UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 739
Distributed Systems

Andrea C. Arpaci-Dusseau

Fail-Stop Processors

One paper:

- Byzantine Generals in Action: Implementing Fail-Stop Processors, Fred Schneider, TOCS, May 1984

Motivation

Goal: Build systems that continue to work in presence of component failure

Difficulty of building those systems depends upon how components can fail

Fail-stop components make building reliable systems easier than components with byzantine failures

Fail-Stop Processors

What is a failure?
- Output (or behavior) that is inconsistent with specification

What is a Byzantine failure?
- Arbitrary, even malicious, behavior
- Components may collude with each other
- Cannot necessarily detect output is faulty

What is a fail-stop processor?
- Halts instead of performing erroneous transformations
- Others can detect halted state
- Other can obtain uncorrupted stable storage

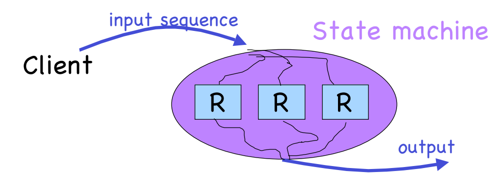
Real processors are not fail-stop; How will we build one?

Building with fail-stop processors is easier; Why?

Distributed State Machine

Common approach for building reliable systems

Idea: Replicate servers, coordinate client interactions with replicas



Failure model of components determines how many replicas, R, are needed and their interactions

How to build t-fault tolerant state machine?

t-fault tolerant: Satisfies specification as long as no more than t components (servers) fail

Inputs

- Key: All replicas receive and process same sequence of inputs
- 1) Agreement: Every nonfaulty replica receives sam request (interactive consistency or byzantine agreement)
- 2) Ordering: Every nonfaulty replica processes requests in same order (logical clocks)

Outputs

	Byzantine	Fail-Stop
Output of SM?		
Number of replicas?		

How to build t-fault tolerant state machine?

t-fault tolerant: Satisfies specification as long as no more than t components (servers) fail

Inputs

- Key: All replicas receive and process same sequence of inputs
- 1) Agreement: Every nonfaulty replica receives sam request (interactive consistency or byzantine agreement)
- 2) Ordering: Every nonfaulty replica processes requests in same order (logical clocks)

Outputs

	Byzantine	Fail-Stop
Output of SM?	Output of majority	
Number of replicas?		

How to build t-fault tolerant state machine?

t-fault tolerant: Satisfies specification as long as no more than t components (servers) fail

Inputs

- Key: All replicas receive and process same sequence of inputs
- 1) Agreement: Every nonfaulty replica receives sam request (interactive consistency or byzantine agreement)
- 2) Ordering: Every nonfaulty replica processes requests in same order (logical clocks)

Outputs

	Byzantine	Fail-Stop
Output of SM?	Output of majority	
Number of replicas?	2t+1	

How to build t-fault tolerant state machine?

t-fault tolerant: Satisfies specification as long as no more than t components (servers) fail

Inputs

- Key: All replicas receive and process same sequence of inputs
- 1) Agreement: Every nonfaulty replica receives sam request (interactive consistency or byzantine agreement)
- 2) Ordering: Every nonfaulty replica processes requests in same order (logical clocks)

Outputs

	Byzantine	Fail-Stop
Output of SM?	Output of majority	Output from any
Number of replicas?	2t+1	

How to build t-fault tolerant state machine?

t-fault tolerant: Satisfies specification as long as no more than t components (servers) fail

Inputs

- Key: All replicas receive and process same sequence of inputs
- 1) Agreement: Every nonfaulty replica receives same request (interactive consistency or byzantine agreement)
- 2) Ordering: Every nonfaulty replica processes requests in same order (logical clocks)

Outputs

	Byzantine	Fail-Stop
Output of SM?	Output of majority	Output from any
Number of replicas?	2t+1	t+1

Building a Fail-Stop Processor

Assumption: Storage

- Volatile: Lost on failure
- Stable
  - Not affected (lost or corrupted) by failure
  - Can be read by any processor
  - Benefit: Recover work of failed process
  - Drawback: Minimize interactions since slow

Can only build approximation of fail-stop processor

- Finite hardware -> Finite failures could disable all error detection hardware

k-fail-stop processor: behaves fail-stop unless k+1 or more failures

Implementation of k-FSP: Overview

Two components

- k+1 p-processes (program)
- 2k+1 s-processes (storage)
- Each process runs on own processor, all connected with network

P-Processes (k+1)

- Each runs program for state machine
- Interacts with s-processes to read and write data
- If any fail (if any disagreement), then all STOP
- Cannot detect k+1 failures

Implementation Continued

S-Processes (2k+1)

- Each contains contents of stable storage
- Provides reliable data with k failures (cannot just stop)
- Detects disagreements/failures across p-processes

Assumptions

- Messages can be authenticated (digital signatures)
- Byzantine agreement protocol??
  - Signed messages
  - Requires k+1 processes
- Synchronized clocks across all processes in FSP

FSP Algorithm: Writes

Each p-process, on a write:

- Byzantine agreement across s-processes (agree on same input value)

Each s-process, on a write:

- Ensure each p-process writes same value within time bound
  - If all okay, then update value in stable storage
  - If not, then halt all p-processes
    - Set failed variable to true
    - Do not allow future writes

FSP Algorithm: Reads

Each p-process, on a read:

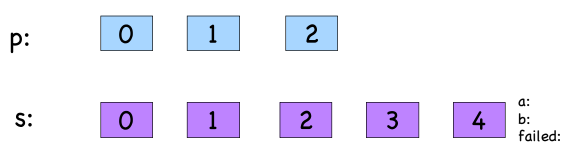
- Broadcast request to all s-processes
- Use result from majority (k+1 out of 2k+1)
- Other FSP can read as well

Each p-process, determine if halted/failed:

- Read failed variable from s-process (use majority)

FSP Example

k=2, SM code: "b=a+1"



How do p-processes read a?

- What if 2 s-processes fail?
- What if 3 s-processes fail?

How do p-processes write b?

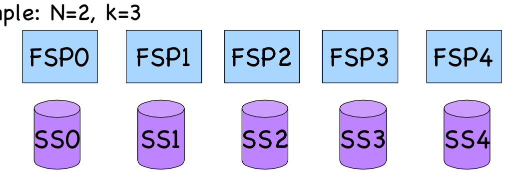
- What if 1 p-process is very slow?
- What if 1 p-process gives incorrect results to all?
- What if 1 p-process gives incorrect results to some?
- What if 3 p-processes give bad result?

Higher-Level Example

Goal: Run app handling k faults, needs N good processors

Solution: Use N+k k-failstop processors

Example: N=2, k=3



What happens if:

- 3 p-processes in FSP0 fail? 4 p-processes in FSP0 fail?
- 1 p-process in FSP0, FSP1, and FSP2 fail? also in FSP3?
- 2 p-processes in FSP0, FSP1, and FSP2 fail?
- 1 s-process in SS0 fails? also in SS1, SS2, and SS3?
- 4 s-processes in SS0 fail?

Should we use Fail Stop Processors?

Metric: Hardware cost for state machines:

- Fail-stop components:
  - Worst-case (assuming 1 process per processor):
    - (N+k) * (3k+2) processors
  - Best-case (assuming s-processes from different FSP share same processor)
    - (N+k)(k+1) + (2k+1) processors
- Byzantine components:
  - N * (2k+1)
- Fail-stop can be better if s-processes share and N>k

Metric: Frequency of byzantine agreement protocol

- Fail-Stop: On every access to stable storage
- Byzantine: On every input read
- Probably fewer input reads

Summary

Why build fail-stop components?

- Easier for higher layers to model and deal with
- Matches assumptions of many distributed protocols

Why not?

- Usually more hardware
- Usually more agreements needed
- Higher-levels may be able to cope with "slightly faulty" components
- End-to-end argument