New Approaches in Byzantine Fault Tolerance

Dahlia Malkhi
The Saddest Moment, Mickens 2013

Bitcoin, Nakamoto 2008
use case ➔ accelerated BFT research

blockchain unlocks innovation

beyond known approaches
HOORAY
I'M RELEVANT
Enable a simple global currency and financial infrastructure that empowers billions of people
Libra Association Founding Members

- Women's World Banking
- kiva
- vodafone
- Union Square Ventures
- Ribbit Capital
- Thrive Capital
- Lyft
- coinbase
- Spotify
- Anchorage
- xapo
- BisonTrails
- Facebook Calibra
- ANDREessen Horowitz
- FARFETCH
- iliad
- Uber
- PayU
- LATTICE
- CREATIVE2
- INSTITUTE
State Machine Replication

Client sends a sequence of commands

The server logs commands and executes them on a state machine
State Machine Replication

A group of servers provide the same interface as a single server even if some of the servers are faulty

Safety: Any two observers learn the same sequence of values

Liveness: A value proposed by a proposer will be eventually executed by every observer
State Machine Replication (SMR)

- State machine
- Sequencer
- State store
State Machine Replication (SMR)

- state machine
- sequencer
- state store
- smart contracts
- DLT
- ledger
problem model

n validators
f faulty
clients
transmission delays
authenticated communication
PKI
Byzantine Fault Tolerance (BFT)

Fig. 1. Lieutenant 2 a traitor.

Fig. 2. The commander a traitor.
Classical Consensus Foundations

Synchronous settings
- [LPS 82] $n > 2f$
- [LPS 82] $n > 3f$ with “oral message” only
- [PSL 80] $\Omega(n)$ rounds
- [GM 98] $\Omega(n^2)$ communication

Asynchronous settings
- [FLP 85] non-terminating executions exist even with a single crash fault
- $n > 3f$ for safety
- [Ben-Or 83] randomized solution, exponential number of rounds in expectation
- [AMS 2019] randomized solution, $O(1)$ expected number of rounds, $O(n^2)$ comm., trusted setup
Safe but not live

[DLS 88]

- Synchronous rounds for progress
- Polynomial communication
- $\Omega(n)$ rounds
<table>
<thead>
<tr>
<th>year</th>
<th>work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>Byzantine Generals</td>
</tr>
<tr>
<td>1988</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; solution</td>
</tr>
<tr>
<td>1999</td>
<td>PBFT</td>
</tr>
<tr>
<td>2007</td>
<td>Fast track</td>
</tr>
<tr>
<td>2016</td>
<td>Tendermint: blockchain BFT</td>
</tr>
<tr>
<td>2017</td>
<td>Fast track broken</td>
</tr>
<tr>
<td>2017</td>
<td>Fast track fixed</td>
</tr>
<tr>
<td>2018</td>
<td>HotStuff: 1&lt;sup&gt;st&lt;/sup&gt; linear solution</td>
</tr>
<tr>
<td>2019</td>
<td>LibraBFT</td>
</tr>
<tr>
<td>Year</td>
<td>Work</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1982</td>
<td>Byzantine Generals</td>
</tr>
<tr>
<td>1988</td>
<td>1st solution</td>
</tr>
<tr>
<td>1999</td>
<td>PBFT</td>
</tr>
<tr>
<td>2007</td>
<td>Fast track</td>
</tr>
<tr>
<td>2016</td>
<td>Tendermint: blockchain BFT</td>
</tr>
<tr>
<td>2017</td>
<td>Fast track broken</td>
</tr>
<tr>
<td>2017</td>
<td>Fast track fixed</td>
</tr>
<tr>
<td>2018</td>
<td>HotStuff: 1st linear solution</td>
</tr>
<tr>
<td>2019</td>
<td>LibraBFT</td>
</tr>
</tbody>
</table>

**Era of blockchains**

**voting rule:**
accept and vote iff 
\[(\text{round}(\text{proposal}) > \text{last-voted-round}) \text{ and } (\text{round}(\text{parent_qc}) \geq \text{preferred-round})\]
Flexible Byzantine Fault Tolerance

a new approach to designing BFT solutions [MNR CCS’19]

- New BFT protocol
  - supports both synchrony and synchrony assumptions
  - supports different observers under the same protocol
- Stronger Resilience
  - new fault model
New Fault Model

cmd_1, cmd_2, cmd_3, ...
Diversity

cmd_1, cmd_2, cmd_3, ...

......

......

......

......
contribution #1: introduce alive-but-corrupt faults

(limitation #1: optimized for Byzantine faults)

alive-but-corrupt: an adversary that attacks safety but not liveness
(a-b-c fault)

total faults: #Byzantine faults + #a-b-c faults

we can tolerate a larger number of total faults
Flexible Byz Quorums

Classical BFT:
• safety: Byz quorum intersection, (2q-1)-resilience
• liveness: (1-q)-resilience
• optimal Byz resilience: (1-q) = (2q-1)

Flexible BFT:
• safety: higher resilience > (1-q)
• liveness: q will participate because they cannot break safety by not responding!
• resilience:
  - (1-q) Byz
  - (3q-2) active-but-corrupt
What can you do with Flexible BFT?

(asyynch) switch to mixed resilience when a-b-c behavior observed

- P active corruptions observed
  ↓
- $(1/3 - P)$ Byz
  ↓
- wait for $q = 1 - (1/3 - P)$ quorum
  ↓
- $1/3 + P$ corruption resilience
contribution #2: support diversity

(observer 1: 20% Byzantine faults and 30% a-b-c faults, partial synchrony)
(observer 2: 10% Byzantine faults and 50% a-b-c faults, partial synchrony)
(observer 3: 30% Byzantine faults and 40% a-b-c faults, synchrony, Δ)

Flexible BFT: a family of protocols that separate fault model from the solution

observers make their own assumptions and interpret the transcript generated by the replicas

(network assumptions, type and number of faults, Δ)

Using one protocol instance
Sync BFT with $\Delta$-agnostic replicas

simplest sync BFT protocol?

steady-state leader:
  broadcast proposal

replicas:
  re-broadcast (first) leader proposal
  wait to collect q-cert

observers:
  commit when q replicas “report”
  $2\Delta$ quiet-period after holding a cert

new leader:
  collect certs from q replicas
Flexible BFT: Safety, Liveness, and Commits

Classical consensus protocols:

(Safety) Any two learners learn the same sequence of values
(Liveness) A value proposed by a proposer will be eventually executed by every learner

Replicas can be learners and can commit

Flexible BFT:

(Safety) Any two learners with correct but potentially different assumptions learn the same sequence of values
(Liveness) A value proposed by a proposer will be eventually executed by every learner with a correct assumption

Commit by learners (and not replicas)
contribution #3: support recovery

(limitation #3: no recovery)

If an observer makes an incorrect assumption and fails, he can recover by picking safer assumptions.
What can you do with Flexible BFT?

Wonders!

(asynch) switch when corruption level approaches 1/3
(synch) switch to asynch mode against asynchrony attack
recover from conflicting commits (forking)
perform periodic safety checkpoint
guard high-stake transactions
Client Assumptions Supported at $q = 2/3$
Assumptions Supported at Different $q$’s
Flexibility Open Questions

- rationality analysis
- shades of corruption faults
- observer market