EECS 591
Distributed Systems

Manos Kapritsos
Fall 2019
**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
A slightly different primary-backup protocol

\((f = 1)\)
Generalizing to more backups

$f$ backups
GENERALIZING TO MORE BACKUPS

$\text{update}$

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

$ f$ backups
GENERALIZING TO MORE BACKUPS

(.active updates)

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

(passive updates)

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

(passive updates)

Primary

\(f\) backups
GENERALIZING TO MORE BACKUPS

\[ f \text{ backups} \]
Generalizing to more backups

$f$ backups

Primary
eply
HANDLING QUERIES

query
Primary

\( f \) backups
Handling queries

Primary

$f$ backups
HANDLING QUERIES

However…

Primary

reply

\( f \) backups
Handling queries

query

Primary

\( f \) backups
The primary cannot respond until it has received all acks for prior updates.
Chain replication

Primary

Head $f + 1$ replicas Tail
Chain replication

update

Head

$\mathbf{f + 1}$ replicas

Tail

query

reply
Chain replication

Head

update

$f + 1$ replicas

Tail
**Chain replication**

- Head
- $f + 1$ replicas
- Tail

(update)
Chain replication

update  Head  \( f + 1 \) replicas  Tail  reply
Tail can respond immediately, without waiting for the new update.
**ADMINISTRIVIA**

- Send me your paper preferences by **tonight**
- Send me your group declaration preferences by **tomorrow**
- Homework #2 will be out on Wednesday
  - due Wednesday, Oct 16, before class
- Implementation project will be out on Wednesday
  - due Wednesday, October 30, end of day
- Research project topics due next Tuesday, 10/08
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?

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

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

$\ldots \quad R_1(x)=1 \quad R_1(y)=1 \quad W_2(y,4) \quad R_1(x)=4$
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) \( T_1: W(x,3) \)
2) \( T_2: W(x,5) \)
   \( T_3: R(x)=3 \)
3) \( T_1: W(x,3) \)
   \( T_2: [W(x,5), R(x)=3] \)
Linearizability

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

1)  

2)
# Consensus

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</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. \( V := V \cup S_j \)

\( \text{decide}(\cdot) \) occurs as follows:

5. if \( k = f + 1 \)
6. decide \( \min(V) \)
Our algorithm implementing consensus in a synchronous setting is correct! That is, it is both safe and live.
The FLP result:

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

Fischer, Lynch, Paterson 1985
The intuition

In an asynchronous setting, a process cannot 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

The FLP result

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

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.