ZooKeeper

Hunt et al. ZooKeeper: Wait-free coordination for Internet-scale systems. ATC 2010

Plan for today

- 1st hour: Discuss ZooKeeper
- last 30 minutes : 4m breakout room project chats
- Reminder:
    - Post your project ideas to piazza/slack.
    - **Proposal drafts due next Friday (Oc 9th)**

ZooKeeper + (Paxos and RSMs)

- ***What is*** the relationship between ZooKeeper and Paxos/RSMs?
- Uses something like Paxos (no details on how different)
- ABcast for ordering operations from leader to replicas
- *Paxos vs. Virtual synchrony (ABcast): ISIS*

ZooKeeper

- What's the provided abstraction?
- *"Coordination service" ~ "Creative File System service", "MultiCore Data Structure Service"*
  - Hierarchical tree of znodes with concurrency control
  - znodes are read in full, and written in full (atomic read/write operations)
  - Ephemeral znodes: depend on lifetime of clients; they are removed when the client session is terminated (always leaf nodes)
  - e-znodes exposes failure of the corresponding session to everyone
  - Create/delete/exist/getData/setData/getChildren/sync / *watch (callback)*

ZooKeeper

- Reads handled by node that client connected to
- Writes sent to leader, which distributes to followers using ABCast (ZAB)
- Reads may be *stale* (“wait free!”; multicore term)
- Ordering constraints using *zxid*
- Writes carry zxid and detect if operating on stale data

ZooKeeper

- Stale reads — good? Ok? Bad?
- ZK designed for read heavy workloads
- 80% reads => stale data chance is low
- Make up for stale reads with *sync*
- Design idea: build simple first, build for common case, more complexity can be optionally added on top (not used by all clients), *don't impose on all clients!*
- ZK is built for use by developers; make it easy for them use! And make it fast.
- Con: this is unexpected for people who assume a "file system" like thing
- *Doesn't work that great with heavy write workload*

ZooKeeper

- What can you build with it? (Layers of synchronization over some state)
- General abstraction (can do ***all*** the things): not as efficient as a more precise abstraction (e.g., lock server)
- Group membership (track who is in the group). Choose znode G for group. A node starts, creates an e-znode below G. Member leaves => e-znode deleted. Nodes can watch for changes/updates (e.g., nodes can watch an e-znode for the leader node)
- Config management: Store config in a znode C. Nodes watch C and detect changes. (Generalizes to hierarchical config)
- *Herd: group of nodes that all do the same thing. (All attempt to lock)*
- Lock herd effect management: sequence of watches where each node watches on a previous node's e-znode, which notifies them when they should do their operation (grab lock). Creates a ordered queue of nodes.
- *Note: all of these require a friendly developer that knows how to structure their application*

ZooKeeper

- Generic abstraction = microkernel for distributed systems?
- Pushes logic to clients/applications

ZooKeeper

- Implementation/design

- “Fuzzy snapshots” — but note, these are not distributed snapshots. Used for faster boot-up of new replicas that might have failed + replay message on top of the snapshot

- Idempotent operations — node translates an API call into an idempotent op before sending to leader (NFS style). Relax re-transmission guarantees: okay to retransmit ops.

- Write ahead logging for recovery (classic DB technique)

- Writes don’t return unless (1) majority nodes know about the write, (2) the write is reflected on disk at each of those nodes

- => rationale for separating *read and write paths*

ZooKeeper

- Evaluation
- More servers => closer ABcast => slower writes (lower write throughput)
- Fewer nodes => less potential for stale data
- *Staleness is a function of the network*
- Fewer nodes => less fault tolerance (can handle f out of 2f+1 failures: same as Paxos)
- Fewer nodes => slower reads (lower read throughput)
- Want: evaluate the stale reads — how often are read stales for different mixes of reads/writes

Next paper: PBFT

- *Practical* Byzantine fault tolerance system
- Another “big system” paper
- Handles byzantine faults!
- Influenced all future generations of BFT systems
- Barbara Liskov — Turing Award winner :-)

Project speed-dating

- I’ll create random breakout rooms, 2 people each for 4-5 min
  - First person presents their idea
  - I’ll send a global msg => signal to switch
  - Second person presents their idea
  - I’ll send a global msg => signal to switch
  - Mutual discussion
  - End break out room => rejoin global session
- Repeat, until end of class.