COM3026: Distributed Systems

Paxos

Gregory Chockler

Consensus with Partial Synchrony

- Partially (Eventually) Synchronous model:
  - The timing assumptions hold eventually
  - Captures the idea that realistic systems are synchronous most of the time
- Abstracted via an Eventually Perfect Failure Detector (◇P)
  - Strong Completeness and Eventual Strong Accuracy
- ◇P makes it possible to eventually elect a stable leader
  - A correct process that is permanently trusted by all correct processes
  - Eventual Leader Detector Ω

Eventual Leader Detector (Ω)

- **Module**:
  - Name: EventualLeaderDetector, instance Ω
- **Events**:
  - **Indication**: $\langle \Omega$, Trust | p$\rangle$: Indicates that process p is trusted to be leader
- **Properties**:
  - **ELD1** (*Eventual Accuracy*)
  - **ELD2** (*Eventual Agreement*)

Eventual Leader Detector (Ω)

- **Properties**:
  - **ELD1** (*Eventual Accuracy*)
    - There is a time after which every correct process trusts some correct process
  - **ELD2** (*Eventual Agreement*)
    - There is a time after which no two correct processes trust different correct processes

Problem Statement

- Solve Uniform Consensus
- Fail-noisy model
  - Crash-stop failure model for processes
  - Eventual Leader Detector Ω
  - Perfect Point-to-Point links
- N processes, up to a minority $f < N/2$ can fail
  - There is a correct majority (Majority Quorum System)

The Paxos Algorithm

- Solves Uniform Consensus in a fail-noisy model with a Eventual Leader Detector and correct majority
- Arguably, the most important algorithm in distributed computing
- Was initially proposed by Leslie Lamport in late 80s and further developed by Lamport and others over several decades
  - Hugely influential in practice — virtually all modern data replication systems have a variant of Paxos in their core
- Our presentation follows “Paxos Made Simple” (Lamport, 2001) with some further simplifications

Leslie Lamport

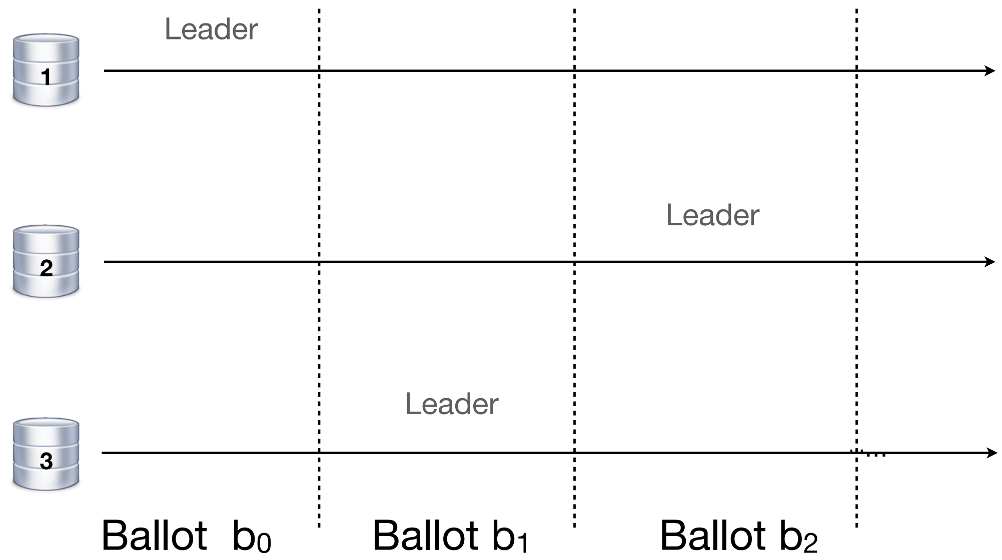
Paxos Consensus (Synod, Single-Decree)

Execution is divided into a series of ballots

Each ballot has a unique number and is associated with a unique leader

At each ballot, the ballot leader tries to reach a decision

If a value V is decided at ballot b, any decision at ballot b’ > b must be V



$b_0 < b_1 < b_2 < \ldots$

Ballot b_0

- All processes start in a default ballot b_0 with a default leader (e.g., p_1)

Ballot b_0

- Leader of ballot b_0 broadcasts a value val to all processes and decides val
  - Does this work?

Ballot b_0

- Leader of ballot b_0 broadcasts a value val to all processes and decides val
    - Does this work? - No
    - If the leader fails, may not reach the leader of ballot b_1

Ballot b_0

- Leader of ballot b_0 broadcasts a value val to all processes and waits for acks
    - How many acks should the leader wait for?

Ballot b_0

- Leader of ballot b_0 broadcasts a value val to all processes and waits for acks
  - How many acks should the leader wait for?

Ballot b_0

- Leader of ballot b_0 broadcasts a value to all processes and waits for acks
- $N - f$ is the maximum number of acks the leader may hope to receive
- The leader of ballot b_1 will be able to discover val by contacting $N - f$ nodes, if any two sets of $N - f$ nodes intersect (i.e., $N - f$ is a quorum)
- $N - 2f > 0 \implies N > 2f$
- This is a necessary condition for solving crash fault-tolerant consensus under partial synchrony and crash failures [DLS88]
- We assume for simplicity: $N = 2f + 1$

Ballot b_0: Accepting a Value

1. Leader of ballot b_0 broadcasts $\langle$ACCEPT, b_0, $val\rangle$ to all processes and waits for acks ($\langle$ACCEPTED, $b_0\rangle$)

2. If a process receives $\langle$ACCEPT, b_0, $val\rangle$, while in ballot b_0, it stores
   - a_bal := b_0
   - a_val := val

3. and responds with $\langle$ACCEPTED, $b_0\rangle$ to the leader

4. Once the leader receives $f+1$ $\langle$ACCEPTED, $b_0\rangle$, it decides val, and broadcasts $\langle$COMMITTED, $val\rangle$

5. Whenever a process receives $\langle$COMMITTED, $val\rangle$, it decides val

Ballot b > b_0: General Case

- The processes store the current ballot in bal
- The leader of ballot b broadcasts ⟨PREPARE, b⟩
- If a process receives ⟨PREPARE, b⟩, and its current ballot bal < b:
  - bal := b //join ballot b
  - send ⟨PREPARED, a_bal, a_val⟩ to the leader of b //to allow the leader to learn about values accepted in the prior ballots
  - From this point onward, the process will not be participating in any ballot < b
- If the leader receives $f + 1$ ⟨PREPARED, a_bal_i, a_val_i⟩ responses, it selects the value a_val_j associated with the highest ballot a_bal_j, and broadcasts ⟨ACCEPT, b, val⟩ where val=a_val_j
  - If all a_val_i=null, the leader broadcasts the value provided in its propose request
- Proceed with the steps 2 - 4 of the accept protocol with b_0 replaced with b

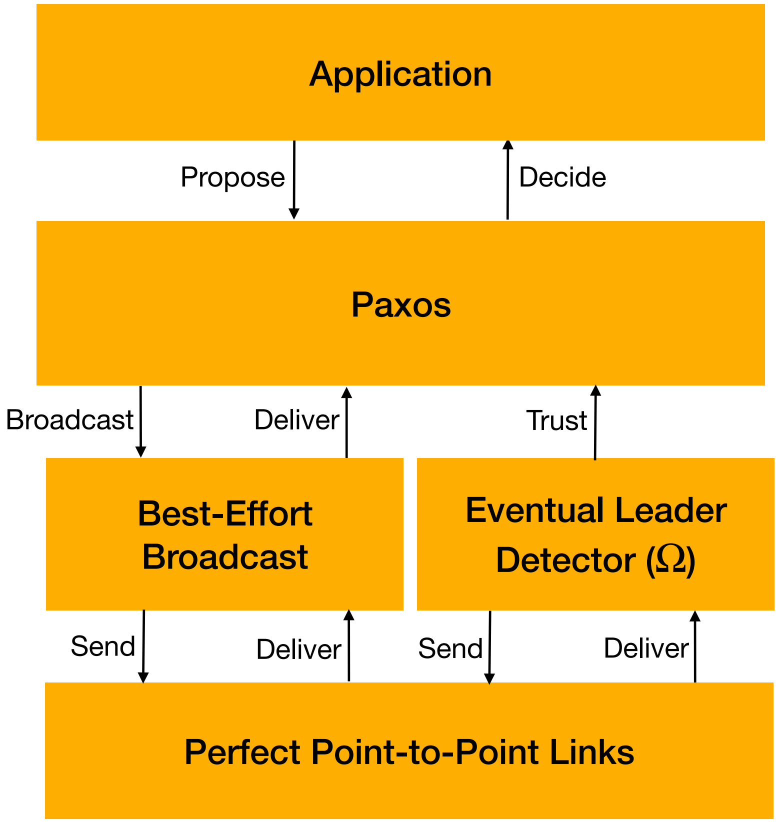
Paxos Consensus: Agreement

- Sequence of ballots in the execution: $b_0 < b_1 < b_2 < \ldots$
- Let b_k be the minimum ballot in which some value val_k is decided
- Prove that for all $l \geq k$: if the leader of v adopts a value val at ballot b_l, then $val = val_k$
- By induction on $l \geq k$:
  - $l = k$: the ballot leader can commit at most one value

Paxos Consensus: Agreement

- By induction on $l \geq k$:
  - Assume that the result holds for all l up to $l = i$, and consider $l = i + 1$
  - The leader of b_{i+1} will hear from at least one process p^* that committed val_k in b_k (quorum intersection)
  - $\Longrightarrow$ the highest ballot $b' \geq b_k$
  - By the inductive hypothesis, val' associated with b' satisfies $val' = val_k$

Component Structure



Sketch of the Leader Protocol

upon event <Ω, *Trust* | *p*> **do**

leader := *p*

while *self* = *leader* ∧ decided = FALSE **do**

- Prepare phase: choose a value *V*, lock majority
- Accept phase: convince a majority to accept V
- If successful: **trigger** <*Decide* | *V*>; *decided* := TRUE

Paxos core

ABORTABLE CONSENSUS

if *decided* = TRUE **then**

trigger <*beb*, *Broadcast* | *V*>

Abortable Consensus Properties

- At all times, at most one proposed value can be chosen by any process
  - Safety: Agreement and Validity (Integrity is achieved via *decided*)
- Succeeds once Ω converges to a permanent correct leader
  - Liveness: Termination

- Common pattern followed by most fail-noisy (partially synchronous) algorithms:
  - Guarantee safety at all times
  - Guarantee liveness once the system becomes stable

Abortable Consensus: Ballot b

Leader

(1) Broadcast (prepare, b) to all processes

(3) **upon** quorum S of (prepared, b, a_bal, a_val) **do**:
    If all a_val = null:
        V= v_0;
    else
        V := a_val with the highest a_bal in S;
    Broadcast (accept, b, V)

(5) **upon** quorum S of (accepted, b) **do**:
    return V

If received (nack, b) while waiting for a quorum in either (3) or (5), **return** ABORT

All Processes

(2) **upon** (prepare, b) from p **do**
    if b > bal **then**
        bal := b
        send (prepared, b, a_bal, a_val) to p
    else
        send (nack, b) to p

(4) **upon** (accept, b, v) from p **do**:
    if b ≥ bal **then**
        bal := b
        (a_bal, a_val) := (b, v)
        send (accepted, b) to p
    else
        send (nack, b) to p

Complete Leader Protocol

Initially:

leader := ⊥; *myBallot* := b_0; decided := FALSE; bal=b_0; a_bal := b_0; a_val :=null; v_0 := value of the Propose request

upon event <Ω, *Trust* | *p*> **do**

leader := *p*

upon *self* = *leader* ∧ decided = FALSE **do**

- *myBallot* := *b* such that *b* is unique and *b* > *myBallot*
- Start **Abortable Consensus** for ballot *myBallot*
- If returned *V* ≠ ABORT: **trigger** <*beb*, *Broadcast* | [DECIDE, *V*]>

upon event <beb, Deliver | [DECIDE, *V*]> such that decided = FALSE **do**

decided := TRUE

trigger <*Decide* | *V*>

Correctness: Termination

- Eventually some correct process *p* is permanently trusted as a leader by all correct processes
- *p* will be the only process executing the code triggered by the condition *self* = *leader* ∧ decided = FALSE
- Since *myBallot* keeps increasing, eventually, p calls Abortable Consensus with a ballot which is > than the value of *bal* of any process
- Once this happens, Abortable Consensus is guaranteed to return V ≠ ABORT causing p to decide V and broadcast V to all processes
- Since p is correct V will eventually reach all correct processes causing them to decide V as well

Further Reading

- "Paxos Made Simple" (Lamport, 2001)
  - A copy is available on the module's web site
- Adapted to the standard setting of the N-process Consensus problem
  - No separate proposer, acceptor, and learner roles
  - Every process can be viewed as simultaneously playing all these three roles

Conclusions

- Paxos is a widely used protocol for solving Uniform Consensus in a partially synchronous system
    - Fail-noisy model with Eventual Leader Detector
    - Tolerates up to a minority $f < N/2$ of process crashes
- Abortable Consensus is the core of Paxos
    - Never violates Agreement, but may fail to reach a decision (abort)
- Terminates once Abortable Consensus is called with a sufficiently high ballot number by a permanently trusted correct leader
    - This will happen once the system becomes synchronous

Next

- Total-order broadcast