can optionally have children.
- ZNode paths:
  - canonical, absolute, slash-separated
  - no relative references.
  - names can have Unicode characters
- ZNodes
- Maintain a stat structure with version numbers for data changes, ACL changes and timestamps.
- Version numbers increases with changes
- Data is read and written in its entirety



ZNode Types

- Persistent Nodes
    - exists till explicitly deleted
- Ephemeral Nodes
    - exists as long as the session is active
    - can’t have children
- Sequence Nodes (Unique Naming)
    - append a monotonically increasing counter to the end of path
    - applies to both persistent & ephemeral nodes

ZNode Operations



Operation	Type
create	Write
delete	Write
exists	Read
getChildren	Read
getData	Read
setData	Write
getACL	Read
setACL	Write
sync	Read

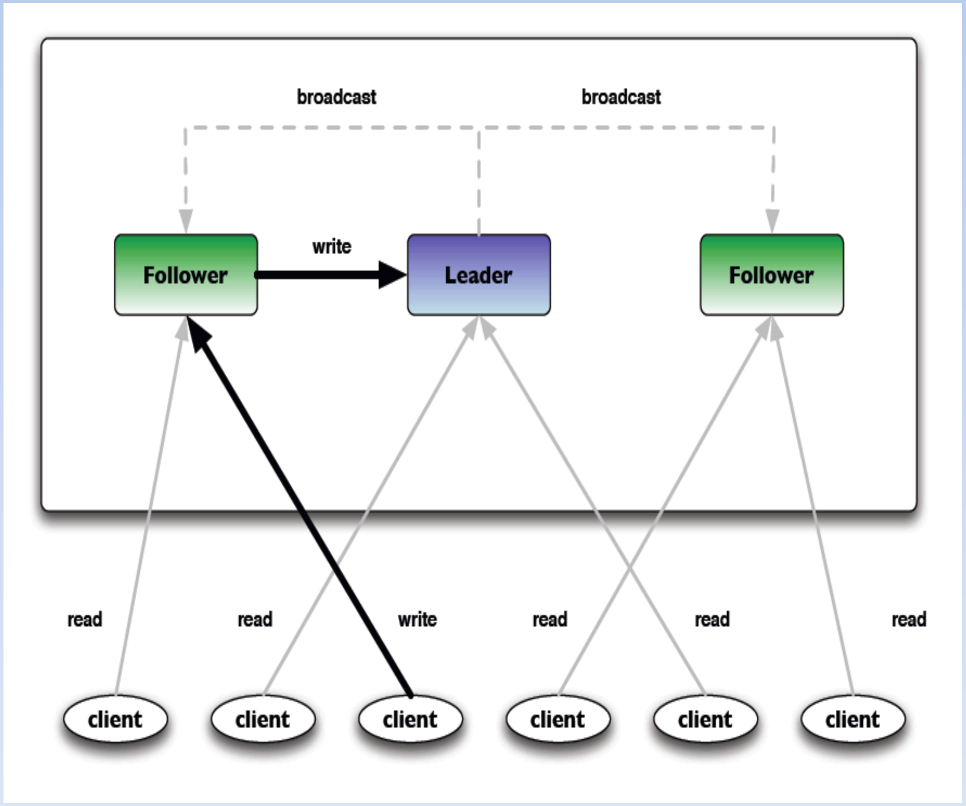
All these operations can be sync as well as async

ZNode Watches

- Clients can set watches on znodes:
  - NodeChildrenChanged
  - NodeCreated
  - NodeDataChanged
  - NodeDeleted
- Changes to a znode trigger the watch and ZooKeeper sends the client a notification.
- Watches are one time triggers.
- Watches are always ordered.
- Client sees watched event before new znode data.
- Client should handle cases of latency between getting the event and sending a new request to get a watch.

ZNode Reads & Writes

- Read requests are processed locally at the ZooKeeper server to which the client is currently connected
- Write requests are forwarded to the leader and go through majority consensus before a response is generated.

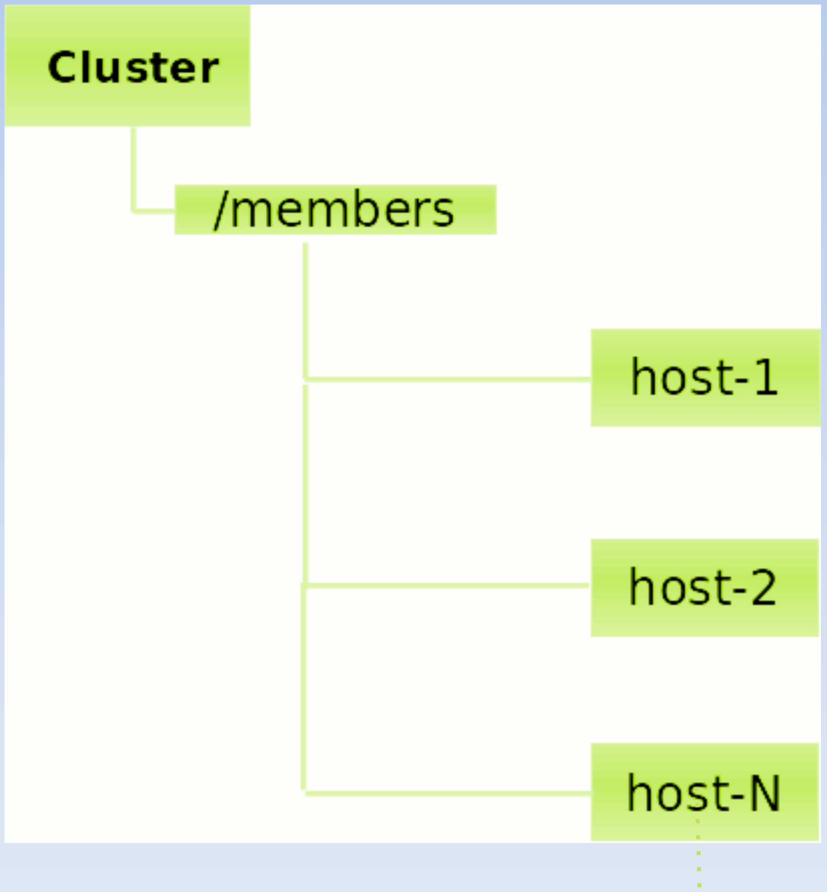


Consistency Guarantees

- **Sequential Consistency:** Updates are applied in order
- **Atomicity:** Updates either succeed or fail
- **Single System Image:** A client sees the same view of the service regardless of the ZK server it connects to.
- **Reliability:** Updates persists once applied, till overwritten by some clients.
- **Timeliness:** The clients' view of the system is guaranteed to be up-to-date within a certain time bound. (Eventual Consistency)

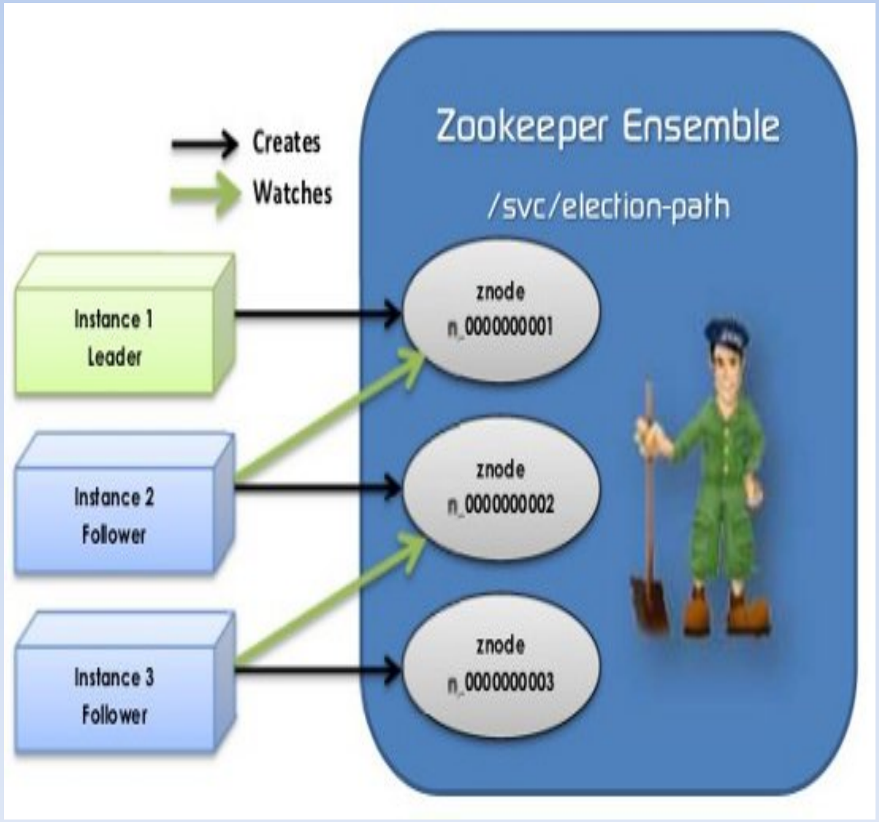
Example #1: Cluster Management

- Each Client Host i, i:=1 .. N
- Watch on /members
- Create /members/host-${i} as ephemeral nodes
- Node Join/Leave generates alert
- Keep updating /members/host-${i} periodically for node status changes
- (load, memory, CPU etc.)



Example #2: Leader Election

- A znode, say “/svc/election-path"
- All participants of the election process create an ephemeral-sequential node on the same election path.
- The node with the smallest sequence number is the leader.
- Each “follower” node listens to the node with the next lower seq. number
- Upon leader removal go to
- election-path and find a new leader,
- or become the leader if it has the lowest sequence number.
- Upon session expiration check the election state and go to election if needed



Recipe #3: Distributed Exclusive Lock

Assuming there are N clients trying to acquire a lock

- Clients creates an ephemeral, sequential znode under the path /Cluster/_locknode_
- Clients requests a list of children for the lock znode (i.e. _locknode_)
- ***The client with the least ID according to natural ordering will hold the lock.***
- Other clients sets watches on the znode with id immediately preceding its own id
- Periodically checks for the lock in case of notification.
- The client wishing to release a lock deletes the node, which triggering the next client in line to acquire the lock.

```
ZK
|---Cluster
    +---config
    +---memberships
    +---_locknode_
        +---host1-3278451
        +---host2-3278452
        +---host3-3278453
        +--- …
        \---hostN-3278XXX
```

ZooKeeper In Action @Twitter

- Used within Twitter for service discovery
- How?
- Services register themselves in ZooKeeper
- Clients query the production cluster for service “A” in data center “XYZ”
- An up-to-date host list for each service is maintained
- Whenever new capacity is added the client will automatically be aware
- Also, enables load balancing across all servers.

Reference: http://engineering.twitter.com/

A few points to remember

- Watches are one time triggers
- Continuous watching on znodes requires reset of watches after every events / triggers
- Too many watches on a single znode creates the “herd effect” - causing bursts of traffic and limiting scalability
- If a znode changes multiple times between getting the event and setting the watch again, carefully handle it!
- Keep session time-outs long enough to handle long garbage-collection pauses in applications.
- Dedicated disk for ZooKeeper transaction log

Reference

- Zookeeper site: http://zookeeper.apache.org/
- Slides:
  - http://www.slideshare.net/sauravhaloi/introduction-to-apache-zookeeper