STATE MACHINE REPLICATION
MODELING FAULTS

- Mean Time To Failure/Mean Time To Recover
  - used mostly for disks
  - of questionable value in expressing reliability

- Threshold: $f$ out of $n$
  - makes condition for correct operation explicit
  - measures fault-tolerance of the architecture, not of individual components

- Enumerate failure scenarios
A HIERARCHY OF FAILURE MODELS

- Fail-stop
- Crash
- Send omission
- Receive omission
- General omission
- Arbitrary (Byzantine) failures

○ = benign failures
A HIERARCHY OF FAILURE MODELS
FAULT TOLERANCE: THE PROBLEM

Clients

Server

Solution: replicate the server
Replication in time

- When a server fails, restart it or replace it
- Failures are detected, not masked
- Lower maintenance, lower availability
- Tolerates only benign failures
Replication in Space

- Run multiple copies of a server (replicas)
- Vote on replica output
- Failures are masked
- High availability and can tolerate arbitrary failures
  - but at high cost
The enemy: non-determinism

An event is non-deterministic if its output is not uniquely determined by its input

The problem with non-determinism:

- Replication in time: must reproduce the original outcome of all non-deterministic events
- Replication in space: each replica must handle non-deterministic events identically
The solution: state machines

Design the server as a deterministic state machine
THE SOLUTION: STATE MACHINES

State machine example: a switch
**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
**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

**non-faulty**

All state machines receive all commands in the same order
**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
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

State Machine Replication
1. Evil Lorenzo Speaks French
2. And was born in Corsica
3. Went to Dartmouth instead of Cornell
4. Rides a Ducati instead of a Moto Guzzi
5. Still listens opera, but he doesn't care for Puccini
5. Evil Lorenzo thinks that $2f+1$ is good enough

When in trouble, cheat!

Voter and client share fate!
PREPARING FOR THE “RESEARCH” PART OF THE COURSE

Look at the papers listed on the course webpage

You each pick one to present
(email me 4 preferences by Monday night)

- ~30-35 minutes presentation
- Send me the slides by October 30th
  (unless you are presenting on Oct 23rd)
THE RESEARCH PROJECT

Sample topics:

Concrete
- Combining Fast Paxos and Flexible Paxos to reduce latency in a geo-replicated storage system
- Proving the correctness of BitCoin

Motivational
- Why the world needs real-time proofs of distributed systems
- Supporting the equivalent instruction hypothesis
- All the things you can do with Flexible Paxos

Survey
- Applying Byzantine Fault Tolerance to blockchains: theory and practice
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: \( \text{sync} = \text{client request(s)} \)

Passive replication: \( \text{sync} = \text{state update} \)
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  

\((f = 1)\)