Failure Detectors

- All our recent protocols have detected failures (then run leader election/view change/...)
- Not the focus of today's topic (despite the name)
- Asynchronous
  - Partially synchronous
  - Synchronous

Several weeks (and a midterm ago)

Asynchronous: cannot distinguish b/w failure & delay

Fault tolerant consensus

Dwork et al. => F.T. consensus

FLP => Fault tolerant consensus
Partially Synchronous

Synchronous

Can distinguish

What do we need to implement F.T. Consensus?

Asynchronous

No

Partially Synchronous

Yes

How?

Use an old idea from TCS: ORACLES

Except we call them FAILURE DETECTORS

Why?

1. The obvious answers

I chose to inflict this on you
2 Useful Technique To Evaluate Whether New Hardware Improves Fault Tolerance/Performance...


→ Do Synchronized Clocks Help with Consensus in Fail-Stop Model? (Ghemawat & Lynch, others)

→

Some Important Points To Remember Before We Start

- Failure Detectors (Or Oracles) Usually Cannot Be Implemented
  → Model To Answer Theoretical Questions (Our Discussion Today)

→ Model New Hardware That Doesn't Obviously Fit in The FLP Model/To Automaton

- Can Sometimes Be Emulated (2nd Paper)
  → But Not Quite The Same
First, an apology about the paper.

**Asynchronous**  No F-to-Consensus  No Delay Bounds

??  Yes  ??

**Partially Synchronous**  Yes F-to-Consensus  Eventual Delay Bounds

---

**Problem**

- Hard to reason about effect of delays

- Instead model additional information as failure information
  (notice, this fits in more cleanly if we just considered synchronous vs asynchronous)

- How to model failure information?
  - Each process/node can call a function $F$:
    
    \[ F : \text{Time} \rightarrow \text{Set of 'suspected' functions processes} \]
    
    \[ T \rightarrow 2^T \]

- Best case $F(T)$ returns set of failed processes at $T$. 

Time $T$

$\Rightarrow$ **Physically Impossible. Why?**

---

**Characterizing Failure Detectors**

**Completeness**: Eventually How many failed nodes are returned

**Accuracy**: Are any correct nodes returned

---

**Completeness**

- **Strong**: Eventually all correct nodes suspect all faulty nodes

$\exists k^0 \quad FCL = \{ P_2, P_3 \}$

$R \quad P_2 \quad A \quad P_4 \quad X$

How do you suspect? Suppose each
- Weak: Eventually some correct node suspects each faulty node

- Strong: No correct node is ever suspected

- Weak: Eventually there is never suspected

**More Accuracy**

Eventually Strong:

Eventually Weak
Putting Them Together

\[
\begin{array}{c|ccc}
\text{Completeness} & S & W \\
\hline
S & P & Q \\
W & S & W \\
\downarrow & \downarrow & \downarrow \\
\diamond S & \diamond P & \diamond Q \\
\diamond W & \diamond S & \diamond W \\
\end{array}
\]

P: Perfect
S: "Strong"
W: Weak

Observe: Perfect $\iff$ Synchronous
Strong \leftrightarrow \text{Leader election?}

<table>
<thead>
<tr>
<th>Completeness</th>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td>w</td>
<td>S</td>
<td>W</td>
</tr>
<tr>
<td>$\diamond$ S</td>
<td>$\diamond$ P</td>
<td>$\diamond$ Q</td>
</tr>
<tr>
<td>$\diamond$ W</td>
<td>$\diamond$ S</td>
<td>$\diamond$ W</td>
</tr>
</tbody>
</table>

Some relations in Fail Stop

- $P \rightarrow \diamond P$
- $S \rightarrow \diamond S$
- $W \rightarrow \diamond W$
- $Q \rightarrow \diamond Q$
- $P \rightarrow Q$
- $P \rightarrow S$
- $S \rightarrow W$
- $Q \rightarrow W$

Maybe surprising
Q → P, \ W → S

\{STRONG COMPLETENESS: All Correct Processes Suspect All Failed Processes
WEAK COMPLETENESS: Some Correct Process Suspects Each Failed Process
\}

Both Are Eventual

Weak Completeness → Strong Completeness

- Take union of FD output from different processes.

How?

- Each process periodically queries F with \( W \cdot c \)
- Broadcasts query output \( b \) times
- All processes track last query response from each process
- Output of \( F \) with \( S \cdot c \) is union of last responses
CLAIM: Does not affect accuracy.

**Completeness**

<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>P ─── Q</td>
</tr>
<tr>
<td>W</td>
<td>S ─── W</td>
</tr>
<tr>
<td>♣S</td>
<td>♣P ─── ♣Q</td>
</tr>
<tr>
<td>♦W</td>
<td>♦S ─── ♦W</td>
</tr>
</tbody>
</table>

**Core Result**

◇S ↔ Consensus

Equivalently ◇S (◇W) is the weakest FD for F.T. Consensus.
Consensus

- Everyone repeatedly queries $\diamondsuit$, computer $\Pi - \diamondsuit$ (correct processes)
- Leader election among processes in $\Pi - \diamondsuit$
  [only wait on votes from all correct processes]
- Eventually elect a correct leader who decides on value $\lambda$ broadcasts.

\[ \text{Consensus} \rightarrow \diamondsuit \]

- Periodically run consensus protocol
  $\Rightarrow$ Each node tracks who participated
- Use consensus to agree on nodes who participate
- $\diamondsuit$ returns consensus value

What can we actually infer from this?
Building FDs in Practice

Problem 1: FLP - previous construction might not work

Bigger Problem 2: DS not useful in many cases
  - Correct nodes might be suspected
  => Protocol must deal with suspected nodes reanimating

- Lemmens et al. solution