FLP Impossibility & Weakest Failure Detector

Consensus Protocols in Theory
Philip Daian - 10/25

slides influenced by Birman FA12 slides
Consensus!

Courtesy of

https://rethinkdb.com
Consensus Example

Clients

Leader

Storage
Consensus Example

Clients

Replicated Leader

Storage
Consensus Summary

● Important problem! We’ve already talked quite a bit about forms of consensus
  ○ State machine replication -> consensus on state of machine
  ○ Leader election in leadered protocols -> consensus on leader
  ○ Paxos, CORFU -> essentially consensus protocols
  ○ Byzantine Generals -> consensus in malicious actor setting

● Applications: “clock synchronization, PageRank, opinion formation, power smart grids, state estimation, control of UAVs, load balancing and so on” (Wiki)

● Conditions: **Termination, Validity, Integrity, Agreement**
  ○ Conditions vary depending on problem model / definitions

● Focus on consensus on a simple bit for simplicity; such protocols can extend
Impossibility of Distributed Consensus with One Faulty Process 1985

- 2001 Dijkstra prize; best paper in distributed systems

- distributed systems, e-voting, oblivious transfer
- distributed algorithms and impossibility results, formal modeling
- algorithms, complexity, theory
asynchronous deterministic distributed consensus impossible with even 1 crash failure
Follow along!

http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/
Communication Model

message buffer

processes
send(p, m)
receive(p)
processes
∅
message buffer
∅(p, m)
receive(p)
receive(p)

processes

message buffer
reliable

(p, m)

receive(p)
Step - Part 1: event

message buffer
reliable

processes

receive(p)

(p, m)
Step - Part 2

processes

message buffer
reliable

# send(p, m)

finite # send(p, m)
Configuration

message buffer
reliable

processes
Schedule - $\sigma$

Event (receipt of $m$ by $p$)

Event (receipt of $m$ by $p$)

Event (receipt of $m$ by $p$)
0-Valent Configuration

All Processes Decide 0
Initial configuration

All Processes Decide 0
1-Valent Configuration

p0  v1  Schedule - \(\sigma_1\)
p1  v2  Schedule - \(\sigma_2\)
p2  v3  Schedule - \(\sigma_3\)
p3  v4

All Processes Decide 1
Bivalent Configuration (Read: Undecided)

- p0: v1
- p1: v2
- p2: v3
- p3: v4

- Schedule - $\sigma_1$
  - Decide 0
- Schedule - $\sigma_2$
  - Decide 1
- Schedule - $\sigma_3$
  - Decide 0
- Schedule - $\sigma_4$
Now, we prove:

Any protocol in our model must have an infinitely long run (that never terminates)
Proof Outline

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be another bivalent configuration reachable from the configuration by applying e last (Lemma 3)
- Since the configuration is bivalent... (Lemma 3)
Lemma 1; Housekeeping

Schedules are commutative
Proof! (Lemma 1) [from the paper]
Lemma 2

There is an initial **bivalent** configuration

(see: bivalent; read: undetermined / undecided)
Initial Configurations - **neighbors**

0-valent  1-valent

<table>
<thead>
<tr>
<th></th>
<th>0-valent</th>
<th>1-valent</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>v1</td>
<td>v1'</td>
</tr>
<tr>
<td>p1</td>
<td>v2</td>
<td>v2</td>
</tr>
<tr>
<td>p2</td>
<td>v3</td>
<td>v3</td>
</tr>
</tbody>
</table>
Initial Configurations

0-valent  1-valent

```
p0  |  v1  |  v1' |
    |
 p1  |  v2  |  v2  |
    |
 p2  |  v3  |  v3  |
```
Initial Configurations

0-valent  1-valent

<table>
<thead>
<tr>
<th>p0</th>
<th>v2</th>
<th>v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3</td>
<td>v3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p1</th>
<th>v2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p2</th>
<th>v3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Initial Configurations

0-valent 1-valent

p0

p1
v2  v2

p2
v3  v3

bivalent OR both 0 OR both 1
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>p1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>p2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3 Processes - Neighbors differ by 1 Process Input

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p0</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>p1</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>p2</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
We want to prove

There is an initial **bivalent** configuration

assume the opposite -

All initial configurations **univalent**

(see: bivalent; read: undetermined / undecided)
3 Processes - A Univalent-Only Scheme
3 Processes - Another Univalent-Only Scheme

```
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
So

Univalent only schemes *don’t work*

Must have initial bivalent configuration!
Reminder

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be another bivalent configuration reachable from the configuration by applying e last (Lemma 3)
- Since the configuration is bivalent... (Lemma 3)
Lemma 3

If C is a bivalent configuration, and e is an event applicable to C, there is a bivalent configuration reachable by applying e last

(this is the big one)
Lemma 3

2 Ingredients:

An event, e (fix any event)

D - all configurations right after e
Lemma 3

We will show:

D has a bivalent configuration

(through series of contradictions)
Lemma 3 - Contradiction 1

\textbf{D} has only 1-valent configurations

\textit{(E0 has seen e)}
Lemma 3 - Contradiction 1

\( \mathbf{D} \) has only 1-valent configurations

\((E0 \text{ has seen } e)\)
Lemma 3 - Contradiction 1

\[ \overline{D} \text{ has only 1 valent configurations} \]

\((E0 \text{ has seen e})\)

Diagram:
- Initial C Bivalent
- Just received e
- 1 Valant?
- Other events
- E0 0 Valant
Lemma 3 - Contradiction 1

\( \mathbb{D} \) has only 1-valent configurations

\((E0 \ has \ not \ seen \ e)\)
Lemma 3 - Contradiction 1

D has only 1-valent configurations

(E0 has not seen e)
Lemma 3 - Contradiction 1

\( D \) has only 1-valent configurations

\((E0 \text{ has not seen } e)\)
Summary

Disproven:

D has only 1-valent configurations

D has only 0-valent configurations (same)

2 Possibilities:

D has only 1, 0 valent configurations (no bivalent) [next]

D has bivalent configurations
Lemma 3 - Contradiction 1

**D** has only 1, 0-valent configurations
Lemma 3 - Contradiction 1

\textbf{D} has only 1, 0-valent configurations

\textit{(e' and e have different destinations)}
Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

(e’ and e have different destinations)
Lemma 3 - Contradiction 1

D has only 1, 0 valent configurations

(e’ and e have different destinations)

Initial C Bivalent

Events (just got e)

C0 1 Valent

(just got e’)
(just became 1-valent)

D0 0 Valent

D1 1 Valent

(just got e)
Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

(e’ and e have same destination, p)

Initial C
Bivalent

D
0 Valent

C0
1 Valent

D0
0 Valent

D1
1 Valent

Events (just got e)

(just got e)
(just became 1-valent)
Lemma 3 - Contradiction 1

D has only 1, 0-valent configurations

(e’ and e have same destination, p)
Lemma 3 - Contradiction 1

**D** has only 1, 0-valent configurations

(e’ and e have same destination, p)
Lemma 3 - Contradiction 1

**D** has only 1, 0-valent configurations

(e’ and e have same destination, p)
Lemma 3 - Contradiction 1

$D$ has only 1, 0 valent configurations

$(e' \text{ and } e \text{ have same destination, } p)$
Summary

Disproven:

D has only 1-valent configurations

D has only 0-valent configurations (same)

D has only 1, 0 valent configurations (no bivalent)

1 Possibility:

D has bivalent configurations
The whole proof!

- Start from the initial guaranteed bivalent configuration (Lemma 2)
- Since the configuration is bivalent, there must be a bivalent configuration (in D) reachable from the configuration by applying \( e \) last (Lemma 3)
- Since the configuration is bivalent... (Lemma 3)
Beyond FLP

Work has continued far beyond the FLP result:

- Relaxing async model; failure detectors
  - New models; partially synchronous, sleepy, etc
  - Coming up next!
- Reducing other problems to consensus
  - SMR, leader election, atomic broadcast, shared log, ...
- New forms of consensus in permissionless models!
  - Bitcoin, blockchains, ByzCoin, etc.
Consensus with Probability 1

I like 1! Cardinality 1

I like 0! Cardinality 1

I like 1! Cardinality 1

I like 0! Cardinality 1

I like 1! Cardinality 1

I like 1! Cardinality 1

I like 1! Cardinality 1
Consensus with Probability 1

I like 0! Cardinality 3

I like 0! Cardinality 3

I like 0! Cardinality 3

I like 1! Cardinality 1

I like 1! Cardinality 1

I like 1! Cardinality 1
Consensus with Probability 1

I like 0! Cardinality 4

I like 0! Cardinality 4

I like 1! Cardinality 1

I like 0! Cardinality 4

I like 0! Cardinality 4

I like 1! Cardinality 1
Consensus with Probability 1

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5

I like 0! Cardinality 5
Consensus with Probability 1

I like 0! Cardinality 6

I like 0! Cardinality 6

I like 0! Cardinality 6

I like 0! Cardinality 6
Wrap Up Discussion; FLP

- Test your understanding: what is the difference between a univalent and bivalent state?
- But what does “impossibility” mean in FLP?
- How can we make sure our models are accurate for the desired setting?
- What are the implications for protocols handling Byzantine faults?
- Which one of these assumptions is easiest to relax in a datacenter?

<table>
<thead>
<tr>
<th>Asynchronous model</th>
<th>Real world</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable message passing, unbounded delays</td>
<td>Just resend until acknowledged; often have a delay model</td>
</tr>
<tr>
<td>No partitioning faults (“wait until over”)</td>
<td>May have to operate “during” partitioning</td>
</tr>
<tr>
<td>No clocks of any kinds</td>
<td>Clocks but limited sync</td>
</tr>
<tr>
<td>Crash failures, can’t detect reliably</td>
<td>Usually detect failures with timeout</td>
</tr>
</tbody>
</table>
Failure Detectors!

**Motivation:** OK, we know FLP impossibility asynchronously.

Can we create *minimal weakening* of model, Achieve *(asynchronous*\(^*\) deterministic) consensus?

**YES:** Failure Detectors
The Weakest Failure Detector for Solving Consensus
1996

- Formalizes “failure detection service”; used by consensus as black box
- Explores types, guarantees, constructions, proofs of failure detectors
Motivation

Diagram from Ken Birman’s slides, ‘12FA
Background, Model, Assumptions

Same as last time!
Failure Detection Guarantees

- Want to achieve two properties:

- **Completeness:** failed processes eventually suspected by correct processes
- **Accuracy:** correct processes are never suspected by other correct processes
  - Can you think of a failure detector that is complete but not accurate and vice versa?

- Incomplete or unreliable **failure detectors** provide some but not perfect satisfaction of above
- **Self test:** How to implement perfect FD in synchronous model? Asynchronous model? What about the weakest imaginable FD?
## Failure Detection Guarantee Variations

<table>
<thead>
<tr>
<th></th>
<th>Strong completeness:</th>
<th>Weak completeness:</th>
<th>Strong accuracy:</th>
<th>Weak accuracy:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>Eventually every process that crashes permanently suspected by every correct process</td>
<td>Eventually every process that crashes permanently suspected by some correct process</td>
<td>Correct processes never suspected</td>
<td>Some correct process never suspected</td>
</tr>
<tr>
<td>Weak</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eventual strong accuracy:</td>
<td>There is a time after which strong accuracy holds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eventual weak accuracy:</td>
<td>There is a time after which weak accuracy holds</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Accuracy

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Strong</th>
<th>Weak</th>
<th>Eventual Strong</th>
<th>Eventual Weak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>Perfect (P)</td>
<td>Strong (S)</td>
<td>Eventually Perfect (♢P)</td>
<td>Eventually Strong (♢S)</td>
</tr>
<tr>
<td>Weak</td>
<td>Quasi-Perfect (Q)</td>
<td>Weak (W)</td>
<td>Eventually Quasi-Perfect (♢Q)</td>
<td>Eventually Weak (♢W)</td>
</tr>
</tbody>
</table>

*Diagram adapted from Ken Birman’s slides, ‘12FA*
◊W for consensus!

- Coordinator
- Propose value
- Token - circulate around ring
- Processes
◊W for consensus!

- Propose value
- Token - circulated around ring
- No change in failure detector

processes - proposal

coordinator
diamond W for consensus!

- processes - proposal
- coordinator

- Decide value

- Token - circulated around ring
- Received by coordinator
- No change in failure detector
◊W for consensus!

- Processes - Proposal
- Decide value
- Token - circulated around ring
- Received by all processes
- No change in failure detector
Real systems and failure detectors

- Most common form of failure detection in real-systems timeout based; can you name a few?
- Usually stronger than weakest failure detector
- **No violation of FLP**; FLP applies to full system, incl. detectors

- Real systems achieve consensus with high probability
  - FLP “doesn’t matter” (can be worked around) in practice
Wrap Up Discussion; Open Problems in Consensus

- For crash fault tolerance, “trusted” setting
  - We have Paxos! We have RAFT! Can we get a simpler protocol?
- Fail-stop model; we can achieve failure detection
  - In practice, is this a workable solution?
- Byzantine faults!
  - PBFT protocol; so much easier than Paxos (look it up!)

- “trustless” setting; lots of work to be done
  - What are the unique challenges?
  - Are there model relaxations possible other than computational bounding?
  - How to identify nodes?
  - Canetti et. al 2005 impossibility result
Takeaway!

- Consensus is hard!
- Subtleties in model of consensus strongly influence results
- Make sure to choose model accurately matching reality
- Minor differences in model -> major differences in results
- The consensus ship is not yet sunk, much work to be done
- Consensus is everywhere in distributed systems