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

Proposer/Learner
Replicated state machine

Proposer/Learner
Replicated state machine

Accepter 0

Accepter 1

Accepter 2

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. Other wise, not chosen.
Paxos Algorithm

Proposer

Proposer client (This will become Write)

```java
// 1. Prepare request and get ack
For (number of acceptors) {
    Send (Pp)
    RecvAck(ack)
}

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

// 3. Accept request
If Promised_cnt >= majority then
    Send (Pp, Vp)

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

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

Acceptor

Acceptor server: main

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

Acceptor server: thread_body

```java
While (Receive(message)) {
    Switch (message.type) {
        Case Prepare:
            If Pp > Pa then Pa = Pp
            Ack with Send(Pa, [Aa, Va])
        Case Accept:
            If Pp >= Pa then Aa = Pp, Va=Vp
            Ack with Send(Accepted/Rejected)
    }
}
```

Implementation
Paxos Algorithm

Learner (Almost same as proposer)

*Pp*: proposal #; *(Vp): Value; Ap: 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 *Promised_cnt++*

3. Accept request
   1. If *Promised_cnt >= majority* then
      Send *(Pp, Vp)*

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

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

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

*Pa*: Promised #; *Aa: Accepted #; Vp: 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 *Value*
      else read *Failed*

and Vp contains a value

Extending to Multi-Paxos

• Create an array of acceptor states

• Add an address to every message
  – *Send(Pp) → Send(addr, Pp)*
  – *Send(Pp, Vp) → Send(addr, Pp, Vp)*
  – *Ack(Pa, Aa, Va) → Ack(addr, Pa, Aa, Va)*
  – *Ack(Accept/Reject) → Ack(addr, Accept/Reject)*

• Add log order writing
**Overall Design**

Replicated state machine

Proposer/Learner

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

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

Acceptors:
- Acceptor 0
- Acceptor 1
- Acceptor 2

**Example**
Paxos

- Proposer \((P_p = 3, V_p = A)\)
- Acceptor 1 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
- Acceptor 2 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
- Acceptor 3 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)

Paxos

- Proposer \((P_p = 3, V_p = A)\)
  \(\text{Prepare}(P_p = 3)\)
- Acceptor 1 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
- Acceptor 2 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
- Acceptor 3 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
Paxos

• Proposer \((P_p = 3, V_p = A)\)

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

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

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

• Acknowledgment: \(P_a = 3, A_a = -1, V_a = \text{Nil}\)

• Acknowledgment: \(P_a = 3, A_a = -1, V_a = \text{Nil}\)

• Acknowledgment: \(P_a = 3, A_a = -1, V_a = \text{Nil}\)
Paxos

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

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

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

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

Value Chosen!
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 \((P_p = 3, V_p = A)\)

• Acceptor 1 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  \[\text{Ack}(P_a = 3, A_a = -1, V_a = \text{Nil})\]
• Acceptor 2 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
  \[\text{Network down}\]
• Acceptor 3 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  \[\text{Ack}(P_a = 3, A_a = -1, V_a = \text{Nil})\]

Paxos

• Proposer 1 \((P_p = 3, V_p = A)\)
  \[\text{Accept}(P_p = 3, V_p = A)\]

• Acceptor 1 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
  \[\text{Network down}\]
• Acceptor 2 \((P_a = -1, A_a = -1, V_a = \text{Nil})\)
  \[\text{Network down}\]
• Acceptor 3 \((P_a = 3, A_a = -1, V_a = \text{Nil})\)
<table>
<thead>
<tr>
<th>Paxos</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Proposer 1 (Pp = 3, Vp = A)</td>
</tr>
</tbody>
</table>
| • 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) |

<table>
<thead>
<tr>
<th>Paxos</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Proposer 1 (Pp = 3, Vp = A)</td>
</tr>
</tbody>
</table>
| • 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

Network down
Paxos

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

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

- Acceptor 2 ($P_a = 4$, $A_a = -1$, $V_a = \text{Nil}$)
  
  Ack($P_a=4$, $A_a=-1$, $V_a=\text{Nil}$)

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

Network down

Paxos

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

  Accept($P_p=4$, $V_p=B$)

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

- Acceptor 2 ($P_a = 4$, $A_a = -1$, $V_a = \text{Nil}$)

- 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)

• Acceptor 3 (Pa = 3, Aa = 3, Va = 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 ($P_p = 4$, $V_p = B$)
  Failed!

• Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = \text{Nil}$)
  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 3 ($P_p = 5$, $V_p = C$)

• Acceptor 1 ($P_a = 4$, $A_a = -1$, $V_a = \text{Nil}$)
  Network down
• Acceptor 2 ($P_a = 4$, $A_a = 4$, $V_a = B$)

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

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

- 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)
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 (Pp = 5, Vp = B)

• 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 = B)
  Accept(Pp=5, Vp=B)

• 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 = 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)