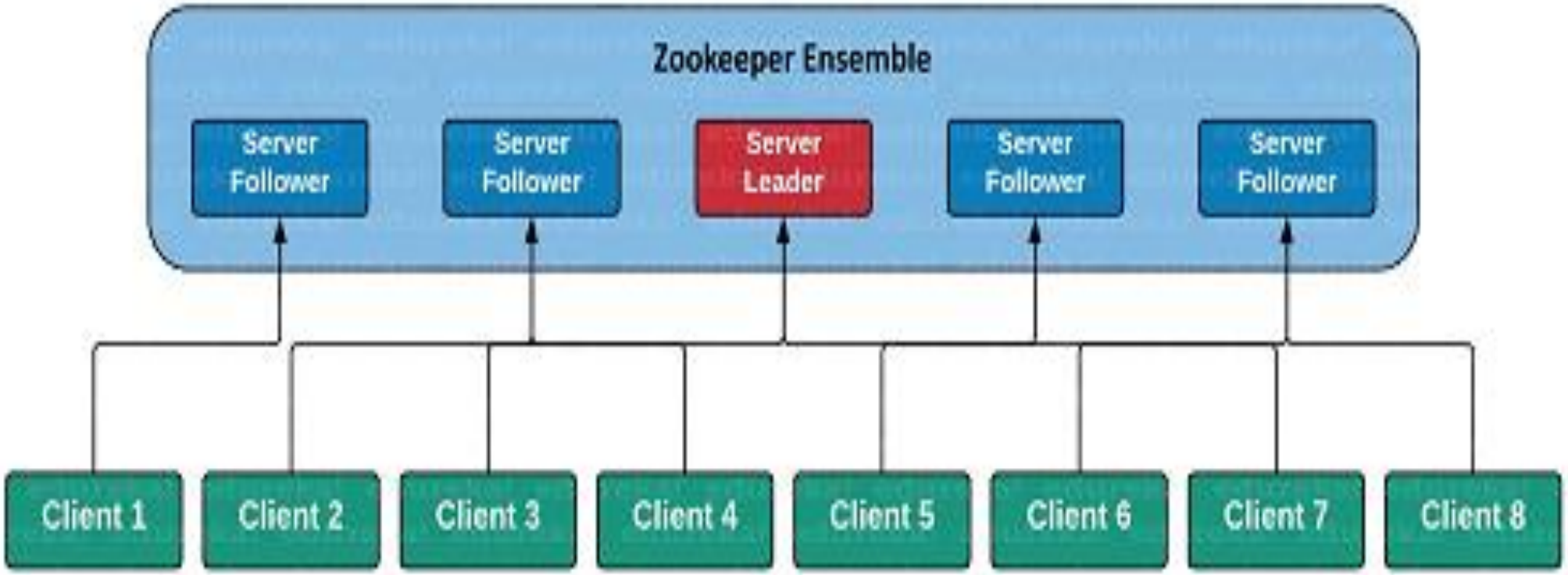
Zookeeper: Most of it

ZooKeeper: Why?

- In a distributed system:
  - Servers can crash.
  - Servers can come and go.
  - They must agree on shared information.
  - They must avoid conflicts (e.g., two servers acting as “leader”).
  - They must know who is alive.
- Without a coordinator, they will fight, get confused, or corrupt data
- ZooKeeper ensures everyone behaves properly :)

ZooKeeper

- ZooKeeper is a small, super-reliable “manager” for distributed systems.
- ZooKeeper is like a class monitor who:
  - Keeps/retains common settings
  - Tells who is alive and dead
  - Decides who will be the leader
  - Coordinates who should do what
  - Sends alerts when something changes.


Zookeeper Ensemble
Server Follower
Server Follower
Server Leader
Server Follower
Server Follower
Client 1
Client 2
Client 3
Client 4
Client 5
Client 6
Client 7
Client 8

Data Flair

ZooKeeper: Core Idea

- Run a small ensemble (3–7) of ZooKeeper servers that replicate a tiny hierarchical data store (znodes)
- Clients read from any server and perform coordinated updates via a leader.
- Leader orders writes with the Zab consensus protocol, giving you consistent configuration, watches, ephemeral and sequential nodes, and primitives.
- You can build locks and leader election on the primitives

Leader

- Only one leader at a time.
- Handles all write requests (create, delete, setData, setACL, multi)
- Coordinates replication of writes to followers.
- Runs leader election algorithms.

Followers

- Usually multiple followers.
- Handle read requests directly.
- Forward write requests to the leader.
- Participate in voting for leader election.

Observers

- Optional servers (you may have 0 or many).
- Handle read requests, just like followers.
- Do NOT vote in leader election.
- Do NOT participate in write quorum.
- Useful for scaling reads without affecting write performance – increase scalability of read operations w/o affecting any writes

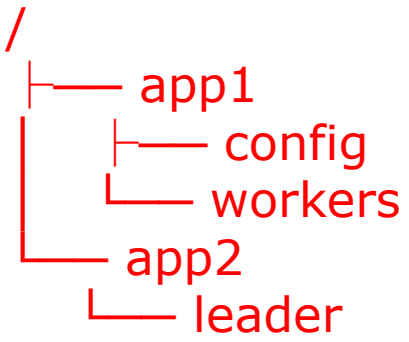
Observers

- Increase scalability of read operations w/o affecting any writes

- Reads can come from any node (leader or follower) - Writes must be approved by a majority (quorum) of servers.

If you keep adding MORE servers to increase read capacity, the quorum size becomes larger:
More servers → more majority → writes become slower because the leader must wait for ack from more followers.

Observers are special ZooKeeper servers that: receive updates from the leader, keep a copy of the data, but do not vote in write quorum – so they do not slow down writes

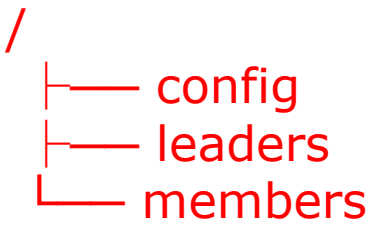


Znode

- A ZNode is a node in ZooKeeper’s data tree.
- Each ZNode:
  - Stores data (small amounts, usually < 1 MB)
  - Has metadata (version, ACL, timestamps)
  - Can have children (like directories)

ZooKeeper: How?

- ZooKeeper keeps a very small tree structure, like folders
- Each item in the tree is called a znode.
- A znode is like a tiny file that stores a few bytes of information.
- Like tiny “notes” for the whole system.



ZooKeeper: How?

- A node (znode) is a path in ZooKeeper where:
- You can store small data (like a config string)
- You can create children (like directories)
- You can set watches, and
- ZooKeeper keeps it consistent across the cluster.

Znode examples

- Store app configuration in ZooKeeper
- A microservice starts and registers its presence
- Application tries to acquire a lock.
- A process participates in leader election.
- Prepare the namespace structure before running the app.

Special Types of znodes

- Persistent Node
  - Stays forever until explicitly deleted.
  - It does NOT disappear when:
    - The client that created it disconnects
    - The session that created it expires
    - The server restarts (assuming the ensemble is running normally)
  - It stays in ZooKeeper **forever** unless deleted manually or programmatically.

/configs/database

```
zk.create("/myNode", "data".getBytes(),
        ZooDefs.Ids.OPEN_ACL_UNSAFE,
        CreateMode.PERSISTENT);
```

Special Types of znodes

- Ephemeral Node
  - Created by a client
  - Exists only as long as that client's session is alive
  - Disappears automatically when that server dies -
  - Used to know which servers are alive
  - Cannot have children
  - If server-1 crashes, the node auto-deletes → everyone knows it died.

/members/server-1 (ephemeral)

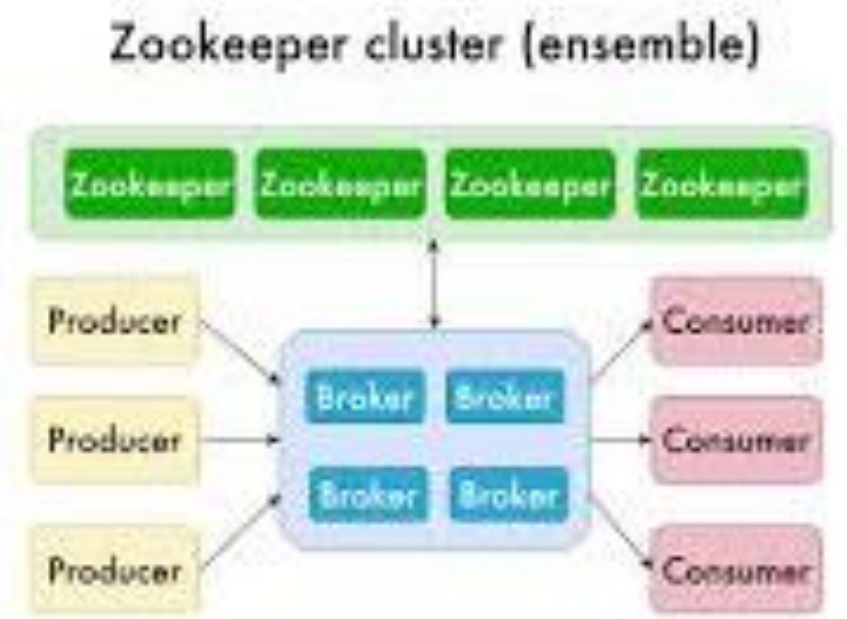
Special Types of znodes

- Sequential Node
  - ZooKeeper adds a number at the end
  - Used for ordering — especially in leader election and locks.

/election/candidate-00001
/election/candidate-00002

ZooKeeper: Key Components

- Ensemble — a group of ZooKeeper servers - availability requires a majority (quorum)
- The group of ZooKeeper servers that work together to form a single, reliable, fault-tolerant ZooKeeper service.
- Think of it as the cluster.



ZooKeeper: Key Components

- Leader / Followers / Observers: one of the 3 possible rows played by each server
- The Leader is the only server that:
  - Processes all write requests (create, delete, set data)
  - Assigns transaction IDs (zxids)
  - Proposes updates to all followers
  - Runs ZAB protocol
- Leader is chosen via the ZooKeeper leader election process when the ensemble starts (or when the leader fails).

ZooKeeper: Key Components

- Leader / Followers / Observers: one of the 3 possible rows played by each server
- Followers are normal voting ZooKeeper servers.
- Handle read requests directly
- Forward write requests to the leader
- Vote on leader proposals (ack)
- Participate in leader election
- Maintain copies of the data tree
- Followers and Leader together form the **quorum** (majority group).
- In 5 node ensemble, if Leader + 2 followers, then ZK operates.

ZooKeeper: Key Components

- Leader / Followers / Observers: one of the 3 possible rows played by each server
- Observers are **non-voting ZooKeeper servers**.
- Handle read requests
- Forward write requests to the leader
- Do not vote in write agreement
- Do not participate in leader election
- Do not count toward quorum
- Purpose: To scale read-heavy workloads without making the ensemble slower.

ZooKeeper: Key Components

- Znodes: A **znode** is a **node (entry)** in ZooKeeper's hierarchical data tree.
- Store data (up to ~1 MB, but usually a few KB)
- Have children (except ephemeral nodes)
- Be watched by clients
- Be persistent or ephemeral
- Be sequential or not
- Every znode is identified by a **path** like /app/config/version.

ZooKeeper: Key Components

- Znodes can store:
- Small pieces of data (e.g., "server=active")
- Metadata for distributed systems
- Information for leader election
- Configuration values
- Service discovery information

/brokers/ids/1 (znode for broker 1)

/config/topics/myTopic (topic config)

ZooKeeper: Key Components

- Ephemeral nodes — automatically removed when a client session ends; used for membership & leader election.

- Sequential nodes — ZooKeeper appends a monotonically increasing sequence number to the node name (useful for ordered locks/election).

ZooKeeper: Key Components

- Watches — lightweight notification mechanism that let a client know when something changes in the data tree.
- They are one of the most important features of ZooKeeper.
- A watch is a *one-time trigger* that a client sets on a znode.
- When that znode changes, ZooKeeper sends a notification to the client.
- Watches allow clients to avoid constant polling.

ZooKeeper: Key Components

- Watches allow clients to avoid constant polling.
- Instead of asking repeatedly:
  - "Has the leader changed?"
  - "Has this node been created?"
  - "Has the child list changed?“
- A client places **one watch**, and ZooKeeper notifies it when something happens.

ZooKeeper: Key Components

- A watch can be triggered by a:
- **Node data change**
- **Node deletion**
- **Node creation**
- **Changes in children list**

set /config "newValue"

delete /leader

exists /election/node-0001 watch create /election/node-0001 ""

ZooKeeper: Key Components

- Zab (ZooKeeper Atomic Broadcast) — the replication/consensus protocol (leader-based total order broadcast optimized for primary/backup crash-recovery).

How Leader Election Works

- Every server writes an ephemeral-sequential node:
  - /election/server-00005
- The one with the smallest number becomes leader.
- If leader dies, its ephemeral node disappears → the next smallest number becomes leader.
- So: No fights, no confusion - ZooKeeper ensures fairness.

Group Membership: Who is alive?

- Each server creates an ephemeral node under /members.
- ZooKeeper keeps the list automatically:

```
/members
   ├── server1
   ├── server2
   └── server3
```

- If a server crashes → its node disappears → all other servers immediately see the updated list.
- This gives instant cluster membership.

Watches (Notifications)

- Clients can tell ZooKeeper: “Please tell me if this node changes.”
- This is called a watch.
- When the node changes, ZooKeeper sends a notification.
- Usage:
  - config change alert
  - membership change
  - leader change

Zookeeper Servers: How?

- ZooKeeper does NOT run as one server: It runs as a small cluster, called an ensemble (usually 3 or 5 servers).
- Why?
  - If 1 server fails, others continue.
  - They vote and agree on updates.
  - It gives very strong reliability.
- ZooKeeper needs a majority to work (called quorum).
- Example:
  - 3 servers → any 2 alive is enough
  - 5 servers → any 3 alive is enough

Internal Information

- ZooKeeper keeps all data in memory for fast reads.
- When you write something:
  - Write goes to the leader.
  - Leader broadcasts it to followers.
  - Majority confirms.
  - Leader commits the change.
- This ensures the same order of changes everywhere.

Guarantees

- Total order of updates (a global ordering of state changes).
- Atomicity of updates (multi ops succeed or fail entirely).
- Single system image — clients see an ensemble that behaves like a single datastore.
- Reliability — once a write is acknowledged by quorum it will persist despite failures.
- Simple timeliness / session semantics — ephemeral nodes are tied to client sessions; session timeout determines liveness.

Common Use cases

- Leader election — each contender creates an ephemeral sequential znode under /election; the lowest sequence becomes leader. If it dies, next lowest wins.
- Group membership / service discovery — services create ephemeral znodes (e.g., /members/node-00001) so consumers can list current members.
- Distributed locks — create ephemeral-sequential nodes under /locks; clients watch predecessor to acquire lock in order.
- Configuration store — store small configuration values under znodes; clients watch for changes to react dynamically.
- Barrier / queue primitives — implement barriers, work queues using znodes and watches.

Typical API

- create(path, data, acl, flags) — flags: EPHEMERAL, SEQUENTIAL.
- delete(path, version)
- getData(path) / setData(path, data, version)
- exists(path, watch)
- getChildren(path, watch)
- multi(ops) — atomic transaction of multiple
- ops.sync(path) — ensure read is up-to-date with leader.

ZooKeeper is..

- Not a database (limited data size).
- Not for big data.
- Not for large files.
- Not for high-write workloads.
- It is ONLY for tiny metadata and coordination.

Dependency on Zookeeper

- Apache Kafka (older versions)
- Hbase
- Hadoop YARN Resource Manager
- Solr
- Storm
- Flink (older versions)

Observations

1) Distributed systems always need some form of coordination
2) Programmers cannot use locks correctly
3) Message based coordination can be hard to use in some applications

Wishes

1) Simple, Robust, Good Performance
2) Tuned for Read dominant workloads
3) Familiar models and interfaces
4) Wait-Free: A slow/failed client will not interfere with the requests of a fast client
5) Need to be able to wait efficiently

Design Starting Point

Start with the File API and model strip out what we don't need:

1)Partial writes/reads (takes with it open/close/seek)

2)Rename

add what we do need:

1)Ordered updates and strong persistence guarantees

2)Conditional updates

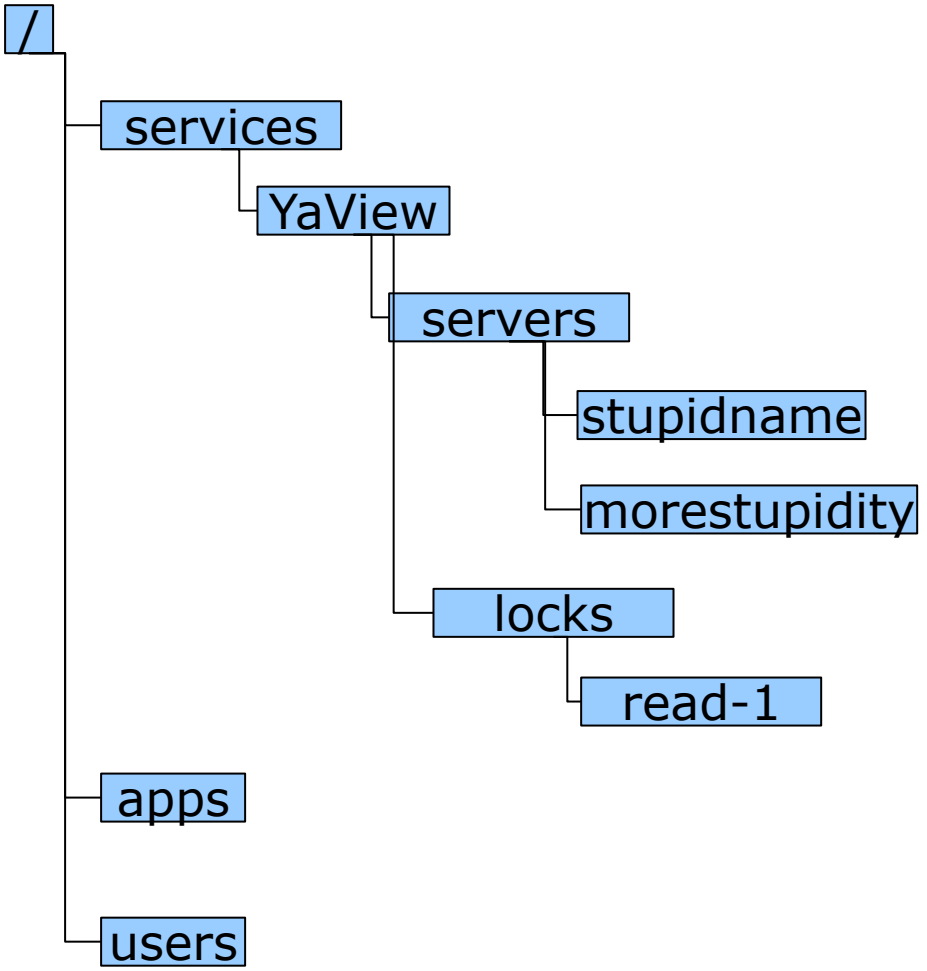
3)Watches for data changes

4)Ephemeral nodes

5)Generated file names

Data Model

1) Hierarchal namespace (like a file system)
2) Each znode has data and children
3) data is read and written in its entirety



ZooKeeper API

String create(path, data, acl, flags)

void delete(path, expectedVersion)

Stat setData(path, data, expectedVersion)

(data, Stat) getData(path, watch)

Stat exists(path, watch)

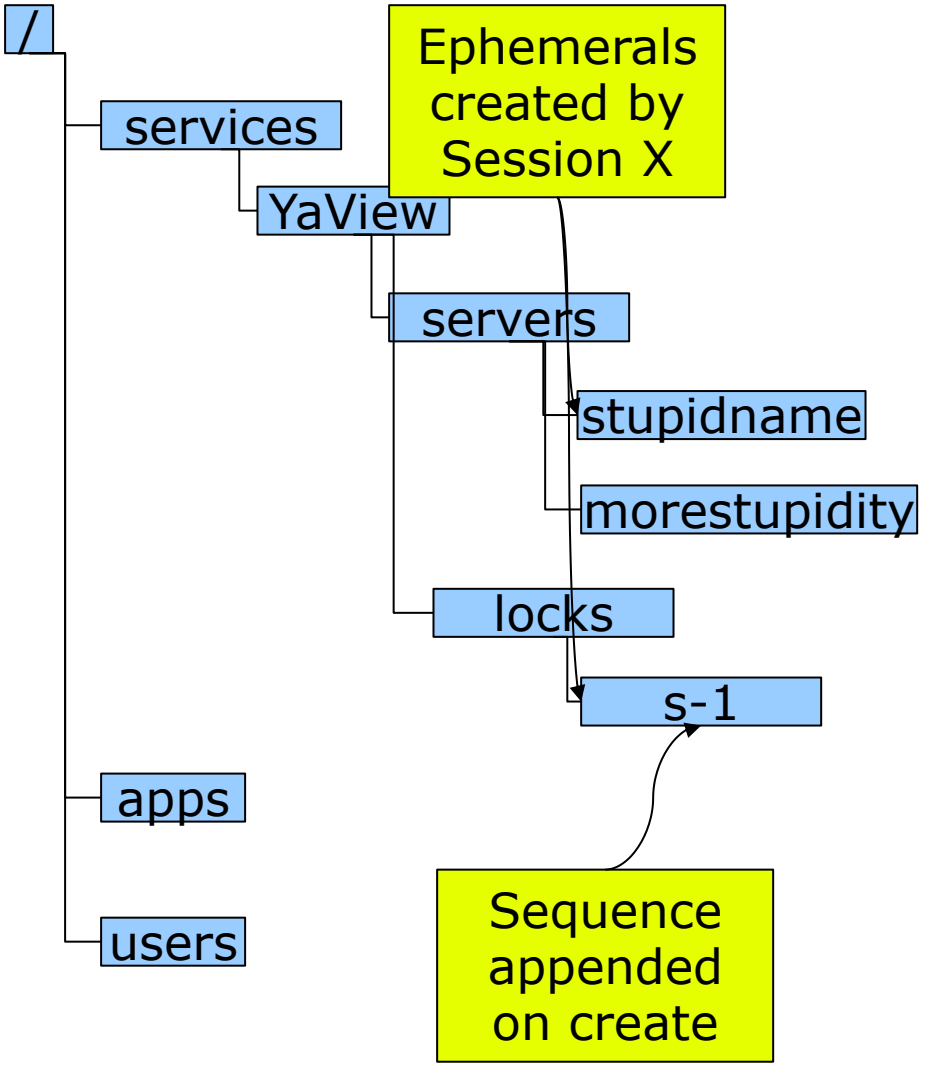
String[] getChildren(path, watch)

void sync(path)

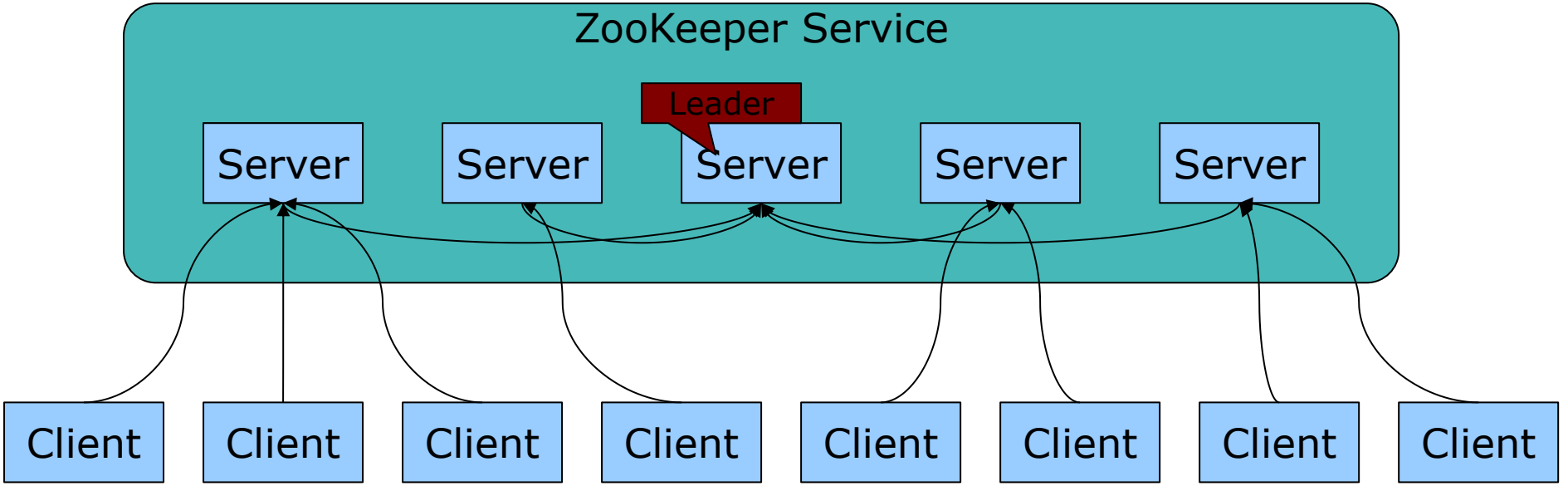
Create Flags

1)Ephemeral: the znode will be deleted when the session that created it times out or it is explicitly deleted

2)Sequence: the the path name will have a monotonically increasing counter relative to the parent appended



ZooKeeper Servers



1)All servers store a copy of the data (in memory)

2)A leader is elected at startup

3)Followers service clients, all updates go through leader

4)Update responses are sent when a majority of servers have persisted the change

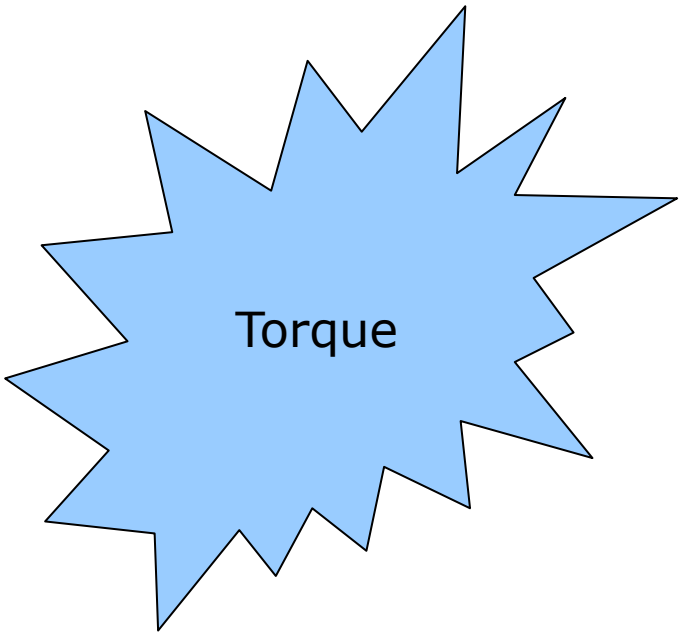
HOD

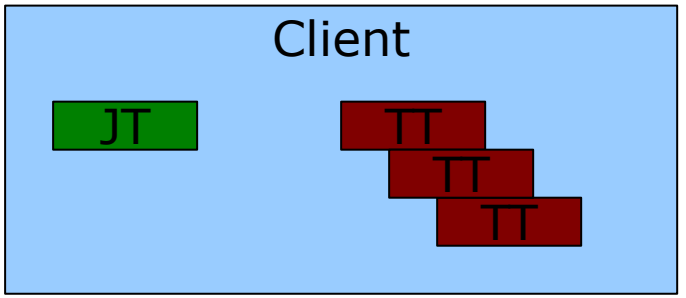
1)A client submits a request to start jobtracker and a set of tasktrackers to torque

2)The ip address and the ports that the jobtracker will bind to is not known apriori

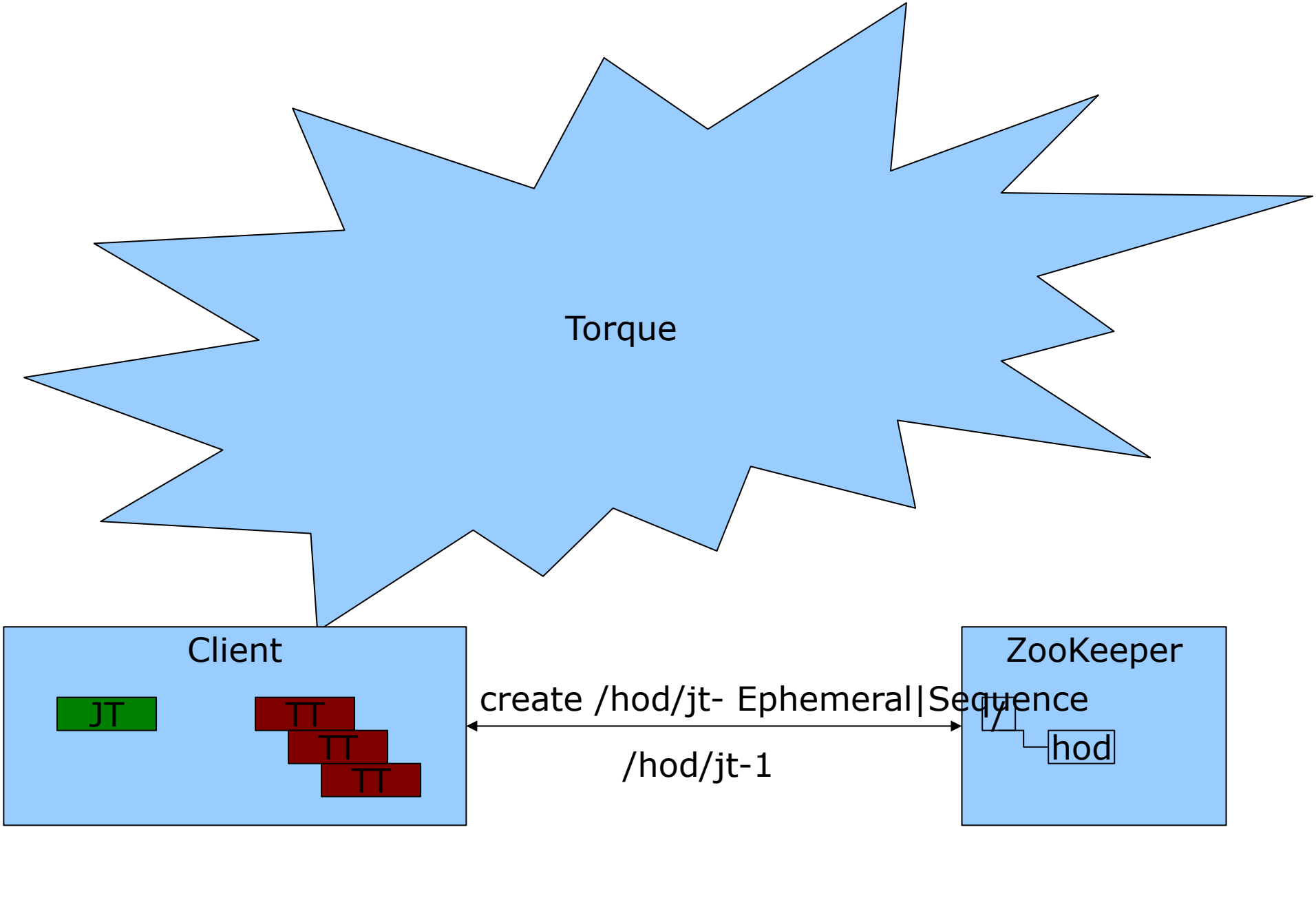
3)The tasktrackers need to find the jobtracker

4)The client needs to find the jobtracker

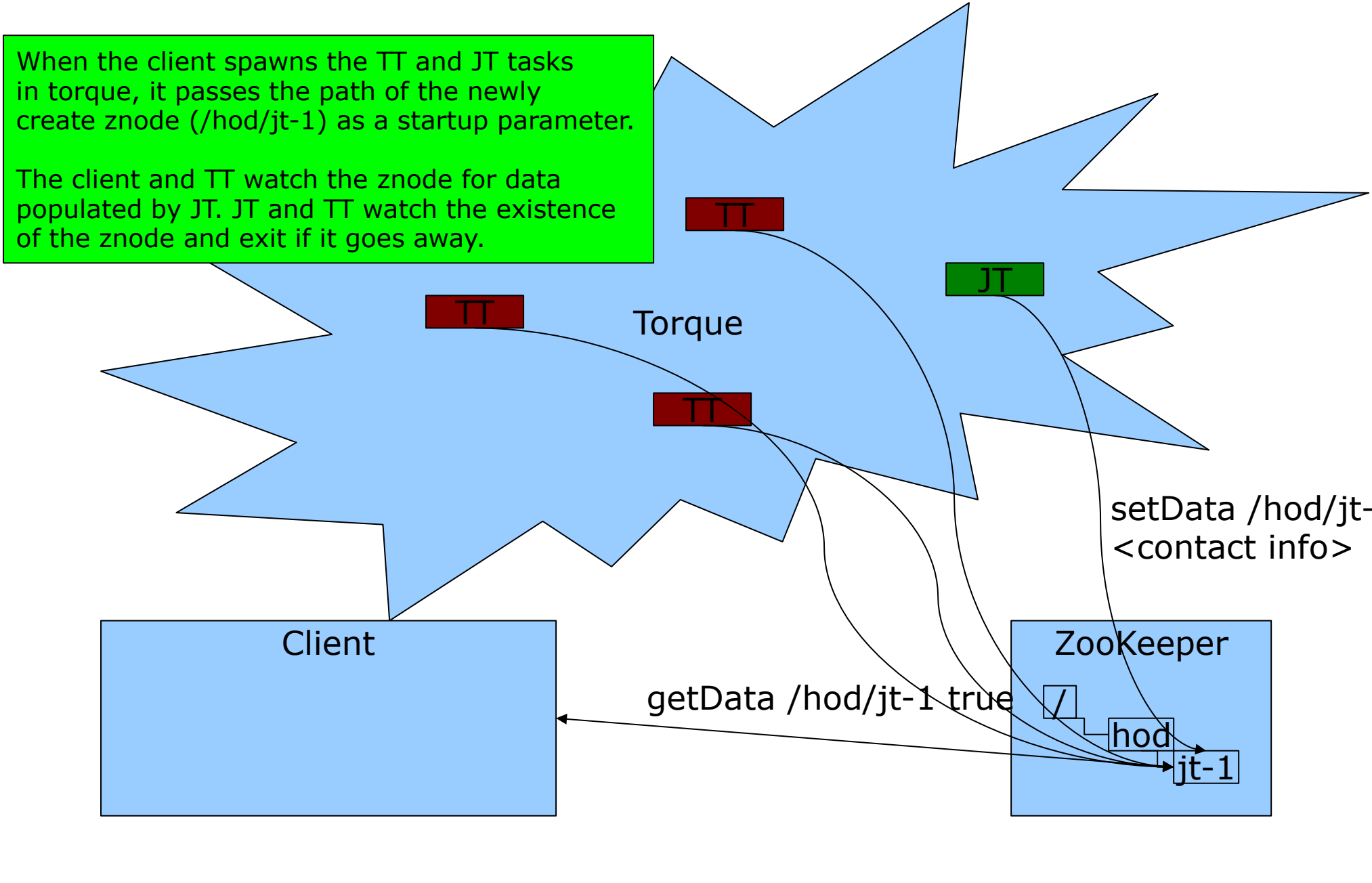


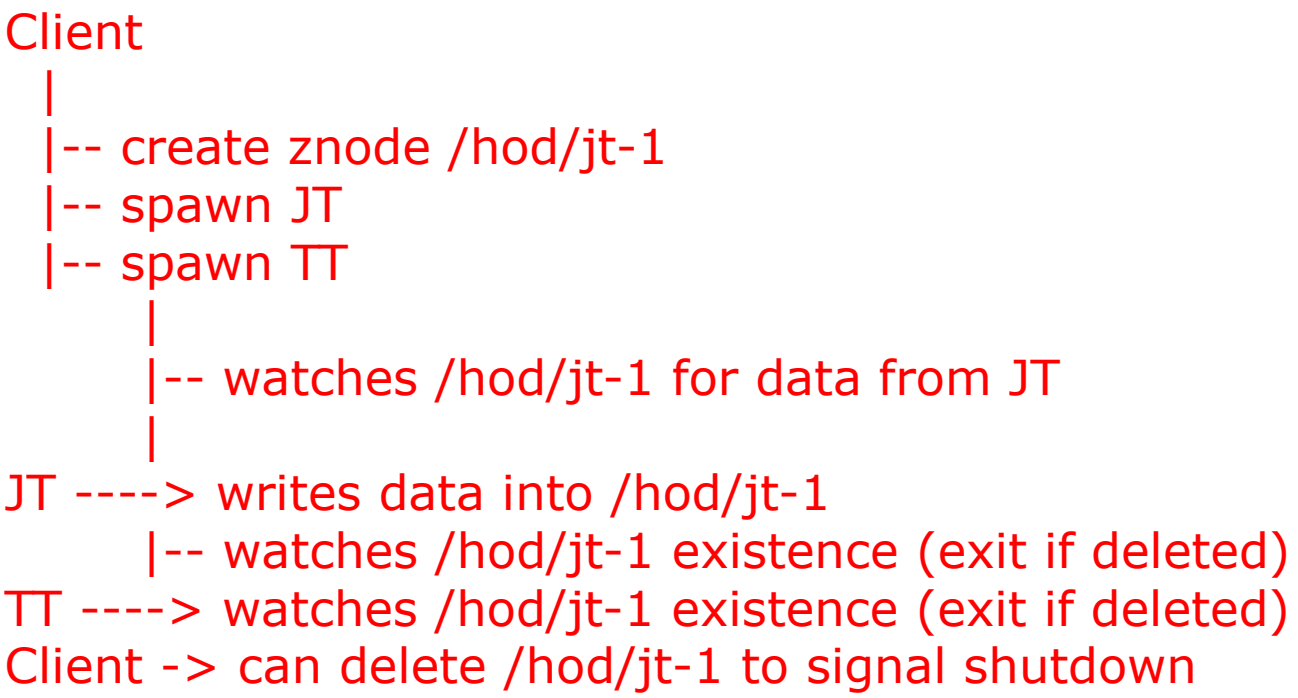


HOD with ZooKeeper

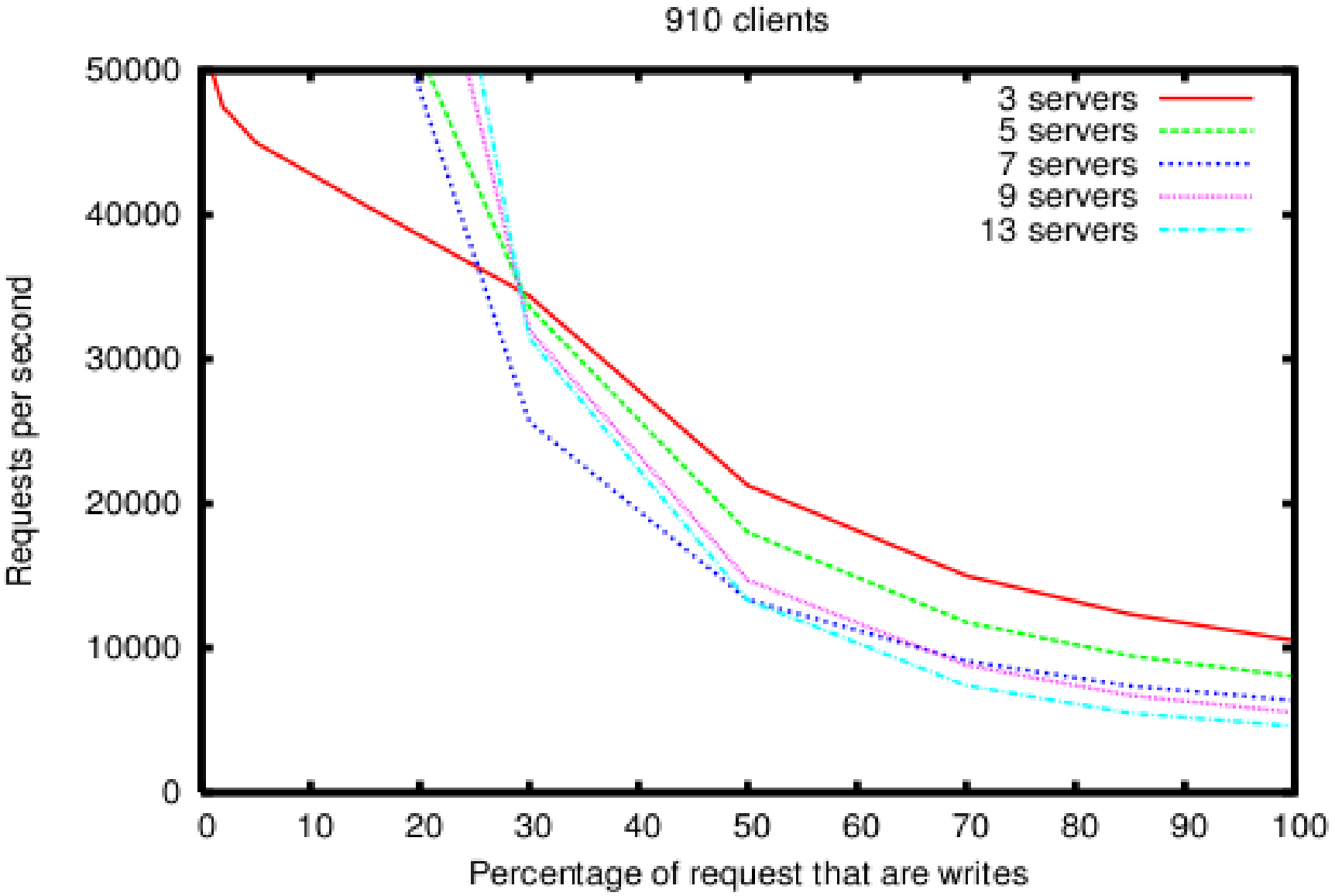


HOD with ZooKeeper




Client
|
|-- create znode /hod/jt-1
|-- spawn JT
|-- spawn TT
|
|-- watches /hod/jt-1 for data from JT
|
JT ----> writes data into /hod/jt-1
|-- watches /hod/jt-1 existence (exit if deleted)
TT ----> watches /hod/jt-1 existence (exit if deleted)
Client -> can delete /hod/jt-1 to signal shutdown

Performance



Performance at Extremes

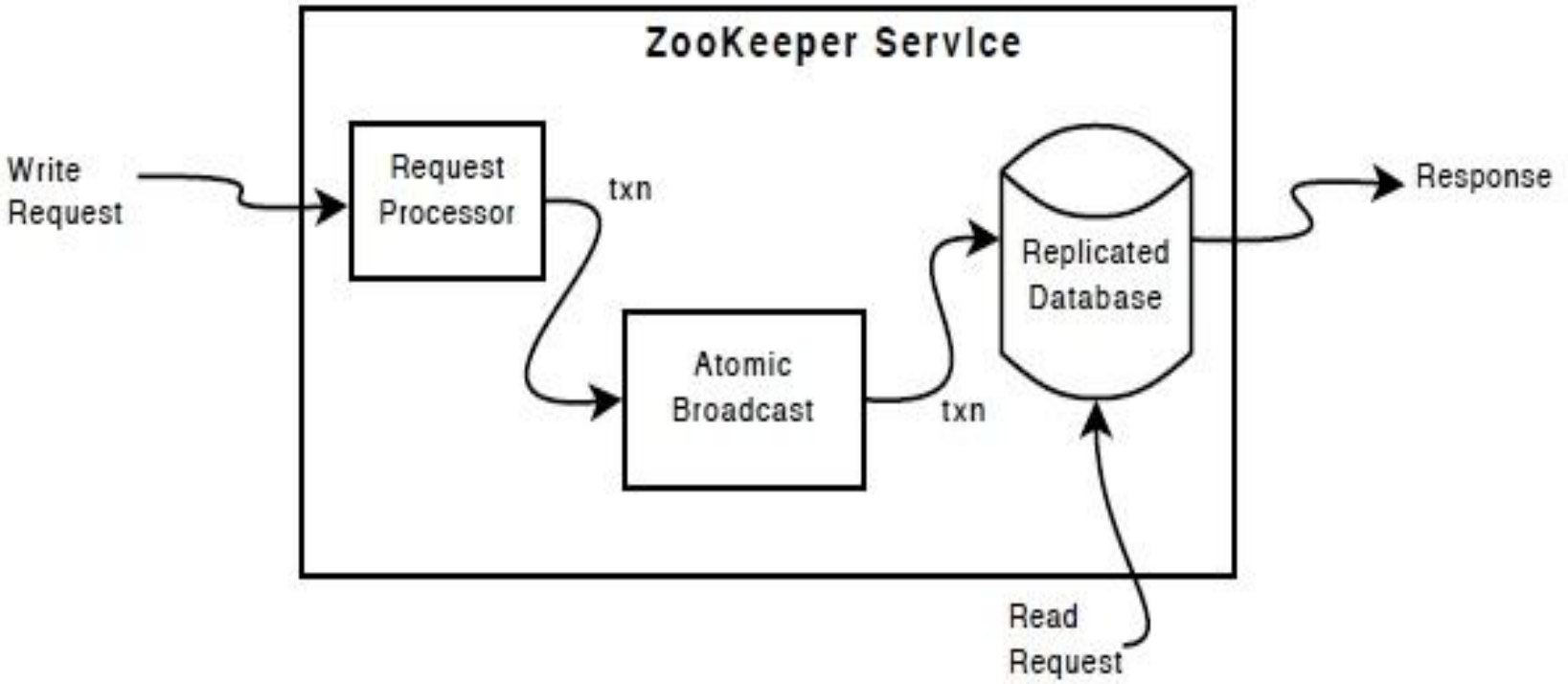
Servers	1% Writes	100% Writes
13	265115	4592
9	195178	5550
7	147810	6371
5	75308	8048
3	49827	10519

Numbers are operations per second

Status

1) Project started October 2006
2) Prototyped in Fall 2006
3) Initial implementation of production service March 2007
4) Code moved to zookeeper.sf.net and Apache License November 2007
5) Java Quorum and Standalone servers
6) Java and C clients available

ZAB



Why Atomic? Guarantee that each update is applied either completely or not at all across all Zookeeper nodes

Why Broadcast?: Updates from Leader sent to all the followers in the cluster to ensure strong consistency

ZAB

- ZooKeeper separates how it handles:
  - WRITE requests → complicated, must go through consensus
  - READ requests → very simple, go directly to memory database
- This part is the heart of ZooKeeper's consistency.
  It uses the ZAB protocol (ZooKeeper Atomic Broadcast).
- This module:
  - sends the proposed transaction to all followers
  - waits for ACKs from a majority (quorum)
  - guarantees that all ZooKeeper servers process transactions in the same order
- This ensures strong consistency.

ZAB

- ZAB is the consensus protocol used by Apache ZooKeeper to ensure that all ZooKeeper servers (nodes) agree on the same state, even in the presence of failures.
- Its main goal is to provide highly reliable replication of updates (writes) across a ZooKeeper ensemble (cluster).

ZAB: How?

- Leader election – One node is chosen as the leader.
- Proposal phase – Leader receives a client write request, assigns it a unique transaction ID (zxid), and proposes it to followers.
- Acknowledgment – Followers persist the proposal and send ACKs to the leader.
- Commit phase – When a majority (quorum) of followers ACK, the leader commits the transaction and notifies followers to commit.
- Recovery after crash – If the leader crashes, followers use ZAB to elect a new leader and continue from the last committed transaction.

ZAB: Making it Easy

- ZAB is a postal system for distributed updates.
- The leader
  - writes the letter (transaction),
  - sends copies to all nodes (followers),
  - waits until enough copies are confirmed (ACKs),
  - and then officially posts it (commit).
- Observers just get a copy without participating in the decision.