EECS 482
Introduction to Operating Systems

Fall 2017

Manos Kapritsos
A DISTRIBUTED SYSTEM
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

Send omission = benign failures

General omission

Receive omission

Crash

Arbitrary (Byzantine) 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

1 2 3 4

1: a 2: c
2: d 3: b
3: e
4: f
THE SOLUTION: STATE MACHINES

State machine example: a switch

```
 off   on
   |   |
   v   v
  click  click
```
STATE MACHINE REPPLICATION
**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
**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

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
4. Vote on replica outputs

Make voter and client share fate!
PROJECT 4 DUE IN ONE WEEK

- Remember: a char* is not a string
- printBytes(char *s, int len)
- Minimize disk accesses (at all times)
- Make sure you unlock
ANNOUNCEMENTS

- Remember to fill course evaluations
- No class on Thursday, 12/7
- My office hours moved to Tuesday, 12/12, 4-7pm
- Extra office hours on Wednesday 12/13, 2:30-4:30pm
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
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

update
Primary

\( f \) backups
GENERALIZING TO MORE BACKUPS

(update) Primary

\[ f \text{ backups} \]

\[ \circ \quad \circ \quad \circ \quad \circ \quad \circ \quad \circ \quad \circ \]
GENERALIZING TO MORE BACKUPS

( active updates )

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

(passive updates)

Primary

\[ f \] backups
GENERALIZING TO MORE BACKUPS

(passive updates)

Primary

\( f \) backups
GENERALIZING TO MORE BACKUPS

$\text{Primary}$

$\text{backups}$

$\text{ack}$

$\text{ack}$

$\text{ack}$

$\text{ack}$

$\text{ack}$
GENERALIZING TO MORE BACKUPS

Primary

reply

\( f \) backups
HANDLING QUERIES

Primary

query

$f$ backups
HANDLING QUERIES

Primary

$f$ backups
HANDLING QUERIES

Primary

reply

However…

\( f \) backups
Handling queries

$\text{Primary}$

$f$ backups
Handling Queries

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

Primary

Query

f backups

ack

ack

ack

ack

ack
**Chain replication**

- **Primary**
- **Head**
- $f + 1$ replicas
- **Tail**
Chain replication

update

Head

$f + 1$ replicas

Tail

query

reply
**Chain replication**

Update

Head

\(f + 1\) replicas

Tail
Chain replication

Head

$\\rightarrow$

update

$\Rightarrow$

$f + 1$ replicas

Tail
Chain replication

Head $f + 1$ replicas Tail

update reply
Chain replication

Tail can respond immediately, without waiting for the new update

Head \( f + 1 \) replicas Tail