Consensus Algorithms
Quiz 2 grades/solutions released

HW2 released - due next Thursday!
Paxos

Citation: Google Tech Talks video on Paxos
Paxos

What do we mean by consensus?

- Consensus is on **one** value.
- Consensus is reached once a **majority** of participants agree.
- Once a consensus is reached, everyone can **eventually** know the result.
- Participants are happy to reach consensus on **any** result, not just the one they propose.
- Communication channels are **not** perfect (messages may be lost).
“Let’s see Finding Nemo”

“Alice”

“Sure, Finding Nemo is great”

“Bob”

“What about Mission Impossible”

“What about Mission Impossible”

“Ugh, OK fine”

“Charlie”

“Finding Nemooooooooo”

“Debbie”

“OK, I guess it’s Finding Nemo”

“Erica”
“Let’s see Finding Nemo”

Alice

“Sure, Finding Nemo is great”

Bob

“Finding Nemooooooooo”

Debbie

“Ugh, OK fine”

Charlie

 Erica

“Yes, Finding Nemo!”

Fred

“Finding Nemoooooo”

Debbie

“What about Mission Impossible”

Charlie
Paxos defines three different roles for the nodes in the system:

- **Proposers**
  - These propose values for consensus.

- **Acceptors**
  - These “vote” on proposals and form the majority.

- **Learners**
  - These record whatever the acceptors have accepted as the decision.

Decisions must be persistent. Nodes must know how many acceptors there are.

Note: we’re talking about them separately, but in practice any single machine can play any number of roles simultaneously.
**Paxos**

- **Proposer** picks a proposal ID ($ID_p$) and sends a PREPARE $ID_p$ message to **Acceptors**
  - $ID_p$ must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher $ID_p$
- **Acceptor** receives PREPARE request. Did it promise to ignore requests with $ID_p$?
  - If yes, then ignore request
  - If no, then promise to ignore $ID < ID_p$ (and send PROMISE $ID_p$ in reply)

- **Proposer** gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST ($ID_p$, value) to **Acceptors**.
  - value can be anything
- **Acceptor** receives ACCEPT_REQUEST. Did it promise to ignore $ID_p$?
  - If yes, then ignore request
  - If no, reply ACCEPT ($ID_p$, value) and also send to **Learners**
**Paxos**

- **Proposer** picks a proposal ID ($ID_p$) and sends a PREPARE $ID_p$ message to **Acceptors**
  - $ID_p$ must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher $ID_p$
- **Acceptors** receive PREPARE request. Did it promise to ignore requests with $ID_p$?
  - If yes, then ignore request
  - If no, then promise to ignore $ID < ID_p$ (and send PROMISE $ID_p$ in reply)

- **Proposer** gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST ($ID_p$, value) to **Acceptors**.
  - value can be anything
- **Acceptors** receive ACCEPT_REQUEST. Did it promise to ignore $ID_p$?
  - If yes, then ignore request
  - If no, reply ACCEPT ($ID_p$, value) and also send to **Learners**
**Paxos**

**Proposer**
- Picks a proposal ID ($ID_p$) and sends a PREPARE $ID_p$ message to **Acceptors**
  - $ID_p$ must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher $ID_p$

**Acceptors**
- Receives PREPARE request. Did it promise to ignore requests with $ID_p$?
  - If yes, then ignore request
  - If no, then promise to ignore $ID < ID_p$. Did we already ACCEPT something?
    - If yes, send PROMISE $ID_p$, ACCEPTED *value*
    - If no, send PROMISE $ID_p$

**Reminder: 'nemo' is already consensus**

**PROMISE 6, ACCEPTED 'nemo'**

**Proposer**
- Gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST ($ID_p$, *value*) to **Acceptors**.
  - *value* can be anything

**Acceptor**
- Receives ACCEPT_REQUEST. Did it promise to ignore $ID_p$?
  - If yes, then ignore request
  - If no, reply ACCEPT ($ID_p$, *value*) and also send to **Learners**
Paxos

- **Proposer** picks a proposal ID \( ID_p \) and sends a PREPARE \( ID_p \) message to **Acceptors**
  - \( ID_p \) must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher \( ID_p \)
- **Acceptors** receives PREPARE request. Did it promise to ignore requests with \( ID_p \)?
  - If yes, then ignore request
  - If no, then promise to ignore \( ID < ID_p \). Did we already ACCEPT something?
    - If yes, send PROMISE \( ID_p \), ACCEPTED value
    - If no, send PROMISE \( ID_p \)
- **Proposer** gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST \( ID_p, value \) to **Acceptors**. Did PROMISES come with **values**?
  - If yes, **Proposer** must use value with highest \( ID_p \).
  - If no, value can be anything
- **Acceptors** receives ACCEPT_REQUEST. Did it promise to ignore \( ID_p \)?
  - If yes, then ignore request
  - If no, reply ACCEPT \( ID_p, value \) and also send to **Learners**

Reminder: ‘nemo’ is already consensus
Paxos: Distributed Storage

Log ID 0
- $100

Log ID 1
+ $20 = $120

Log ID 2
- $50 = $70

Log ID 3
+ $200 = $270

Server 1

Server 2

Server 3

Acceptance

Proposal
Paxos: Distributed Storage

<table>
<thead>
<tr>
<th>Log ID 0</th>
<th>Log ID 1</th>
<th>Log ID 2</th>
<th>Log ID 3</th>
<th>Log ID 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$100</td>
<td>$120</td>
<td>$70</td>
<td>$270</td>
<td>$230</td>
</tr>
</tbody>
</table>

- **Proposal**: $+200 = $270

- **Accepted**: $+200 = $270

Server 1

Server 2

Server 3
### Paxos: Distributed Storage

<table>
<thead>
<tr>
<th>Log ID</th>
<th>Transaction 1</th>
<th>Transaction 2</th>
<th>Transaction 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$100</td>
<td>+$20 = $120</td>
<td>+$200 = $270</td>
</tr>
<tr>
<td>1</td>
<td>-$50 = $70</td>
<td>-$50 = $70</td>
<td>-$40 = $230</td>
</tr>
<tr>
<td>2</td>
<td>+$200 = $270</td>
<td>+$20 = $120</td>
<td>+$200 = $270</td>
</tr>
<tr>
<td>3</td>
<td>-$40 = $230</td>
<td>+$1000 = $1070</td>
<td>+$20 = $120</td>
</tr>
</tbody>
</table>

Server 1 proposes $40 = $230 to Server 2 and Server 3. Server 2 accepts the proposal, and Server 3 also accepts it.
Paxos: Distributed Storage

Log ID 0
$100

Log ID 1
+$20 = $120
-$50 = $70

Log ID 2
+$200 = $270
-$50 = $70

Log ID 3
+$200 = $270
-$40 = $230

Log ID 4
-$40 = $230
+$200 = $270

Log ID 5
+$200 = $270
-$40 = $230

+$1000 = $1070

Server 1
Server 2
Server 3

Acceptance
Proposal
Paxos: Distributed Storage

<table>
<thead>
<tr>
<th>Log ID</th>
<th>Servers</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server 1</td>
<td>+$20 = $120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$50 = $70</td>
</tr>
<tr>
<td>1</td>
<td>Server 2</td>
<td>+$20 = $120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$50 = $70</td>
</tr>
<tr>
<td>2</td>
<td>Server 3</td>
<td>+$200 = $270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$40 = $230</td>
</tr>
<tr>
<td>3</td>
<td>Server 1</td>
<td>+$200 = $270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$40 = $230</td>
</tr>
<tr>
<td>4</td>
<td>Server 2</td>
<td>+$200 = $270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$40 = $230</td>
</tr>
<tr>
<td>5</td>
<td>Server 3</td>
<td>+$1000 = $1070</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-$40 = $230</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+$1000 = $1070</td>
</tr>
</tbody>
</table>
Paxos: Master Election

Peer-to-peer

Primary/Secondary
Paxos

PREPARE 5

ACCEPT-REQUEST (5, 'nemo')

PREPARE 9

PREPARE 7

ACCEPT-REQUEST (7, 'star wars')

PREPARE 11

PROMISE 5

(ignored)

PROMISE 9

(ignored)

PROMISE 11

PROMISE 7

(ignored)

PROMISE 9

(ignored)

PROMISE 11

PROMISE 5

PROMISE 7

PROMISE 9

PROMISE 11
Digression: Contention

- Contention is a general issue in concurrent algorithms.
  - Race conditions
  - Deadlock
  - Livelock
- Concurrency is HARD!
Digression: Contention (Race Condition)

- What happens if two machines run this code at once?

```python
def incrementSharedValue(value_server):
    x = value_server.get_value()
    y = x + 1
    value_server.set_value(y)
```
Digression: Contention (Race Condition)

- What happens if two machines run this code at once?

```python
def incrementSharedValue(value_server):
    value_server.lock()
    x = value_server.get_value()
    y = x + 1
    value_server.set_value(y)
    value_server.unlock()
```

This will block if some other machine has the lock, until they release it.
Digression: Contention (Deadlock)

- What happens if two machines are calling these functions?

```python
def incrementSharedValue(s1, s2):
    s1.lock()
    s2.lock()
    x = s1.get_value() + 1
    y = s2.get_value() + 1
    s1.set_value(x)
    s2.set_value(y)
    s2.unlock()
    s1.unlock()
```

```python
def incrementSharedValue(s1, s2):
    s2.lock()
    s1.lock()
    x = s1.get_value() + 1
    y = s2.get_value() + 1
    s1.set_value(x)
    s2.set_value(y)
    s1.unlock()
    s2.unlock()```
Digression: Contention (Livelock)

- Sort of like deadlock, but state is changing
  - Paxos example from earlier
  - Two people walking toward each other in a hallway

- Possible solutions
  - Exponential backoff
  - Backoff fuzzing

- Both of those solutions are generally good practice for request retries