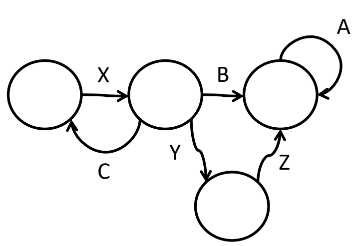
Paxos

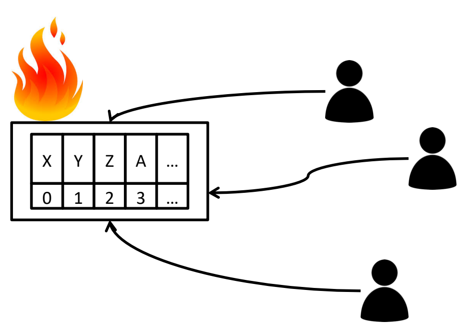
- Distributed consensus protocol
  – Group decision making by majority voting
    - *F+1* out of *2F+1* can make the decision
    - Can handle *F* node failures
    - Can handle network failures
    - Can handle network delays (reordering)
  – Once decision is made the decision does not change (write-once register concept)
- Paxos: reaching consensus on ONE case
- Multi-Paxos: extension to multiple cases

1

Original Intended Use of Paxos

- State machine approach
  – Gets user input then changes state and emits output
  – Record of input can represent current state

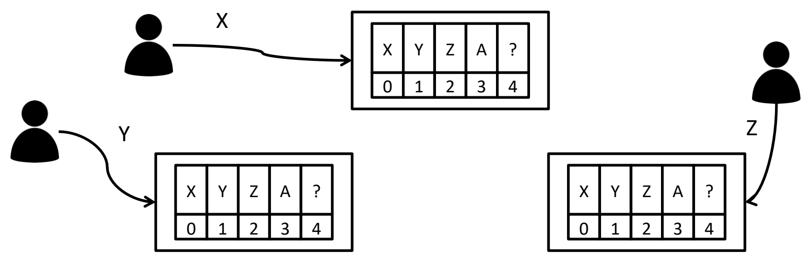




2

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



3

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

4

Sources of Paxos Confusion

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

Most Important Core

Everybody invents their own mechanisms

5

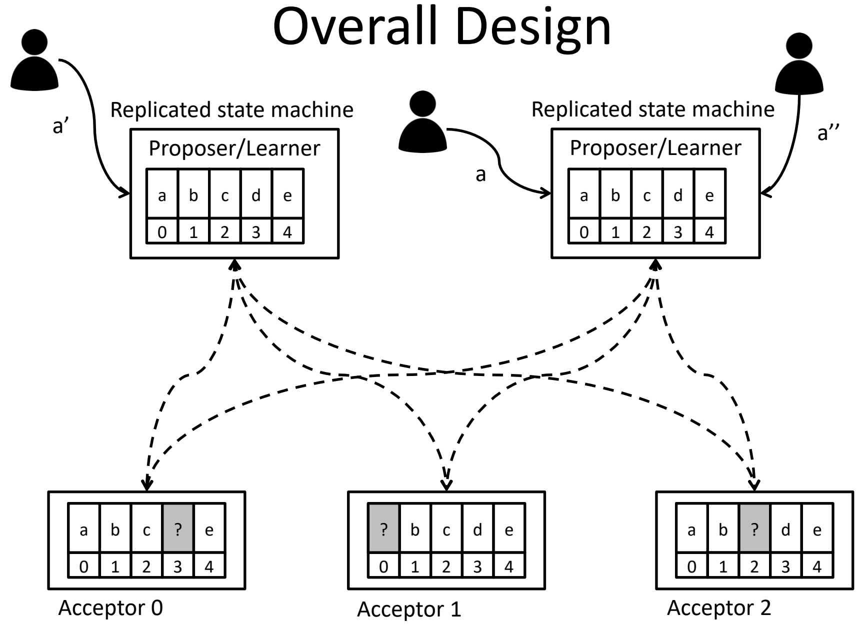
Modular Paxos

- 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

6

Overall Design



7

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.

8

Paxos Algorithm Proposer/Acceptor

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**

Acceptor (total= *2F+1; majority = F+1*)
***Pa*: Promised #; *Aa*: Accepted #; *Va*: Value**

1. Handle prepare
   1. If ***Pp > Pa*** *then* ***Pa = Pp***
   2. Ack with (***Pa***, (***Aa, Va***))
2. Handle accept
   1. If ***Pp >= Pa*** then ***Aa = Pp, Va=Vp***
   2. Ack with (***Accepted/Rejected***)

9

Implementation

Proposer client (This will become Write)

```
// 1. Prepare request and get ack
For (number of acceptors) {
        Send (Pp)
        RecvAck(ack)
}
// 2. Handle prepare ack
Foreack (acks) {
       If (Aa, Va) exists and Aa > Ap then Ap = Aa and Vp = Va
       If Pp == Pa then Promised_cnt++
}
// 3. Accept request and get ack
If Promised_cnt >= majority then
       for (number of acceptors) {
              Send (Pp, Vp)
              RecvAck(ack)
       }
else return;
// 4. Handle accept ack
Foreach (acks) {
       If Accepted then Accepted_cnt++
}
// 5. Decide success/failure
If Accepted_cnt >= Majority then Success
else Failure
```

Acceptor server: main

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

Acceptor server: thread_body

```
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)
        }
}
```

10

Paxos Algorithm
Learner

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**

Acceptor (total= *2F+1; majority = F+1*)
***Pa*: Promised #; *Aa*: Accepted #; *Va*: 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

cept
1. If ***Pp*** >= ***Pa*** then ***Aa = Pp, Va=Vp***
2. Ack with ***Accepted/Rejected***

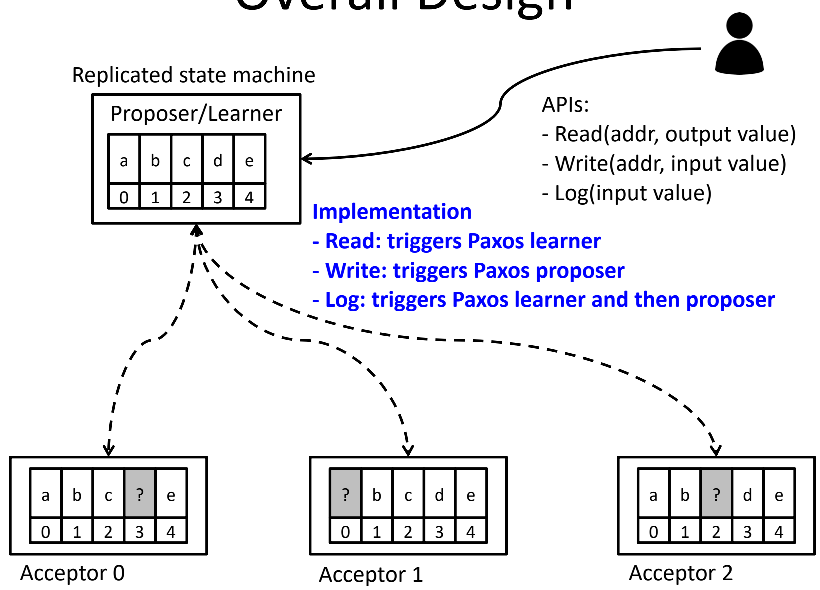
11

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

12

Overall Design



13

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

14

Example

15

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)

16

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)

17

Paxos

- Proposer (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)

18

Paxos

- Proposer (Pp = 3, Vp =A)

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

19

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)

20

Paxos

- Proposer (Pp = 3, Vp =A)

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

21

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)

22

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)

23

24

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)

25

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)

26

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)

27

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)

28

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)

29

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)

30

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)

31

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)

32

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

33

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

34

Paxos

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

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

- Acceptor 2 (Pa = 4, Aa = -1, Va = Nil)

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

  Network down

35

Paxos

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

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

  Ack(Pa=4, Aa=-1, Va=Nil)
- Acceptor 2 (Pa = 4, Aa = -1, Va = Nil)

  Ack(Pa=4, Aa=-1, Va=Nil)
- Acceptor 3 (Pa = 3, Aa = 3, Va = A)

  Network down

36

Paxos

- Proposer 2 (Pp = 4, Vp =B)
  Accept(Pp=4, Vp=B)
- Acceptor 1 (Pa = 4, Aa = -1, Va = Nil)
  Network down
- Acceptor 2 (Pa = 4, Aa = -1, Va = Nil)
- Acceptor 3 (Pa = 3, Aa = 3, Va = A)
  Network down

37

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

38

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

39

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

40

Paxos

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

- 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)

41

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)

42

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)

43

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)

44

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)

45

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)

46

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)

47

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)

48