Building High Throughput Permissioned Blockchain Fabrics: Challenges and Opportunities

Suyash Gupta
Jelle Hellings
Sajjad Rahnama
Mohammad Sadoghi
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).

Annual Size of the Global Datasphere

Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018
Towards high-performance resilient data processing:

What is the relation with blockchains?
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.

\[
\text{hash}_1 \text{puzzle}_1, \ldots, \text{hash}_{100} \text{puzzle}_{100}, \ldots, \text{hash}_{200} \text{puzzle}_{200}, \ldots, \text{hash}_{400} \text{puzzle}_{400}
\]

Many participants hold a copy of the blockchain.

Blockchain structure is tamper-proof by design.
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.

<table>
<thead>
<tr>
<th>Block $B_1$</th>
<th>Block $B_2$</th>
<th>Block $B_3$</th>
<th>Block $B_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>$\text{hash}_v$ puzzle$_1$</td>
<td>$\text{hash}_1$ puzzle$_2$</td>
<td>$\text{hash}_2$ puzzle$_3$</td>
</tr>
<tr>
<td>$T_1, \ldots, T_{100}$</td>
<td>$T_{101}, \ldots, T_{200}$</td>
<td>$T_{201}, \ldots, T_{300}$</td>
<td>$T_{301}, \ldots, T_{400}$</td>
</tr>
</tbody>
</table>

Longest chain has highest incentives.

Making blocks (solving puzzles) is very costly.

Malicious attempt leads to a dead end.
What is a blockchain? - Malicious behavior

Bitcoin: Preventing malicious behavior

- Malicious attempts to change a chain.

<table>
<thead>
<tr>
<th>Block B₁</th>
<th>Block B₂</th>
<th>Block B₃</th>
<th>Block B₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashᵥ puzzle₁</td>
<td>hash₁ puzzle₂</td>
<td>hash₂ puzzle₃</td>
<td>hash₃ puzzle₄</td>
</tr>
<tr>
<td>ℑ , ..., T₁₀₀</td>
<td>ℑ₁₀₁ , ..., T₂₀₀</td>
<td>ℑ₂₀₁ , ..., T₃₀₀</td>
<td>ℑ₃₀₁ , ..., T₄₀₀</td>
</tr>
</tbody>
</table>

- 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.
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 $\rho$-th client request $T$.
2. Append $T$ as the $\rho$-th entry to the ledger.
3. Execute $T$ as the $\rho$-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 \( T_j \).

\[
\begin{align*}
\text{Interactive} & \rightarrow \text{consistency} \\
R_1 & \rightarrow T_1 \\
R_2 & \rightarrow T_2 \\
R_3 & \rightarrow T_3 \\
R_4 & \rightarrow T_4 \\
\end{align*}
\]

(As \( R_3 \) is faulty: \( \times \) 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.

⇒ Consistency not guaranteed (e.g., forks).
The CAP Theorem and Blockchains

Permissioned Blockchains
Consistency at all costs.
⇒ Availability when communication is reliable.
⇒ Some network failure when replicas are reliable.
What else do we have?

- A lot of \textit{theory} on consensus: consensus is costly.
- \textbf{PBFT}: A practical Byzantine fault-tolerant consensus protocol.
- Tamper-proof \textit{ledgers}.

\begin{align*}
\text{\begin{tabular}{|c|c|c|c|}
\hline
\text{hash}_v \text{ puzzle}_1 & \text{hash}_1 \text{ puzzle}_2 & \text{hash}_2 \text{ puzzle}_3 & \text{hash}_3 \text{ puzzle}_4 \\
\hline
T_1, \ldots, T_{100} & T_{101}, \ldots, T_{200} & T_{201}, \ldots, T_{300} & T_{301}, \ldots, T_{400} \\
\hline
\end{tabular}}
\end{align*}

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

- Many \textit{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*.

\[
\begin{array}{cccc}
\text{hash}_v \text{ puzzle}_1 & \text{hash}_1 \text{ puzzle}_2 & \text{hash}_2 \text{ puzzle}_3 & \text{hash}_3 \text{ puzzle}_4 \\
T_1, \ldots, T_{100} & T_{101}, \ldots, T_{200} & T_{201}, \ldots, T_{300} & T_{301}, \ldots, T_{400}
\end{array}
\]

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 $f$ malicious failures requires $n > 3f$ replicas.
- Worst-case: at least $\Omega (f + 1)$ phases of communication.
- Worst-case: at least $\Omega (nf)$ signatures and $\Omega (n + f^2)$ messages.
- Network must stay connected when removing $2f$ replicas.

Consensus in practice

Asynchronous communication, $n > 3f$, clique network:

$\Rightarrow$ 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 $\nu$

If receive $\text{PREPREPARE}$ message $m$: $\text{PREPARE}(m)$. 
PBFT: Normal-case protocol in view $\nu$

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

**PBFT: Normal-case protocol in view $\nu$**
**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 $\rho$.*
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, \( n = 4, f = 1, n - f = 3 \))

What to do?
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, \( n = 4, f = 1, n - f = 3 \))
PBFT: A normal-case property when \( n > 3f \)

Theorem (Castro et al.)

If replicas \( R_i, i \in \{1, 2\} \), commit to \( m_i = \text{PREPARE}(\langle T_i \rangle_{c_i}, v, \rho) \),
then \( \langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2} \).
PBFT: A normal-case property when $n > 3f$

**Theorem (Castro et al.)**

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

**Proof.**

Replica $r_i$ commits to $m_i$:

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

Proof diagram:
PBFT: A normal-case property when $n > 3f$

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 $m_i$:

- $n - f$ messages $\text{PREPARE}(m_i)$
- $\geq n - 2f$ non-faulty
- $\leq f$ faulty

$$2(n - f) \leq n - f \iff 2n - 4f \leq n - f \iff n \leq 3f.$$
PBFT: A normal-case property when \( n > 3f \)

Theorem (Castro et al.)

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

Proof.

Replica \( r_i \) commits to \( m_i \):

\[
\text{n} - \text{f} \text{ messages PREPARE}(m_i) \\
\geq \text{n} - 2\text{f} \text{ non-faulty} \\
\leq \text{f} \text{ faulty}
\]

If \( \langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2} \), then \( B_1 \cap B_2 = \emptyset \) and \( |B_1 \cup B_2| \geq 2(n - 2f) \).
PBFT: A normal-case property when \( n > 3f \)

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 \( m_i \):

\[
\begin{align*}
\text{\( n - f \) messages} & \quad \text{PREPrepare}(m_i) \\
\geq \text{\( n - 2f \) non-faulty} & \quad B_i \\
\leq \text{\( f \) faulty} & \quad F_i \\
\end{align*}
\]

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

\[
2(n - 2f) \leq n - f
\]
**PBFT: A normal-case property when \( n > 3f \)**

**Theorem (Castro et al.)**

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

**Proof.**

Replica \( r_i \) commits to \( m_i: \)

\[
\begin{align*}
n - f \text{ messages } & \text{PREPARE}(m_i) \\
\geq n - 2f \text{ non-faulty} & \rightarrow B_i \\
\leq f \text{ faulty} & \rightarrow F_i \\
\end{align*}
\]

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

\[
2(n - 2f) \leq n - f \quad \text{iff} \quad 2n - 4f \leq n - f
\]
**PBFT: A normal-case property when** \( n > 3f \)

**Theorem (Castro et al.)**

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

**Proof.**

Replica \( r_i \) commits to \( m_i \):

\[
\begin{align*}
\text{n} - f \text{ messages } & \text{PREPARE} (m_i) \\
\geq n - 2f \text{ non-faulty} & \to B_i \\
\leq f \text{ faulty} & \to F_i
\end{align*}
\]

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

\[
2(n - 2f) \leq n - f \quad \text{iff} \quad 2n - 4f \leq n - f \quad \text{iff} \quad n \leq 3f.
\]

\( \square \)
PBFT: Primary failure versus malicious replicas

Primary $p$ is faulty

*ignores* $r_3$
PBFT: Primary failure versus malicious replicas

Primary \( p \) is faulty

*ignores* \( r_3 \)
PBFT: Primary failure versus malicious replicas

Primary \( p \) is faulty

ignores \( r_3 \)

Replica \( r_3 \) 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 \( > f \) non-faulty replicas, then failure of the primary is detectable.

Replacing the primary: View-change at replica \( R \)

1. \( R \) detects \textit{failure} of the current primary \( P \).
2. \( R \) chooses a new primary \( P' \) (the next replica).
3. \( R \) provides \( P' \) with its \textit{current state}.
4. \( P' \) proposes a \textit{new view}.
5. If the new view is valid, then \( R \) switches to this view.
Send $\text{ViewChange}(E, \nu)$ 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 $n - f$ valid $\text{ViewChange}(E, v)$ messages: $\text{NewView}(v + 1, \mathcal{E}, N)$.

- $\mathcal{E}$ contains $n - f$ valid $\text{ViewChange}$ messages.
- $\mathcal{N}$ contains no-op proposals for missing rounds.
PBFT: A view-change in view $v$

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

- $\mathcal{E}$ contains $n - f$ valid ViewChange messages.
- $\mathcal{N}$ contains no-op proposals for missing rounds.
PBFT: A property of view-changes when $n > 3f$

Theorem (Castro et al.)

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

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

Proof.
The $\text{VIEWCHANGE}$ messages in $E$:

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

$\geq n - 2f$ non-faulty $B$

$\leq f$ faulty $F$
PBFT: A property of view-changes when $n > 3f$

Theorem (Castro et al.)

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

Proof.
The $\text{VIEWCHANGE}$ messages in $\mathcal{E}$:

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

$\geq n - 2f$ non-faulty $\quad B$

$\leq f$ faulty $\quad F$

if $S \cap B = \emptyset$, then $|S \cup B| \geq 2(n - 2f)$, a contradiction! \qed
PBFT: Further dealing with failures

1. *Undetected failures*: e.g., ignored replicas.
   At least $n - 2f > f$ non-faulty replicas participate: *checkpoints.*
PBFT: Further dealing with failures

1. **Undetected failures**: e.g., ignored replicas.
   
   At least \( n - 2f > f \) non-faulty replicas participate: checkpoints.

2. **Detected failures**: primary replacement.
   
   Worst-case: a sequence of \( f \) view-changes (\( \Omega(f) \) phases).
PBFT: Further dealing with failures

1. **Undetected failures**: e.g., ignored replicas.
   At least \( n - 2f > f \) non-faulty replicas participate: checkpoints.

2. **Detected failures**: primary replacement.
   Worst-case: a sequence of \( f \) view-changes (\( \Omega(f) \) 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 \( n - 2f > f \) non-faulty replicas participate: *checkpoints*.

   Worst-case: a sequence of \( f \) view-changes (\( \Omega (f) \) phases).

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

   Worst-case: replacements until communication becomes *reliable*. 
PBFT: Modeling real-world performance

(Maximum throughput of any primary-backup broadcast protocol)

1Bandwidth: 100 MiB/s, PrePrepare message size: 1024 B, Prepare and Commit message size: 256 B.
PBFT: Modeling real-world performance

(Number of Replicas)

(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**

(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**

(Maximum throughput of out-of-order PBFT)

1 Bandwidth: 100 MiB/s, PrePrepare message size: 1024 B, Prepare and Commit message size: 256 B.

---

**Throughput (txn/s, 15 ms delay)**

- $T_{\text{max}}$
- $T_{\text{PBFT}}$
- $T_{\text{PBFT-256}}$
- $T_{\text{ooo-PBFT}}$

---

**Throughput (txn/s, 7 replicas)**

- Maximum throughput of out-of-order PBFT

---

**Message Delay (s)**

- Bandwidth: 100 MiB/s, PrePrepare message size: 1024 B, Prepare and Commit message size: 256 B.
PBFT: Modeling real-world performance

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

1Bandwidth: 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 $\implies$ adding performance.*

Full replication: adding resources (replicas) $\implies$ less performance!
Sharding and Geo-scale aware sharding

System (All Data) ⇒ Shard (European Data) & Shard (American Data)

Adding shards ⇒ adding throughput (parallel processing), adding storage.
Role Specialization: Read-only workloads

Specializing roles $\implies$ 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 $C_1$ to a cluster $C_2$ such that

- all non-faulty replicas in $C_2$ \textit{receive} the value $v$;
- all non-faulty replicas in $C_1$ \textit{confirm} that the value $v$ was received; and
- $C_2$ only receives a value $v$ if all non-faulty replicas in $C_1$ \textit{agree} upon sending $v$.

\textit{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 $(f_1 + 1)(f_2 + 1) \approx f_1 \times f_2$ messages.
Global communication versus local communication

Straightforward cluster-sending solution (crash failures)

Pair-wise broadcasting with \((f_1 + 1)(f_2 + 1) \approx f_1 \times f_2\) messages.

<table>
<thead>
<tr>
<th></th>
<th>Ping round-trip times (ms)</th>
<th>Bandwidth (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OR</td>
<td>IA</td>
</tr>
<tr>
<td>Oregon</td>
<td>≤ 1</td>
<td>38</td>
</tr>
<tr>
<td>Iowa</td>
<td>≤ 1</td>
<td>33</td>
</tr>
<tr>
<td>Montreal</td>
<td>≤ 1</td>
<td>82</td>
</tr>
<tr>
<td>Belgium</td>
<td>≤ 1</td>
<td>252</td>
</tr>
<tr>
<td>Taiwan</td>
<td>≤ 1</td>
<td>137</td>
</tr>
<tr>
<td>Sydney</td>
<td>≤ 1</td>
<td></td>
</tr>
</tbody>
</table>
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

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 $\max(n_1, n_2)$ messages to do cluster-sending.

Theorem (Cluster-sending lower bound, crash failures)
Assume $n_1 \geq n_2$ and let

$$q = (f_1 + 1) \div nf_2; \quad r = (f_1 + 1) \mod nf_2.$$  

We need to exchange at least $qn_2 + r + f_2 sgn r \approx n_1$ messages to do cluster-sending.
An optimal cluster-sending algorithm (crash failures)

**Protocol for the sending cluster $C_1$, $n_1 \geq n_2$, $n_1 \geq \sigma$:**

1: \textit{AGREE} on sending $v$ to $C_2$.
2: Choose replicas $\mathcal{P} \subseteq C_1$ with $|\mathcal{P}| = \sigma$.
3: Choose a $n_2$-partition $\text{partition}(\mathcal{P})$ of $\mathcal{P}$.
4: \textbf{for} $P \in \text{partition}(\mathcal{P})$ \textbf{do}
5: 
6: Choose replicas $Q \subseteq C_2$ with $|Q| = |P|$. 
7: Choose a bijection $b : P \rightarrow Q$.
8: \textbf{for} $R_1 \in P$ \textbf{do}
9: 
10: Send $v$ from $R_1$ to $b(R_1)$.

**Protocol for the receiving cluster $C_2$:**

9: \textbf{event} $R_2 \in C_2$ receives $w$ from a replica in $C_1$ \textbf{do}
10: Broadcast $w$ to all replicas in $C_2$.
11: \textbf{event} $R_2 \in C_2$ receives $w$ from a replica in $C_2$ \textbf{do}
12: $R_2$ considers $w$ \textit{RECEIVED}.
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = 7$, $n_2 = 4$, $f_1 = 3$, $f_2 = 1$, $\sigma = 6$

\[
\begin{align*}
C_2 & \quad \{ R_{2,1}, R_{2,2}, R_{2,3}, R_{2,4} \} \\
C_1 & \quad \{ R_{1,1}, R_{1,2}, R_{1,3}, R_{1,4}, R_{1,5}, R_{1,6}, R_{1,7} \}
\end{align*}
\]
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = 7$, $n_2 = 4$, $f_1 = 3$, $f_2 = 1$, $\sigma = 6$
An optimal cluster-sending algorithm—visualized

Crash failures, \( n_1 = 7, n_2 = 4, f_1 = 3, f_2 = 1, \sigma = 6 \)

Decide on sending \( v \)
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = 7$, $n_2 = 4$, $f_1 = 3$, $f_2 = 1$, $\sigma = 6$

Decide on sending $v$
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = 7, n_2 = 4, f_1 = 3, f_2 = 1, \sigma = 6$
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 $n$ pieces $v'$ such that $v$ can be decoded from every set of $n - f$ such pieces.

Theorem (Rabin 1989)
The IDA algorithm is an optimal information dispersal algorithm:
- Each piece $v'$ has size $\left\lceil \frac{\|v\|}{n-f} \right\rceil$.
- The $n - f$ pieces necessary for decoding have a total size of $(n - f) \left\lceil \frac{\|v\|}{(n-f)} \right\rceil \approx \|v\|$.
The delayed-replication algorithm

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

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

Observation (\( n > 2f \))

- Replica \( r_i \) sends at most \( B = \left\lfloor \frac{\|S\|}{n-f} \right\rfloor + c \leq \frac{2\|S\|}{n} + 1 + c = O\left( \frac{\|S\|}{n} + c \right) \) bytes.
- Learner \( \mathcal{L} \) receives at most \( n \cdot B = O\left( \|S\| + cn \right) \) bytes.
Communication by the delayed-replication algorithm

Update decision in ledger $\mathcal{L}$ →

No dispersal  First 4 updates  Second 4 updates

Learned $\mathcal{L}[0:4]$  Learned $\mathcal{L}[4:8]$
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 $R_5$).

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

![Merkle-tree diagram](image)
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.
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 $O(\|\mathcal{L}\| + |\mathcal{L}| \log n)$;
3. $L$ will only need at most $|\mathcal{L}|/n$ decode steps;
4. $R$ will send at most $|\mathcal{L}|/n$ messages to $L$ of size $O\left(\frac{\|\mathcal{L}\|+|\mathcal{L}| \log n}{n}\right)$.

Adding replicas to cluster $C \implies$ less communication per replica!
Application: Scalable storage for resilient systems

- Clusters typically need a view $V$ on the data to decide whether updates are valid.
- Clusters only need the full ledger $L$ 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

$$O(\|L\| + \|V\|)$$

to

$$O\left(\frac{\|L\|}{n - f} + \frac{|L|}{n} \log(n) + \|V\|\right).$$
Towards high-performance resilient data processing:

Concluding remarks
Conclusion

We need an extensive toolbox!

- Consensus (permissioned) (permissionless)
  - 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


More about us and our work

▶ Jelle Hellings  https://jhellings.nl/.
▶ ExpoLab  https://expolab.org/.
▶ ResilientDB  https://resilientdb.com/.
References I


References IV


References V


References VII


References IX


References


References XI
