Paxos Made Live

an engineering perspective

Authored by Tushar Chandra, Robert Griesemer, Joshua Redstone

Presented by John Hudson

Paxos made live

- Paper describes the engineering challenges faced building fault tolerant database on top of Paxos algorithm
- Description of their solutions
- Measurement and analysis of final system
- Thoughts on state of field

Why Paxos

- Want fault tolerance
  - resilience to hardware failures
  - maintain data integrity, liveliness of system
- Solution is replication
  - many machines, who cares if one goes down
- Another problem
  - how to ensure consistency
  - all replicas agree on stored values

Enter Paxos

- Solution to the replication problem
  - Consensus algorithm
  - Replicas agree on one value for each instance of algorithm
- Fault tolerant log
  - Paxos produces a consistent sequence of values across replicas
  - Build database on this log

Paxos

- Proposers: picks unique id N
  - Broadcasts three types of messages to acceptors
    - Propose – begins the consensus process, includes proposal id N
    - Accept – contains value and id (N) of proposal
    - Commit – informing acceptors that consensus has been reached
- Acceptors
  - Responds to proposers with two types of messages
    - Promise – upon receiving 'propose', respond with 'promise' to ignore accepts from proposers with id less than N, and value of previously highest id 'accept'
    - Accepted – upon receiving 'accept', responds with accepted if id equal or larger than previous 'promise' id.

Paxos algorithm

- Phase 1
  - Proposer broadcasts 'propose' messages with a unique id
  - Acceptor receives 'propose' messages
- Phase 2
  - Proposer receives 'promises'
  - Proposer broadcasts 'accept' messages with id, value
  - Acceptors respond with 'accepted'

Paxos algorithm

- Proposer waits for 'accepted' from majority of acceptors
    - Broadcasts 'commit' message

```
Proposer        Acceptor      Learner
   |            |   |   |        |   |
   |            |   |   |        |   |  Request
   X--------->|->|->|        |   |  Prepare(1)
   |<---------X--X--X        |   |  Promise(1,{Va,Vb,Vc})
   X--------->|->|->|        |   |  Accept!(1,Vn)
   |<---------X--X--X------>|->|  Accepted(1,Vn)
```

Paxos: in action

```
Proposer         Acceptor     Learner

  |             |  |  |       |  |
  |             |  |  |       |  |
  X------------>|->|->|       |  |  Prepare(1)
  |<------------X--X--X       |  |  Promise(1,{null,null,null})
  !             |  |  |       |  |  !! PROPOSER FAILS
     |          |  |  |       |  |  !! NEW PROPOSER (knows last number was 1)
     X--------->|->|->|       |  |  Prepare(2)
     |<---------X--X--X       |  |  Promise(2,{null,null,null})
  |  |          |  |  |       |  |  !! OLD PROPOSER recovers
  |  |          |  |  |       |  |  !! OLD PROPSOSER tries 2, denied
  X------------>|->|->|       |  |  Prepare(2)
  |<------------X--X--X       |  |  Nack(2)
  |  |          |  |  |       |  |  !! OLD PROPOSER tries 3
  X------------>|->|->|       |  |  Prepare(3)
  |<------------X--X--X       |  |  Promise(3,{null,null,null})
  |  |          |  |  |       |  |  !! NEW PROPOSER proposes, denied
  |  X--------->|->|->|       |  |  Accept!(2,Va)
  |  |<---------X--X--X       |  |  Nack(3)
  |  |          |  |  |       |  |  !! NEW PROPOSER tries 4
  |  X--------->|->|->|       |  |  Prepare(4)
  |  |<---------X--X--X       |  |  Promise(4,{null,null,null})
  |  |          |  |  |       |  |  !! OLD PROPOSER proposes, denied
  X------------>|->|->|       |  |  Accept!(3,Vb)
  |<------------X--X--X       |  |  Nack(4)
  |  |          |  |  |       |  |  ... and so on ...
```

Paxos algorithm

- Prepare broadcast will necessarily hear about previous consensus
- Guarantees only 1 value chosen.
  - Any majority overlaps with any other majority by at least 1 element
- That's good!

Chubby

- Chubby is Google's fault tolerant distributed lock and file system
  - Use replication to achieve fault tolerance
  - Built on 3DB replicated database
- 3DB's poor performance motivated new system
- Use Paxos to implement fault tolerant log, on which fault tolerant db and chubby will run

Chubby

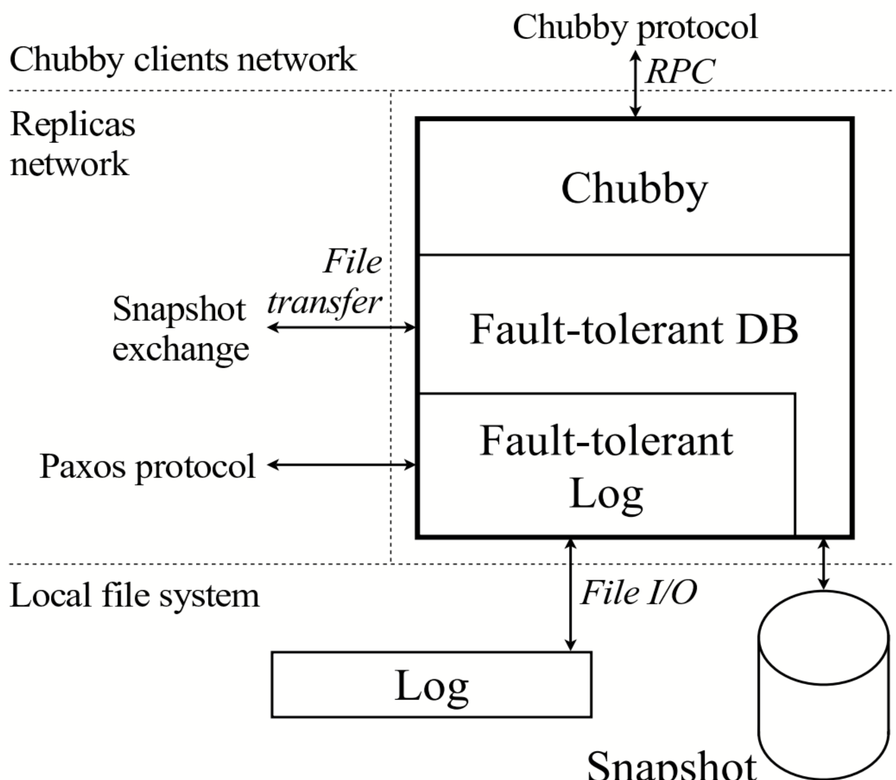

Figure 1: A single Chubby replica.

Challenges

- HD failure/corruption
- Master leases
- Group membership
- Snapshots, truncating log
- Database transactions

Challenges: HD failure

- File corruption. Checked with checksum
- File's inaccessible/deleted
  - In order to distinguish from new replica, have new replicas drop marker in GFS
  - Check GFS for marker, to determine if new or corrupted
- If corrupted, rebuild state, observe one complete Paxos round, so no promises are reneged.

Challenges: Improving read throughput

- Naive Paxos: reads require running a Paxos instance
    - Serialize read with respect to system updates
    - Too slow
- Solution: master lease
    - Only master can submit values to Paxos
    - Guarantees master has most up-to-date DB
    - Allows read to be served out of master's local DB

Challenges: growth of log

- Each completed Paxos instance contributes an entry to the log
    - In Chubby each entry is a DB operation
    - Datastructure that log entries apply to is application specific
- Solution: snapshot DB state and truncate log
- If recovering replica is too far behind to use available logs, can use snapshots, then logs

Challenges: Nature of Fault-tolerant systems

- By their nature mask failures
- Mis-configurations and bugs masked by fault tolerance
- How to distinguish between failure types?
- No systematic solution

Engineering Practices

- Implement transaction through Paxos
  - Submitted values are serialized wrt other Paxos instances
- Minimize concurrency
  - Repeatable tests, easier debugging
  - Eventually application requirements required more concurrency
- Periodically compute and compare replica DB checksums
- Described core algorithm in custom language that then compiled to C++
  - Core algorithm fit on 1 page

Measurements

- Paxos implementation is better than 3DB

Test	# workers	file size (bytes)	Paxos-Chubby (100MB DB)	3DB-Chubby (small database)	Comparison
Ops/s Throughput	1	5	91 ops/sec	75 ops/sec	1.2x
Ops/s Throughput	10	5	490 ops/sec	134 ops/sec	3.7x
Ops/s Throughput	20	5	640 ops/sec	178 ops/sec	3.6x
MB/s Throughput	1	8 KB	345 KB/s	172 KB/s	2x
MB/s Throughput	4	8 KB	777 - 949 KB/s	217 KB/s	3.6 - 4.4x
MB/s Throughput	1	32 KB	672 - 822 KB/s	338 KB/s	2.0 - 2.4x

Table 1: Comparing our system with 3DB (higher numbers are better).

Their Conclusions

- "There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system. In order to build a real-world system... needs to ... make several relatively small protocol extensions. The cumulative effort will be substantial and the final system will be based on an unproven protocol."

Distributed Systems

- Their criticisms:
  - Current tools are inadequate
  - Not enough focus on testing
- Written in 2007, since then:
  - CrystalBall, D3S
  - GO

Thoughts?

- A lot of their concerns have been addressed