Distributed Systems, Consensus and Replicated State Machines

Ali Ghodsi – UC Berkeley
alig(at)cs.berkeley.edu

What’s a distributed system?

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. “

Leslie Lamport

Two Generals' Problem

- Two generals need to coordinate an attack
  - Must agree on time to attack
  - They'll win only if they attack simultaneously
  - Communicate through messengers
  - Messengers may be killed on their way

Two Generals' Problem

- Lets try to solve it for general g1 and g2
- g1 sends time of attack to g2
  - Problem: how can g1 ensure g2 received msg?
  - Solution: let g2 ack receipt of msg, then g1 attacks
  - Problem: how can g2 ensure g1 received ack?
  - Solution: let g1 ack the receipt of the ack…
  - …
- This problem is impossible to solve!

Teaser: Two Generals' Problem

- Applicability to distributed systems
  - Two nodes need to agree on a value
  - Communicate by messages using an unreliable channel
- Agreement is a core problem…

How it all started

- Commit protocols for databases
  - Needed to implement Atomicity in ACID
  - Every node needs to agree on COMMIT/ABORT
- Two-phase commit known since 1976
  - Problem?
  - Centralized and blocking if coordinator fails

Bad news theorem

- After years of people solving the problem, bunch of authors proved that a basic version of it is impossible in most circumstances

Consensus: agreeing on a number

- Consensus problem
  - All nodes propose a value
  - Some nodes might crash & stop responding
- The algorithm must ensure:
  - All correct nodes eventually decide
  - Every node decides the same
  - Only decide on proposed values

Bad news theorem (FLP85)

- In an asynchronous system, even with only one failure, Consensus cannot be solved
  - In asynchronous systems, message delays can be arbitrary, i.e. not bounded
  - If cannot do it with 1 failure, definitely cannot do it with n>1 failures
  - Internet is essentially an asynchronous system
- But Consensus != Atomic Commit

Consensus is Important

- Atomic Commit
  - Only two proposal values {commit, abort}
  - Only decide commit if all nodes vote commit

- This related problem is even harder to solve than consensus
  - Also impossible in asynchronous systems ☹

Reliable Broadcast Problem

- Reliable Broadcast Problem
  - A node broadcasts a message
  - If sender correct, all correct nodes deliver msg
  - All correct nodes deliver same messages
- Very simple solution, works in any environment
  - Algo: Every node broadcast every message $O(N^2)$

Atomic Broadcast Problem

- Atomic Broadcast
  - A node broadcasts a message
  - If sender correct, all correct nodes deliver msg
  - All correct nodes deliver same messages
  - Messages delivered in the same order

Atomic Broadcast=Consensus

- Given Atomic broadcast
  - Can use it to solve Consensus. How?

- Every node broadcasts its proposal
  - Decide on the first received proposal
  - Messages received same order
    - All nodes will decide the same

- Given Consensus
  - Can use it to solve Atomic Broadcast. How?

- Atomic Broadcast equivalent to Consensus

Ali Ghodsi, alig@cs

Possibility of Consensus

- Consensus solvable in synchronous system with up to N/2 crashes
  - Synchronous system has a bound on message delay

- Intuition behind solution
  - Accurate crash detection
    - Every node sends a message to every other node
    - If no msg from a node within bound, node has crashed

- Not useful for Internet, how to proceed?

Modeling the Internet

- But Internet is mostly synchronous
  - Bounds respected mostly
  - Occasionally violate bounds (congestion/failures)
  - How do we model this?

- Partially synchronous system
  - Initially system is asynchronous
  - Eventually the system becomes synchronous

Ali Ghodsi, alig@cs

Failure detectors

- Let each node use a failure detector
  - Detects crashes
  - Implemented by heartbeats and waiting
  - Might be initially wrong, but eventually correct

- Consensus and Atomic Broadcast solvable with failure detectors
  - Obviously, those FDs are impossible too
  - But useful to encapsulate all asynchrony assumptions inside FD algorithm

Useless failure detectors

- How do we create a failure detector with no false-negatives?
  - i.e., never say a failed node is correct
- How do we create a failure detector with no false-positives?
  - i.e., never say a correct node is failed

Eventual Perfect FD

- Eventually perfect failure detector
  - Every failed node is eventually detected as failed
    - How to implement?
  - Eventually, no correct node is detected as failed
- Properties
  - Initially, all bets are off and the FD might output anything
  - Eventually, all nodes will not give false-positives

Failure detection and Leader Election

- Leader election (LE) is a special case of failure detection
  - Always suspect every node, but one correct node, as failed
- Implement LE with eventual perfect FD
  - How?
  - Pick highest ID **correct** node as leader

All problems solved

- With LE we can solve
  - Atomic Commit
  - Atomic Broadcast
  - Eventual Perfect Failure Detection
  - Consensus

- Consensus algorithm Paxos implemented with LE

One special failure detector

- Omega failure detector
  - Every failed node is eventually detected as failed
    - How to implement?
  - Eventually, at least one correct node will not be suspected as failed by any node

- Properties
  - Initially, all bets are off and the FD might output anything
  - Eventually, all nodes will not give false-positives w.r.t. at least one node

Failure Detection and Consensus

- Omega is the weakest failure detector needed to solve Consensus
  - Second most important results of distributed systems

RSM?

- So many problems
  - All interrelated
  - How should we build distributed systems?
- Replicated State Machine (RSM) approach
  - Model your application as an RSM
  - Replicate your RSM to N servers
  - Clients/users submits inputs to all servers
  - Servers run agree on the order of inputs
  - All servers will have the same state and output

RSM (2)

- Advantage of RSM?
  - Make any application trivially fault tolerant
- Distributed file system example
  - Each server implements a filesystem
  - Each input (read/write) run through consensus
  - Voila: fault tolerant FS

Paxos vs RSM

- Paxos vs RSM?
  - Use Paxos to agree on input order to RSM

- Adapting Paxos for RSM
  - Paxos takes 2 round-trips, but RSM optimizations make it 1 round trip, how?
  - Prepare phase doesn't need actual values, run Prepare for thousands of inputs at once
  - How do you add/remove nodes to RSM? (Reconfig)
  - Use Paxos to agree on set of nodes in the system

Raft vs Paxos

- Paxos initially for Consensus
  - Easy to understand correctness, but harder o know how to implement the algorithm
  - Need all optimizations to make it perfect for RSM
  - Need to implement reconfiguration on-top
  - A family of Paxos algorithms

Raft vs Paxos

- Raft purpose made algorithm for RSM
  - Less declarative, more imperative
  - Leader election, leader replicates log
  - Supports reconfiguration
  - Many implementations

Summary

- Distributed systems are hard to build
  - Failures
  - Parallelism
- Many problems interrelated
  - Consensus, Atomic Commit
  - Atomic Broadcast
  - Leader election, Failure detection
- Replicated state machine
  - Generic approach to make any DS fault-tolerant
  - Can be implemented with Paxos or Raft
  - Raft more straight forward to implement