ZooKeeper

Yahoo Inc!

The problem: Coordination

Group membership

Leader election

Configuration

Status monitoring

Queuing

Critical sections

Develop different services for each need

OR

Implement primitives that can be used to implement other higher-level primitives

E.g. Chubby

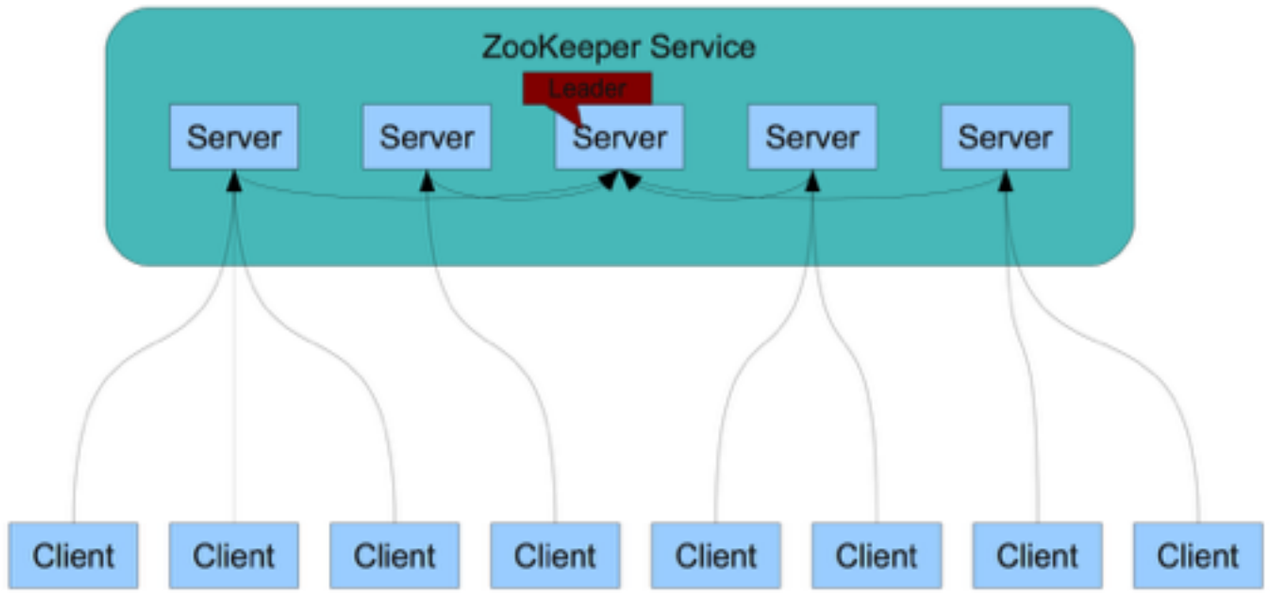
- Google's distributed lock service
- Locks can be used to implement other coordination needs (e.g. leader election, group membership)
- Emphasis on availability and reliability, not high performance
- All requests are directed to the leader

Goals

- High performance
- General
- Reliable

ZooKeeper

- Replicated over a set of machines
- Each replica has a copy of the data in memory
- Clients connect to a single replica over TCP
- Reads are local; writes go through the leader and need consensus (Zab protocol)
- Writes are logged to persistent storage for reliability; read-dominant workload



Wait-free

+

Event ordering

+

Notifications

Wait-free

Pros - no locks!

- Slow processes cannot slow down fast ones
- No deadlocks
- No blocking in the implementations

Cons - no locks!

- Some coordination primitives are blocking
- Need to be able to efficiently wait for conditions

Event ordering

Guarantees

- Writes are linearizable (strongest guarantee)
- FIFO client ordering of all operations

Cons

- Reads can be stale

Notifications (watches)

Properties

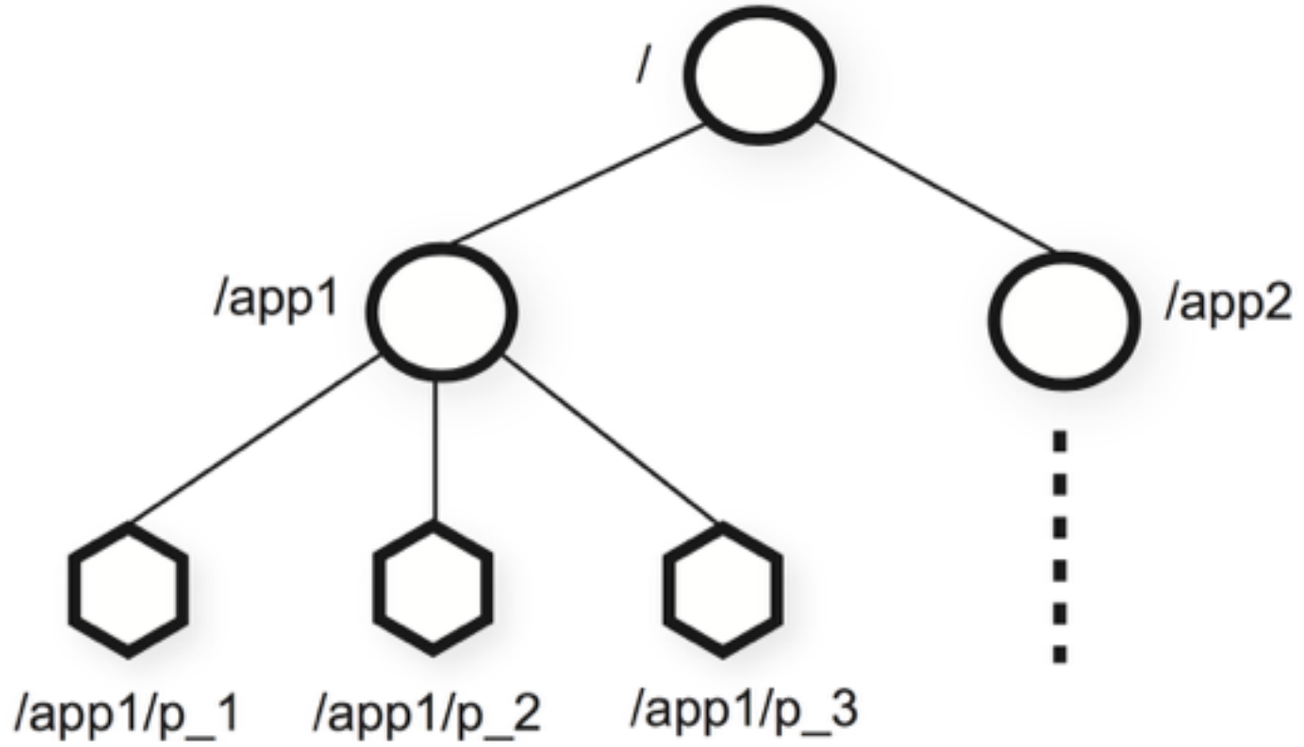
- Clients can request notifications on updates
- Notifications do not block write requests
- Clients are notified before they read the updated value

Cons

- One-time triggers

Data Model

- Hierarchical namespace (akin to a file system)
- Znodes are data objects that clients can manipulate
- Map to abstractions of the client apps, and store metadata



Znode flags

Ephemeral

- Znode deleted when creator fails or explicitly deleted

Sequence

- Append a monotonically increasing counter

API

- create (path, data, flags)
- delete (path, version)
- exists (path, watch)
- getData (path, watch)
- setData (path, data, version)
- getChildren (path, watch)
- sync (path)

Recipe: Configuration

- Workers get configuration
  getData (path=…/app/config, watch=true)
- Administrator changes configuration
  setData (path=…/app/config, newConfig, …)
- Workers get notified of change and get new config

Recipe: Group membership

- Register workers in the group
  create (path=…/workers/w1, data, EPHEMERAL)

- List group members
  getChildren (path=…/workers, watch=true)

Recipe: Locks (!!)

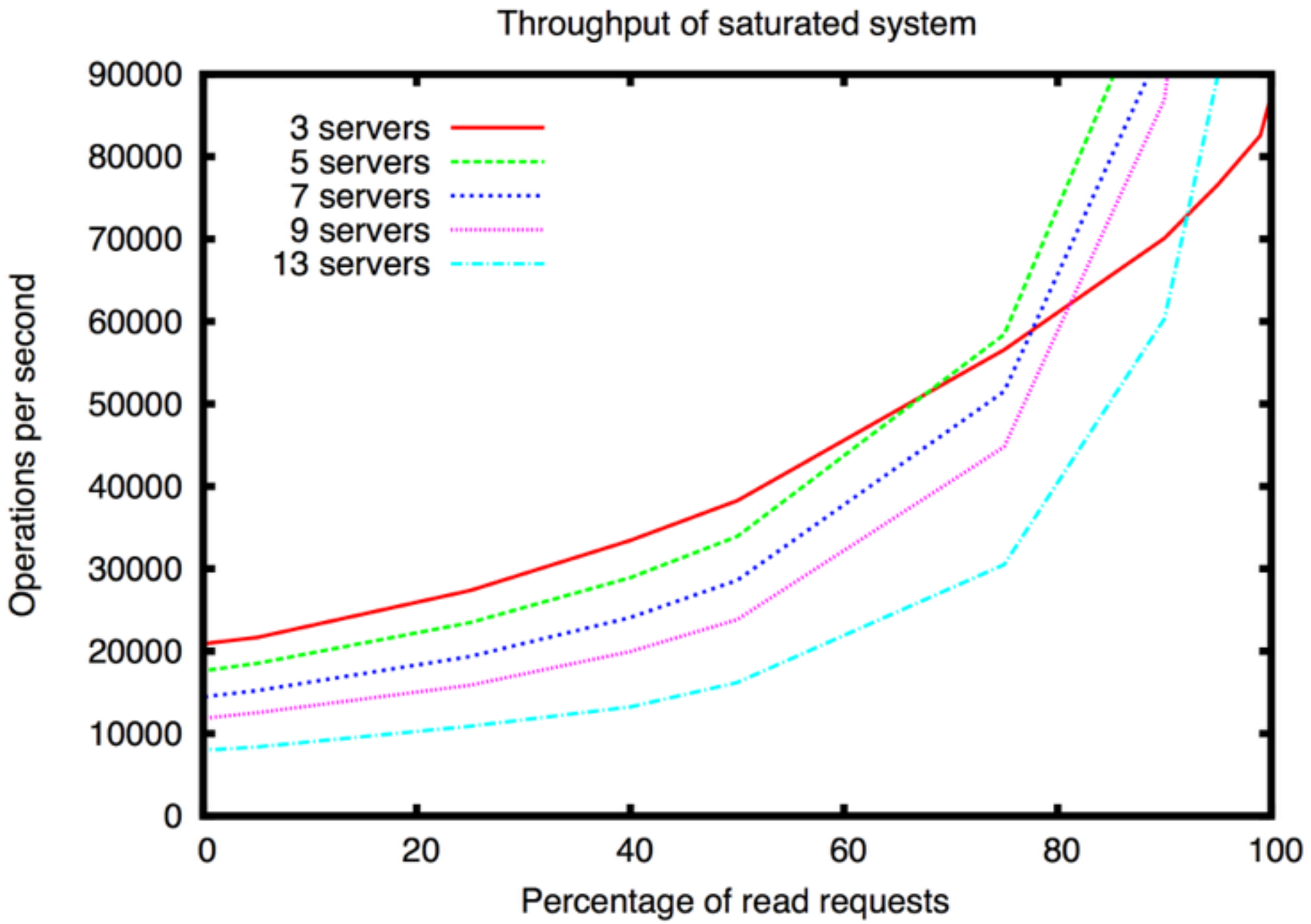
- n = create (“…/locks/x-”, SEQUENCE | EPHEMERAL)
- getChildren (“…/locks”)
- if n is the first child, exit /*(i.e. lock acquired)*/
- p = znode in list of children just before n
- if exists (p, true) wait for watch event
- goto step 2

Similar recipe can be used to implement shared locks as well

Tradeoffs

- Read v/s write throughput as size of ensemble is changed
- Performance v/s reliability — writes are logged to persistent storage

Performance



Thoughts

- ZooKeeper punts the ball to the clients, which can cause errors. Scope for a better system?
- Complete replication limits the size of the data ZooKeeper can handle. Problem?
- How about using a database with notifications?
- Random thought: is ZooKeeper CP or AP or neither? Does it matter?