Paxos Made Live

An Engineering Perspective

Authors: Tushar Chandra, Robert Griesemer, Joshua Redstone

Presented By: *Dipendra Kumar Jha*

Consensus Algorithms

- Consensus: process of agreeing on one result among a group of participants
- Use a consensus algorithm to ensure that all replicas are mutually consistent
- Build an identical log of values on each replica
- Using “Paxos” to build a fault-tolerant database for Chubby

Chubby

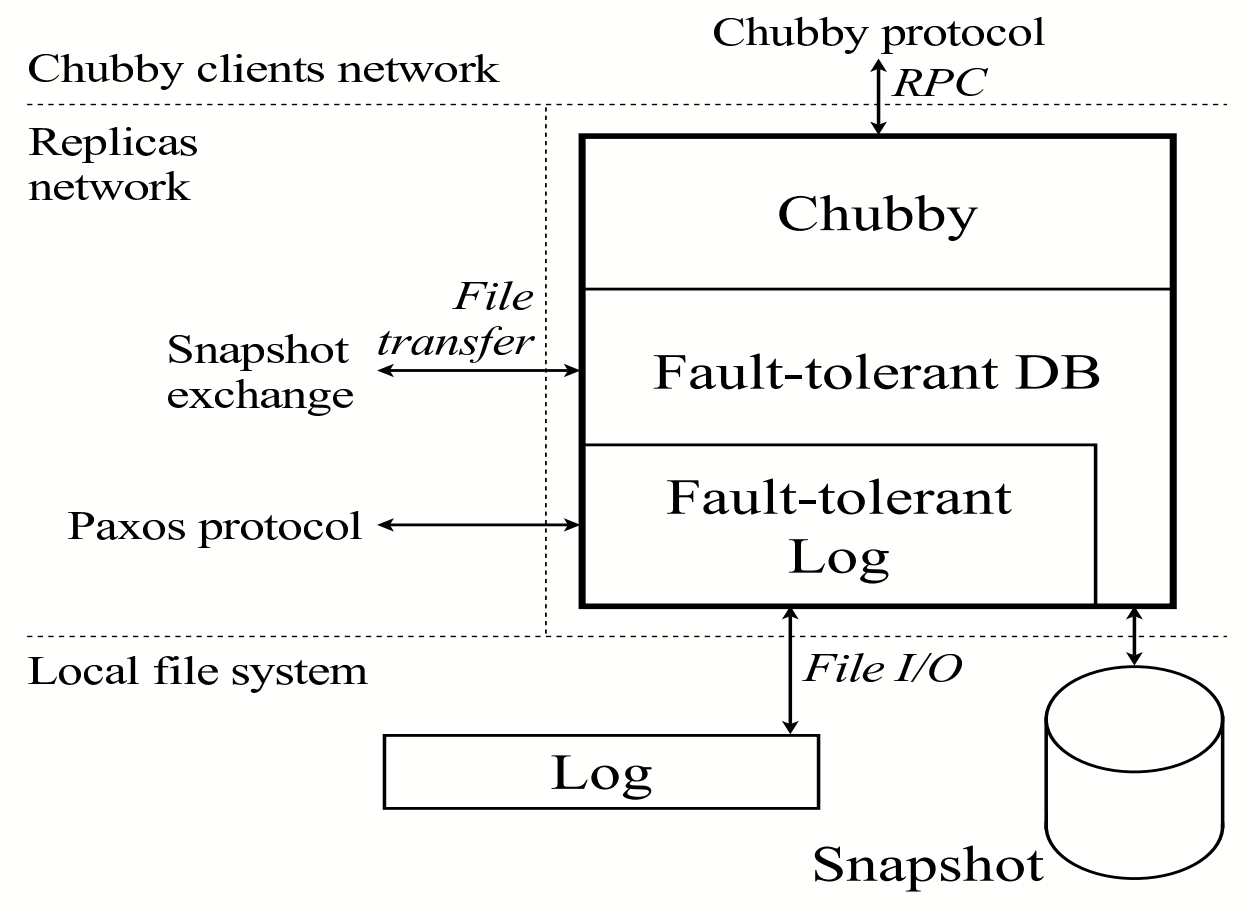

Figure 1: A single Chubby replica.

Paxos

Three phases:

1. Election of coordinator (Propose and Promise)
   - When a replica wants to become a coordinator, generates a unique sequence number higher than any it has seen
   - Broadcasts sequence number to all replicas in a *propose message*
   - If majority of replicas *promise (*reply and indicate they have not see a higher sequence number), then the replica acts as a coordinator
   - *Promise* message include the most recent value they have heard, if any, along with sequence number from whom they heard it

Paxos *(continued...)*

Three phases:

1. Accept and Acknowledge:
   - Coordinator selects a value and broadcasts it to all replicas in "*accept*" messages.
   - Other replicas either acknowledge or reject the message
3. Commit:
   - If majority replicas acknowledge, consensus is reached
   - Coordinator broadcasts a commit message to notify replicas.

- *There are multiple replicas working as coordinators at once.*

Multi-Paxos

- Use Paxos to achieve consensus on a sequence of values
- Chain together multipe *instance* (execution) of Paxos
- Use a *catch-up* mechanism for *lagging* replicas
- Each replica maintains a locally persistent log to record all Paxos actions for *catch-up*
- May select a *master* for a long time for optimization
- Propose messages may be omitted if the coordinator identity does not change between instances
- Possible to batch a collection of values submitted by different application threads into a single Paxos instance

Implementation of Paxos Algorithmic Challenges

- Handling disk corruption (*checksum, marker, catch-up*)
- Master leases (*Server up-to-date info for read operations*)
- Epoch numbers (*Detect master turnover and abort operations*)
- Group Membership (*Change in the set of replicas*)
- Snapshots (*Snapshot handle*)
- Database transactions(*MultiOp*)

Implementation of Paxos
Software Engineering

- Expressing the algorithm effectively
  - Simple state machine specification language
  - Compiler to translate specification to C++
- Runtime consistency checking
  - Assert statements, checksum requests
- Testing
  - Safety mode
  - Liveness mode
- Concurrency
  - Multithreading – database, to take snapshots, computer checksums and process iterators

Implementation of Paxos
Unexpected failures

- Rapid master failover due to high multi-threading
- Failure in upgrading Chubby cell
- Different semantics from Chubby
- One database replica different from others
- Failure in upgrade script for migrating cells from 3DB to Paxos version
- Failure due to bugs in the underlying operating system

Measurements

- Designed the tests to be write intensive
- If the contents of the file are small, the test measures latency of the system
- If the contents of the file are large, the test measures throughput of the system
- The system take snapshot whenever the replicated log size exceeds 100MB
- Performance comparison improves with size of file and number of workers
- Not optimized for performance, possible to make it faster

Summary and open problems

- Implementation of a fault-tolerant database
- Based on the Paxos consensus algorithm
- Significantly harder to build this system than anticipated
  - Significant gap between Paxos Algorithm and real-world system
  - No tools to implement fault-tolerant algorithms
  - Not enough testing criteria for building fault-tolerant systems
- Significant gap between theory and practice and not enough tools in fault-tolerant computing

Thank You

Questions???