An In-Depth Look of BFT Consensus in Blockchain: Challenges and Opportunities

Suyash Gupta *Jelle Hellings* Sajjad Rahnam Mohammad Sadoghi

Exploratory Systems Lab
Department of Computer Science
University of California, Davis
Davis, CA 95616-8562, USA

Goal: High-performance resilient data processing

Questions

1. Why?
2. What do we already have?
3. Where can we improve?
4. What new tools do we need?

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.

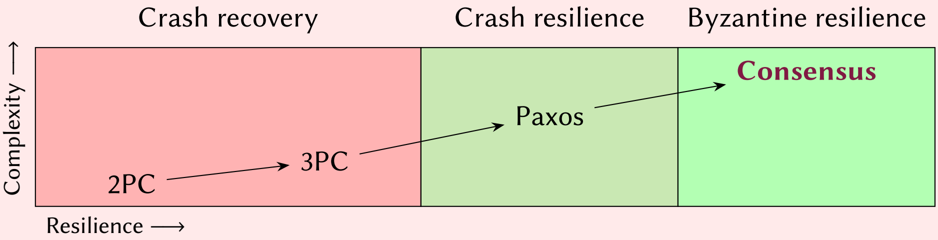
Many ideas also apply to permissionless blockchains.

Towards high-performance resilient data processing:

Why?

Why resilient data processing?

Go beyond assumptions of traditional transaction processing!

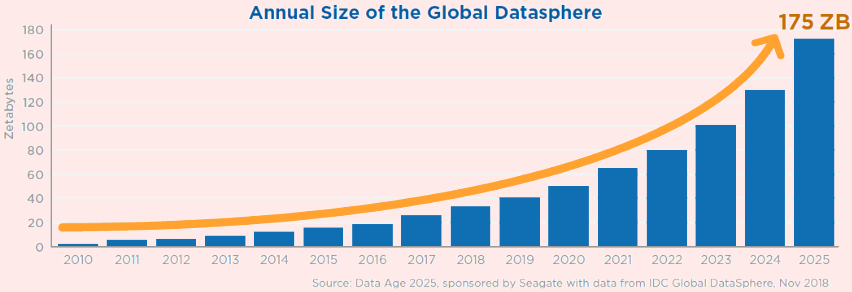


Example

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

Why high-performance?

Support requirements of future applications!

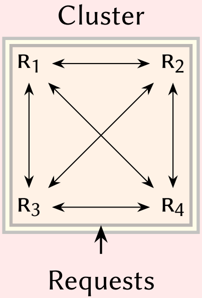


- Ever-growing volumes of data (e.g., sensor networks).
- Ever-growing demands of applications (e.g., machine learning).

Towards high-performance resilient data processing:

What do we already have?

Resilient data processing: Fully-replicated ledgers



- All participants (replicas) hold *all data*.
- All operations by *consensus*, e.g., via majority-vote.
- All operations executed in a unique ordering as specified by the *ledger* (journal).

We have consensus: PBFT, Paxos, PoW, ...

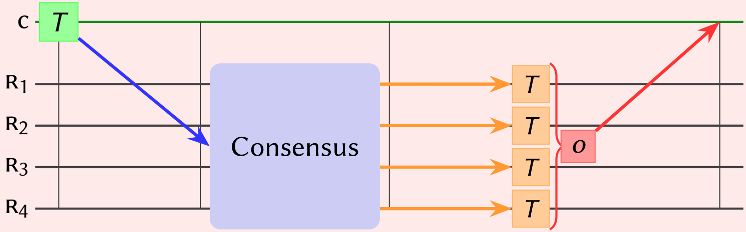
Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.

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.

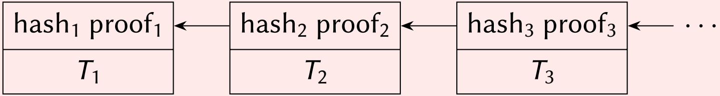
We have consensus: PBFT, Paxos, PoW, ...

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.



What else do we have?

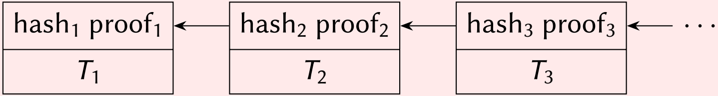
- A lot of *theory* on consensus: consensus is costly.
- Variations on consensus: Byzantine broadcasts, interactive consistency, …
- Tamper-proof *ledgers*.



  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.
- Variations on consensus: Byzantine broadcasts, interactive consistency, ...
- Tamper-proof *ledgers*.



Exact details: depend on consensus, application, attack model, ...

- Many *cryptographic tools*.

What about high-performance?

Towards high-performance resilient data processing:

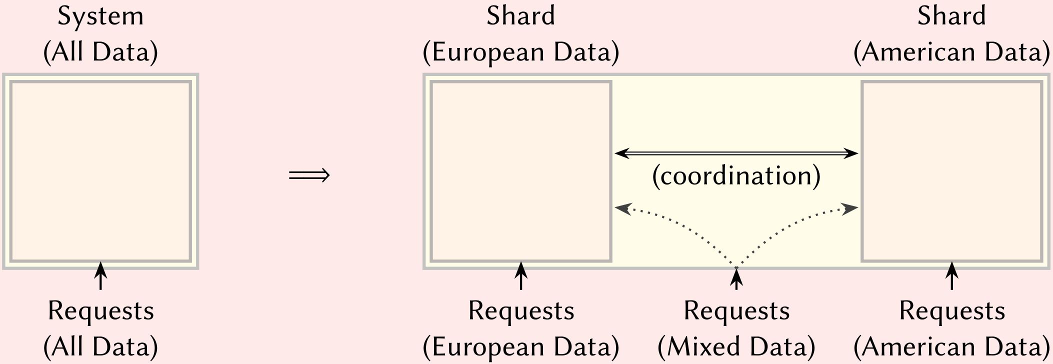
Where can we improve?

A look at high-performance data processing

Scalability: adding resources $\Longrightarrow$ adding performance.

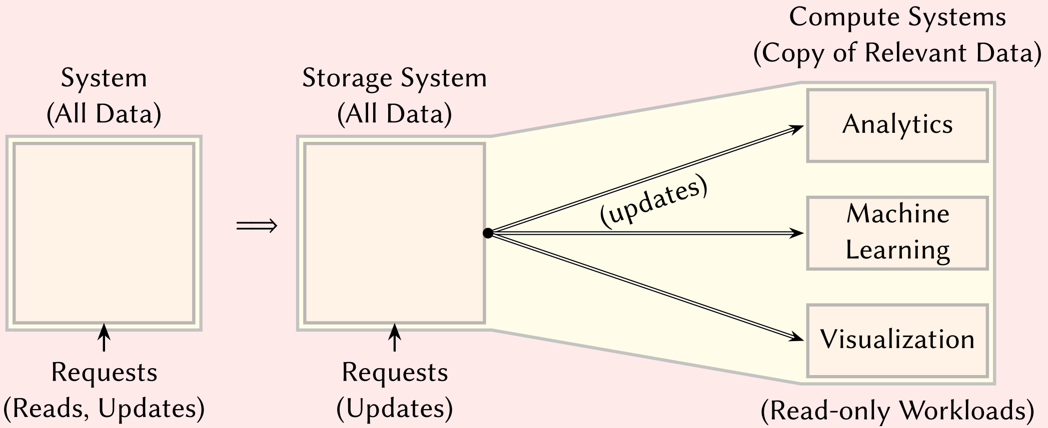
Full replication: adding resources (replicas) $\Longrightarrow$ less performance!

Sharding and Geo-scale aware sharding



Adding shards ⟹ adding throughput (parallel processing), adding storage.

Role Specialization: Read-only workloads



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

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!

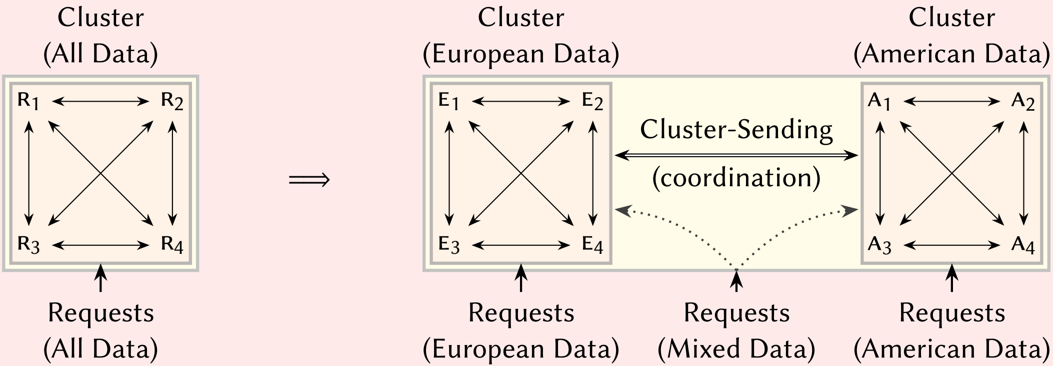
- **Role Specialization**: make the storage system a blockchain.
  Requires: *reliable read-only updates of the blockchain.*
  Permissionless blockchains: light clients!

Consensus is of no use here if we want efficiency.

Towards high-performance resilient data processing:

What new tools do we need?

Sharding: Reliable communication between blockchains



The Byzantine cluster-sending problem.

The Byzantine cluster-sending problem

The problem of sending a value v from a cluster C_1 to a cluster C_2 such that

- all non-faulty replicas in C_2 RECEIVE the value v;
- all non-faulty replicas in C_1 CONFIRM that the value v was received; and
- C_2 only receives a value v if all non-faulty replicas in 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

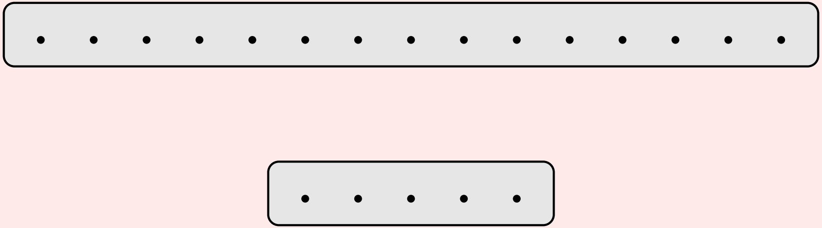
Lower bounds for cluster-sending: Example

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

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



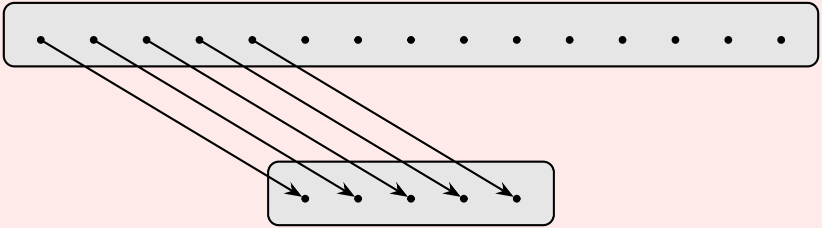
Lower bounds for cluster-sending: Example

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

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



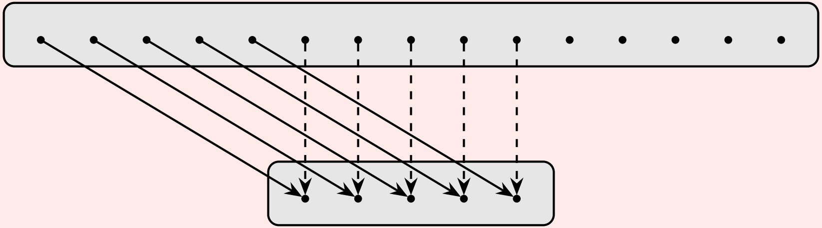
Lower bounds for cluster-sending: Example

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

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



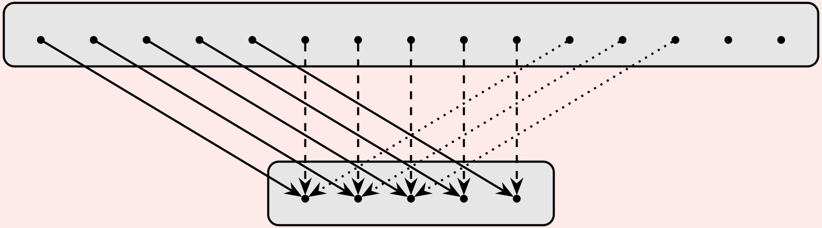
Lower bounds for cluster-sending: Example

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

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



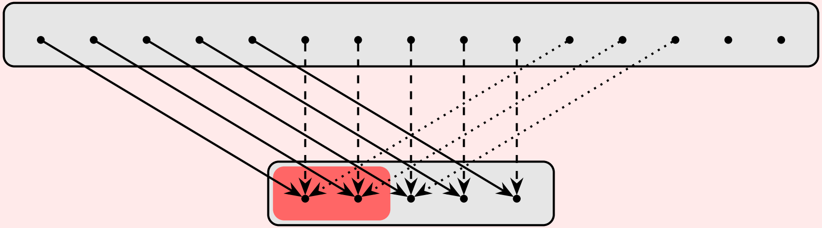
Lower bounds for cluster-sending: Example

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

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



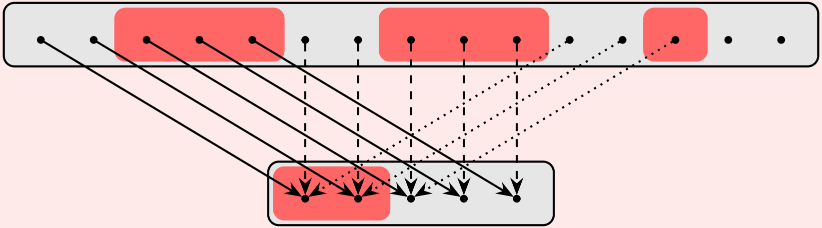
Lower bounds for cluster-sending: Example

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

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

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 $\mathcal{C}_1$, $\mathbf{n}_1 \geq \mathbf{n}_2$, $\mathbf{n}_1 \geq \sigma$:

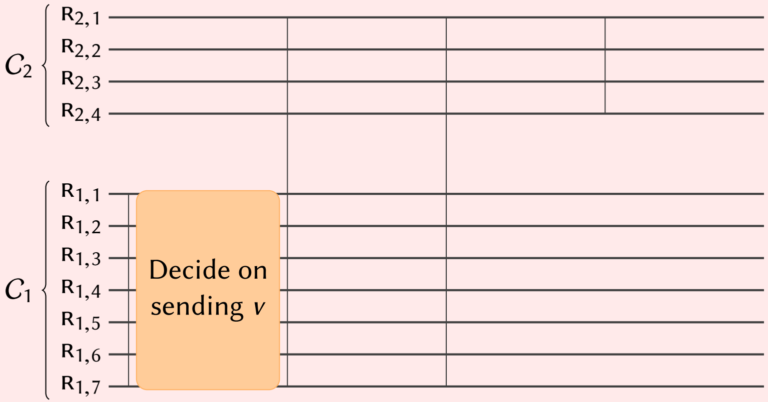
Agree on sending v to $\mathcal{C}_2$.
Choose replicas $\mathcal{P} \subseteq \mathcal{C}_1$ with $|\mathcal{P}| = \sigma$.
Choose a $\mathbf{n}_2$-partition partition($\mathcal{P}$) of $\mathcal{P}$.
for $P \in$ partition($\mathcal{P}$) **do**
    Choose replicas $Q \subseteq \mathcal{C}_2$ with $|Q|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

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

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

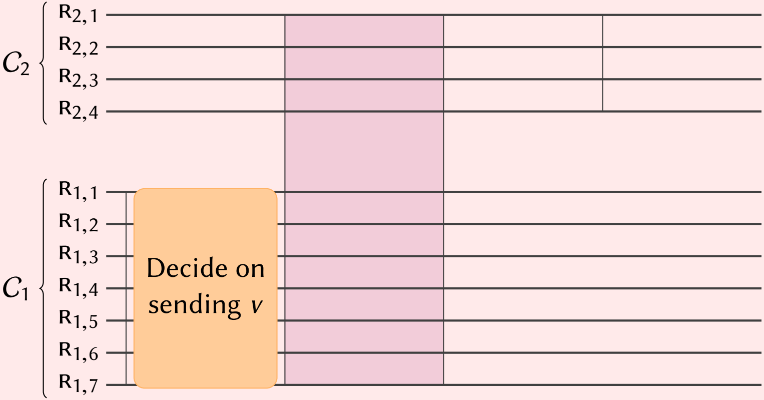
An optimal cluster-sending algorithm—visualized

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



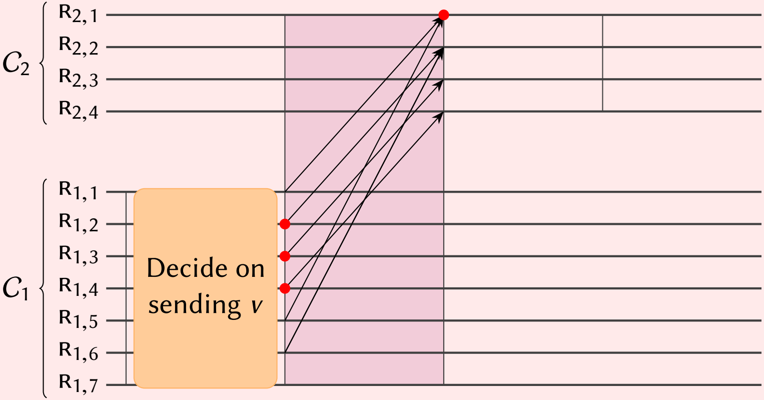
An optimal cluster-sending algorithm—visualized

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



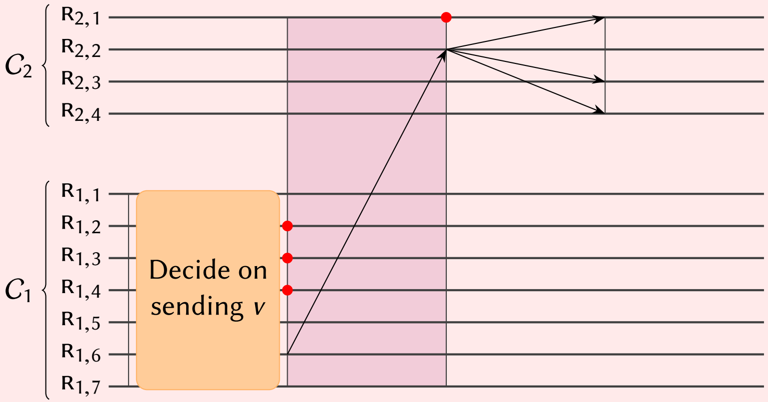
An optimal cluster-sending algorithm—visualized

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



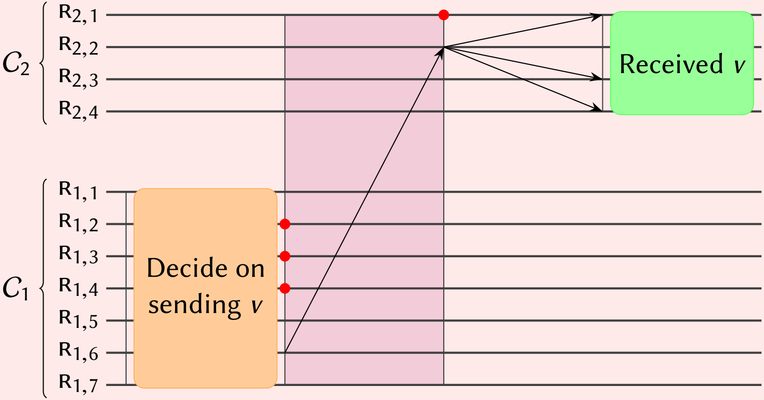
An optimal cluster-sending algorithm—visualized

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

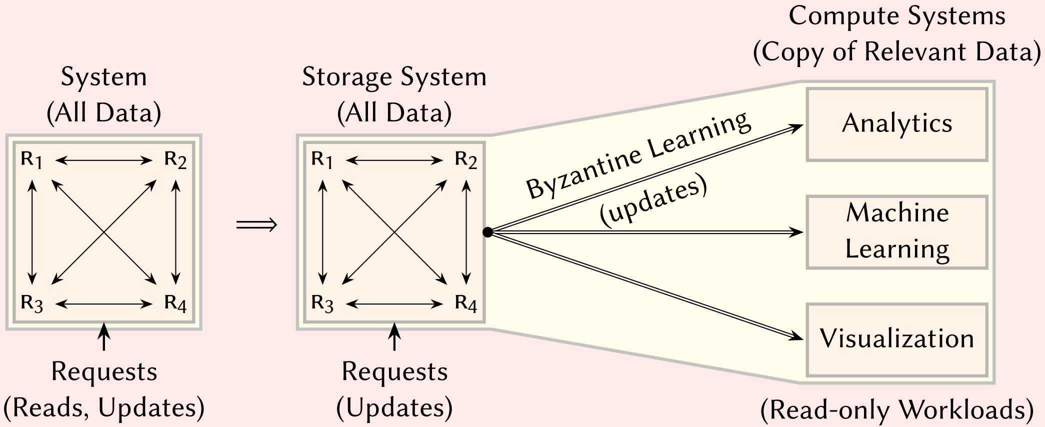
Pessimistic

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).

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

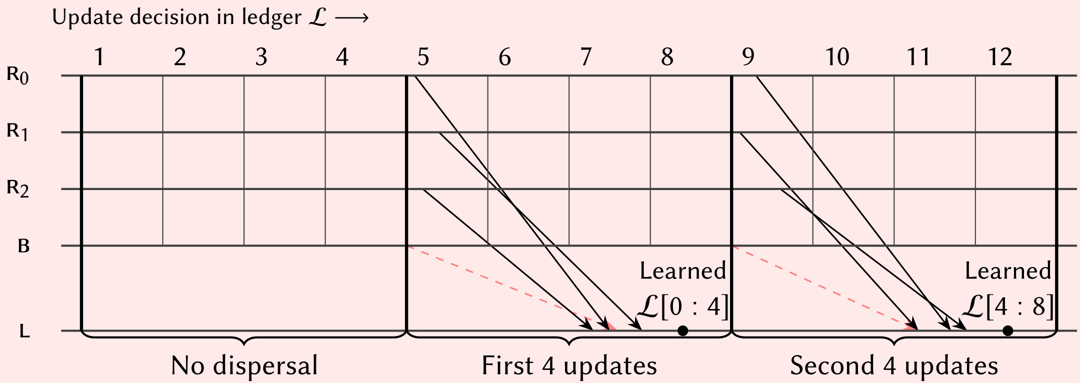
Idea: $\mathcal{C}$ sends a ledger $\mathcal{L}$ to learner L

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

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

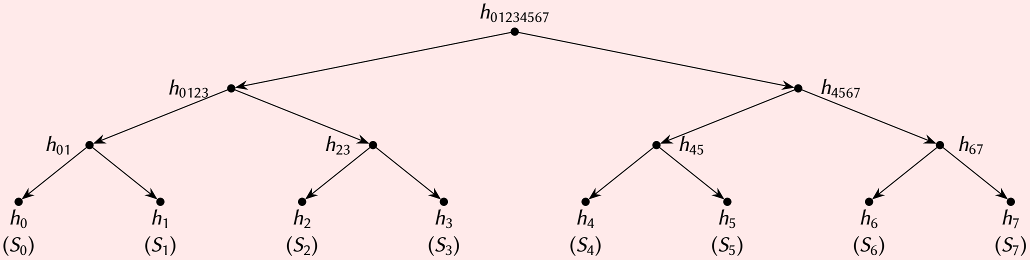
- Replica R_i sends at most $B = \left\lceil \frac{\|S\|}{\mathbf{n}-\mathbf{f}} \right\rceil + c \leq \frac{2\|S\|}{\mathbf{n}} + 1 + c = \mathcal{O}\left(\frac{\|S\|}{\mathbf{n}} + c\right)$ bytes.
- Learner L receives at most $\mathbf{n} \cdot B = \mathcal{O}(\|S\| + c\mathbf{n})$ bytes.

Communication by the delayed-replication algorithm



Checksums: Use Merkle-trees to construct checksums

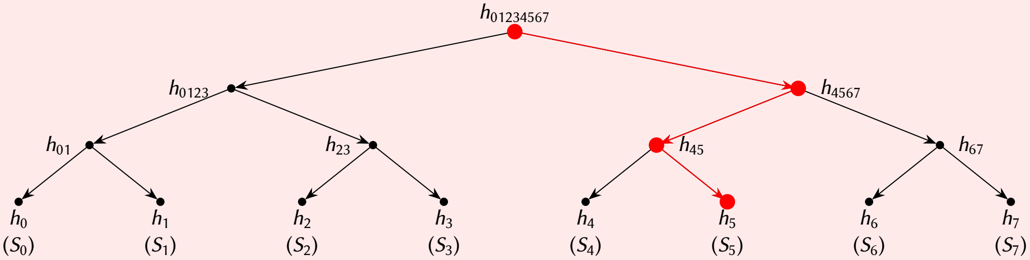
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by R_5).



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

Checksums: Use Merkle-trees to construct checksums

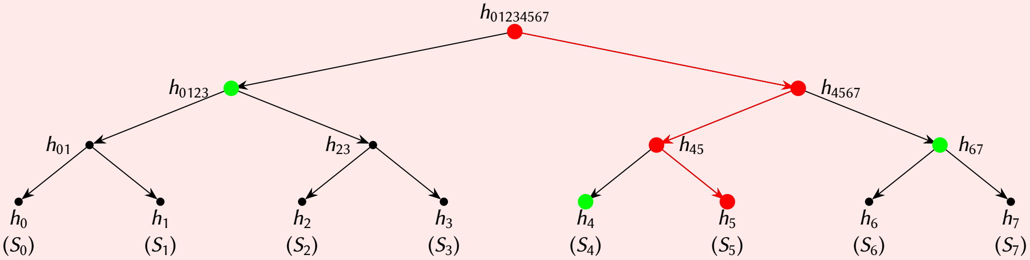
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by R_5).



Determine the path from root to S_5.

Checksums: Use Merkle-trees to construct checksums

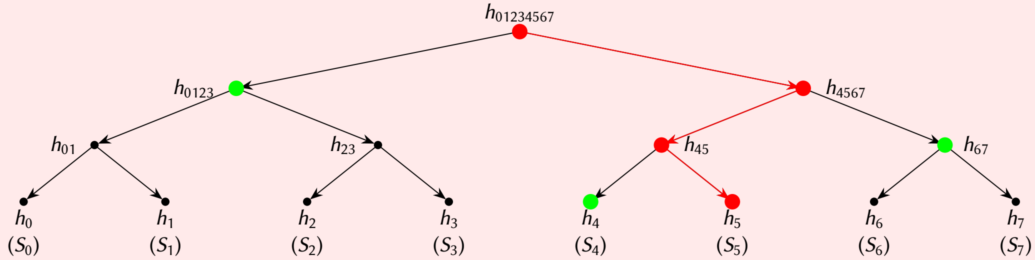
Consider 8 replicas and a sequence S.
We construct the checksum $C_5(S)$ of S (used by $\textsc{r}_5$).



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 $\textsc{r}_5$).



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/`.