Cristina Nita-Rotaru

# 7610: Distributed Systems

Chubby. Zookeeper.

# REQUIRED READING

- The Chubby Lock Service for Loosely-Coupled Distributed Systems OSDI 2006.
- ZooKeeper: Wait-free coordination for Internet-scale systems. Usenix 2010
- Zab: High-performance broadcast for primary-backup systems. DSN 2011

- Slides prepare from talks of Chubby and Zookeeper authors

# 1: Chubby

# Chubby

- A coarse-grained lock service
  - Provides a means for distributed systems to synchronize access to shared resources
  - Uses advisory locks
- Intended for use by "loosely-coupled distributed systems"
- Goals
  - High availability
  - Reliability
  - Small storage
  - Easy-to-understand semantics

# Advisory vs. Mandatory Locking

- Advisory (unenforced) locking:
  - Requires cooperation from the participating processes to ensure serialization.
  - Each process tries to acquire a lock before writing.
- Mandatory locking:
  - Does not require cooperation from the participating processes.
  - Kernel checks every open, read, and write to verify that the calling process is not violating a lock on the given file.

# Why Not Mandatory Locks?

- Locks represent client-controlled resources; how can Chubby enforce this?
- Mandatory locks imply shutting down client apps entirely to do debugging
  - Shutting down distributed applications much trickier than in single-machine case

# How is Chubby Used at Google

- GFS: Elect a master
- BigTable: master election, client discovery, table service locking
- Well-known location to bootstrap larger systems:  store small amount of meta-data, as the root of the distributed data structures
- Partition workloads
- Name service because of its consistent client caching
- Locks are coarse: held for hours or days

# External Interface

- Organized as cells (5 replicas)
- Presents a simple distributed file system
- Clients can open/close/read/write files
  - Reads and writes are whole-file
  - Supports advisory reader/writer locks
  - Clients can register for notification of file update

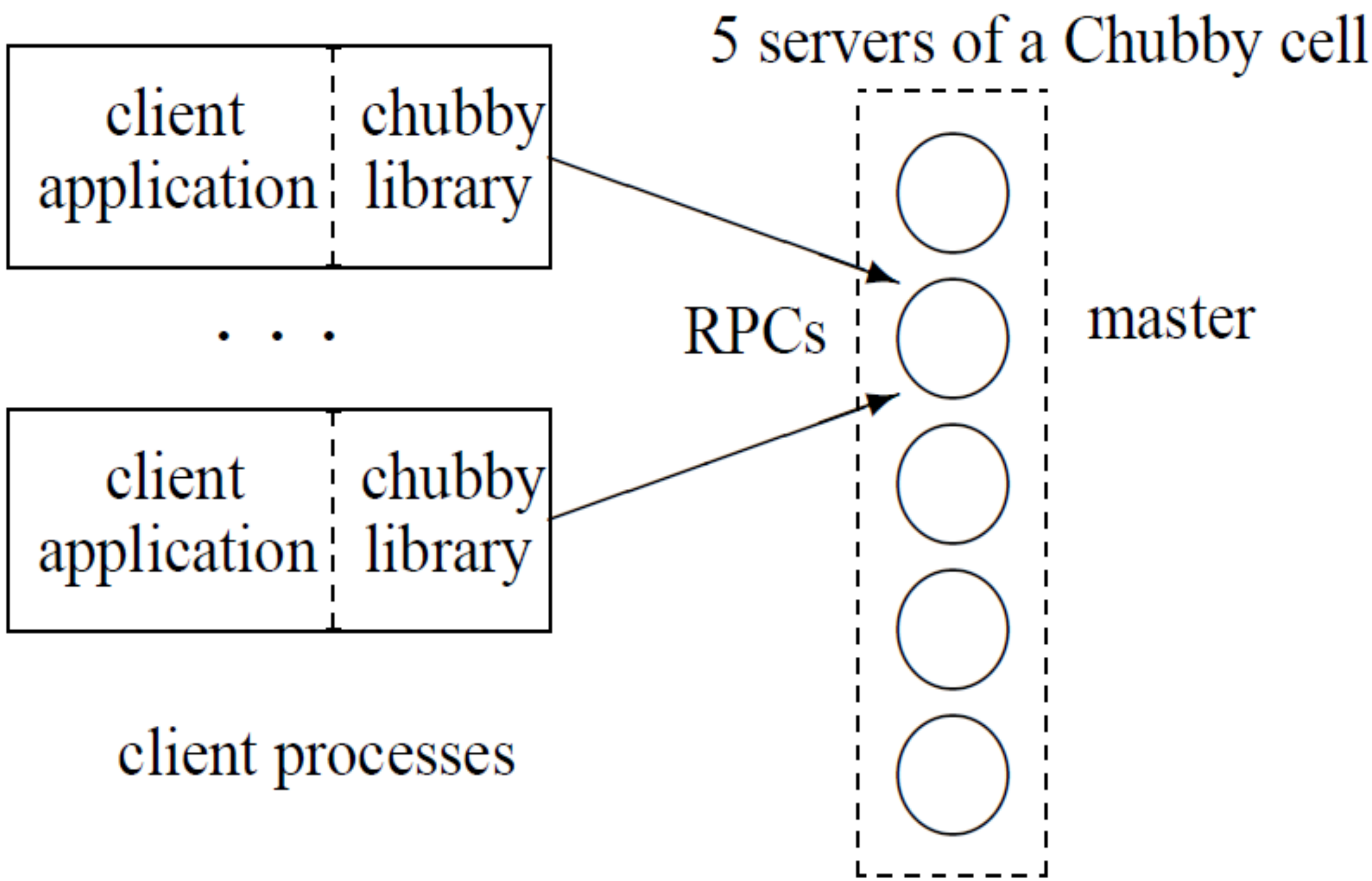
# How are Files used as Locks

- Files can have several attributes
  - The contents of the file is one (primary) attribute
  - Owner of the file
  - Permissions
  - Date modified
  - Whether the file is locked or not

# Example: Use Chubby for Master Election

- All replicas try to acquire a write lock on a designated file.
- The replica who gets the lock is the master.
- Master can then write its address to file; other replicas can read this file to discover the chosen master name.
- Chubby can also be used as a name service.

# Chubby Cell


5 servers of a Chubby cell
client application
chubby library
. . .
client application
chubby library
RPCs
master
client processes

# Chubby and Consensus

- Chubby cell is usually 5 replicas (2f+1), tolerates 2 failures
  - 3 replicas must be alive for cell to work (otherwise it blocks)
- Replicas in Chubby must agree on their own master and official lock values
- Uses PAXOS algorithm (provides consensus in an asynchronous system)
  - Memory for individual "facts" in the network
  - A fact is a binding from a variable to a value

# Paxos: Processor Assumptions

- Operate at arbitrary speed
- Independent, random failures
- Process with stable storage may rejoin protocol after failure
- Do not lie, collude, or attempt to maliciously subvert the protocol

# Paxos: Network Assumptions

- All processors can communicate with  one another
- Messages are sent asynchronously and may take arbitrarily long to deliver
- Order of messages is not guaranteed: they may be lost, reordered, or duplicated
- Messages, if delivered, are not corrupted in the process

# Paxos in Chubby

- Replicas in a cell initially use Paxos to establish the leader.
- Majority of replicas must agree
- Replicas promise not to try to elect new master for at least a few seconds ("master lease")
- Master lease is periodically renewed

# Client Updates

- All replicas are listed in DNS
- All client updates go through master
- Master updates official database; sends copy of update to replicas
  - Majority of replicas must acknowledge receipt of update before master writes its own value
- Clients find master through DNS
  - Contacting replica causes redirect to master

# Replica Failure

- If a replica fails and does not recover for a long time (a few hours), a fresh machine is selected to be a new replica, replacing the failed one
- New replica
  - Updates the DNS
  - Obtains a recent copy of the database
- Current master polls DNS periodically to discover new replicas

# Chubby File System

- Looks like simple UNIX FS: /ls/foo/wombat
  - All filenames start with '/ls' ("lockservice")
  - Second component is Chubby cell ("foo")
  - Rest of the path is anything you want
- No inter-directory move operation
- Permissions use ACLs, non-inherited
- No symlinks/hardlinks
- Files have version numbers attached
- Opening a file receives handle to file
  - Clients cache all file data including file-not-found

# ACLs and File Handles

- Access Control List (ACL)
  - A node has three ACL names (read/write/change )
  - An ACL name is a name to a file in the ACL directory
  - The file lists the authorized users
- File handle:
  - Has check digits encoded in it; cannot be forged
  - Sequence number: a master can tell if this handle is created by a previous master
  - Mode information at open time: If previous master created the handle, a newly restarted master can learn the mode information

# Use of Sequences

- Lock problems in distributed systems
  - A holds a lock L, issues request write W, then fails
  - B acquires L (because A fails), performs actions
  - W arrives (out-of-order) after B's actions
- One approach is to prevent other clients from getting the lock if a lock become inaccessible or the holder has failed
- Another approach: Sequencer
  - A lock holder can obtain a sequencer from Chubby
  - It attaches the sequencer to any requests that it sends to other servers (e.g., Bigtable)
  - The other servers can verify the sequencer information

# Chubby Events

- Master notifies clients if files modified, created, deleted, lock status changes, etc
- Clients can subscribe to events (up-calls from Chubby library)
  - File contents modified: if the file contains the location of a service, this event can be used to monitor the service location
  - Master failed over
  - Child node added, removed, modified
  - Handle becomes invalid: probably communication problem
  - Lock acquired (rarely used)
  - Locks are conflicting (rarely used)
- Push-style notifications decrease bandwidth from constant polling

# APIs

- Open()
  - Mode: read/write/change ACL; Events; Lock-delay
  - Create new file or directory?
- Close()
- GetContentsAndStat(), GetStat(), ReadDir()
- SetContents(): set all contents; SetACL()
- Delete()
- Locks: Acquire(), TryAcquire(), Release()
- Sequencers: GetSequencer(), SetSequencer(), CheckSequencer()

# Example: Primary Election

```
Open("write mode");
If (successful) {
        // primary
  SetContents("identity");
}
Else {
  // replica
  open ("read mode", "file-modification event");
  when notified of file modification:
          primary= GetContentsAndStat();
}
```

# Client Caching

- Clients cache all file content
- Strict consistency:
  - Lease based
  - Master will invalidate cached copies upon a write request
- Client must send respond to Keep-Alive message from server at frequent interval
- Keep-Alive messages include invalidation requests
  - Responding to Keep-Alive implies acknowledgement of cache invalidation
- Modification only continues after all caches invalidated or Keep-Alive time out

# Client Sessions

- Sessions maintained between client and server
  - Keep-alive messages required to maintain session every few seconds
  - A client sends keep-alive requests to a master
  - A master responds by a keep-alive response
- If session is lost, server releases any client-held handles.
- What if master is late with next keep-alive?
  - Client has its own (longer) timeout to detect server failure

# Master Failure

- If client does not hear back about Keep-Alive in local lease timeout, session is in jeopardy
  - Clear local cache
  - Wait for "grace period" (about 45 seconds)
  - Continue attempt to contact master
  - Successful attempt => ok; jeopardy over
  - Failed attempt => session assumed lost
- If replicas lose contact with master
  - They wait for grace period (4—6 secs)
  - On timeout, hold new election

# Master Fail-over: Grace Period

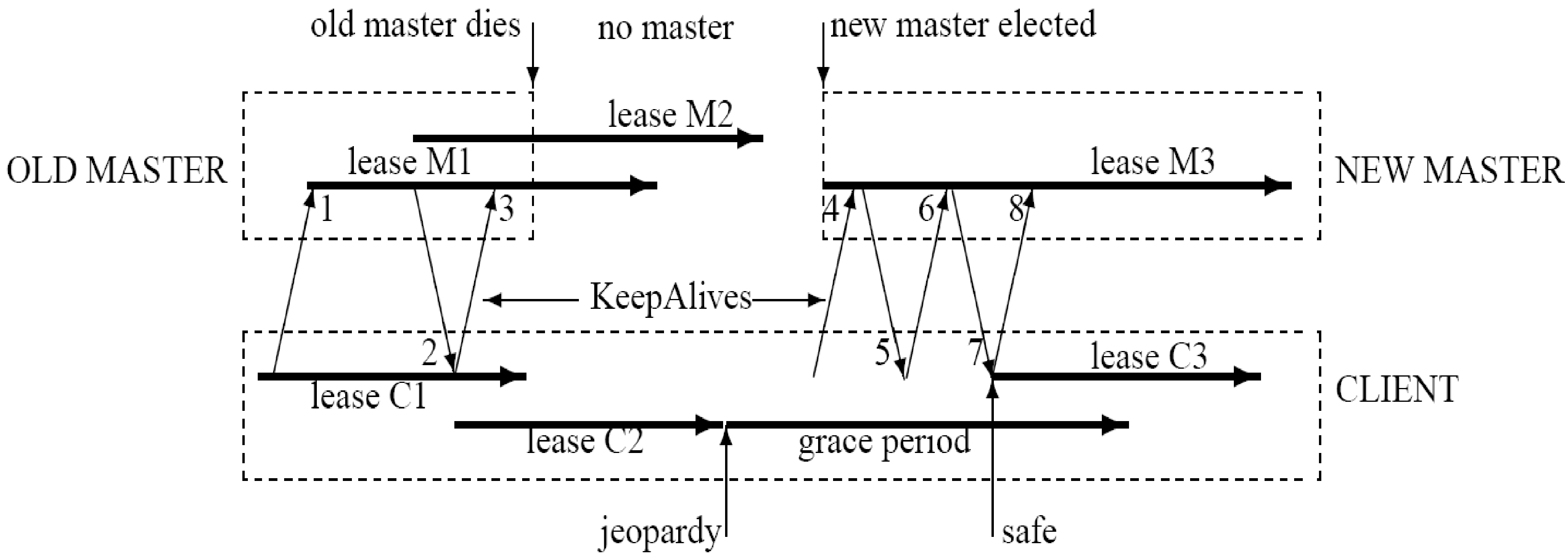



Figure 2: The role of the grace period in master fail-over

# Reliability

- Started out using replicated Berkeley DB
- Now uses custom write-thru logging DB
- Entire database periodically sent to GFS
  - In a different data center
- Chubby replicas span multiple racks

# Scalability

- 90K+ clients communicate with a single Chubby master (2 CPUs)
- System increases lease times from 12 sec up to 60 secs under heavy load
- Clients cache virtually everything
- Data is small – all held in RAM (as well as disk)

# 2: Zookeeper

# ZooKeeper

- Provides to HDSF functionality similar to that provided by Chubby to GFS
- Design inspired from Chubby
- Zookeeper is used to manage master election and store other process metadata
- Chubby and Zookeeper are both much more than a distributed lock service: implementations of highly available, distributed metadata file systems

# ZooKeeper

- Aims to provide a simple and high performance kernel for building more complex client
- Wait free
- FIFO
- No lock
- Pipeline architecture

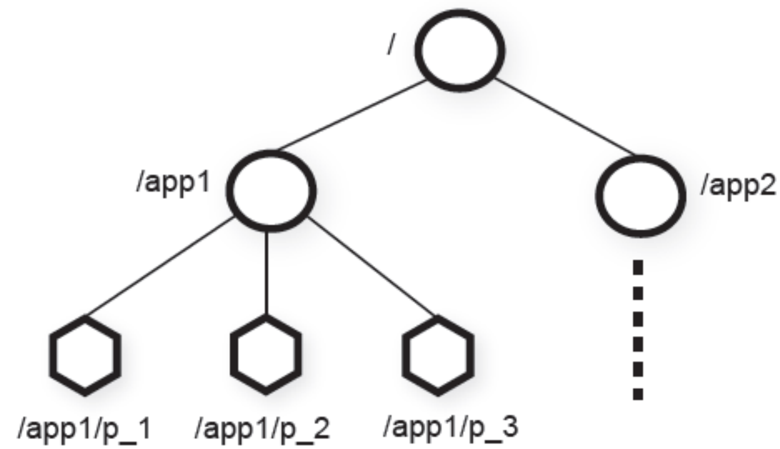
# What is coordination?

- Group membership
- Leader election
- Dynamic configuration
- Status monitoring
- Queuing
- Critical sections

# Contributions

- Coordination kernel
  - Wait-free coordination
- Coordination recipes
  - Build higher primitives
- Experience with Coordination
  - Some application use ZooKeeper

# Zookeeper Service




- Znode
  - In-memory data node in the Zookeeper data
  - Have a hierarchical namespace
  - UNIX like notation for path
- Types of Znode
  - Regular: Clients manipulate regular znodes by creating and deleting them explicitly;
  - Ephemeral: Clients create such znodes, and they either delete them explicitly, or let the system remove them automatically
- Flags of Znode
  - Sequential flag: Nodes created with the sequential flag set have the value of a monotonically increasing counter appended to its name. If n is the new znode and p is the parent znode, then the sequence value of n is never smaller than the value in the name of any other sequential znode ever created under p.

# Zookeeper Service

- Watch Mechanism
  - Get notification
  - One time triggers
- Other properties of Znode
  - Znode is not designed for data storage, instead it stores meta-data or configuration
  - Can store information like timestamp version
- Session
  - A connection to server from client is a session
  - Timeout mechanism

# Client API

- Create(path, data, flags)
- Delete(path, version)
- Exist(path, watch)
- getData(path, watch)
- setData(path, data, version)
- getChildren(path, watch)
- Sync(path)
- Two versions
  - Synchronous: I when it needs to execute a single ZooKeeper operation and it has no concurrent tasks to execute,
  - Asynchronous: multiple outstanding ZooKeeper operations and other tasks executed in parallel.

# Guarantees

- Linearizable writes
  - All requests that update the state of ZooKeeper data are serializable and respect precedence
- FIFO client order
  - All requests are in order that they were sent by client

# Configuration Management

- Problem:  dynamic configuration propose
- Solution:
  - Simplest way is to make up a znode c for saving configuration
  - Other processes set the watch flag on c
  - The notification just indicates there is an update without telling how many time updates occurs

# Rendezvous

- Problem: Configuration of the system may not be sure at the beginning (For example, a client may want to start a master process and several worker processes, but the starting processes is done by a scheduler, so the client does not know ahead of time information such as addresses and ports that it can give the worker processes to connect to the master)
- Solution
  - Create a znode r as a rendezvous point
  - When master starts he fills the configuration in r
  - Workers watch node r

# Group Membership

- Create a znode g
- Each process create a znode under g in ephemeral mode: e. If the process fails or ends, the znode that represents it under zg is automatically removed.
- Watch g for group information
- Processes can obtain group information by simply listing the children of zg

# Simple Lock

- Create a znode l for locking
- If one gets to create l he gets the lock
- Others who fail to create watch l, waiting for the lock to be released
- A client releases the lock when it dies or explicitly deletes the znode.
- Problems: herd effect

# Simple Lock without herd effect

- We line up all the clients requesting the lock and each client obtains the lock in order of request arrival

**Lock**
```
1 n = create(l + "/lock-", EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event
6 goto 2
```

**Unlock**
```
1 delete(n)
```

# Read/Write Lock

**Write Lock**

```
1  n = create(l + "/write-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if n is lowest znode in C, exit
4  p = znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 2
```

**Read Lock**

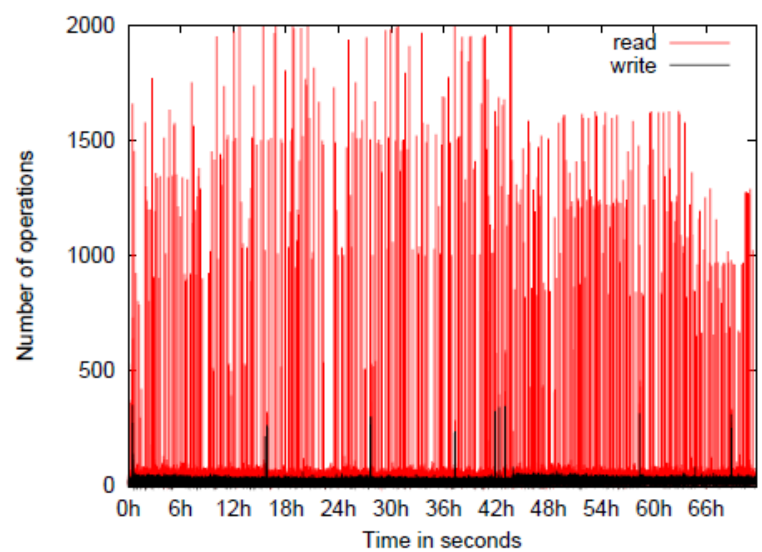
```
1  n = create(l + "/read-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if no write znodes lower than n in C, exit
4  p = write znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 3
```

# Double Barrier

- To synchronize the beginning and the end of computation
- Create a znode b, and every process needs to register on it, by adding a znode under b
- Set a threshold that starts the process

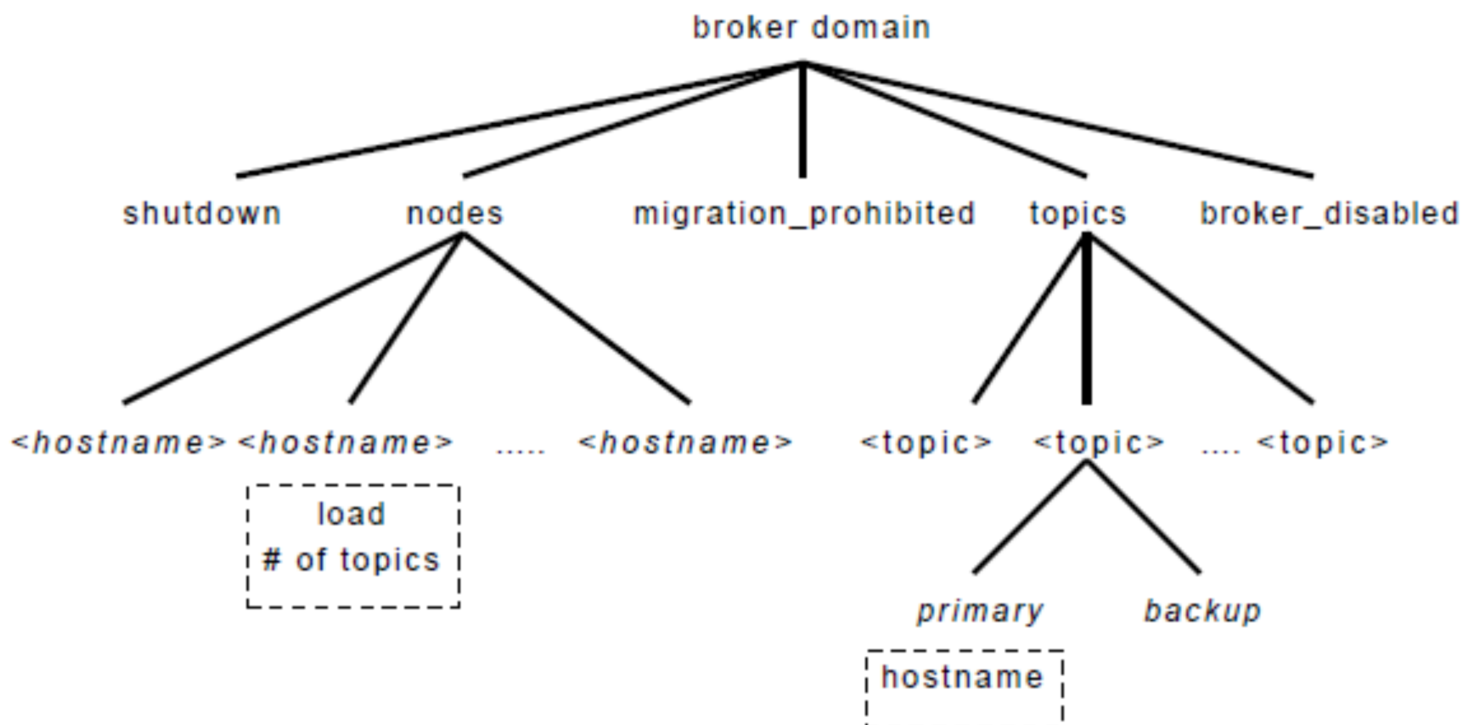
# Application

- Fetching Service
  - Using ZooKeeper for recovering from failure of masters
  - Configuration metadata and leader election



# Application

- Yahoo Message Broker
  - A distributed publish-subscribe system



# Implementation

- Provides high availability by replicating the ZooKeeper data on each server that composes the service
  - an in-memory database containing the entire data tree
- Servers fail by crashing, and such faulty servers may later recover
- Clients submit request:
  - Write requests require coordination among the servers; they use an agreement protocol (an implementation of atomic broadcast)
  - Read requests do not require coordination; , a server reads the state of the local database and generates a response to the request

# 3: Zab

# Zab

- It provides an important service for Zookeeper
- Atomic broadcast for primary-backup schemes
- Addresses the scenario when the primary (i.e the leader) fails
- Semantics
  - Primary order: similar but different from causal order
- Assumes that state changes are idempotent, i.e. applying the same state multiple times does not lead to inconsistencies
  - At least once semantics is enough

# Zab vs Group Communication

- Zab does borrow some concepts from group communication
- Group communication also uses the notion of VIEW – to define membership
  - View changes take place because of join/leave, process crashes and network partitions
- Zab uses VIEWs to identify leadership of primaries
  - View changes take place when a primary crashed or lost support from a quorum

# Other features

- Support for prefix of transactions submitted concurrently by a client are applied in FIFO order
- Fast recovery from primary crashes: allows the primary to identify the sequence of transactions to recover the application state
  - Does not need to reexecute orderings for pending transactions

# Process roles

- All process either Lead or Follow
- Followers
  - Maintain a history of transactions
- Leader
  - Can change
- Transactions are identified by <e, c>
  - e is the epoch number of the leader
  - c: epoch counter

# Properties of the PO Broadcast

- Integrity
  - Only broadcast transactions are delivered
  - Leaders recovers before broadcasting new transactions
- Total order
- Agreement
  - Followers deliver the same transaction and in the same order
- They are defined with respect to the leadership of a leader
  - Similar with the way such properties were defined in the context of Virtual Synchrony

# Primary Order

- Local order:
  - Order in which transactions are accepted by the leader
- Global order:
  - Defined by the order of epochs

# Zab

- Phase 0 – Leader election
  - Prospective leader L elected
- Phase 1 – Discovery
- Phase 2
  - Followers promise not to go back to previous epochs
  - Followers send to the leader L their last epoch and history
  - L selects longest history of latest epoch
- Phase 3 – Synchronization
  - Sends new history to followers
  - Followers confirm leadership
- Phase 3 – Broadcast
  - Proposes new transactions
  - Commits if quorum acknowledges

# Zab vs Paxos vs Viewstamped Replication

- Paxos, VSR, and Zab are three well-known replication protocols for asynchronous environments that admit bounded numbers of crash failures.
- Compute-intensive services are better off with a passive replication strategy, such as used in VSR and Zab (provided that state updates are of a reasonable size).
- To achieve predictable low-delay performance for short operations during both normal case execution and recovery, an active replication strategy without designated majorities, such as used in Paxos, is the best option.