EECS 591
Distributed Systems

Manos Kapritsos
Fall 2018

Slides by: Lorenzo Alvisi
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
PRIMARY-Backup
THE MODEL

Failure model: crash

Network model: synchrony
  - Reliable, FIFO channels
  - All messages are delivered within $\delta$ time

Tolerates $f$ crash failures
THE IDEA

- Clients communicate with a single replica (primary)

- Primary:
  - sequences and processes clients’ requests
  - updates other replicas (backups)

- Backups use timeouts to detect failure of primary

- On primary failure, a backup becomes the new primary
A SIMPLE PRIMARY-BACKUP PROTOCOL

\((f = 1)\)

Active replication: sync = client request(s)
Passive replication: sync = state update
A simple primary-backup protocol

\((f = 1)\)

Active replication: \(\text{sync} = \text{client request(s)}\)
Passive replication: \(\text{sync} = \text{state update}\)
WEAKENING THE MODEL

Failure model: crash

Network model: synchrony
- Unreliable, FIFO channels
- Channels may drop messages
- All messages are delivered within $\delta$ time
  - (looks paradoxical)

Tolerates $f$ crash failures
A slightly different primary-backup protocol \((f = 1)\)
GENERALIZING TO MORE BACKUPS

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

$\text{Primary}$

$\text{update}$

$f$ backups
GENERALIZING TO MORE BACKUPS

update

Primary

\[ f \] backups
**Generalizing to more backups**

(active updates)

![Diagram](Image)
GENERALIZING TO MORE BACKUPS

(passive updates)

Primary

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

(passive updates)

Primary

$f$ backups
GENERALIZING TO MORE BACKUPS

![Diagram showing a primary node connected to multiple backup nodes with acknowledgment (ack) messages.](image)
GENERALIZING TO MORE BACKUPS

$\text{reply}$

Primary

$f$ backups
HANDLING QUERIES

$\text{ Primary} \quad \text{query} \quad \text{backups}$
Handling queries

Primary

$f$ backups
Handling queries

However…

\[ f \text{ backups} \]
HANDLING QUERIES

Primary

query

\( f \) backups
Handling queries

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

Primary

Head $f + 1$ replicas Tail
Chain replication

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

update

query \quad \text{reply}
Chain replication

Update

Head

\( f + 1 \) replicas

Tail
**Chain replication**

- **Head**
- **$f + 1$ replicas**
- **Tail**

The diagram shows a chain of nodes, with the head node being updated and propagating the update to the following nodes. The notation $f + 1$ refers to the number of replicas beyond the head node.
**Chain replication**

![Diagram of chain replication with nodes labeled Head, \( f + 1 \) replicas, and Tail, connected by arrows for update and reply.]
Chain replication

Tail can respond immediately, without waiting for the new update

![Diagram showing the concept of chain replication with Head, f + 1 replicas, and Tail.]
ADMINISTRIVIA

• Send me your group declaration preferences by **tonight**

• Homework #1 graded
  - Regrade: hand back homework, along with **written** request

• Homework #2 handed out
  - due Thursday, Oct 11, to Remzi in his office hours

• Traveling to OSDI next week, no class

• Research project topics due next Monday, 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?

\[
\begin{align*}
W_1(x,3) & \quad R_1(x)=4 \quad W_2(x,4) & \quad R_2(x)=4 \\
R_1(x)=1 & \quad R_1(y)=1 \quad W_2(y,4) & \quad R_1(x)=4 \\
\ldots & \quad R_1(x)=1 \quad R_1(y)=1 \quad W_2(y,4) & \quad R_1(x)=4
\end{align*}
\]
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
SERIALIZABILITY

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**

**Validity**
If all processes that propose a value propose \( v \), then all correct processes eventually decide \( v \)

**Agreement**
If a correct process decides \( v \), then all correct processes eventually decide \( v \)

**Integrity**
Every correct process decides at most one value, and if it decides \( v \), then some process must have proposed \( v \)

**Termination**
Every correct process eventually decides some value
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.
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
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

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