EECS 591
DISTRIBUTED SYSTEMS

Manos Kapritsos
Winter 2018
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
A Slightly Different PRIMARY-BACKUP Protocol

\((f = 1)\)
GENERALIZING TO MORE BACKUPS

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

update

Primary

\[ f \text{ backups} \]
GENERALIZING TO MORE BACKUPS

$\text{Primary} \quad \text{update} \quad \text{f backups}$
GENERALIZING TO MORE BACKUPS
(active updates)

Primary

\( f \) backups
GENERALIZING TO MORE BACKUPS

(passive updates)

$\text{Primary}$

$f$ backups
GENERALIZING TO MORE BACKUPS

(passive updates)

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

Primary

ack

ack

ack

ack

ack

\( f \) backups
Generalizing to more backups

$\text{Primary}$

$f$ backups
HANDLING QUERIES

query

Primary

f backups
Handling queries

Primary

f backups
Handling queries

However…

\[ f \text{ backups} \]
Handling queries

Primary

$f$ backups
**Handling Queries**

The primary cannot respond until it has received all acks for prior updates.

Diagram:
- A query is sent to the primary.
- The primary sends acks to each of the $f$ backups.
- The backups respond with acks.

The primary waits for all acks before it can respond.
Chain replication

Primary

Head $f + 1$ replicas Tail
Chain replication
Chain replication

Head

\[ f + 1 \] replicas

Tail
CHAIN REPLICAATION

update

Head → $f + 1$ replicas → Tail
Chain replication

Head \quad f + 1 \text{ replicas} \quad \text{Tail}
Chain replication

Tail can respond immediately, without waiting for the new update

head

Tail
Upcoming deadlines

- Send me your presentation preferences by **tonight**
- Homework #1 due Monday before class
- Research project descriptions due next Wednesday, 02/07
CONSISTENCY

Is the server’s response correct?

(are all the server’s responses consistent with each other?)
Consistency is a property of the execution; a constraint on the values of the reads and writes returned by the server.
Monotonic read consistency

If a client reads the value of a data item $x$, any successive read operation on $x$ by that client will always return that same value or a more recent value.

Are these runs monotonic read consistent?

$W_1(x,3) \ R_1(x)=4 \ W_2(x,4) \ R_2(x)=4$

$R_1(x)=1 \ R_1(y)=1 \ W_2(y,4) \ R_1(x)=4$

.... $R_1(x)=1 \ R_1(y)=1 \ W_2(y,4) \ R_1(x)=4$
Causal consistency

All processes see causally related events in the same order.

A student removes advisor from friends list and then posts Spring Break photos

The advisor should not be able to see the pictures
A *concurrent* execution of transactions is equivalent to one that executes the transactions serially in *some sequential order*.

Are these runs serializable?

1)  
   \[ T_1: W(x,3) \]
   \[ T_2: W(x,5) \]
   \[ T_3: R(x)=3 \]

2)  
   \[ T_1: W(x,3) \]
   \[ T_2: [W(x,5), R(x)=3] \]
LINEARIZABILITY

Same as serializability, but the sequential order must preserve the **real-time** constraints of non-overlapping operations.

1) $W(x,3)$

2) $W(x,3)$
## Consensus

<table>
<thead>
<tr>
<th>Property</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validity</strong></td>
<td>If all processes that propose a value propose ( v ), then all correct processes eventually decide ( v )</td>
</tr>
<tr>
<td><strong>Agreement</strong></td>
<td>If a correct process decides ( v ), then all correct processes eventually decide ( v )</td>
</tr>
<tr>
<td><strong>Integrity</strong></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><strong>Termination</strong></td>
<td>Every correct process eventually decides some value</td>
</tr>
</tbody>
</table>
The algorithm

Process \( p_i \):
Initially \( V = \{v_i\} \)

To execute \( \text{propose}(v_i) \):

round \( k, 1 \leq k \leq f + 1 \)

1. Send \( \{v \in V : p_i \text{ has not already sent } v\} \) to all
2. for all \( j, 0 \leq j \leq n + 1, j \neq i \), do
3. receive \( S_j \) from \( p_j \)
4. \( V := V \cup S_j \)

\( \text{decide}( ) \) occurs as follows:
5. if \( k = f + 1 \)
6. decide \( \min(V) \)
Good news

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
In an asynchronous setting, a process cannot \textit{tell the difference} between a crashed process and one whose messages take long to arrive.

How long should the process wait before deciding?

- It can’t wait forever: that would violate liveness.
- If it gives up on a process, but it turns out that process is just slow, that would violate safety.
GETTING AROUND THE IMPOSSIBILITY RESULT OF FLP

You can’t be both safe and live in the presence of asynchrony

The FLP result

Fine, then I’ll just be safe! I will only be live when the network behaves synchronously.
ENTER PAXOS
Abstract

The Paxos algorithm, when presented in plain English, is very simple.
**Three types of processes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Proposers</strong></td>
<td>A proposer is a process that has a value to propose</td>
</tr>
<tr>
<td><strong>Acceptors</strong></td>
<td>Acceptors are the processes that ultimately choose which proposed value will be decided</td>
</tr>
<tr>
<td><strong>Learners</strong></td>
<td>A learner only cares about learning which value was decided</td>
</tr>
</tbody>
</table>

\[ (2f+1) \]
HOW THE GAME IS PLAYED

• **Election:** Proposers first try to get a majority of acceptors to follow them.

• **Legislating:** After acquiring a majority, a proposer can *try* to enforce her value, by getting acceptors to accept it, *but*…

• **Playing nice:** If an elected proposer finds that some previous value has been proposed, she proposes that value instead.

• **Winning the game:** once a majority of acceptors have accepted a value, the value is *chosen/decided*
How it is supposed to work

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

We are with you, oh mighty leader!

We are with you, oh wise leader!

My first decree is:

The value should be 12

Sounds good to me!

We are with you, oh wise leader!
HOW IT IS SUPPOSED TO WORK
DEALING WITH MULTIPLE PROPOSERS

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

We are with you, oh mighty leader!

We are with you, oh wise leader!

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

We are with you, oh wise leader #2!
DEALING WITH MULTIPLE PROPOSERS

- I swear I won’t follow an earlier leader!
- And, btw, here is my current accepted value (if any) by leader x.
I swear I won’t follow an earlier leader!
And, btw, here is my current accepted value (if any) by leader x.