Info

• Quiz 5 will be Tuesday Nov. 10, 2015
• It will cover Byzantine Agreement and Paxos
  • I will provide pseudocode for both algorithms

• After quiz, we will discuss specs for project 2
  • Please prepare by brainstorming how you will use Paxos in the distributed Calendar application.
Some Requirements for Project 1

• No conflicting appointment should be inserted into the log.
  • So there is no need for a conflict resolution protocol.

• Every event should appear in the log (at most) once.
  • Single event for all participants.

• No delete appointment event should appear in the log unless the create appointment event precedes it.
  • Do we really need/want this?
PAXOS
The Problem

- We need to store a log of events (decrees)

| Event 1 | Event 2 | Event 3 | Event 4 | Event 5 | …. |

- Want to replicate this log at multiple sites.
  - Need the event to appear in the same order in every log.
  - Need every event to eventually appear in every log.

- Challenges
  - Concurrency – different processes want to write to log at same time
  - Failures – processes and channels
System Model

- Processes may fail by crashing and may restart.
  - Both crash failures and crash/recovery processes.
  - Processes do not lose state when they crash and recover.
  - No Byzantine failures.

- Messaging is asynchronous.
  - Messages may take arbitrarily long to be delivered.
  - Messages can be duplicated.
  - Messages can be lost.
  - Messages cannot be corrupted.
The Paxos Algorithm

• Initial approach: consider the replicated log problem as a sequence of consensus problems, one per log position.

| Event 1 | Event 2 | Event 3 | Event 4 | Event 5 | ... |

• The algorithm used to reach consensus on a single log entry is called the **Synod Algorithm**.

• Later: show how to make this initial approach more efficient – this is **Paxos**.
Roles in the Consensus Algorithm

- **Proposers**: propose value to chosen (the consensus value)
- **Acceptors**: decide whether to accept a proposed value
- The value is **chosen** when enough acceptors accept it.
  - Enough = a majority
- **Learners**: must all learn the consensus value (after it is chosen)
  - For our discussion, Acceptors = Learners
Requirements for Consensus Algorithm

• Safety
  • Only a value that has been proposed may be chosen (decided as consensus value).
  • Only a single value is chosen.
  • A process never learns that a value has been chosen unless it actually has been (no changing a decision).

• Liveness
  • No precise requirement
  • Goal is that eventually some proposed value is chosen, and then eventually a process can learn the chosen value.
Algorithm Properties

• **(P1)** An acceptor must accept the first value it receives.

• Why?
  • We want algorithm to work well when there are no competing proposals and no failures.
Algorithm Properties (2)

- Suppose two values are proposed (2m+1 total acceptors).
  - One is accepted by m acceptors.
  - The other is accepted by m+1 acceptors.
  - Single failure can make it impossible to learn which value was chosen.

- An acceptor must be able to accept more than one proposal.

- Keep track of different proposals by assigning each a unique natural number.
Dealing with Multiple Proposals

• A value is chosen when a single proposal with that value has been accepted by a majority of acceptors.
  • Also say the proposal has been chosen.

• Multiple proposals can be chosen, but in that case, they must all have the same value.

• **(P2)** If a proposal with value v is chosen, then every higher numbered proposal that is also chosen has value v.
How to satisfy (P2)?

- **(P2)** If a proposal with value \(v\) is chosen, then every higher numbered proposal that is also chosen has value \(v\).

- **(P2a)** If a proposal with value \(v\) is chosen, then every higher numbered proposal accepted by any acceptor has value \(v\).

- (P2a) implies (P2).
Combining (P1) and (P2a)

- **(P1)** An acceptor must accept the first value it receives.
- **(P2a)** If a proposal with value v is chosen, then every higher numbered proposal accepted by any acceptor has value v.

To satisfy (P1) and (P2a), we must strengthen (P2a).

- **(P2b)** If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.
- Clearly (P2b) implies (P2a)
How to satisfy (P2b)?

• If a new proposal issued, how can we choose a “safe value” for it?
  • If no value has yet been chosen, then any value is safe.
  • If a value has been chosen, then new proposal must use this value.

• How can we tell if no value has been chosen yet?
  • Look at set S consisting of majority of acceptors.
  • If none have accepted a proposal, then no value has been chosen.
  • So new proposal can use a new value.
  • If some have accepted a proposal, then a value may have been chosen.
How to satisfy (P2b)?

- If a new proposal issued, how can we choose a “safe value” for it?
  - If no value has yet been chosen, then any value is safe.
  - If a value has been chosen, then new proposal must use this value.

- If proposal was chosen, a majority of acceptors accepted it.
  - So we can pick any set S consisting of a majority of acceptors.
  - At least one of them will have accepted the chosen value.
  - But some may have accepted other values.
  - Which value do we choose?
How to satisfy (P2b)?

- (P2b) If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

- (P2c) For any v and n, if proposal number n is issued with value v, then there is a majority set S such that either
  - (a) No accepter in S has accepted any value OR
  - (b) v is the value of the highest numbered proposal with number less than n that was accepted by any acceptor in S.

- (P2c) implies (P2b)
  - Why?


Final Requirements

• (P1) An acceptor must accept the first value it receives.

• (P2c) For any \( v \) and \( n \), if proposal number \( n \) is issued with value \( v \), then there is a majority set \( S \) such that either
  • (a) No accepter in \( S \) has accepted any value OR
  • (b) \( v \) is the value of the highest numbered proposal with number less than \( n \) that was accepted by any acceptor in \( S \).

• How to translate these to an algorithm?
Algorithm for Proposer

• When a proposer wants to propose a value.
  • Picks a new proposal number n
  • Sends “prepare” message to set of acceptors requesting:
    • It promise not to accept any proposals with lower numbers than n.
    • It respond with highest number proposal (less than n) that is has accepted.

• If proposer receives response from majority of acceptors, it issues a proposal (with a value) and its number n.
  • Picks its own value if no responder has accepted a value.
  • Otherwise, picks value with largest proposal number from responders.
  • Sends “accept” request message with value and n to set of acceptors.
Algorithm for Acceptor

- Acceptor can receive two types of requests “prepare” and “accept”.

- When can it respond to a “prepare” request?
  - Respond to “prepare” if request number is greater than that of any prepare request it has responded to.

- When can it accept a value (in response to an “accept” request)?
  - If and only if it has not responded to a “prepare” request with a larger proposal number.
Synod Algorithm Implementation

- Proposers are stateless
  - Safety is not violated if a proposer crashes.
  - If proposer recovers, it can start over.
  - To ensure liveness, we need a proposer to stay alive “long enough”.

- Each acceptor stores
  - maxPrepare: largest proposal number for which it has responded to a prepare message (initially 0)
  - accNum: largest proposal number of proposal it has accepted (initially null)
  - accVal: value of proposal numbered accNum (initially null)

- The acceptor state must survive crash/recovery
Execution of Synod Algorithm (1)

1. Choose new proposal number \( m \).
   Send \( \text{prepare}(m) \) to all acceptors.

2. If \( m > \text{maxPrepare} \), reply with \( (\text{accNum}, \text{accVal}) \).

3. If receive response from majority, choose value \( v \), send \( \text{accept} \) to all acceptors.
   Else, start over.

4. If \( m \geq \text{maxPrepare} \), record accept:
   \( \text{accNum} = m \)
   \( \text{accVal} = v \)

\( v \) = value with largest \( \text{accNum} \) number.

Only if all \( \text{accVal} \) values are \( \text{null} \), choose own value.
• A value is chosen when a majority of acceptors accept it.
• How can the acceptors (learners) determine when this has happened?
Learning the Accepted Value

- A value is chosen when a majority of acceptors accept it.
- How can the acceptors (learners) determine when this has happened?

- Only once a value has been chosen can it be written in the log.

- With message loss, there is no guarantee that all acceptors will eventually learn the decided value.
Execution of Synod Algorithm (2)

4. Record $v$ in log.

5. Send $\text{ack}(\text{accNum}, \text{accVal})$ after saving $\text{accNum}$ and $\text{accVal}$

6. If receive $\text{ack}$ from majority, send $\text{commit}(v)$
Correctness of Synod Algorithm

- Safety is guaranteed by algorithm construction.

- What about liveness?
  - Do a majority of acceptors eventually accept a value?
  - Even if acceptors do not fail and messages are not lost?

- Multiple proposers can keep issuing prepare requests with higher numbers.
  - Possible no proposal gets majority of “promises”.
Ensuring Progress

• Can ensure liveness by having a distinguished proposer.
  • All proposer funnel proposals through this distinguished proposer.
  • Distinguished proposer issues proposals one at a time.

• Use leader election algorithm to elect this distinguished process.
Full Paxos Algorithm

- The Synod algorithm is used to determined consensus value for a single log entry.

- The Paxos algorithm is a sequence of Synod algorithms.
  - With some optimizations.
Full Paxos Algorithm

- Use leader election algorithm of your choice to elect a leader (to act as distinguished proposer).

- When leader elected, it may have missing log entries.
  - It runs Synod algorithm (acts as proposer and acceptor) to learn consensus value for those entries.
  - Uses a dummy value for these entries.

- Once holes in log are filled, leader can start processing new requests to create log entries.
Paxos Optimization

- If leader is stable, only need to do “prepare-promise” phase once (when first elected)
  - Until leader’s first value is accepted.

- After first value is accepted, leader just uses “accept-ack-commit” phase for subsequent values.
Another Reference