Dynamic Reconfiguration of Primary/Backup Clusters (Apache ZooKeeper)

Alex Shraer
Yahoo! Research

In collaboration with:



Benjamin Reed
Yahoo! Research



Dahlia Malkhi
Microsoft Research



Flavio Junqueira
Yahoo! Research

1

Configuration of a Distributed Replicated System

- Membership
- Role of each server
  - E.g., deciding on changes (participant) or learning the changes (observer)
- Quorum System spec
  - majorities / hierarchical (server votes have different weight)
- Network addresses & ports
- Timeouts, directory paths, etc.

Dynamic Membership Changes

- Necessary in every long-lived system!
- Examples:
  - Cloud computing: adopt to changing load, don't pre-allocate!
  - Failures: replacing failed nodes with healthy ones
  - Upgrades: replacing out-of-date nodes with up-to-date ones
  - Free up storage space: decreasing the number of replicas
  - Moving nodes: within the network or the data center
  - Increase resilience by changing the set of servers

  Example: asynch. replication works as long as > #servers/2 are up:



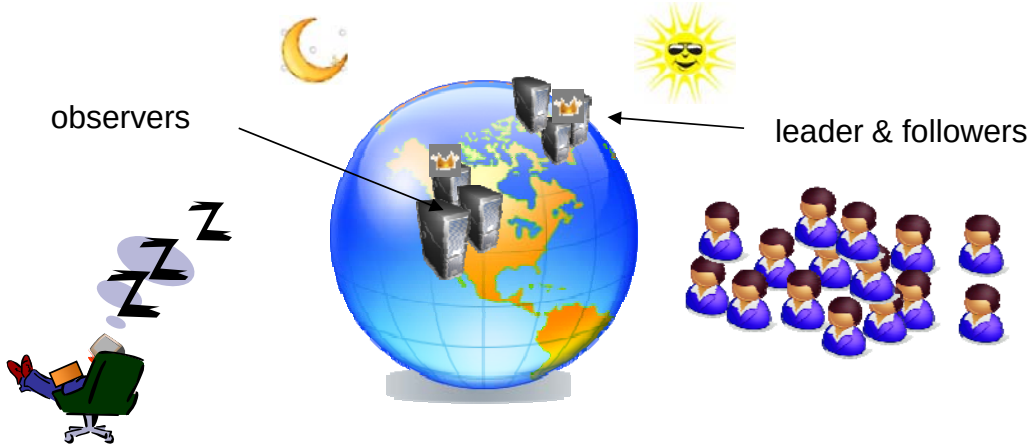
Other Dynamic Configuration Changes

- Changing server addresses/ports
- Changing server roles:



Other Dynamic Configuration Changes

- Changing server addresses/ports
- Changing server roles:

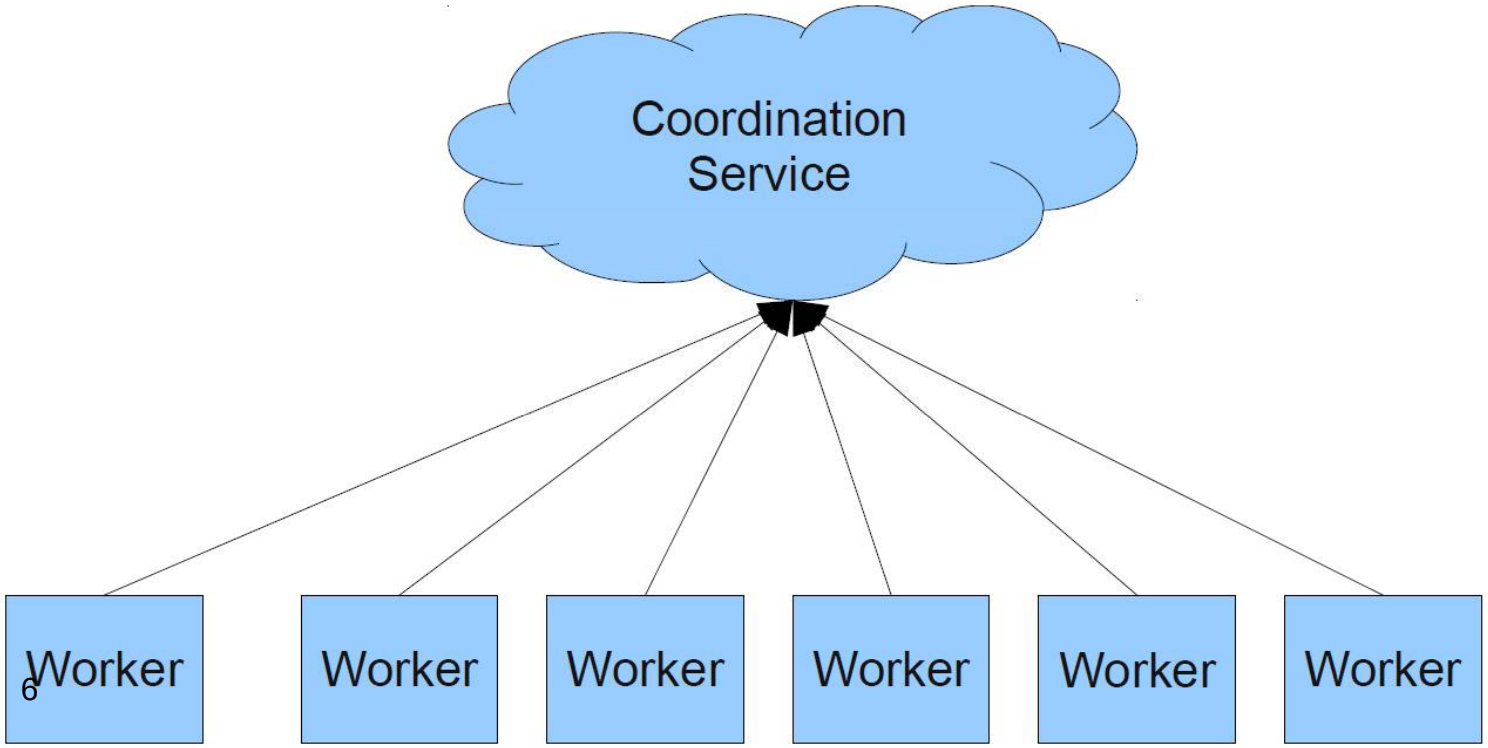


- Changing the Quorum System
  - E.g., if a new powerful & well-connected server is added

Industry Approach to Reconfiguration

Reconfiguration in Distributed Systems is difficult!

→ use external Coordination Service



Industry Approach to Reconfiguration

Reconfiguration in Distributed Systems is difficult!

→ use external Coordination Service

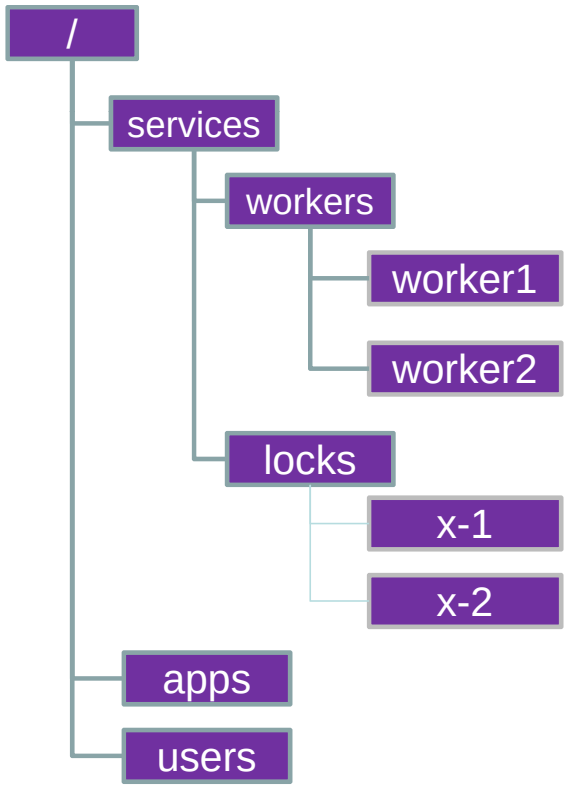
- Leading coordination services:
  - Chubby: Google
  - Apache Zookeeper:
    - Yahoo!, Linkedin, Twitter, Facebook, VMWare, UBS, Goldman Sachs, Netflix, Box, Cloudera, MapR, Nicira, …



- Configuration management, metadata store, failure detection, distributed locking, leader election, message queues, task assignment

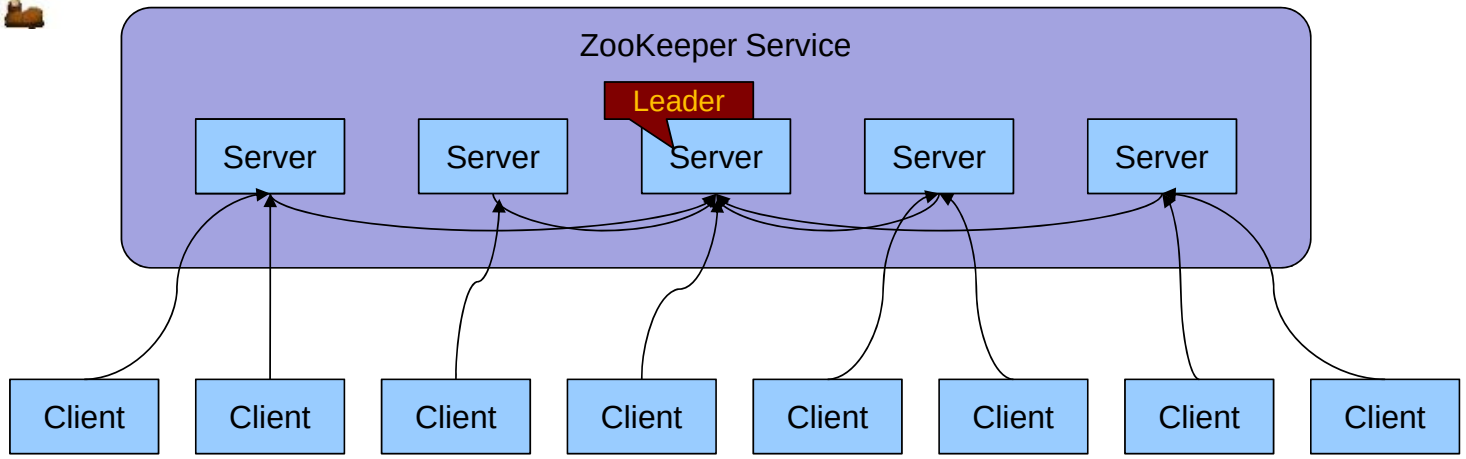
Zookeeper data model

- A tree of data nodes (znodes)
- Hierarchical namespace (like in a file system)
- Znode = <data, version, creation flags, children>





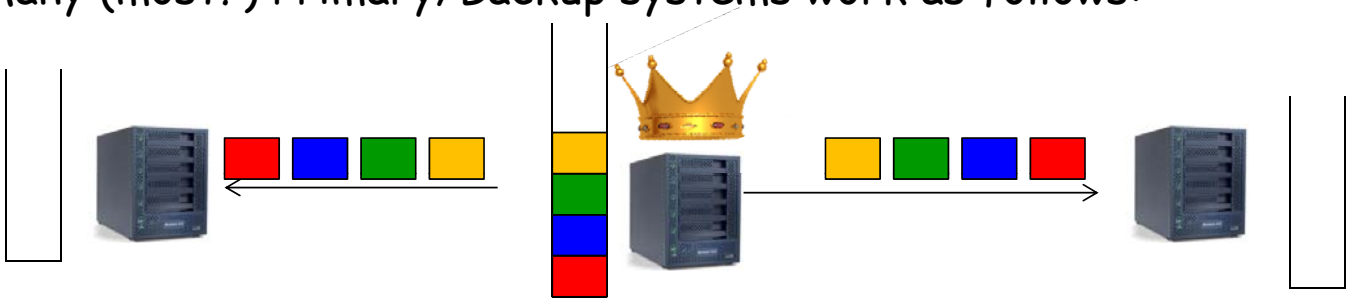
Zookeeper - distributed and replicated

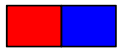


- All servers store a copy of the data (in memory)
- A leader is elected at startup
- Reads served by followers, all updates go through leader
- Update acked when a quorum of servers have persisted the change (on disk)
- Zookeeper uses ZAB - its own atomic broadcast protocol
  - Borrows a lot from Paxos, but conceptually different

Zookeeper is a Primary/Backup system

- Important subclass of State-Machine Replication
- Many (most?) Primary/Backup systems work as follows:



- Primary executes operations, sends idempotent state updates to backups
  -  "makes sense" only in the context of 
  - Primary speculatively executes and sends out  but it will only appear in a backup's log after 
  - In general SMR (Paxos), a backup's log may become 

- Primary order: each primary commits a consecutive segment in the log
  - Preserved by many (most?) primary/backup systems
    - Zookeeper, Chubby, GFS, Boxwood, Chain Replication, Harp, Echo, PacificA, etc.
  - Not preserved by Paxos / general state machine replication

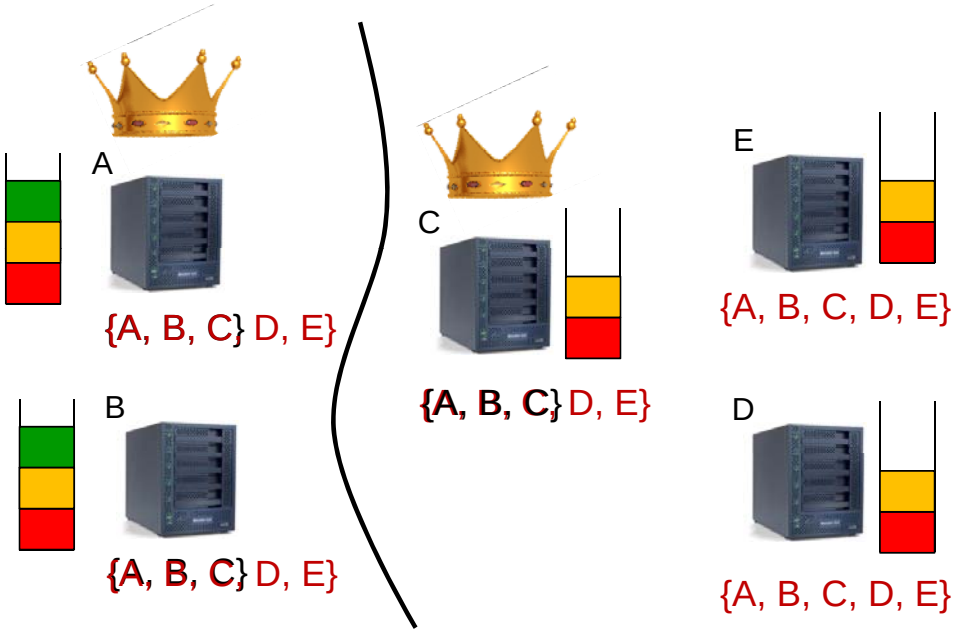
Reconfiguring Zookeeper

- Not supported
- All config settings are static – loaded during boot
- Zookeeper users repeatedly asking for reconfig. since 2008
  - Several attempts found incorrect and rejected

Manual Reconfiguration

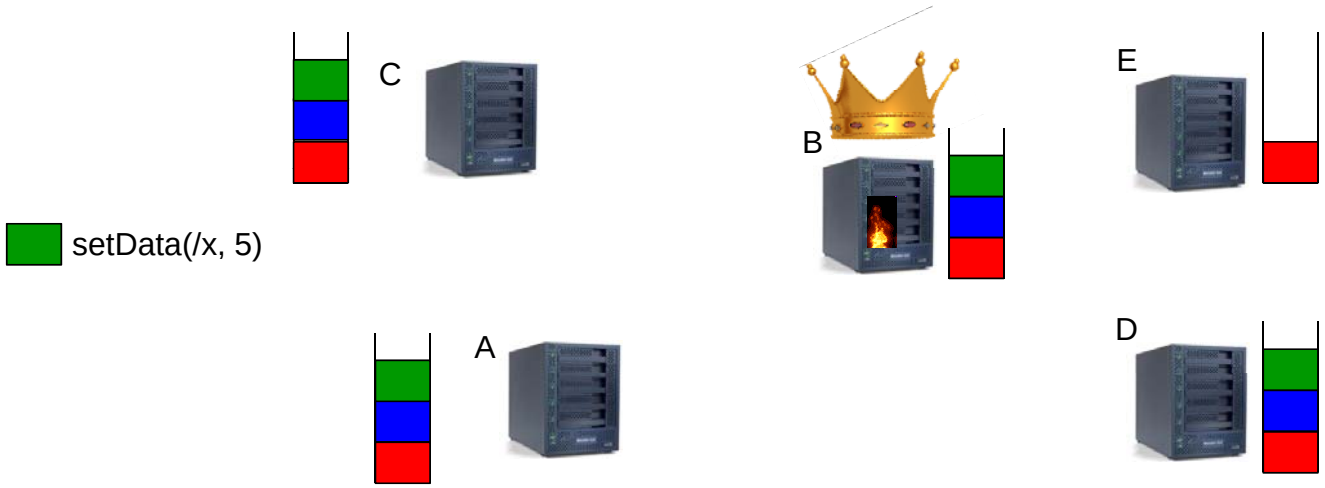
- Bring the service down, change configuration files, bring it back up
- Wrong reconfiguration caused split-brain & inconsistency in production
- Questions about manual reconfig are asked several times each week
- Admins prefer to over-provision than to reconfigure
  [LinkedIn talk @Yahoo, 2012]
  - Doesn't help with many reconfiguration use-cases
  - Wastes resources, adds management overhead
  - Can hurt Zookeeper throughput (we show)
- Configuration errors primary cause of failures in production systems
  [Yin et al., SOSP'11]

Hazards of Manual Reconfiguration



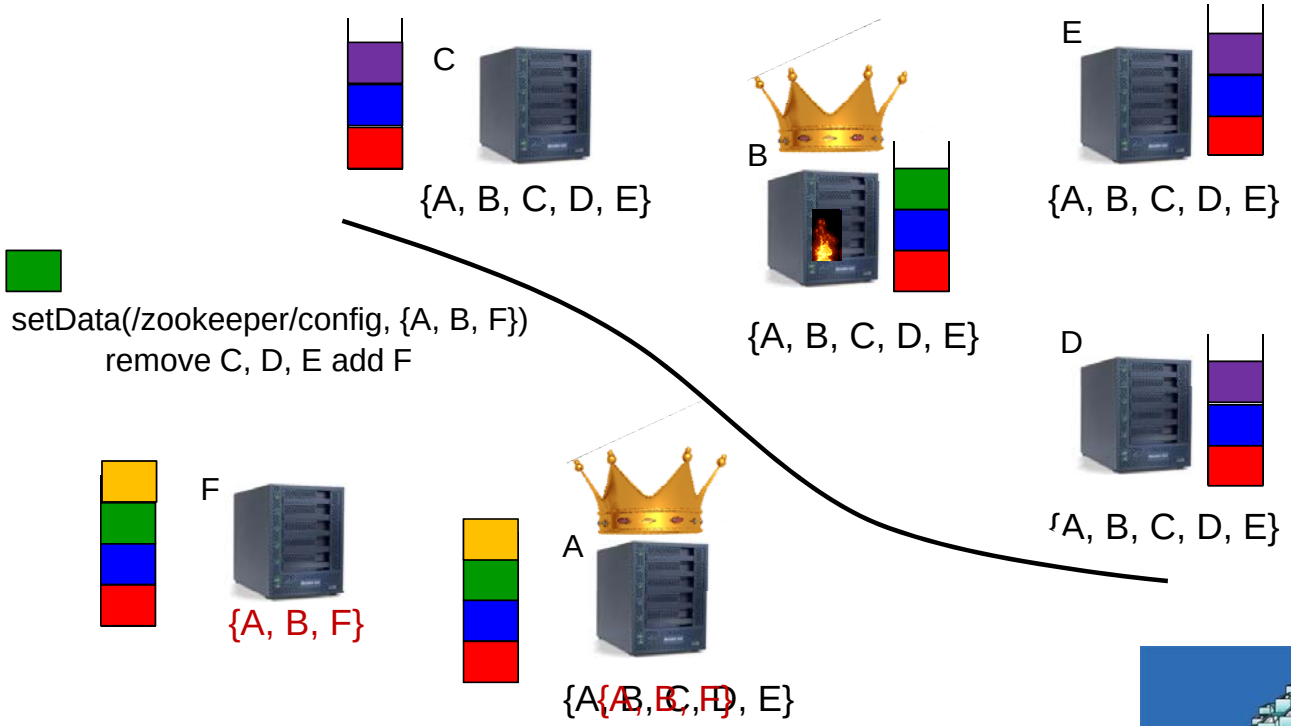
- Goal: add servers E and D
- Change configuration files
- Restart all servers
- We lost  and  !!

Can't we just store configuration in Zoookeeper ?
Recap of Recovery in Zookeeper

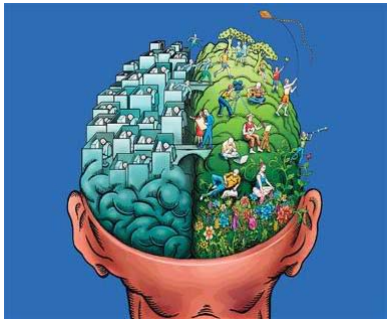


- Leader failure activates leader election & recovery

This doesn't work for reconfigurations!



- Must persist the decision to reconfigure in the old config before activating the new config!
- Once such decision is reached, must not allow further ops to be committed in old config



Principles of Reconfiguration

A reconfiguration S -> S' should do the following:

1. Commit reconfig in a quorum of S

2. Deactivate S (make sure no more updates committed in S)

3. Transfer state from S to S'
   - Identify all committed/potentially committed updates in S
   - Transfer state to a quorum of S'

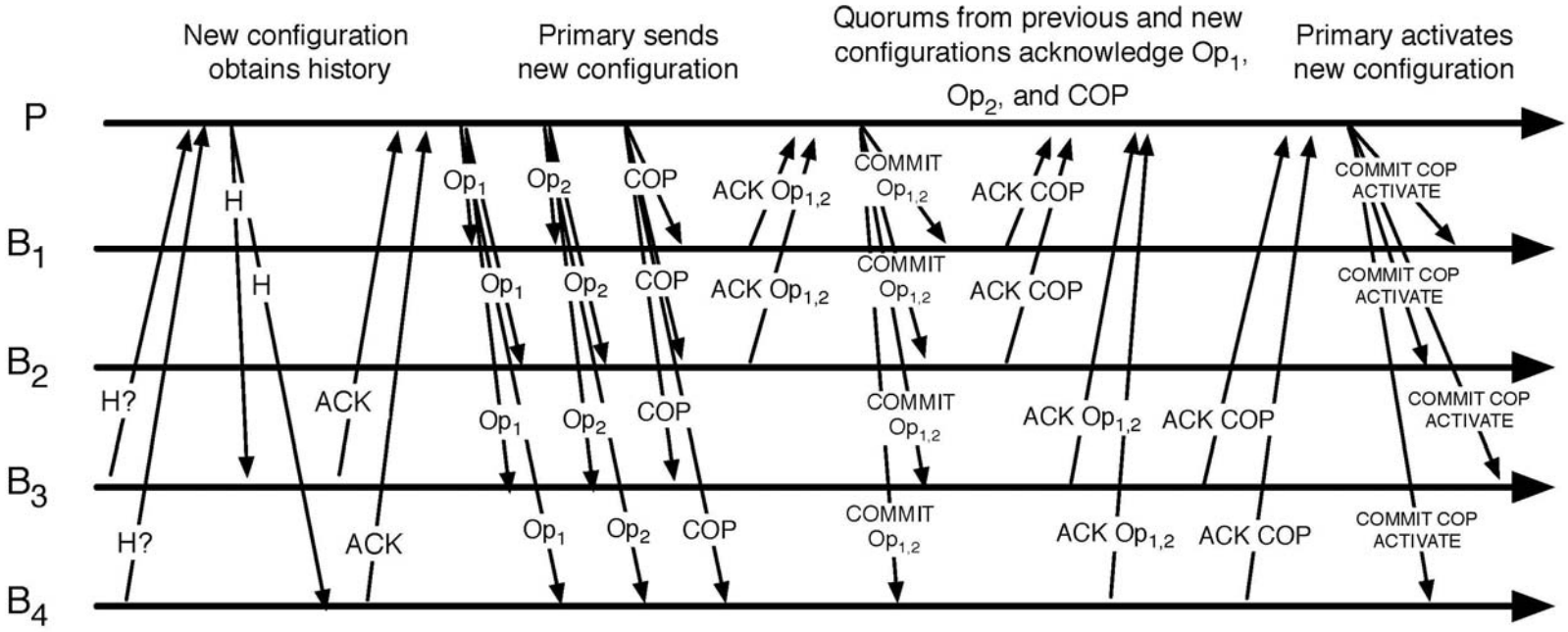
4. Activate S', so that it can process and commit client ops

Primary/Backup Reconfiguration

A reconfiguration S -> S' should do the following:

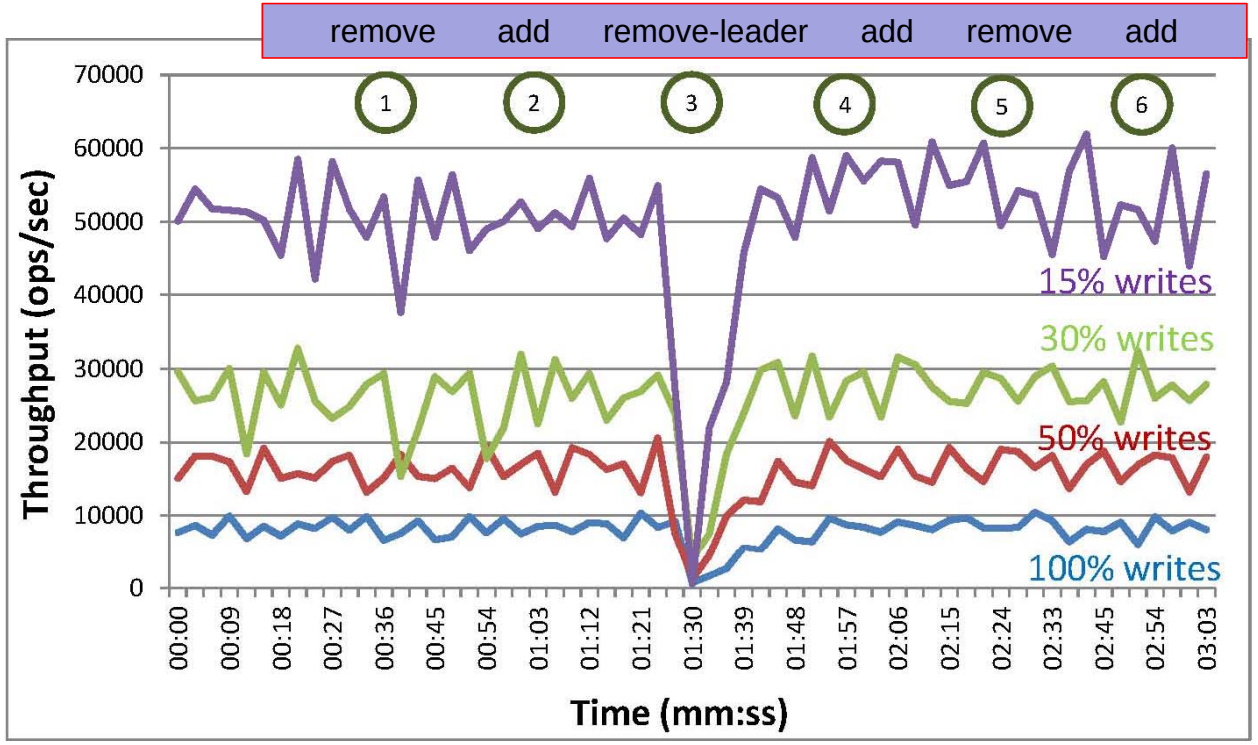
1. Submit reconfig op just like any other update in S
2. Primary-order guarantees that further updates committed in S')
3. Transfer state from S to S'
   - All important updates are in primary's log
   - Transfer ahead of time; here make sure transfer complete – need quorum of S' to ack all history up to reconfig
4. Activate S', so that it can process and commit client ops

Failure-Free Flow



Current config $S = \{P, B1, B2\}$
New configuration $S' = \{P, B3, B4\}$

Usually unnoticeable to clients

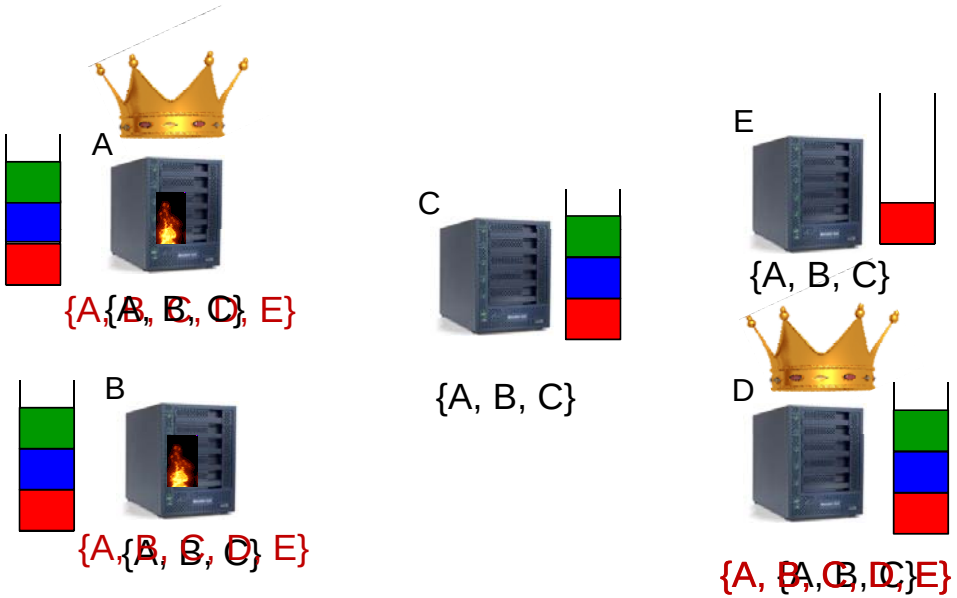


Protocol Features

- After reconfiguration is proposed, leader schedules & executes operations as usual
  - Leader of the new configuration is responsible to commit these
- If leader of old config is in new config and "able to lead", it remains the leader
- Otherwise, old leader nominates new leader (saves leader election time)
- We support multiple concurrent reconfigurations
  - Activate only the "last" config, not intermediate ones
  - In the paper, not in production ☺

20

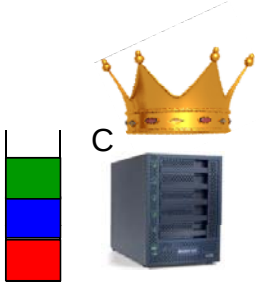
Gossiping activated configurations

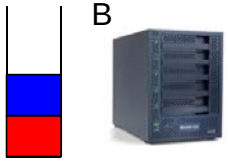


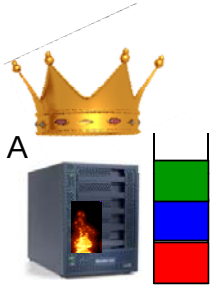
- : add servers E and D
- D should be leader (has latest state)
- But D doesn't have support of a quorum (3 out of 5)

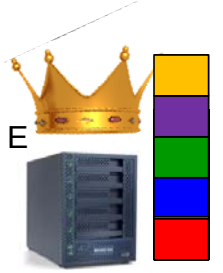
Recovery – Discovering Decisions

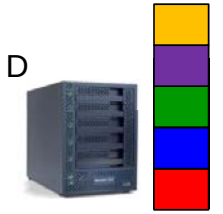








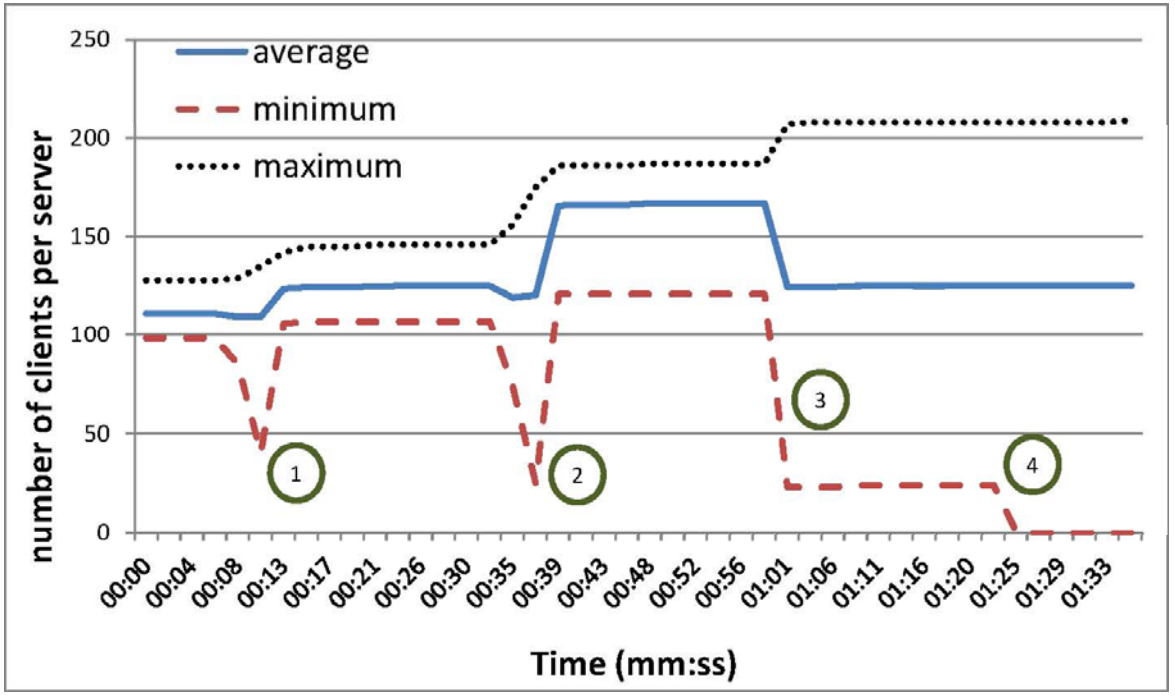




- : replace B, C with E, D
- C must 1) discover possible decisions in {A, B, C}
  (find out about {A, D, E})
  2) discover possible activation decision in {A, D, E}
  - If {A,D, E} is active, C mustn't attempt to transfer state
  - Otherwise, C should transfer state & activate {A, D, E}

The "client side" of reconfiguration

- When system changes, clients need to stay connected
  - The usual solution: directory service (e.g., DNS)
- Re-balancing load during reconfiguration is also important!
- Goal: uniform #clients per server with minimal client migration
  - Migration should be proportional to change in membership



Our approach - Probabilistic Load Balancing

- Example 1 :



  - Each client moves to a random new server with probability 0.4
    - 1 – 3/5 = 0.4
  - Exp. 40% clients will move off of each server

- Example 2 :



  - Clients connected to D and E don't move
  - Clients connected to A, B, C move to D, E with probability 4/9
    - $|S \cap S'|(|S|-|S'|)/|S'||S' \setminus S| = 2(5-3)/3*3 = 4/9$
  - Exp. 8 clients will move from A, B, C to D, E and 10 to F

Probabilistic Load Balancing

When moving from config. S to S':

$$\underbrace{E(load(i,S'))}_{\text{expected \#clients connected to i in S' (10 in last example)}} = \underbrace{load(i,S)}_{\text{\#clients connected to i in S}} + \underbrace{\sum_{j \in S \wedge j \neq i} load(j,S) \cdot \Pr(j \rightarrow i)}_{\text{\#clients moving to i from other servers in S}} - \underbrace{load(i,S) \sum_{j \in S' \wedge j \neq i} \Pr(i \rightarrow j)}_{\text{\#clients moving from i to other servers in S'}}$$

Solving for Pr we get case-specific probabilities.

Input: each client answers locally

Question 1: Are there more servers now or less ?

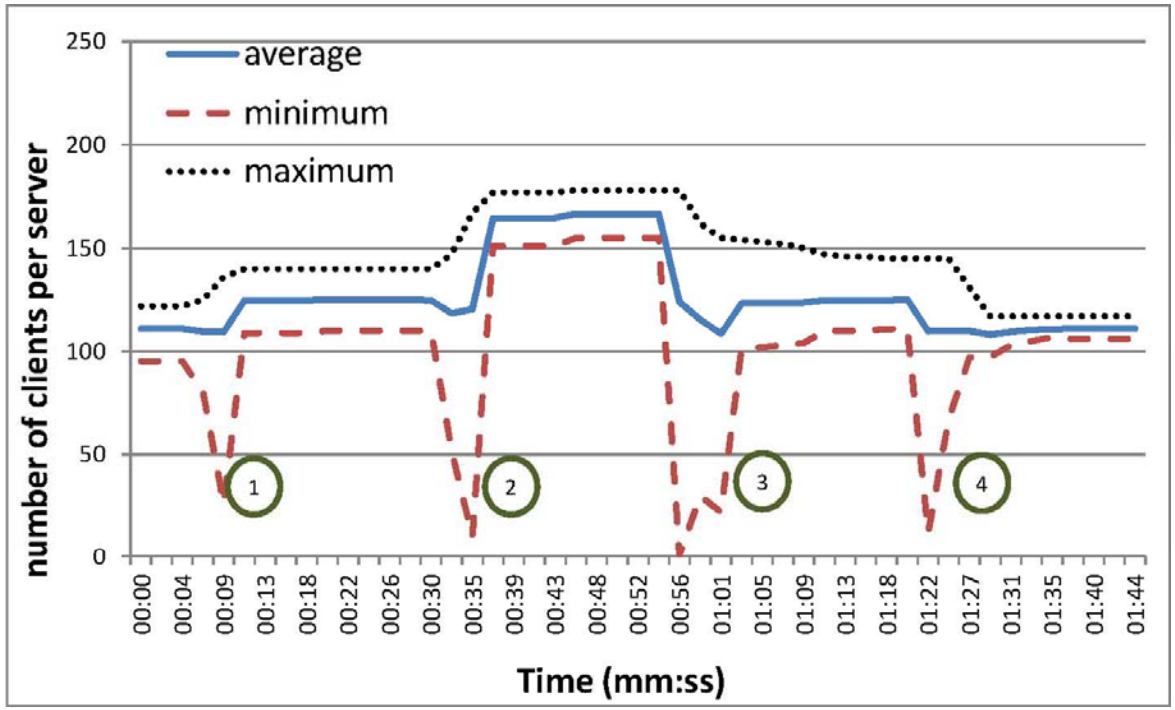
Question 2: Is my server being removed?

Output: 1) disconnect or stay connected to my server

if disconnect 2) Pr(connect to one of the old servers)

and Pr(connect to newly added server)

Probabilistic Load Balancing



Implementation

- Implemented in Zookeeper (Java & C), integration ongoing
  - 3 new Zookeeper API calls: reconfig, getConfig, updateServerList
  - feature requested since 2008
- Dynamic changes to:
  - Membership
  - Quorum System
  - Server roles
  - Addresses & ports
- Reconfiguration modes:
  - Incremental (add servers E and D, remove server B)
  - Non-incremental (new config = {A, C, D, E})
  - Blind or conditioned (reconfig only if current config is #5)
- Subscriptions to config changes using watches
  - Client can invoke client-side re-balancing upon change

Example - reconfig using CLI

reconfig −add 1=host1.com:1234:1235:observer;1239
−add 2=host2.com:1236:1237:follower;1231 −remove 5

- Change follower 1 to an observer and change its ports
- Add follower 2 to the ensemble
- Remove follower 5 from the ensemble

reconfig −file myNewConfig.txt −v 234547

- Change the current config to the one in myNewConfig.txt
- But only if current config version is 234547

getConfig −w −c

- set a watch on /zookeeper/config
- –c means we only want the new connection string for clients
  - host1:port1, host2:port2, host3:port3…

Summary

- Primary/Backup easier to reconfigure than general SMR
- We have a new algorithm, implemented in ZooKeeper
  - Being contributed to ZooKeeper codebase
- First practical algorithm for Speculative Reconfiguration
  - Using the primary order property

- Many nice features:
  - doesn't limit concurrency
  - reconfigures immediately
  - preserves primary order
  - doesn't stop client ops
  - Clients work with a single configuration at a time
  - No external services
  - Includes client-side rebalancing

Acknowledgements

- ZooKeeper open source community

- Marshall McMullen (SolidFire)
- Vishal Kher (VMWare)
- Mahadev Konar (Horton Works)
- Patrick Hunt (Cloudera)
- Rakesh Radhakrishnan (Huawei)
- Raghu Shastry