Paxos

- Distributed consensus protocol
  - Group decision making by majority voting
    - $F+1$ out of $2F+1$ can make the decision
    - Can handle $F$ node failures
    - Can handle network failures
    - Can handle network delays (reordering)
  - Once decision is made the decision does not change
    (write-once register concept)

- Paxos: reaching consensus on ONE case
- Multi-Paxos: extension to multiple cases

Original Intended Use of Paxos

- State machine approach
  - Gets user input then changes state and emits output
  - Record of input can represent current state
Original Intended Use of Paxos

• State machine approach with replication
  – All replicas should be in the same state
  – All inputs should be given in the same order
• Making decisions for one value is not easy

Sources of Paxos Confusion

• The part-time parliament
  – Described in a form of a story of Aegean island of Paxos
  – Annotation from the journal:
    “The author appears to be an archeologist with only a passing
    interest in computer science.”
  – Paper submitted in 1990, accepted in 1998

• Paxos made simple, 2001
  – Abstract:
    “The Paxos algorithm, when presented in plain English,
    is very simple.”
  – Maybe too simplified
Sources of Paxos Confusion

• Paxos made simple paper
  – Clearly described
    • Acceptor
    • Proposer
  – Vaguely described
    • Leader
    • Learner
    • Membership change
    • Unique proposal number

Modular Paxos

• Focus on the essence
  – Implement only clearly described parts
    • Acceptor
    • Proposer
    • Learner implemented as proposer
  – Ignore optimizations
    • Performance may be moderate
    • Liveness guarantee can be an issue

* Optimizations can be added later as needed
Overall Design

Paxos

• Phase 1
  – Proposer: sends prepare message with proposal #
  – Acceptors: record proposal # as promised # if proposal # > promised #
  – Acceptors: ack with promised #, accepted # and values

• Phase 2
  – Proposer: from received acks,
    • Count the # of promises (promised # matches proposal #)
    • Find the value with highest accepted #
      (If a value is found, proposer MUST use this value for the proposal)
  – Proposer: if # of promise >= # of majority of acceptors, send accept message with proposal # and value
  – Acceptor: accepts if proposal # >= promised #
  – Acceptor: ack with accept/rejected message
  – Proposer: if majority responded accepted, value is chosen. Otherwise, not chosen.
Paxos Algorithm
Proposer/Acceptor

**Proposer**

1. Prepare Request
   1. Send ($P_p$)

2. Handle prepare ack
   1. If $(A_a, V_a)$ exists and $A_a > A_p$ then $A_p = A_a$ and $V_p = V_a$
   2. If $P_p = P_a$ then Promised_cnt++

3. Accept request
   1. If Promised_cnt >= majority then
      Send ($P_p, V_p$)

4. Handle accept ack
   1. If Accepted then Accepted_cnt++

5. Decide success/failure
   1. If Accepted_cnt >= Majority then Success else Failure

**Acceptor**

(total= $2F+1$; majority = $F+1$)

1. Handle prepare
   1. If $P_p > P_a$ then $P_a = P_p$

   2. Ack with $(P_a, (A_a, V_a))$

2. Handle accept
   1. If $P_p = P_a$ then $A_a = P_p, V_a = V_p$

   2. Ack with (Accepted/Rejected)

Implementation

**Proposer client (This will become Write)**

```java
// 1. Prepare request and get ack
For (number of acceptors) {
    Send ($P_p$)
    RecvAck(ack)
}
// 2. Handle prepare ack
Foreach (ack) {
    If $(A_a, V_a)$ exists and $A_a > A_p$ then $A_p = A_a$ and $V_p = V_a$
    If $P_p = P_a$ then Promised_cnt++
}
// 3. Accept request and get ack
If Promised_cnt >= majority then
    for (number of acceptors) {
        Send ($P_p, V_p$)
        RecvAck(ack)
    }
else return;
// 4. Handle accept ack
Foreach (ack) {
    If Accepted then Accepted_cnt++
}
// 5. Decide success/failure
If Accepted_cnt >= Majority then Success else Failure
```

**Acceptor server: main**

```java
While [true] {
    AcceptConnections(socket);
    AcceptConnections(thread_body, args);
}
```

**Acceptor server: thread_body**

```java
While [Receive(message)] {
    Switch (message.type) {
        Case Prepare:
            If $P_p > P_a$ then $P_a = P_p$
            Ack with Send($P_a, (A_a, V_a))$
        Case Accept:
            If $P_p = P_a$ then $A_a = P_p, V_a = V_p$
            Ack with Send(Accepted/Rejected)
    }
}
```
Paxos Algorithm
Learner (Almost same as proposer)

\(Pp: \text{proposal #}; (Vp: \text{Value}; Ap: \text{Accepted #})\)

1. Prepare Request
   1. Send \((Pp)\)

2. Handle prepare ack
   1. If \((Aa, Va)\) exists and \(Aa > Ap\) then \(Ap = Aa\) and \(Vp = Va\)
   2. If \(Pp == Pa\) then \(\text{Promised_cnt}++\)

3. Accept request
   1. If \(\text{Promised_cnt} >= \text{majority}\) then
      Send \((Pp, Vp)\)

4. Handle accept ack
   1. If \(\text{Accepted}\) then \(\text{Accepted_cnt}++\)

5. Decide success/failure
   1. If \(\text{Accepted_cnt} >= \text{Majority}\) then \(\text{Success}\)
      else \(\text{Failure}\)

Accepter (total= \(2F+1\); majority = \(F+1\))

\(Pa: \text{Promised #}; Aa: \text{Accepted #}; Va: \text{Value}\)

1. Handle prepare
   1. If \(Pp > Pa\) then \(Pa = Pp\)
   2. Ack with \((Pa, (Aa, Va))\)

2.5 Learn value
   1. For all \((Aa, Va)\) pairs find the majority pair
   2. If found then return the majority \text{Value} else read \text{Failed}

\(\text{and Vp contains a value}\)

2. Ack with Accepted/Rejected

Extending to Multi-Paxos

- Create an array of acceptor states

- Add an address to every message
  - Send(Pp) -> Send\((\text{addr}, Pp)\)
  - Send(Pp, Vp) -> Send\((\text{addr}, Pp, Vp)\)
  - Ack(Pa, Aa, Va) -> Ack\((\text{addr}, Pa, Aa, Va)\)
  - Ack(Accept/Reject) -> Ack\((\text{addr}, \text{Accept/Reject})\)

- Add log order writing
Overall Design

Replicated state machine

Proposer/Learner

Acceptor 0

Acceptor 1

Acceptor 2

APIs:
- Read(addr, output value)
- Write(addr, input value)
- Log(input value)

Implementation
- Read: triggers Paxos learner
- Write: triggers Paxos proposer
- Log: triggers Paxos learner and then proposer

Status and Ongoing Direction

- Status
  - Code for Paxos ready in C
  - Compiles in Linux with gcc and CompCert C

- To do
  - Create read/write wrapper APIs
  - Make multi-paxos as a logging layer
  - Divide the code into verifiable layers
  - Write specifications and verifications
  - Port to CertiKOS

- Unknowns
  - Missing storage and networking stack in CertiKOS
  - Some Paxos optimizations that may be needed for liveness proofs
Example

Paxos

• Proposer (Pp = 3, Vp = A)

• Acceptor 1 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 3 (Pa = -1, Aa = -1, Va = Nil)
Paxos

- Proposer (Pp = 3, Vp = A)
  Prepare(Pp = 3)

- Acceptor 1 (Pa = -1, Aa = -1, Va = Nil)
- Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
- Acceptor 3 (Pa = -1, Aa = -1, Va = Nil)
Paxos

- Proposer \((P_p = 3, V_p = A)\)

- Acceptor 1 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  
  Ack\((P_a=3, A_a=-1, V_a=\text{Nil})\)

- Acceptor 2 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  
  Ack\((P_a=3, A_a=-1, V_a=\text{Nil})\)

- Acceptor 3 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  
  Ack\((P_a=3, A_a=-1, V_a=\text{Nil})\)
Paxos

• Proposer (Pp = 3, Vp = A)

• Acceptor 1 (Pa = 3, Aa = 3, Va = A)  
  Ack(Accepted)

• Acceptor 2 (Pa = 3, Aa = 3, Va = A)  
  Ack(Accepted)

• Acceptor 3 (Pa = 3, Aa = 3, Va = A)  
  Ack(Accepted)
Paxos

• Proposer (Pp = 3, Vp = A)
  Value Chosen!

• Acceptor 1 (Pa = 3, Aa = 3, Va = A)

• Acceptor 2 (Pa = 3, Aa = 3, Va = A)

• Acceptor 3 (Pa = 3, Aa = 3, Va = A)
Paxos

• Proposer 1 (Pp = 3, Vp = A)

• Acceptor 1 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 3 (Pa = -1, Aa = -1, Va = Nil)

Paxos

• Proposer 1 (Pp = 3, Vp = A)
  Prepare(Pp=3)

• Acceptor 1 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
  Network down

• Acceptor 3 (Pa = -1, Aa = -1, Va = Nil)
Paxos

• Proposer 1 (Pp = 3, Vp = A)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

  Network down

• Acceptor 3 (Pa = 3, Aa = -1, Va = Nil)

Paxos

• Proposer 1 (Pp = 3, Vp = A)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)

  Ack(Pa=3, Aa=-1, Va=Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

  Network down

• Acceptor 3 (Pa = 3, Aa = -1, Va = Nil)

  Ack(Pa=3, Aa=-1, Va=Nil)
Paxos

• Proposer 1 (Pp = 3, Vp = A)
  Accept(Pp=3, Vp=A)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)
  Network down

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
  Network down

• Acceptor 3 (Pa = 3, Aa = -1, Va = Nil)

Paxos

• Proposer 1 (Pp = 3, Vp = A)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)
  Network down

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
  Network down

• Acceptor 3 (Pa = 3, Aa = 3, Va = A)
Paxos

- Proposer 1 (Pp = 3, Vp = A)

- Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)
  Network down

- Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
  Network down

- Acceptor 3 (Pa = 3, Aa = 3, Va = A)
  Ack(Accepted)

Paxos

- Proposer 1 (Pp = 3, Vp = A)
  Failed!

- Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)
  Network down

- Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)
  Network down

- Acceptor 3 (Pa = 3, Aa = 3, Va = A)
Paxos

• Proposer 2 (Pp = 4, Vp = B)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 3 (Pa = 3, Aa = 3, Va = A)
  Network down

Paxos

• Proposer 2 (Pp = 4, Vp = B)
  Prepare(Pp=4)

• Acceptor 1 (Pa = 3, Aa = -1, Va = Nil)

• Acceptor 2 (Pa = -1, Aa = -1, Va = Nil)

• Acceptor 3 (Pa = 3, Aa = 3, Va = A)
  Network down
Paxos

- Proposer 2 ($P_p = 4$, $V_p = B$)

- Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)

- Acceptor 2 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)

- Acceptor 3 ($P_a = 3$, $A_a = 3$, $V_a = A$)

  Network down

---

Paxos

- Proposer 2 ($P_p = 4$, $V_p = B$)

- Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)

  Ack($P_a = 4$, $A_a = -1$, $V_a = Nil$)

- Acceptor 2 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)

  Ack($P_a = 4$, $A_a = -1$, $V_a = Nil$)

- Acceptor 3 ($P_a = 3$, $A_a = 3$, $V_a = A$)

  Network down
Paxos

• Proposer 2 ($P_p = 4$, $V_p = B$)
  
  Accept($P_p=4$, $V_p=B$)

• Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)
  
  Network down

• Acceptor 2 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)

• Acceptor 3 ($P_a = 3$, $A_a = 3$, $V_a = A$)
  
  Network down

Paxos

• Proposer 2 ($P_p = 4$, $V_p = B$)

• Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = Nil$)
  
  Network down

• Acceptor 2 ($P_a = 4$, $A_a = 4$, $V_a = B$)

• Acceptor 3 ($P_a = 3$, $A_a = 3$, $V_a = A$)
  
  Network down
Paxos

- Proposer 2 \((Pp = 4, Vp = B)\)

- Acceptor 1 \((Pa = 4, Aa = -1, Va = Nil)\)
  
  *Network down*

- Acceptor 2 \((Pa = 4, Aa = 4, Va = B)\)
  
  *Ack(Accepted)*

- Acceptor 3 \((Pa = 3, Aa = 3, Va = A)\)
  
  *Network down*

Paxos

- Proposer 2 \((Pp = 4, Vp = B)\)
  
  *Failed!*

- Acceptor 1 \((Pa = 4, Aa = -1, Va = Nil)\)
  
  *Network down*

- Acceptor 2 \((Pa = 4, Aa = 4, Va = B)\)

- Acceptor 3 \((Pa = 3, Aa = 3, Va = A)\)
  
  *Network down*
Paxos

• Proposer 3 \((Pp = 5, Vp = C)\)

• Acceptor 1 \((Pa = 4, Aa = -1, Va = \text{Nil})\)
  
  **Network down**

• Acceptor 2 \((Pa = 4, Aa = 4, Va = B)\)

• Acceptor 3 \((Pa = 3, Aa = 3, Va = A)\)

---

Paxos

• Proposer 3 \((Pp = 5, Vp = C)\)
  
  `Prepare(Pp=5)`

• Acceptor 1 \((Pa = 4, Aa = -1, Va = \text{Nil})\)
  
  **Network down**

• Acceptor 2 \((Pa = 4, Aa = 4, Va = B)\)

• Acceptor 3 \((Pa = 3, Aa = 3, Va = A)\)
Paxos

- Proposer 3 (Pp = 5, Vp = C)

- Acceptor 1 (Pa = 4, Aa = -1, Va = Nil)
  
  Network down

- Acceptor 2 (Pa = 5, Aa = 4, Va = B)

- Acceptor 3 (Pa = 5, Aa = 3, Va = A)

Paxos

- Proposer 3 (Pp = 5, Vp = C)

- Acceptor 1 (Pa = 4, Aa = -1, Va = Nil)
  
  Network down

- Acceptor 2 (Pa = 5, Aa = 4, Va = B)
  
  Ack(Pa=5, Aa=4, Va=B)

- Acceptor 3 (Pa = 5, Aa = 3, Va = A)
  
  Ack(Pa=5, Aa=3, Va=A)
Paxos

- Proposer 3 (\(P_p = 5\), \(V_p = B\))

- Acceptor 1 (\(P_a = 4\), \(A_a = -1\), \(V_a = \text{Nil}\))
  Network down

- Acceptor 2 (\(P_a = 5\), \(A_a = 4\), \(V_a = B\))

- Acceptor 3 (\(P_a = 5\), \(A_a = 3\), \(V_a = A\))

Accept(\(P_p = 5\), \(V_p = B\))
Paxos

- Proposer 3 \((Pp = 5, Vp = B)\)

- Acceptor 1 \((Pa = 4, Aa = -1, Va = Nil)\)
  - Network down

- Acceptor 2 \((Pa = 5, Aa = 5, Va = B)\)
  - Ack(Accepted)

- Acceptor 3 \((Pa = 5, Aa = 5, Va = B)\)
  - Ack(Accepted)

---

Paxos

- Proposer 3 \((Pp = 5, Vp = B)\)
  - Value Chosen!

- Acceptor 1 \((Pa = 4, Aa = -1, Va = Nil)\)
  - Network down

- Acceptor 2 \((Pa = 5, Aa = 5, Va = B)\)

- Acceptor 3 \((Pa = 5, Aa = 5, Va = B)\)