VLDB2020Tokyo

Building High Throughput Permissioned Blockchain Fabrics: Challenges and Opportunities




Suyash Gupta *Jelle Hellings* Sajjad Rahnama Mohammad Sadoghi

UC DAVIS UNIVERSITY OF CALIFORNIA

ExpoLab Creativity Unfolded

ResilientDB Security, Privacy Reloaded

About Us

Exploratory Systems Lab at UC Davis

Goal: High-performance resilient data processing.

- 1 Professor, 1 Postdoc, 3 Ph.D. students, 6 M.Sc. and B.Sc. students.
- Recent papers at VLDB, ICDCS, ICDT, DISC, EDBT, and more.
- Intersection of blockchain and database technology.
- ResilientDB: A pioneering new data platform.

Goal: High-performance resilient data processing

Questions

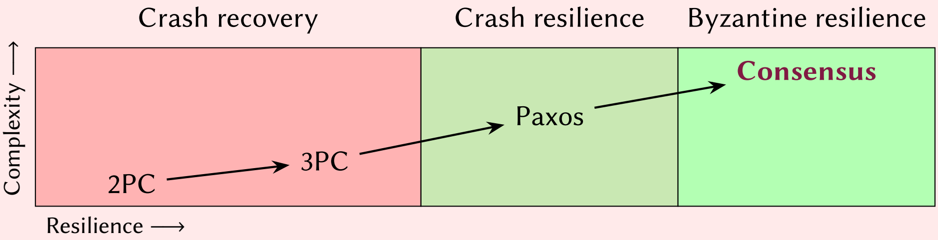
1. Why?
2. What is the relation with blockchains?
3. What do we already have?
4. Where can we improve?
5. What new tools do we need?

Towards high-performance resilient data processing:

Why?

Why resilient data processing?

Go beyond assumptions of traditional transaction processing!

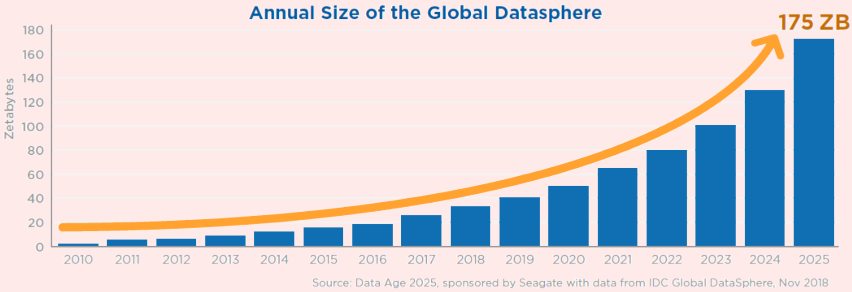


Example

- Provide continuous services during failures.
- Provide services in federated environments.

Why high-performance?

Support requirements of future applications!



- Ever-growing volumes of data (e.g., sensor networks).
- Ever-growing demands of applications (e.g., machine learning).

Towards high-performance resilient data processing:

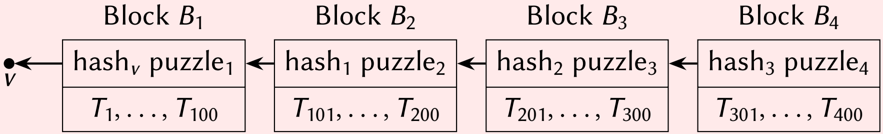
What is the relation with blockchains?

What is a blockchain?

What is a blockchain?

Bitcoin: *Management of monetary tokens* (Bitcoins)

- Open and decentralized transfer of tokens (*transactions*).
- History of transactions (*ledger*) stored in the blockchain.

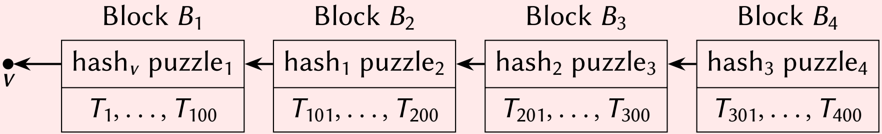


- *Many participants* hold a copy of the blockchain.
- Blockchain structure is *tamper-proof* by design.

What is a blockchain? - Malicious behavior

Bitcoin: Preventing malicious behavior

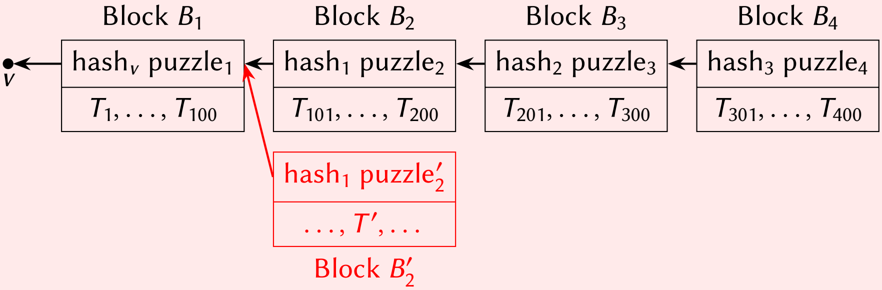
- Malicious attempts to change a chain.



What is a blockchain? - Malicious behavior

Bitcoin: Preventing malicious behavior

- Malicious attempts to change a chain.



- Longest chain has highest incentives.
- Making blocks (solving puzzles) is very costly.
- Malicious attempt leads to a *dead end*.

What is a blockchain? - A definition

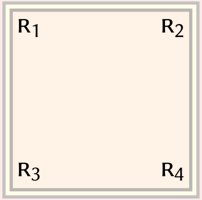
A resilient tamper-proof ledger maintained by many participants.

- *Ledger*.
  Append-only sequence of transactions.
  In database terms: a journal or log.
- *Resilient*.
  High availability via full replication among participants.
- *Tamper-proof*.
  Changes can only be made with majority participation.

Blockchains are *distributed fully-replicated systems*!

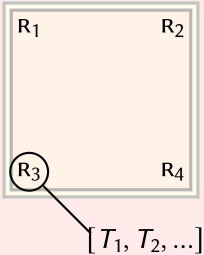
Components of blockchain systems

1. Replicas.



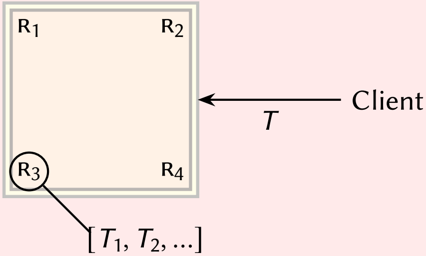
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.



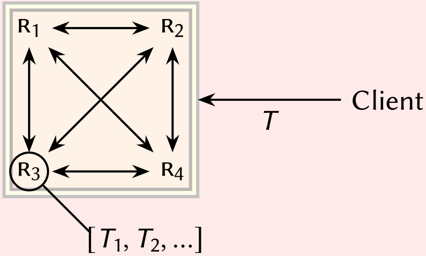
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.



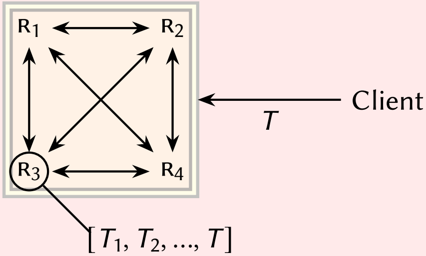
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.



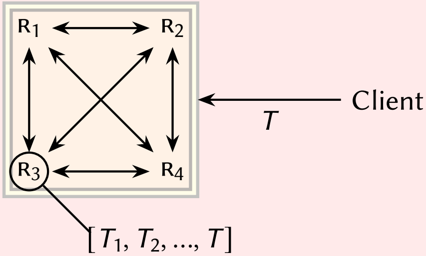
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.
5. Append-only updates to ledger.



Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.
5. Append-only updates to ledger.
6. Cryptography.



Bitcoin: A permissionless blockchain

The participants are not known and can change.

Rationale: Fully decentralized and open cryptocurrencies

- Bitcoin, Ethereum, ...
- Scale to thousands of participants.
- Low transaction processing throughput.
- Very high transaction latencies.

We focus on permissioned blockchains

All participants are known.

Rationale: Data processing in managed environment

- Support different attack models than cryptocurrencies.
- Easier to support low latencies and high throughputs.
- Downside: changing participants is hard.

Many ideas also apply to permissionless blockchains.

Towards high-performance resilient data processing:

What do we already have?

We have consensus: PBFT, PAXOS, POW, ...

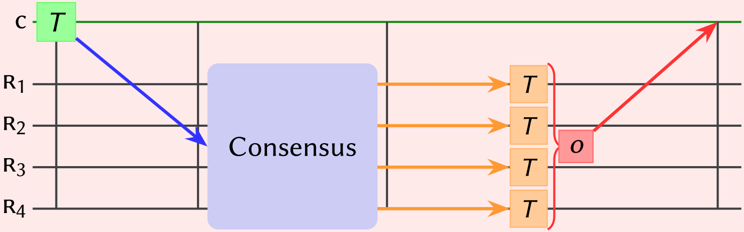
Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.

We have consensus: PBFT, PAXOS, POW, ...

Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.
Validity	Every decided-on transaction is a client request.
Response	Clients learn about the outcome of their requests.
Service	Every client will be able to request transactions.

We have consensus: PBFT, PAXOS, POW, …

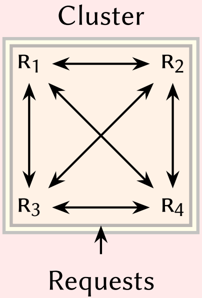
Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.
Validity	Every decided-on transaction is a client request.
Response	Clients learn about the outcome of their requests.
Service	Every client will be able to request transactions.



Operating a fully-replicated ledger using consensus

Each replica maintains a copy of the ledger:
Append-only sequence of transactions.

1. Use consensus to select the ρ-th client request T.
2. Append T as the ρ-th entry to the ledger.
3. Execute T as the ρ-th entry, inform client.



Consistent state: Linearizable order and deterministic execution
On identical inputs, execution of transactions at all non-faulty replicas *must produce identical outputs*.

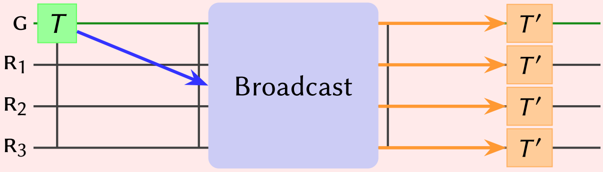
Variations on consensus: Byzantine Broadcast (Generals)

Assume a replica G is the general and holds transaction T.
A *Byzantine broadcast algorithm* is an algorithm satisfying:

Termination Each non-faulty replica decides on a transaction.

Non-divergence Non-faulty replicas decide on the same transaction.

Dependence If the general G is non-faulty, then non-faulty replicas will decide on T.



($T' = T$ if the general G is non-faulty).

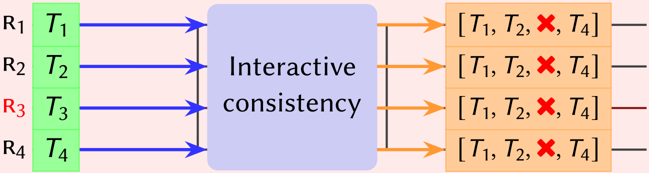
Variations on consensus: Interactive consistency

Assume **n** replicas and each replica R_i holds a transaction T_i.
An *interactive consistency algorithm* is an algorithm satisfying:

Termination Each non-faulty replica decides on **n** transactions.

Non-divergence Non-faulty replicas decide on the same transactions.

Dependence If replica R_j is non-faulty, then non-faulty replicas will decide on T_j.



(As R_3 is faulty: ✖ can be anything)

Distributed fully-replicated systems: The CAP Theorem

Consistency	Does every participant have exactly the same data?
Availability	Does the system continuously provide services?
Partitioning	Can the system cope with network disturbances?

Theorem (The CAP Theorem)

Can provide at most two-out-of-three of these properties.

Distributed fully-replicated systems: The CAP Theorem

Consistency Does every participant have exactly the same data?

Availability Does the system continuously provide services?

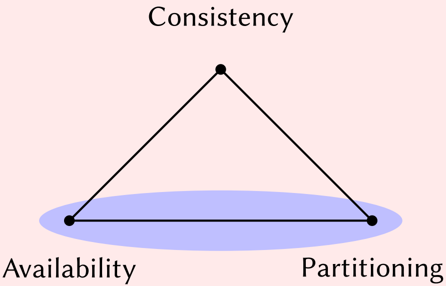
Partitioning Can the system cope with network disturbances?

Theorem (The CAP Theorem)

Can provide at most two-out-of-three of these properties.

CAP Theorem uses narrow definitions!

The CAP Theorem and Blockchains

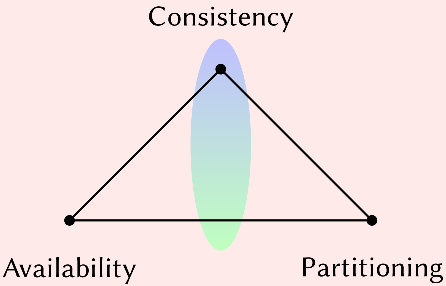


Permissionless Blockchains

Open membership focuses on Availability and Partitioning.

$\Longrightarrow$ Consistency not guaranteed (e.g., forks).

The CAP Theorem and Blockchains



Permissioned Blockchains

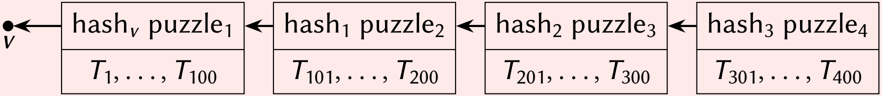
Consistency at all costs.

$\Longrightarrow$ Availability when communication is reliable.

$\Longrightarrow$ Some network failure when replicas are reliable.

What else do we have?

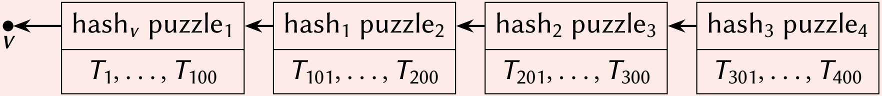
- A lot of *theory* on consensus: consensus is costly.
- PBFT: A practical Byzantine fault-tolerant consensus protocol.
- Tamper-proof *ledgers*.



  Exact details: depend on consensus, application, attack model, ...
- Many *cryptographic tools*.

What else do we have?

- A lot of *theory* on consensus: consensus is costly.
- PBFT: A practical Byzantine fault-tolerant consensus protocol.
- Tamper-proof *ledgers*.



  Exact details: depend on consensus, application, attack model, …
- Many *cryptographic tools*.

What about high-performance?

Theory on consensus: Summary

Limitations of practical consensus

- No asynchronous communication!
- Dealing with $\mathbf{f}$ malicious failures requires $\mathbf{n} > 3\mathbf{f}$ replicas.
- Worst-case: at least $\Omega(\mathbf{f} + 1)$ phases of communication.
- Worst-case: at least $\Omega(\mathbf{nf})$ signatures and $\Omega\left(\mathbf{n} + \mathbf{f}^2\right)$ messages.
- Network must stay connected when removing $2\mathbf{f}$ replicas.

Consensus in practice

Asynchronous communication, $\mathbf{n} > 3\mathbf{f}$, clique network:

$\implies$ termination only when communication is reliable.

Towards high-performance resilient data processing:

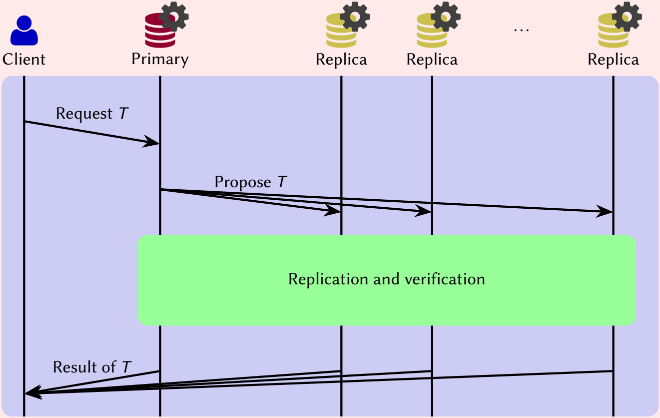
What do we already have?

PBFT

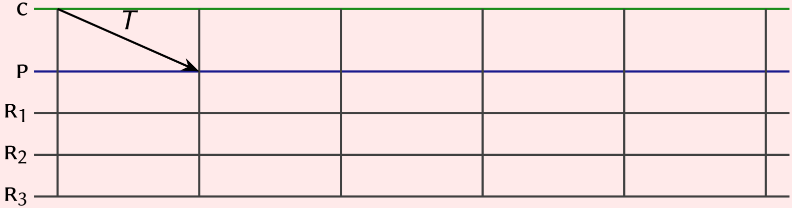
PBFT: Practical Byzantine Fault Tolerance

Primary Coordinates consensus: propose transactions to replicate.

Backup Accept transactions and verifies behavior of primary.

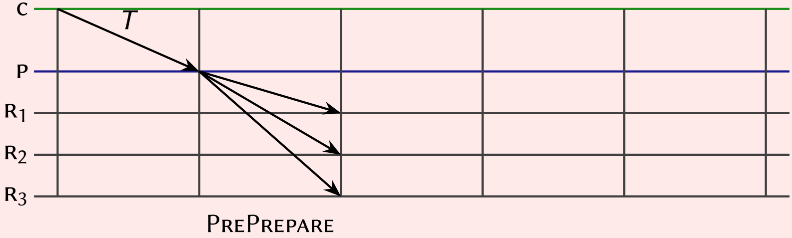


PBFT: Normal-case protocol in view v



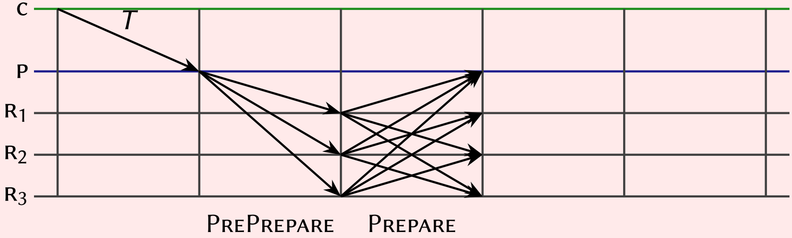
$$\langle T \rangle_{c}.$$

PBFT: Normal-case protocol in view v



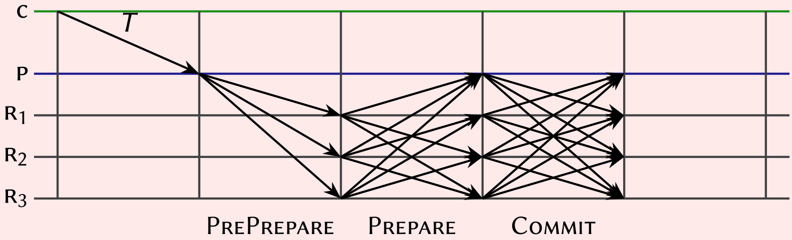
$$\text{PREPREPARE}(\langle T \rangle_c, v, \rho).$$

PBFT: Normal-case protocol in view v



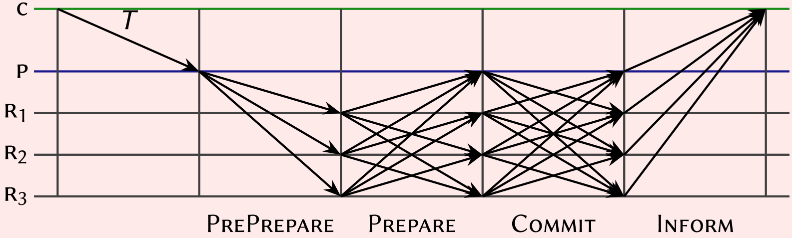
If receive PREPREPARE message m: PREPARE(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical PREPARE(m) messages: COMMIT(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical COMMIT(m) messages: execute, INFORM($\langle T \rangle_c, \rho, r$).

PBFT: Normal-case consensus

Theorem

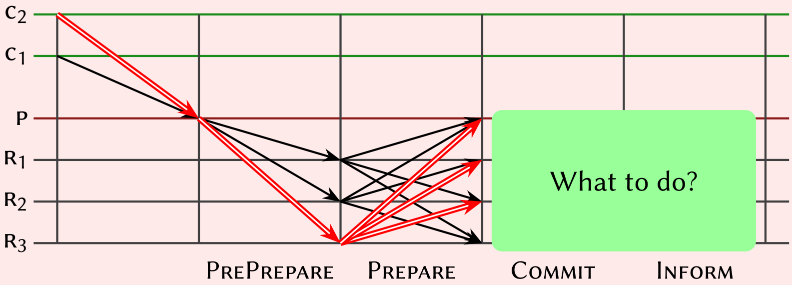
If the primary is non-faulty and communication is reliable,
then the normal-case of PBFT *ensures consensus on T in round ρ.*

PBFT: Normal-case consensus

Theorem

If the primary is non-faulty and communication is reliable, then the normal-case of PBFT *ensures consensus on T in round ρ.*

Example (Byzantine primary, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)

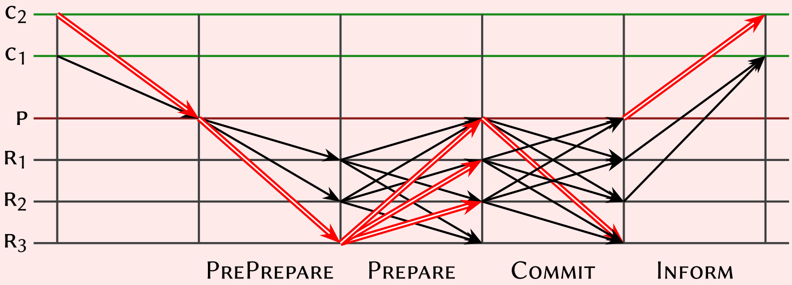


PBFT: Normal-case consensus

Theorem

If the primary is non-faulty and communication is reliable, then the normal-case of PBFT *ensures consensus on T in round ρ.*

Example (Byzantine primary, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)



PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas $\textsc{r}_i$, $i \in \{1, 2\}$, *commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{\mathrm{c}_i}, v, \rho)$, *then* $\langle T_1 \rangle_{\mathrm{c}_1} = \langle T_2 \rangle_{\mathrm{c}_2}$.

PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

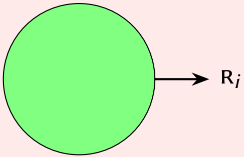
Theorem (Castro et al.)

If replicas R_i, $i \in \{1, 2\}$, *commit to* $m_i = \text{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$, *then* $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica R_i commits to m_i:

$\mathbf{n} - \mathbf{f}$ messages $\text{Prepare}(m_i)$



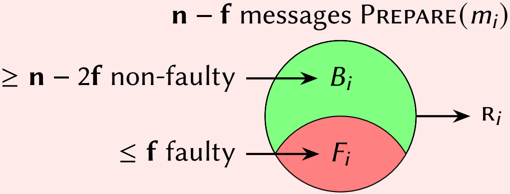
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas R_i, $i \in \{1, 2\}$, *commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$,
then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica R_i commits to m_i:



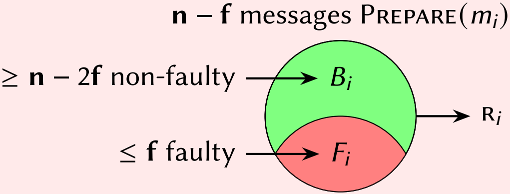
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas R_i, $i \in \{1, 2\}$, *commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$,
then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica R_i commits to m_i:



If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(\mathbf{n} - 2\mathbf{f})$.

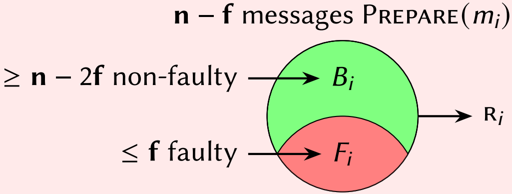
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas R_i, $i \in \{1, 2\}$, *commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$,
then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica R_i commits to m_i:



If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(\mathbf{n} - 2\mathbf{f})$.

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f}$$

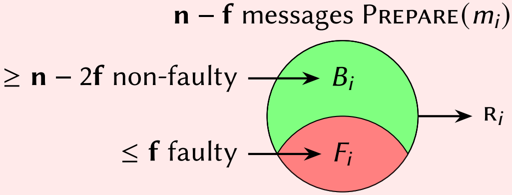
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas R_i, $i \in \{1, 2\}$, *commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$, *then* $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica R_i commits to m_i:



If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(\mathbf{n} - 2\mathbf{f})$.

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f} \qquad \text{iff} \qquad 2\mathbf{n} - 4\mathbf{f} \leq \mathbf{n} - \mathbf{f}$$

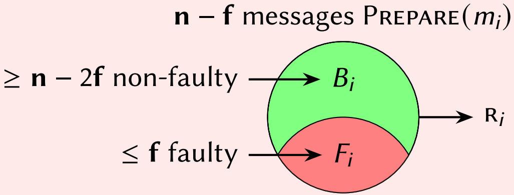
Pbft: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

If replicas $\textsc{r}_i$*,* $i \in \{1, 2\}$*, commit to* $m_i = \textsc{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$*,*
then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$*.*

Proof.

Replica $\textsc{r}_i$ commits to m_i:



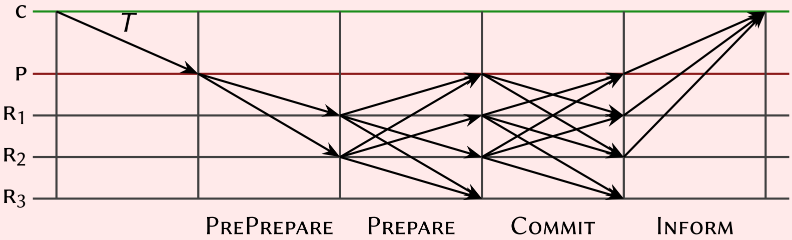
If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(\mathbf{n} - 2\mathbf{f})$.

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f} \qquad \text{iff} \qquad 2\mathbf{n} - 4\mathbf{f} \leq \mathbf{n} - \mathbf{f} \qquad \text{iff} \qquad \mathbf{n} \leq 3\mathbf{f}. \qquad \square$$

PBFT: Primary failure versus malicious replicas

Primary P is faulty

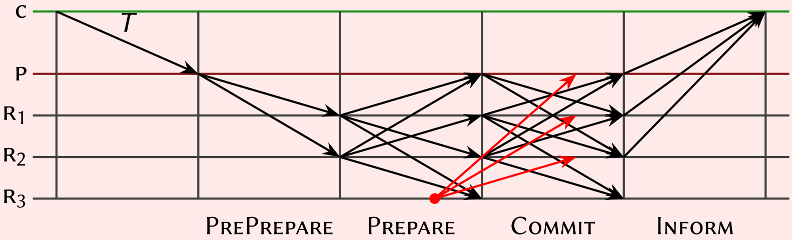
ignores R3



PBFT: Primary failure versus malicious replicas

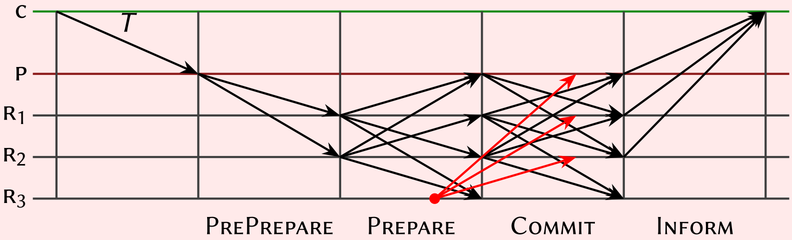
Primary P is faulty

ignores R3

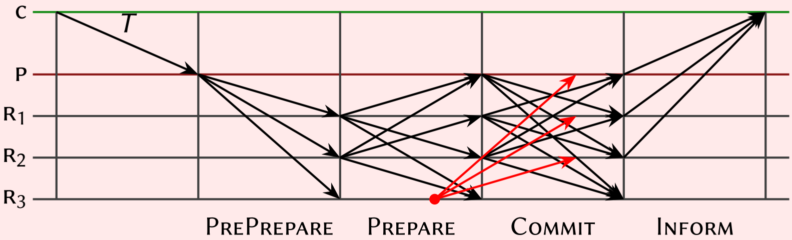


PBFT: Primary failure versus malicious replicas

Primary P is faulty
ignores R3

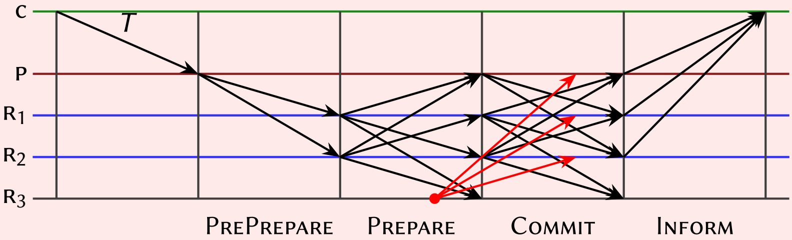


Replica R3 is malicious
pretends to be ignored

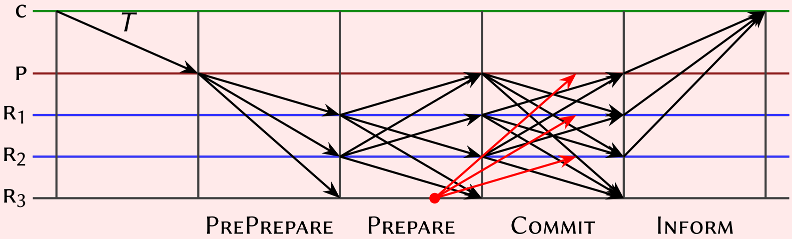


Pbft: Primary failure versus malicious replicas

Primary p is faulty
ignores R_3



Replica R_3 is malicious
pretends to be ignored



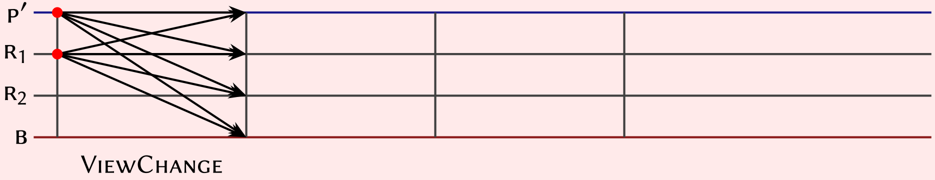
PBFT: Detectable primary failures

If the primary behaves faulty to $> \mathbf{f}$ non-faulty replicas, then failure of the primary is detectable.

Replacing the primary: View-change at replica R

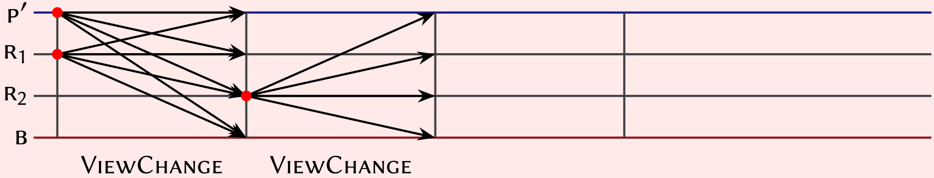
1. R detects *failure* of the current primary P.
2. R chooses a new primary P′ (the next replica).
3. R provides P′ with its *current state*.
4. P′ proposes a *new view*.
5. If the new view is valid, then R switches to this view.

PBFT: A view-change in view v



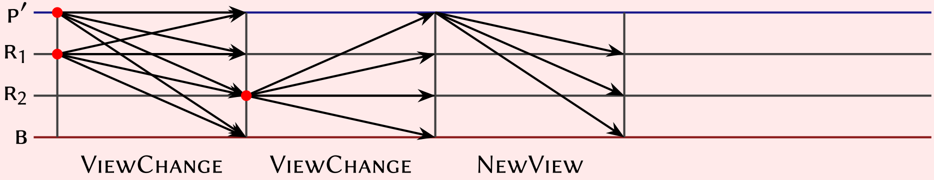
Send VIEWCHANGE(E, v) with E all prepared transactions.

PBFT: A view-change in view v



Indirect failure detection by R_2.

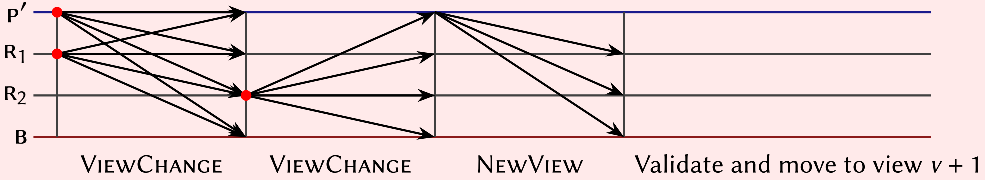
PBFT: A view-change in view v



If $\mathbf{n} - \mathbf{f}$ valid VIEWCHANGE(E, v) messages: NEWVIEW$(v + 1, \mathcal{E}, \mathcal{N})$.

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid VIEWCHANGE messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

PBFT: A view-change in view v



Move to view $v + 1$ if NewView$(v + 1, \mathcal{E}, \mathcal{N})$ is valid.

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid ViewChange messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

PBFT: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

Let NEWVIEW$(v', \mathcal{E}, \mathcal{N})$ *be a well-formed* NEWVIEW *message.*
If a set S of $\mathbf{n} - 2\mathbf{f}$ non-faulty replicas committed to m in view $v < v'$,
then $\mathcal{E}$ contains a VIEWCHANGE *message preparing m.*

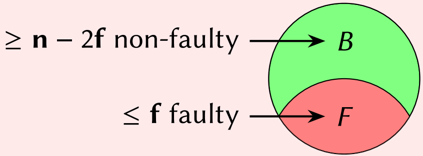
Theorem (Castro et al.)

Let NEWVIEW$(v', \mathcal{E}, \mathcal{N})$ *be a well-formed* NEWVIEW *message.*
If a set S of $\mathbf{n} - 2\mathbf{f}$ non-faulty replicas committed to m in view $v < v'$,
then $\mathcal{E}$ contains a VIEWCHANGE *message preparing m.*

Proof.

The VIEWCHANGE messages in $\mathcal{E}$:

$\mathbf{n} - \mathbf{f}$ messages VIEWCHANGE$(E, v' - 1)$



Pbft: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

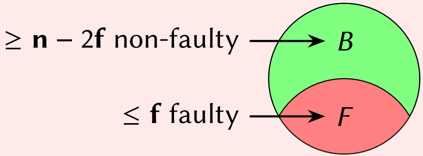
Theorem (Castro et al.)

Let NewView$(v', \mathcal{E}, \mathcal{N})$ *be a well-formed* NewView *message.*
If a set S of $\mathbf{n} - 2\mathbf{f}$ non-faulty replicas committed to m in view $v < v'$,
then $\mathcal{E}$ contains a ViewChange *message preparing m.*

Proof.

The ViewChange messages in $\mathcal{E}$:

$\mathbf{n} - \mathbf{f}$ messages ViewChange$(E, v' - 1)$



if $S \cap B = \emptyset$, then $|S \cup B| \geq 2(\mathbf{n} - 2\mathbf{f})$, a contradiction! $\square$

PBFT: Further dealing with failures

1. *Undetected failures*: e.g., ignored replicas.
   At least $\mathbf{n} - 2\mathbf{f} > \mathbf{f}$ non-faulty replicas participate: *checkpoints*.

PBFT: Further dealing with failures

1. *Undetected failures*: e.g., ignored replicas.
   At least $\mathbf{n} - 2\mathbf{f} > \mathbf{f}$ non-faulty replicas participate: *checkpoints*.

2. *Detected failures*: primary replacement.
   Worst-case: a sequence of $\mathbf{f}$ view-changes ($\Omega(\mathbf{f})$ phases).

PBFT: Further dealing with failures

1. *Undetected failures*: e.g., ignored replicas.
   At least $\mathbf{n} - 2\mathbf{f} > \mathbf{f}$ non-faulty replicas participate: *checkpoints*.

2. *Detected failures*: primary replacement.
   Worst-case: a sequence of $\mathbf{f}$ view-changes ($\Omega(\mathbf{f})$ phases).

3. *View-change cost*: includes all previous transactions!
   Checkpoints: view-change includes *last successful* checkpoint.

PBFT: Further dealing with failures

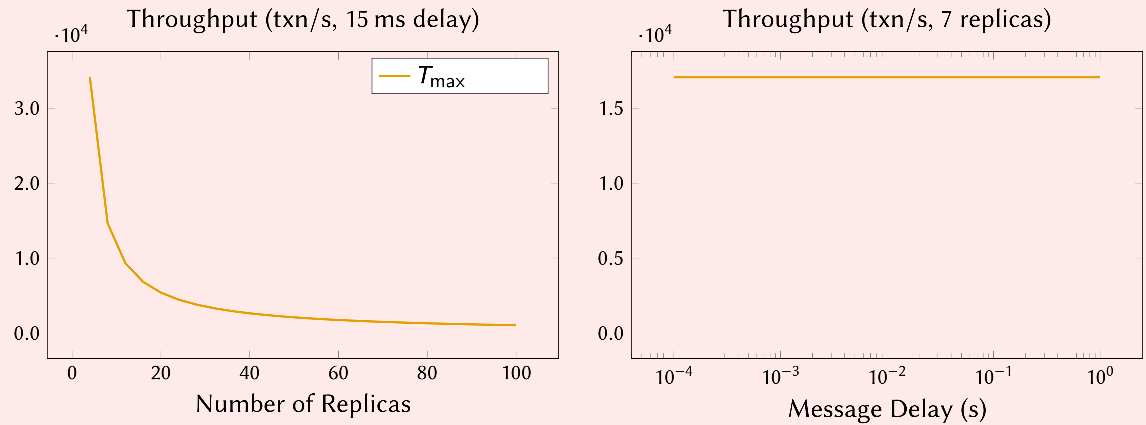
1. *Undetected failures*: e.g., ignored replicas.
   At least $\mathbf{n} - 2\mathbf{f} > \mathbf{f}$ non-faulty replicas participate: *checkpoints*.

2. *Detected failures*: primary replacement.
   Worst-case: a sequence of $\mathbf{f}$ view-changes ($\Omega(\mathbf{f})$ phases).

3. *View-change cost*: includes all previous transactions!
   Checkpoints: view-change includes *last successful* checkpoint.

4. *Unreliable communication*: replacement of non-faulty primaries.
   Worst-case: replacements until communication becomes *reliable*.

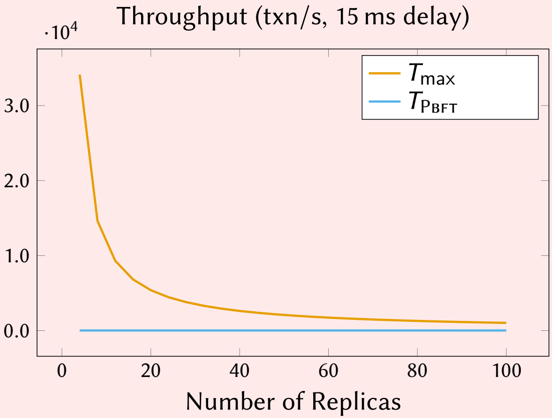
PBFT: Modeling real-world performance[1]

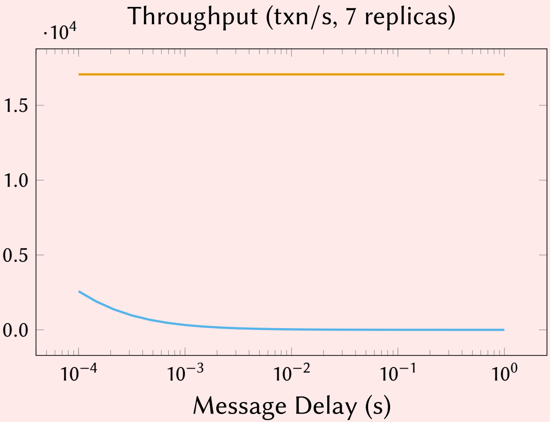


(Maximum throughput of any primary-backup broadcast protocol)

[1] Bandwidth: 100 MiB/s, PREPREPARE message size: 1024 B, PREPARE and COMMIT message size: 256 B.

PBFT: Modeling real-world performance[1]

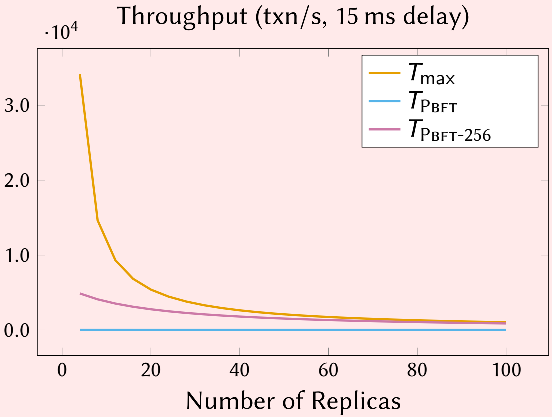


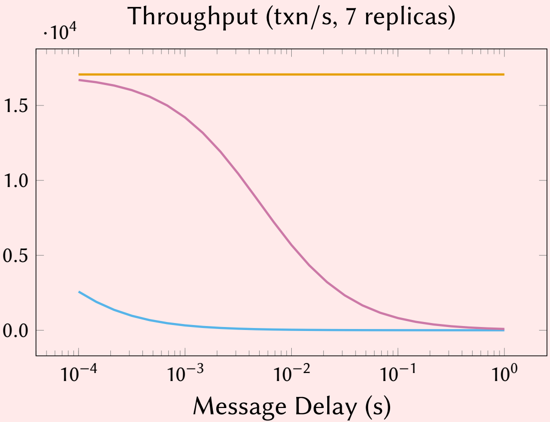


(Maximum throughput of in-order PBFT)

[1] Bandwidth: 100 MiB/s, PREPREPARE message size: 1024 B, PREPARE and COMMIT message size: 256 B.

PBFT: Modeling real-world performance[1]

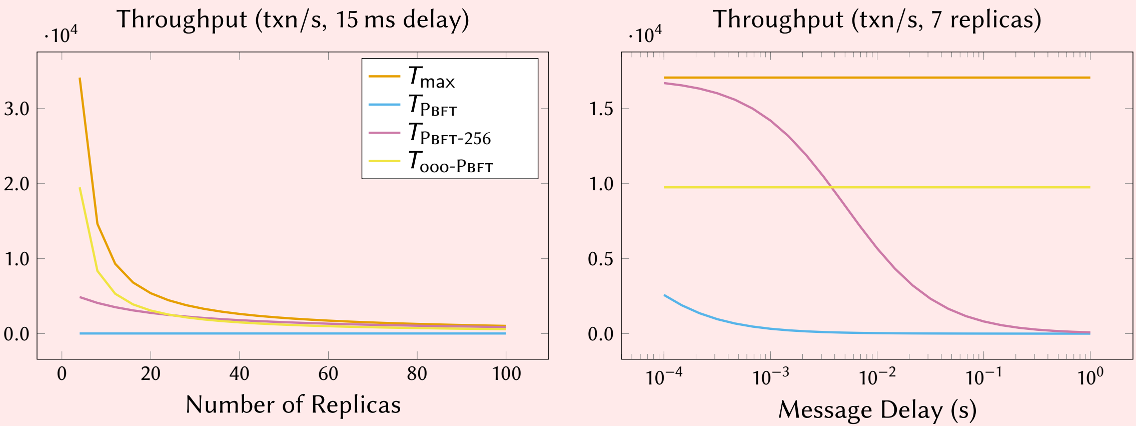




(Maximum throughput of in-order PBFT with batching, 256 txn/batch)

[1] Bandwidth: 100 MiB/s, PREPREPARE message size: 1024 B, PREPARE and COMMIT message size: 256 B.

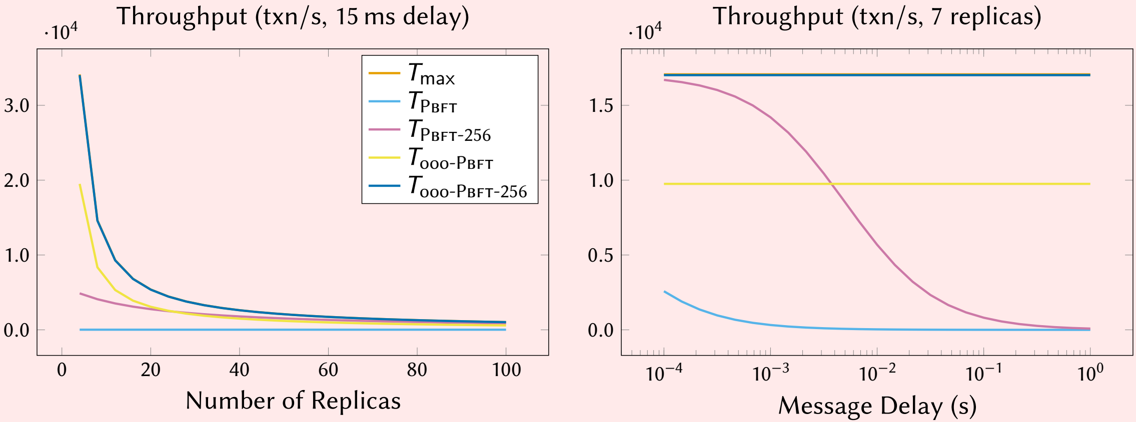
PBFT: Modeling real-world performance[1]



(Maximum throughput of out-of-order PBFT)

[1] Bandwidth: 100 MiB/s, PREPREPARE message size: 1024 B, PREPARE and COMMIT message size: 256 B.

PBFT: Modeling real-world performance[1]



(Maximum throughput of out-of-order PBFT with batching, 256 txn/batch)

[1] Bandwidth: 100 MiB/s, PREPREPARE message size: 1024 B, PREPARE and COMMIT message size: 256 B.

Towards high-performance resilient data processing:

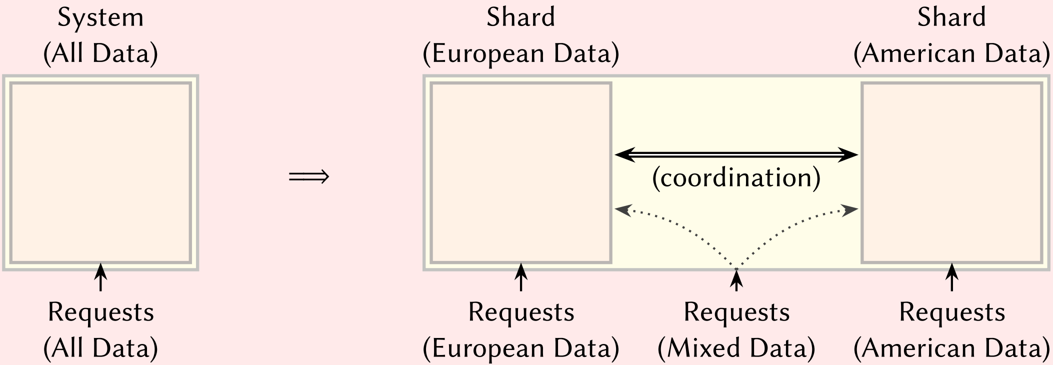
Where can we improve?

A look at high-performance data processing

Scalability: adding resources $\Longrightarrow$ adding performance.

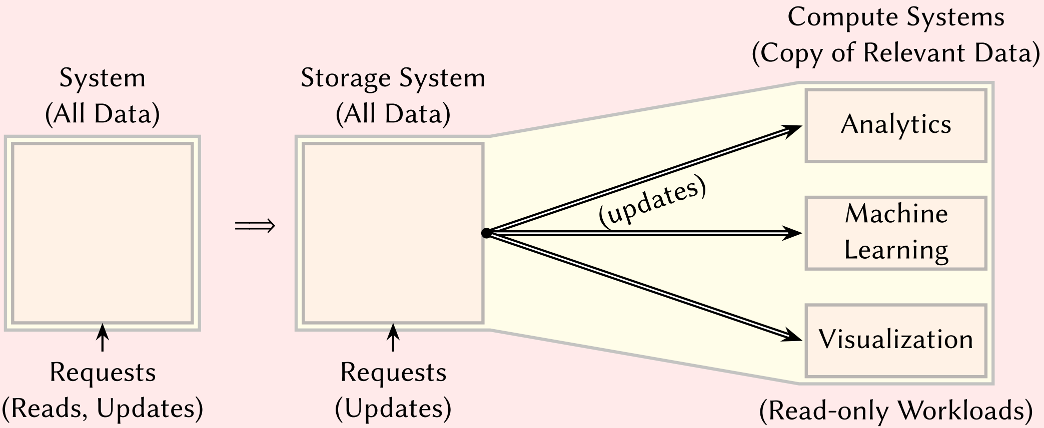
Full replication: adding resources (replicas) $\Longrightarrow$ less performance!

Sharding and Geo-scale aware sharding



Adding shards $\Longrightarrow$ adding throughput (parallel processing), adding storage.

Role Specialization: Read-only workloads



Specializing roles ⟹ adding throughput (parallel processing, specialized hardware, . . .).

Towards high-performance resilient data processing:

What new tools do we need?

Central ideas for improvement

Reminder

We can make a resilient cluster that manages data: *blockchains*.

- **Sharding**: make each shard an independent blockchain.
  Requires: *reliable communication between blockchains.*
  Permissionless blockchains: relays, atomic swaps!
- **Role Specialization**: make the storage system a blockchain.
  Requires: *reliable read-only updates of the blockchain.*
  Permissionless blockchains: light clients!

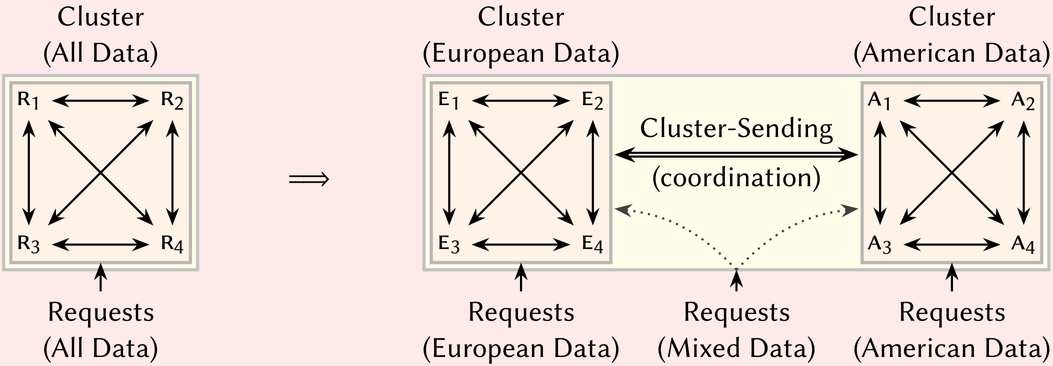
Consensus is of no use here if we want efficiency.

Towards high-performance resilient data processing:

What new tools do we need?

Sharding

Sharding: Reliable communication between blockchains



The Byzantine cluster-sending problem.

The Byzantine cluster-sending problem

The problem of sending a value v from a cluster $\mathcal{C}_1$ to a cluster $\mathcal{C}_2$ such that

- all non-faulty replicas in $\mathcal{C}_2$ *RECEIVE* the value v;
- all non-faulty replicas in $\mathcal{C}_1$ *CONFIRM* that the value v was received; and
- $\mathcal{C}_2$ only receives a value v if all non-faulty replicas in $\mathcal{C}_1$ *AGREE* upon sending v.

Requirements to provide reliable communication between clusters with Byzantine replicas.

Global communication versus local communication

Straightforward cluster-sending solution (crash failures)

Pair-wise broadcasting with $(\mathbf{f}_1 + 1)(\mathbf{f}_2 + 1) \approx \mathbf{f}_1 \times \mathbf{f}_2$ messages.

Global communication versus local communication

Straightforward cluster-sending solution (crash failures)

Pair-wise broadcasting with $(\mathbf{f}_1 + 1)(\mathbf{f}_2 + 1) \approx \mathbf{f}_1 \times \mathbf{f}_2$ messages.

	Ping round-trip times (ms)						*Bandwidth (Mbit/s)*					
	OR	IA	Mont.	BE	TW	Syd.	OR	IA	Mont.	BE	TW	Syd.
Oregon	≤ 1	38	65	136	118	161	7998	669	371	194	188	136
Iowa		≤ 1	33	98	153	172		10004	752	243	144	120
Montreal			≤ 1	82	186	202			7977	283	111	102
Belgium				≤ 1	252	270				9728	79	66
Taiwan					≤ 1	137					7998	160
Sydney						≤ 1						7977

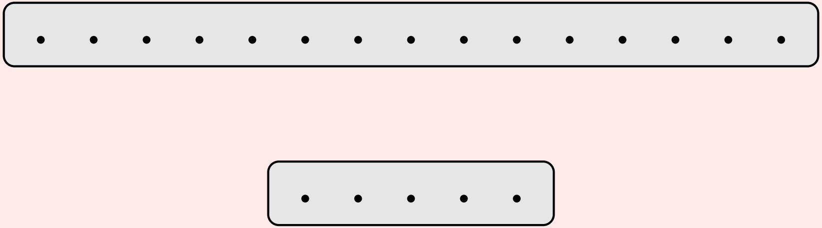
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



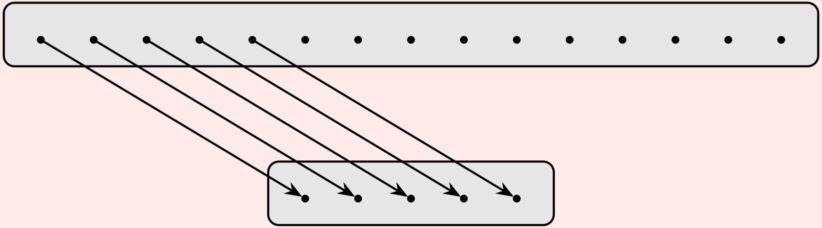
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



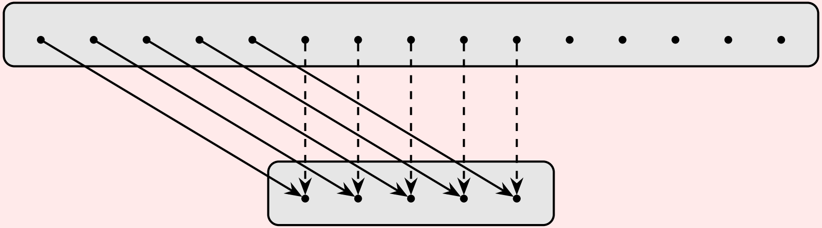
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



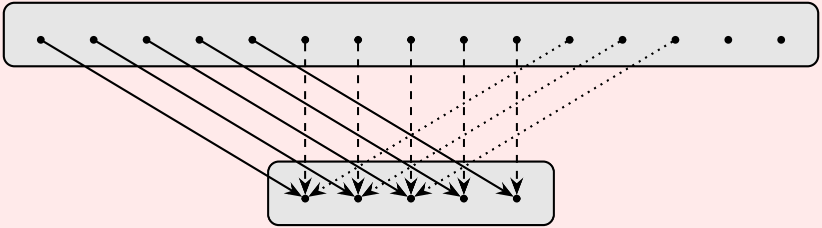
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



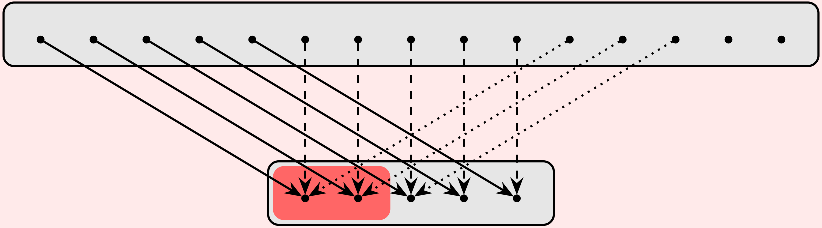
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



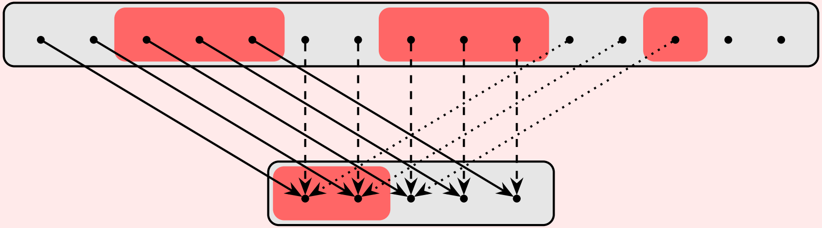
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



Lower bounds for cluster-sending: Results

Theorem (Cluster-sending lower bound, simplified)

We need to exchange $\max(\mathbf{n}_1, \mathbf{n}_2)$ *messages to do cluster-sending.*

Theorem (Cluster-sending lower bound, crash failures)

Assume $\mathbf{n}_1 \geq \mathbf{n}_2$ *and let*

$$q = (\mathbf{f}_1 + 1) \operatorname{div} \mathbf{nf}_2; \qquad r = (\mathbf{f}_1 + 1) \bmod \mathbf{nf}_2.$$

We need to exchange at least $q\mathbf{n}_2 + r + \mathbf{f}_2 \operatorname{sgn} r \approx \mathbf{n}_1$ *messages to do cluster-sending.*

An optimal cluster-sending algorithm (crash failures)

Protocol for the sending cluster $\mathcal{C}_1$, $\mathbf{n}_1 \geq \mathbf{n}_2$, $\mathbf{n}_1 \geq \sigma$:

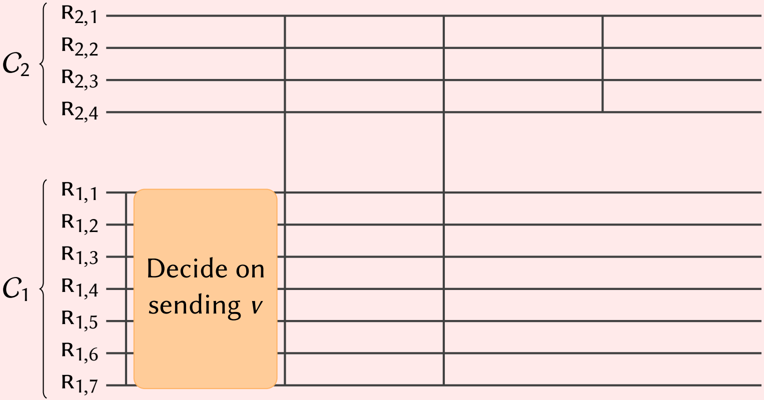
AGREE on sending v to $\mathcal{C}_2$.
Choose replicas $\mathcal{P} \subseteq \mathcal{C}_1$ with $|\mathcal{P}| = \sigma$.
Choose a $\mathbf{n}_2$-partition $\text{partition}(\mathcal{P})$ of $\mathcal{P}$.
for $P \in \text{partition}(\mathcal{P})$ **do**
    Choose replicas $Q \subseteq \mathcal{C}_2$ with $|Q| = |P|$.
    Choose a bijection $b : P \to Q$.
    for $\text{R}_1 \in P$ **do**
        Send v from R_1 to $b(\text{R}_1)$.

Protocol for the receiving cluster $\mathcal{C}_2$:

event $\text{R}_2 \in \mathcal{C}_2$ receives w from a replica in $\mathcal{C}_1$ **do**
    Broadcast w to all replicas in $\mathcal{C}_2$.
event $\text{R}_2 \in \mathcal{C}_2$ receives w from a replica in $\mathcal{C}_2$ **do**
    R_2 considers w *RECEIVED*.

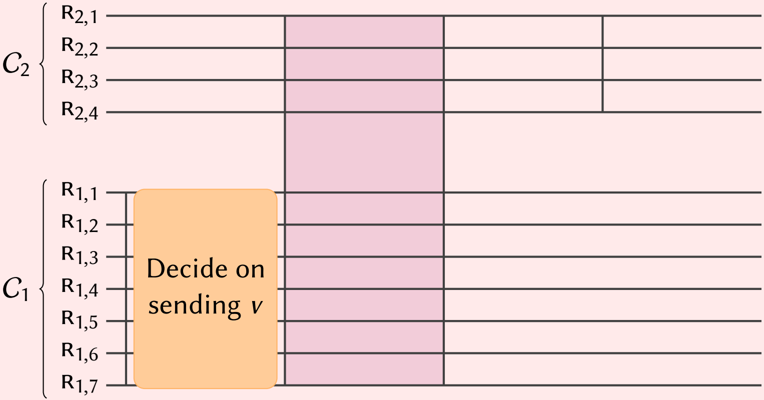
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = 7$, $\mathbf{n}_2 = 4$, $\mathbf{f}_1 = 3$, $\mathbf{f}_2 = 1$, $\sigma = 6$



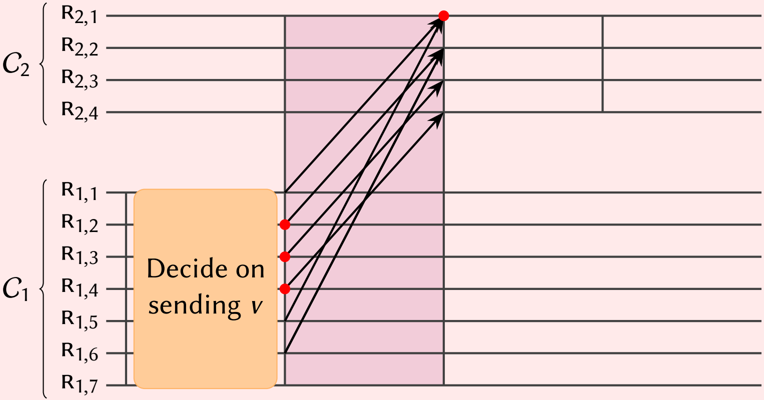
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = 7$, $\mathbf{n}_2 = 4$, $\mathbf{f}_1 = 3$, $\mathbf{f}_2 = 1$, $\sigma = 6$



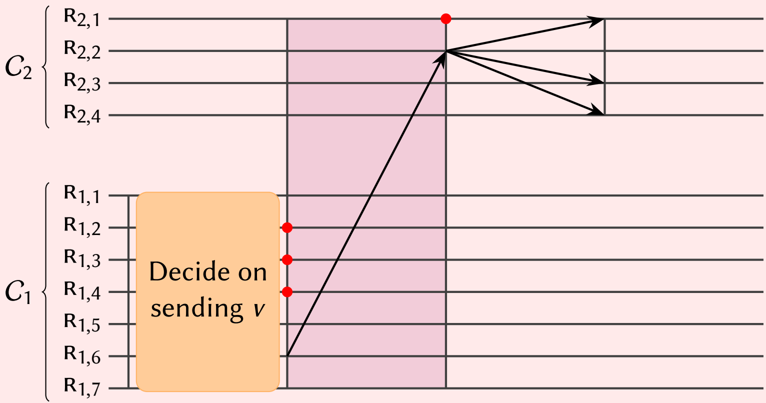
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = 7$, $\mathbf{n}_2 = 4$, $\mathbf{f}_1 = 3$, $\mathbf{f}_2 = 1$, $\sigma = 6$



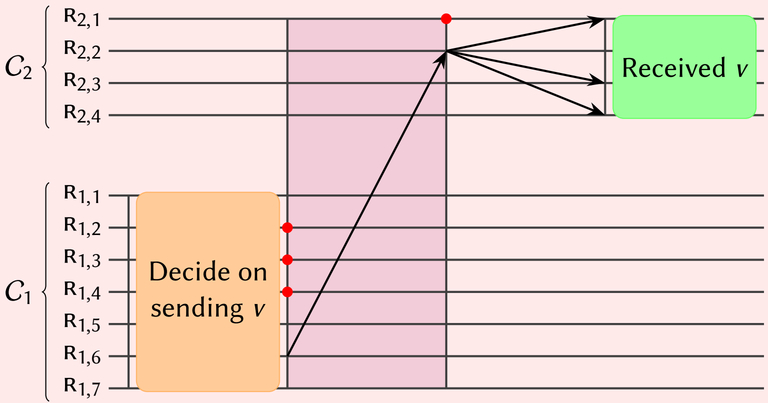
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = 7$, $\mathbf{n}_2 = 4$, $\mathbf{f}_1 = 3$, $\mathbf{f}_2 = 1$, $\sigma = 6$



An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = 7$, $\mathbf{n}_2 = 4$, $\mathbf{f}_1 = 3$, $\mathbf{f}_2 = 1$, $\sigma = 6$



Cluster-sending: Can we do better

Pessimistic

No: these protocols are worst-case optimal.

Cannot do better than *linear communication* in the size of the clusters.

Cluster-sending: Can we do better

Pessimistic

No: these protocols are worst-case optimal.
Cannot do better than *linear communication* in the size of the clusters.

Optimistic—upcoming results

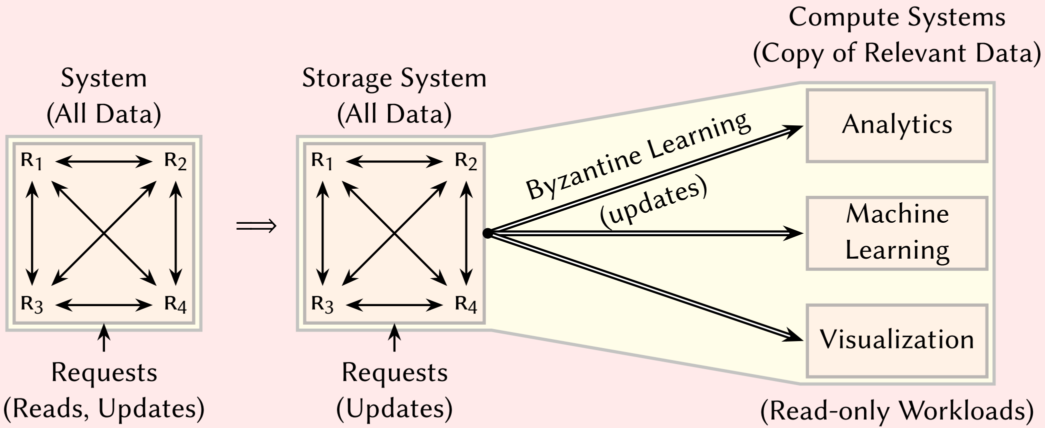
Yes: if we randomly choose sender and receiver, then we often do much better!
Probabilistic approach: expected-case only *constant communication* (four steps).

Towards high-performance resilient data processing:

What new tools do we need?

Role Specialization

Role Specialization: Reliable read-only updates of the blockchain



The Byzantine learner problem.

The Byzantine learner problem

The problem of sending a ledger $\mathcal{L}$ maintained by a cluster C to a learner L such that:

- the learner L will eventually *RECEIVE ALL* transactions in $\mathcal{L}$; and
- the learner L will *ONLY RECEIVE* transactions in $\mathcal{L}$.

Practical requirements

- Minimizing overall communication.
- Load balancing among all replicas in C.

Background: Information dispersal algorithms

Definition

Let v be a value with storage size $s = \|v\|$.
An *information dispersal algorithm* can encode v in $\mathbf{n}$ pieces v' such that v can be *decoded* from every set of $\mathbf{n} - \mathbf{f}$ such pieces.

Theorem (Rabin 1989)

The IDA algorithm is an optimal information dispersal algorithm:

- *Each piece v' has size* $\left\lceil \frac{\|v\|}{\mathbf{n}-\mathbf{f}} \right\rceil$.
- *The* $\mathbf{n} - \mathbf{f}$ *pieces necessary for decoding have a total size of* $(\mathbf{n} - \mathbf{f}) \left\lceil \frac{\|v\|}{(\mathbf{n}-\mathbf{f})} \right\rceil \approx \|v\|$.

The delayed-replication algorithm

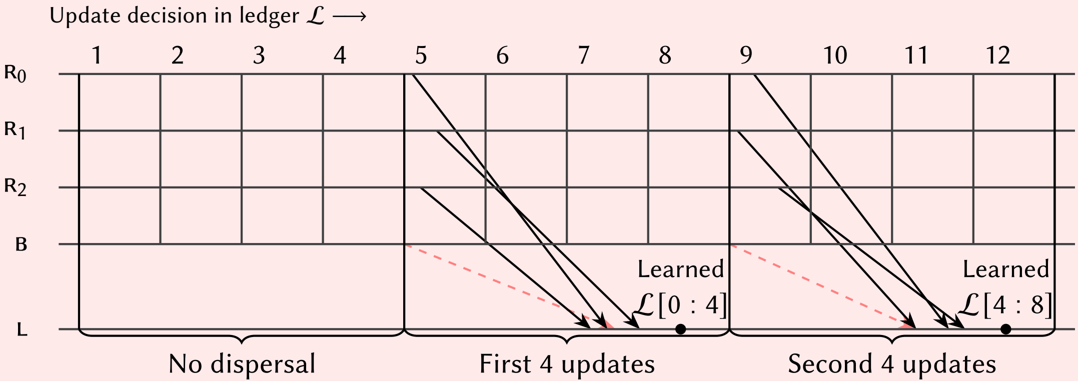
Idea: $\mathcal{C}$ sends a ledger $\mathcal{L}$ to learner $\textsc{l}$

1. Partition the ledger $\mathcal{L}$ in sequences S of $\mathbf{n}$ transactions.
2. Replica $\textsc{r}_i \in \mathcal{C}$ encodes S into the i-th IDA piece S_i.
3. Replica $\textsc{r}_i \in \mathcal{C}$ sends S_i with a checksum $C_i(S)$ of S to learner $\textsc{l}$.
4. *Learner $\textsc{l}$ receives at least $\mathbf{n} - \mathbf{f}$ distinct and valid pieces and decodes S.*

Observation ($\mathbf{n} > 2\mathbf{f}$)

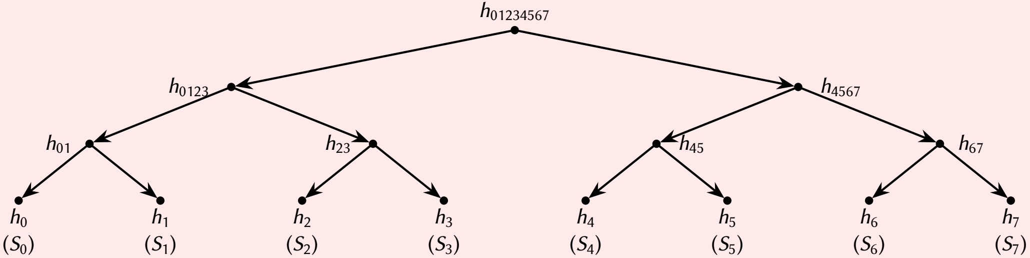
- Replica $\textsc{r}_i$ sends at most $B = \left\lceil \frac{\|S\|}{\mathbf{n}-\mathbf{f}} \right\rceil + c \leq \frac{2\|S\|}{\mathbf{n}} + 1 + c = \mathcal{O}\left(\frac{\|S\|}{\mathbf{n}} + c\right)$ bytes.
- Learner $\textsc{l}$ receives at most $\mathbf{n} \cdot B = \mathcal{O}\left(\|S\| + c\mathbf{n}\right)$ bytes.

Communication by the delayed-replication algorithm



Checksums: Use Merkle-trees to construct checksums

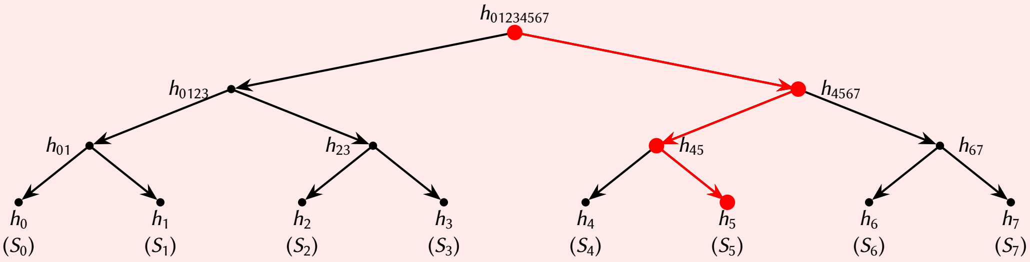
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by $\textsc{r}_5$).



Construct a Merkle tree for pieces $S_0, \ldots, S_7$.

Checksums: Use Merkle-trees to construct checksums

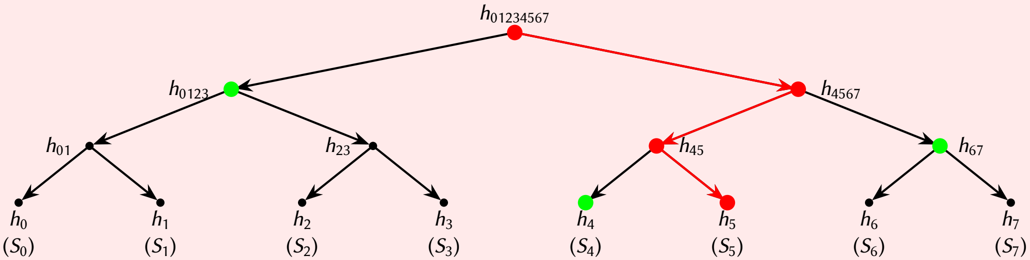
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by $\textsc{r}_5$).



Determine the path from root to S_5.

Checksums: Use Merkle-trees to construct checksums

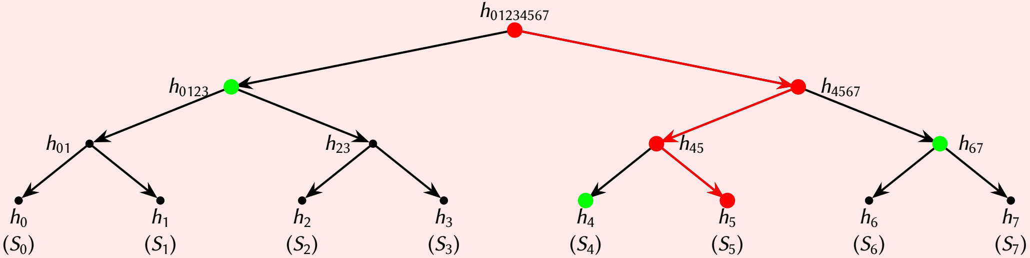
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by $\textsc{r}_5$).



Select *root* and *neighbors*: $C_5(S) = [h_4, h_{67}, h_{0123}, h_{01234567}]$.

Checksums: Use Merkle-trees to construct checksums

Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by R_5).



If one knows the root: *validity* of individual pieces can be determined.

Delayed-replication: Main result ($\mathbf{n} > 2\mathbf{f}$)

Theorem

Consider the learner L, *replica* R $\in C$, *and ledger* $\mathcal{L}$.
The delayed-replication algorithm with tree checksums guarantees

1. L *will learn* $\mathcal{L}$;
2. L *will receive at most* $|\mathcal{L}|$ *messages with a total size of* $\mathcal{O}\left(\|\mathcal{L}\| + |\mathcal{L}| \log \mathbf{n}\right)$;
3. L *will only need at most* $|\mathcal{L}|/\mathbf{n}$ *decode steps;*
4. R *will sent at most* $|\mathcal{L}|/\mathbf{n}$ *messages to* L *of size* $\mathcal{O}\left(\frac{\|\mathcal{L}\|+|\mathcal{L}|\log \mathbf{n}}{\mathbf{n}}\right)$.

Adding replicas to cluster $C \implies$ less communication per replica!

Application: Scalable storage for resilient systems

- Clusters typically need a *view* $\mathcal{V}$ on the data to decide whether updates are valid.
- Clusters only need the full ledger $\mathcal{L}$ for *recovery*.
- We can use *delayed-replication* to reduce the data each replica has to store.

Theorem

The storage cost per replica can be reduced from

$$\mathcal{O}\left(\|\mathcal{L}\| + \|\mathcal{V}\|\right) \quad \textit{to} \quad \mathcal{O}\left(\frac{\|\mathcal{L}\|}{\mathbf{n}-\mathbf{f}} + \frac{|\mathcal{L}|}{\mathbf{n}}\log(\mathbf{n}) + \|\mathcal{V}\|\right).$$

Towards high-performance resilient data processing:

Concluding remarks

Conclusion

We need an extensive toolbox!

	(permissioned)	(permissionless)
▶ Consensus	PBFT, Paxos, …	PoW, PoS, …
▶ Cross-blockchain communication	Cluster-sending	Relays, atomic swaps
▶ Read-only participation	Byzantine learning	Light clients

High-performance resilient data processing is nearby.

Ongoing work

Initial results are available

- Cluster-sending: DISC 2019, doi: 10.4230/LIPIcs.DISC.2019.45.
- Byzantine learning: ICDT 2020, doi: 10.4230/LIPIcs.ICDT.2020.17.
- Geo-aware consensus: VLDB 2020, doi: 10.14778/3380750.3380757.

More about us and our work

- Jelle Hellings https://jhellings.nl/.
- ExpoLab Creativity Unfolded https://expolab.org/.
- ResilientDB Security, Privacy Reloaded https://resilientdb.com/.

References I

- Ittai Abraham et al. *Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience*. 2018.
- Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. "CAPER: A Cross-application Permissioned Blockchain". In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1385–1398. ISSN: 2150-8097. DOI: 10.14778/3342263.3342275.
- Elli Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538.
- Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. "RBFT: Redundant Byzantine Fault Tolerance". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53.
- Pierre-Louis Aublin et al. "The Next 700 BFT Protocols". In: *ACM Transactions on Computer Systems* 32.4 (2015), 12:1–12:45. DOI: 10.1145/2658994.
- Eric Brewer. "CAP twelve years later: How the "rules" have changed". In: *Computer* 45.2 (2012), pp. 23–29. DOI: 10.1109/MC.2012.37.

References II

Eric A. Brewer. "Towards Robust Distributed Systems (Abstract)". In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. ACM, 2000, pp. 7–7. DOI: 10.1145/343477.343502.

Christian Cachin and Marko Vukolic. "Blockchain Consensus Protocols in the Wild (Keynote Talk)". In: *31st International Symposium on Distributed Computing*. Vol. 91. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 1:1–1:16. DOI: 10.4230/LIPIcs.DISC.2017.1.

Michael Casey et al. *The Impact of Blockchain Technology on Finance: A Catalyst for Change*. Tech. rep. International Center for Monetary and Banking Studies, 2018. URL: https://www.cimb.ch/uploads/1/1/5/4/115414161/geneva21_1.pdf.

Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Transactions on Computer Systems* 20.4 (2002), pp. 398–461. DOI: 10.1145/571637.571640.

Miguel Correia et al. "Low complexity Byzantine-resilient consensus". In: *Distributed Computing* 17.3 (2005), pp. 237–249. DOI: 10.1007/s00446-004-0110-7.

Richard A. DeMillo, Nancy A. Lynch, and Michael J. Merritt. "Cryptographic Protocols". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. ACM, 1982, pp. 383–400. DOI: 10.1145/800070.802214.

References III

Tien Tuan Anh Dinh et al. "Untangling Blockchain: A Data Processing View of Blockchain Systems". In: *IEEE Transactions on Knowledge and Data Engineering* 30.7 (2018), pp. 1366–1385. DOI: 10.1109/TKDE.2017.2781227.

D. Dolev. "Unanimity in an unknown and unreliable environment". In: *22nd Annual Symposium on Foundations of Computer Science*. IEEE, 1981, pp. 159–168. DOI: 10.1109/SFCS.1981.53.

D. Dolev and H. Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM Journal on Computing* 12.4 (1983), pp. 656–666. DOI: 10.1137/0212045.

Danny Dolev. "The Byzantine generals strike again". In: *Journal of Algorithms* 3.1 (1982), pp. 14–30. DOI: 10.1016/0196-6774(82)90004-9.

Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. "On the Minimal Synchronism Needed for Distributed Consensus". In: *Journal of the ACM* 34.1 (1987), pp. 77–97. DOI: 10.1145/7531.7533.

Danny Dolev and Rüdiger Reischuk. "Bounds on Information Exchange for Byzantine Agreement". In: *Journal of the ACM* 32.1 (1985), pp. 191–204. DOI: 10.1145/2455.214112.

Partha Dutta, Rachid Guerraoui, and Marko Vukolic. *The Complexity of Asynchronous Byzantine Consensus*. Tech. rep. EPFL, 2004. URL: http://infoscience.epfl.ch/record/52690.

References IV

Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *Journal of the ACM* 35.2 (1988), pp. 288–323. DOI: 10.1145/42282.42283.

Paul Feldman and Silvio Micali. "Optimal Algorithms for Byzantine Agreement". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. ACM, 1988, pp. 148–161. DOI: 10.1145/62212.62225.

Michael J. Fischer and Nancy A. Lynch. "A lower bound for the time to assure interactive consistency". In: *Information Processing Letters* 14.4 (1982), pp. 183–186. DOI: 10.1016/0020-0190(82)90033-3.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *Journal of the ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121.

Juan A. Garay and Yoram Moses. "Fully Polynomial Byzantine Agreement for Processors in Rounds". In: *SIAM Journal on Computing* 27.1 (1998), pp. 247–290. DOI: 10.1137/S0097539794265232.

Yossi Gilad et al. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68. DOI: 10.1145/3132747.3132757.

References V

Seth Gilbert and Nancy Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services”. In: *SIGACT News* 33.2 (2002), pp. 51–59. DOI: 10.1145/564585.564601.

William J. Gordon and Christian Catalini. “Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Driven Interoperability”. In: *Computational and Structural Biotechnology Journal* 16 (2018), pp. 224–230. DOI: 10.1016/j.csbj.2018.06.003.

Jim Gray. “Notes on Data Base Operating Systems”. In: *Operating Systems, An Advanced Course*. Springer-Verlag, 1978, pp. 393–481. DOI: 10.1007/3-540-08755-9_9.

Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. “Brief Announcement: Revisiting Consensus Protocols through Wait-Free Parallelization”. In: *33rd International Symposium on Distributed Computing (DISC 2019)*. Vol. 146. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 44:1–44:3. DOI: 10.4230/LIPIcs.DISC.2019.44.

Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. *Fault-Tolerant Distributed Transactions on Blockchains*. (to appear). 2020.

References VI

- Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. "Permissioned Blockchain Through the Looking Glass: Architectural and Implementation Lessons Learned". In: *Proceedings of the 40th International Conference on Distributed Computing Systems*. IEEE, 2020.
- Suyash Gupta et al. "An In-Depth Look of BFT Consensus in Blockchain: Challenges and Opportunities". In: *Proceedings of the 20th International Middleware Conference Tutorials*. ACM, 2019, pp. 6–10. DOI: 10.1145/3366625.3369437.
- Suyash Gupta et al. "ResilientDB: Global Scale Resilient Blockchain Fabric". In: *Proceedings of the VLDB Endowment* 13.6 (2020), pp. 868–883. DOI: 10.14778/3380750.3380757.
- Suyash Gupta et al. "Tutorial: Blockchain Consensus Unraveled: Virtues and Limitations". In: *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. ACM, 2020. DOI: 10.1145/3401025.3404099.
- Jelle Hellings and Mohammad Sadoghi. "Brief Announcement: The Fault-Tolerant Cluster-Sending Problem". In: *33rd International Symposium on Distributed Computing (DISC 2019)*. Vol. 146. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 45:1–45:3. DOI: 10.4230/LIPIcs.DISC.2019.45.

References VII

Jelle Hellings and Mohammad Sadoghi. "Coordination-Free Byzantine Replication with Minimal Communication Costs". In: *23rd International Conference on Database Theory (ICDT 2020)*. Vol. 155. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 17:1–17:20. DOI: 10.4230/LIPIcs.ICDT.2020.17.

Maurice Herlihy. "Blockchains from a Distributed Computing Perspective". In: *Communications of the ACM* 62.2 (2019), pp. 78–85. DOI: 10.1145/3209623.

Matt Higginson et al. *The promise of blockchain*. Tech. rep. McKinsey&Company, 2017. URL: https://www.mckinsey.com/industries/financial-services/our-insights/the-promise-of-blockchain.

Muhammad El-Hindi et al. "BlockchainDB – Towards a Shared Database on Blockchains". In: *Proceedings of the 2019 International Conference on Management of Data*. Amsterdam, Netherlands: ACM, 2019, pp. 1905–1908. DOI: 10.1145/3299869.3320237.

Muhammad El-Hindi et al. "BlockchainDB: A Shared Database on Blockchains". In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1597–1609. DOI: 10.14778/3342263.3342636.

Dan Holtby, Bruce M. Kapron, and Valerie King. "Lower bound for scalable Byzantine Agreement". In: *Distributed Computing* 21.4 (2008), pp. 239–248. DOI: 10.1007/s00446-008-0069-x.

References VIII

- Maged N. Kamel Boulos, James T. Wilson, and Kevin A. Clauson. "Geospatial blockchain: promises, challenges, and scenarios in health and healthcare". In: *International Journal of Health Geographics* 17.1 (2018), pp. 1211–1220. DOI: 10.1186/s12942-018-0144-x.
- Rüdiger Kapitza et al. "CheapBFT: Resource-efficient Byzantine Fault Tolerance". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. ACM, 2012, pp. 295–308. DOI: 10.1145/2168836.2168866.
- Ramakrishna Kotla et al. "Zyzzyva: Speculative Byzantine Fault Tolerance". In: *ACM Transactions on Computer Systems* 27.4 (2009), 7:1–7:39. DOI: 10.1145/1658357.1658358.
- Leslie Lamport. "Paxos Made Simple". In: *ACM SIGACT News, Distributed Computing Column 5* 32.4 (2001), pp. 51–58. DOI: 10.1145/568425.568433.
- Leslie Lamport. "The implementation of reliable distributed multiprocess systems". In: *Computer Networks (1976)* 2.2 (1978), pp. 95–114. DOI: 10.1016/0376-5075(78)90045-4.
- Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176.

References IX

- Laphou Lao et al. "A Survey of IoT Applications in Blockchain Systems: Architecture, Consensus, and Traffic Modeling". In: *ACM Computing Surveys* 53.1 (2020). DOI: 10.1145/3372136.
- Jean-Philippe Martin and Lorenzo Alvisi. "Fast Byzantine Consensus". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 202–215. DOI: 10.1109/TDSC.2006.35.
- Shlomo Moran and Yaron Wolfstahl. "Extended impossibility results for asynchronous complete networks". In: *Information Processing Letters* 26.3 (1987), pp. 145–151. DOI: 10.1016/0020-0190(87)90052-4.
- Arvind Narayanan and Jeremy Clark. "Bitcoin's Academic Pedigree". In: *Communications of the ACM* 60.12 (2017), pp. 36–45. DOI: 10.1145/3132259.
- Senthil Nathan et al. "Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database". In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1539–1552. DOI: 10.14778/3342263.3342632.
- Faisal Nawab and Mohammad Sadoghi. "Blockplane: A Global-Scale Byzantizing Middleware". In: *35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 124–135. DOI: 10.1109/ICDE.2019.00020.

References X

Michael Okun. "On the round complexity of Byzantine agreement without initial set-up". In: *Information and Computation* 207.12 (2009), pp. 1351–1368. DOI: 10.1016/j.ic.2009.07.002.

M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 2020. DOI: 10.1007/978-3-030-26253-2.

M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults". In: *Journal of the ACM* 27.2 (1980), pp. 228–234. DOI: 10.1145/322186.322188.

Michael Pisa and Matt Juden. *Blockchain and Economic Development: Hype vs. Reality*. Tech. rep. Center for Global Development, 2017. URL: https://www.cgdev.org/publication/blockchain-and-economic-development-hype-vs-reality.

Dale Skeen. *A Quorum-Based Commit Protocol*. Tech. rep. Cornell University, 1982.

Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. 3th. Maarten van Steen, 2017. URL: https://www.distributed-systems.net/.

Gadi Taubenfeld and Shlomo Moran. "Possibility and impossibility results in a shared memory environment". In: *Acta Informatica* 33.1 (1996), pp. 1–20. DOI: 10.1007/s002360050034.

References XI

Gerard Tel. *Introduction to Distributed Algorithms*. 2nd. Cambridge University Press, 2001.

Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356. DOI: 10.1145/3293611.3331591.