6.852 Lecture 24, part 1

- Paxos (continued)
- Reading:
  - Lamport: The Part-Time Parliament
- Part 2: Self-stabilization
Paxos consensus algorithm

• Consensus in asynchronous network
  – impossible if a single process may fail
  – need to solve for real applications
    • weaken requirements
• Strategy: “safe” protocol, contingent termination
  – guarantee validity and agreement always
  – guarantee termination if system “stabilizes”
    • no more failures, recoveries, message losses
    • time for message delivery/process steps within “normal” bounds
  – termination should be fast when system is stable
    • only need system to be stable long enough to terminate
Paxos consensus protocol

- Paxos algorithm implements replicated state machine
  - tolerates stopping failures/recoveries, message loss/duplication
- Heart of Paxos algorithm is “synod” consensus protocol
  - use consensus to agree on sequence of steps
    - as in Herlihy's wait-free universal construction from consensus
Paxos consensus protocol

- Ballot: \((b,d) \in BId \times V \cup \{ \bot \}\)
  - an attempt to reach consensus
  - \(V\) is consensus domain, \(d\) is “decree” (a value or nothing yet)
  - ballot created by any process at any time (restrict later)
    - new ballot must have new id, initially no associated value (i.e., \(\bot\))
    - value assigned later, satisfying certain conditions
  - ballot ids totally ordered
  - process may vote for or abstain from a ballot (but not both)
    - can abstain from sets of ballots, including ones not yet initiated
  - ballot **succeeds** if a write quorum votes for it
  - ballot is **dead** if a read quorum abstains from it
    - read quorum has nonempty intersection with every write quorum
Paxos consensus protocol

- Each ballot processed in three phases of messages
  - initiate new ballot, choose decree for ballot (need read quorum)
  - try to get ballot to succeed (need write quorum to vote)
  - let everyone know if successful
- Initiator “drives” processing of ballot
  - other processes only respond to messages from initiator
- Anyone can ignore/neglect any ballot at any time
  - only affects progress
- Many ballots can be processed concurrently
  - ballots can be initiated at any time
  - ballots with larger ids are “later”
Paxos consensus protocol

• Phase 1:
  – NextBallot(b), where b not previously used ballot id
    • sent by some process p to some read quorum (or more)
  – LastVote(b,v), sent by q to p in reply to NextBallot(b) from p
    • v is vote by q with largest ballot id smaller than b (null if none)
    • q promises not to vote for (i.e., abstains from) ballots with ids between v's and b's (must keep track of abstentions).
  – p selects value when it gets a read quorum of responses
    • decree of latest ballot that had a vote (among LastVote responses)
    • if all LastVote responses are null, choose own decree
Paxos consensus protocol

• Phase 2:
  – BeginBallot(b,d), where d is determined in Phase 1
    • sent by p to a write quorum (or more)
  – Voted(b,q), sent by q to p in reply to BeginBallot(b,d) from p
    • q must not have abstained from b (by LastVote for some other ballot)
  – p decides on d if it gets a write quorum of votes (i.e., responses)

• Phase 3
  – Success(d), sent by p to everyone
    • p can terminate after sending if channels are reliable
  – any process decides on d upon receiving Success(d) from anyone
    • can it terminate if channels are reliable?
Paxos consensus protocol

- Communication pattern for a ballot
  - like 3-phase commit

initiate ballot

select decree

succeed

Phase 1, collect abstention information

Phase 2, collect votes

Phase 3, propagate decision
Paxos consensus protocol

- Recall:
  - ballot **succeeds** if a write quorum votes for it
  - ballot is **dead** if a read quorum abstains from it
  - read quorum has nonempty intersection with every write quorum
    - no ballot can be both dead and successful
- Lemma: For initiated ballots \((b,d)\) and \((b',d')\), if \(b > b'\), then either \(d = d'\) or \(b'\) is dead.
Modifying the ** condition for assigning ballot values

- Instead of checking:
  
  ** For every $b' < b$, either $\text{val}(b') = v$ or $b'$ is dead.

- Check the apparently-weaker condition:
  
  *** Either:
  
  Every $b' < b$ is dead, or there exists $b' < b$ with $\text{val}(b') = \text{val}(b)$, and such that every $b''$ with $b' < b'' < b$ is dead.

- *** is easier to check in a distributed algorithm (will show how).
- And *** implies **, by easy induction on the number of steps in an execution.
Ensuring ***

*** Either every $b' < b$ is dead, or there exists $b' < b$ with $\text{val}(b') = \text{val}(b)$, such that every $b''$ with $b' < b'' < b$ is dead.

• Phase 1:
  – Originator process $i$ tells other processes the new ballot number $b$.
  – Each recipient $j$ abstains from all smaller-numbered ballots it hasn’t yet voted for.
  – Each $j$ sends back to $i$:
    • The largest ballot number $< b$ that it has ever voted for, if any, together with its value $v$.
    • Else a message saying there is no such ballot.
  – When originator $i$ collects this information from a read-quorum $R$, it assigns a value $v$ to ballot $b$:
    • If anyone in $R$ says it voted for a ballot $< b$, then $v = \text{the value associated with the largest-numbered of these ballots}$.
    • If not, $v = \text{any initial value}$.

• Claim this choice satisfies ***:
Ensuring ***

- *** Either every $b' < b$ is dead, or there exists $b' < b$ with $\text{val}(b') = \text{val}(b)$, such that every $b''$ with $b' < b'' < b$ is dead.

- Why does this choice satisfy ***?

- Case 1: Someone in R says it voted for a ballot $< b$.
  - Say $b'$ is the largest such ballot number.
  - Then everyone in R has abstained from all ballots between $b'$ and $b$.
  - So, choosing $\text{val}(b) = \text{val}(b')$ ensures the second clause of ***.

- Case 2: Everyone in R says it did not vote for a ballot $< b$.
  - Then everyone in R has abstained from all ballots $< b$, ensuring they are all dead.
  - Satisfies the first clause of ***.
Paxos consensus protocol

• Protocol requires:
  – ballot id for new ballot has never been used
  – not voting for ballots previously abstained from
  – remembering previous votes (for LastVote)

• Simplify by restricting processes further:
  – ballot id is sequence number plus process id (to break ties)
  – remember largest b sent in LastVote(b,v)
    • never vote for ballots with ids less than b
    • also ignore NextBallot(b') when b' ≤ b
  – remember only latest ballot voted for (ballot id and decree)
    • send in response to NextBallot (if not ignored)
Liveness

• To guarantee termination when the system stabilizes, we must restrict its nondeterminism. Say that process initiates ballot in response to BallotTrigger.

• Most importantly, must restrict when BallotTrigger so that, after stabilization:
  – It asks only one process to start ballots (leader).
  – It doesn’t tell the leader to start new ballots too often—allows enough time for ballot to complete.

• E.g., BallotTrigger might:
  – Use knowledge of “normal case” time bounds to try to detect who is failed.
  – Choose smallest-index non-failed process as leader (refresh periodically).
  – Tell the leader to try a new ballot every so often—allows enough “normal case” message delays to finish the protocol.

• Note the BallotTrigger uses time—not purely asynchronous.

• But we know we can’t solve the problem otherwise.

• Algorithm tolerates inaccuracies in BallotTrigger: If it “guesses wrong” about failures or delays, termination may be delayed, but safety properties are still guaranteed.
Replicated state machines

- Paper also deals with repeated consensus, in particular, on a sequence of operations for a replicated state machine.
- Use infinitely many instances of Paxos to agree on first operation, second, third,…
- Strategy similar to Herlihy’s universal construction, which uses repeated consensus to decide on successive operations for an atomic object.
- Lamport’s paper also includes various optimizations, LTTR.
- Considerable follow-on work, engineering Paxos to work for maintaining real data.
  - Disk Paxos
  - HP, Microsoft, Google,…