EECS 591
Distributed Systems
Review of Part One: Fundamentals
Manos Kapritsos
Fall 2018
Two generals’ problem

Both generals must attack together or face defeat

Communication is only by messengers sneaking through the valley

Messengers may not make it through…
Question 1 (true or false)

a. $e \rightarrow d$

b. $a \rightarrow j$

c. $g \rightarrow b$
Lamport clocks

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

the Clock condition
Vector clocks

\[ VC(e_i)[i] \] = number of events executed by process \( i \) (including \( e_i \))

\[ VC(e_i)[j] \] = number of events executed by process \( j \) that causally precede \( e_i \)

\[ p \rightarrow q \iff \theta(p) \subset \theta(q) \]

Strong clock condition
**Vector clocks**

\[ VC(e_i)[i] = \text{number of events executed by process } i \text{ (including } e_i) \]

\[ VC(e_i)[j] = \text{number of events executed by process } j \text{ that causally precede } e_i \]

Question 2: what is the VC of:

a. event \( d \)

b. event \( g \)
Cristian's algorithm

\[ \text{time} = \min + \alpha \]

\[ \text{time} = \min + \beta \]

\[ t = x \]

\[ \alpha, \beta \geq 0 \]

\[ Q(t) = ? \]

slave

\[ P(t) \]

master

\[ Q(t) \]

\[ t \]

\[ 2d \]

\[ 2D \]

\[ \text{"time=?"} \]

\[ \text{"time=}\ T\text{"} \]
2-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

Participant $p_i$

2. sends $vote_i$ to Coordinator
   
   if $vote_i = \text{No}$ then
   
   $decision_i := \text{Abort}$
   
   halt

3. if (all votes are $\text{Yes}$) then
   
   $decision_c := \text{Commit}$
   
   send $\text{Commit}$ to all

   else
   
   $decision_c := \text{Abort}$
   
   send $\text{Abort}$ to all who voted $\text{Yes}$

   halt

4. if received $\text{Commit}$ then
   
   $decision_i := \text{Commit}$

   else
   
   $decision_i := \text{Abort}$

   halt
3-Phase Commit

Coordinator \( c \)

1. sends VOTE-REQ to all participants

Participant \( p_i \)

2. sends \( vote_i \) to Coordinator

   if \( vote_i = \text{No} \) then

   \( decision_i := \text{Abort} \)

   halt

3. if (all votes are Yes) then

   send Precommit to all

   else

   \( decision_c := \text{Abort} \)

   send Abort to all who voted Yes

   halt

4. if received Precommit then

   send Ack

5. collect Ack from all participants

   When all Ack’s have been received:

   \( decision_c := \text{Commit} \)

   send Commit to all

6. When \( p_i \) receives Commit, sets \( decision_i := \text{Commit} \) and halts
### 3PC: WHICH STATES ARE COMPATIBLE?

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Committable</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncertain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Committable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Committed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A hierarchy of failure models
**State Machine Replication**

Ingredients: a server

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure that all replicas go through the same sequence of state transitions
4. Vote on replica outputs

\[ x=1 \quad x=2 \]
A primary-backup protocol

\((f = 1)\)
Chain replication

Tail can respond immediately, without waiting for the new update

Head $f + 1$ replicas Tail

- Update
- Query
- Reply
## Consensus

<table>
<thead>
<tr>
<th>Validity</th>
<th>If all processes that propose a value propose $v$, then all correct processes eventually decide $v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement</td>
<td>If a correct process decides $v$, then all correct processes eventually decide $v$</td>
</tr>
<tr>
<td>Integrity</td>
<td>Every correct process decides at most one value, and if it decides $v$, then some process must have proposed $v$</td>
</tr>
<tr>
<td>Termination</td>
<td>Every correct process eventually decides some value</td>
</tr>
</tbody>
</table>
Our algorithm implementing consensus in a synchronous setting is correct! That is, it is both safe and live.
BAD NEWS

The FLP result:
There is no protocol that solves consensus in an asynchronous system where one process may crash.

Fischer, Lynch, Paterson 1985
Abstract

The Paxos algorithm, when presented in plain English, is very simple.
PAXOS AT WORK

Learner

IAmLeader YouAreLeader Decree

Accept

Proposer

Acceptors
### ACCEPTOR STATES
(as leader #50 comes to power)

<table>
<thead>
<tr>
<th>Acceptors</th>
<th>Value</th>
<th>By leader</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x$</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td></td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td></td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td></td>
<td>$y$</td>
<td>41</td>
</tr>
</tbody>
</table>

**Question 4:**
What is the set of possible values that leader #50 can propose?
THE THREAT TO LIVENESS:
DUELING PROPOSERS

Greetings, peasants! I am your fearless leader #1! Grant me your blessing!

Greetings, peasants! I am your fearless leader #2! Grant me your blessing!

Greetings, peasants! I am your fearless leader #3! Grant me your blessing!

Greetings, peasants! I am your fearless leader #4! Grant me your blessing!

Greetings, peasants! I am your fearless leader #5! Grant me your blessing!

Greetings, peasants! I am your fearless leader #6! Grant me your blessing!

Greetings, peasants! I am your fearless leader #7! Grant me your blessing!

Greetings, peasants! I am your fearless leader #8! Grant me your blessing!

. . .
Proposers, acceptors and learners are all collocated on $2f + 1$ replicas.
PBFT

Primary

Replica 1

Replica 2

Replica 3

Pre-prepare phase | Prepare phase | Commit phase | Reply phase
EXECUTE-VERIFY

First execute...
(multithreaded and without agreeing on the order)

...then verify
(that replicas agree on the outcome)