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!

- Crash recovery
- Crash resilience
- Byzantine resilience

Resilience $\rightarrow$ Complexity $\rightarrow$

- 2PC
- 3PC
- Paxos
- Consensus

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 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*.

```
| hash₁ proof₁ | hash₂ proof₂ | hash₃ proof₃ | ⋮ |
| T₁           | T₂           | T₃           |   |
```

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*.

```
\begin{array}{|c|}
\hline
\text{hash}_1 \text{ proof}_1 \quad \text{hash}_2 \text{ proof}_2 \quad \text{hash}_3 \text{ proof}_3 \\
\hline
T_1 \quad T_2 \quad T_3 \\
\hline
\end{array}
```

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 $\implies$ adding performance.*

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

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

Specializing roles $\implies$ adding throughput (parallel processing, specialized hardware, …).

System (All Data) $\implies$ Storage System (All Data) $\implies$ Compute Systems (Copy of Relevant Data)

- Analytics
- Machine Learning
- Visualization

(Requests (Reads, Updates) $\implies$ Requests (Updates) $\implies$ (Read-only Workloads))
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

Cluster (All Data)

$\text{Cluster} \ (\text{All Data})$

\begin{align*}
R_1 & \leftrightarrow R_2 \\
R_3 & \leftrightarrow R_4
\end{align*}

Requests (All Data)

\[ \Rightarrow \]

Cluster (European Data)

\begin{align*}
E_1 & \leftrightarrow E_2 \\
E_3 & \leftrightarrow E_4
\end{align*}

Requests (European Data)

Cluster-Sending (coordination)

Requests (Mixed Data)

Cluster (American Data)

\begin{align*}
A_1 & \leftrightarrow A_2 \\
A_3 & \leftrightarrow A_4
\end{align*}

Requests (American Data)

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

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]
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 $qn_2 + r + f_2 \text{sgn } r \approx n_1$ 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: **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: **for** $P \in \text{partition}(\mathcal{P})$ **do**
5: Choose replicas $Q \subseteq C_2$ with $|Q| = |P|$.
6: Choose a bijection $b : P \rightarrow Q$.
7: **for** $r_1 \in P$ **do**
8: Send $v$ from $r_1$ to $b(r_1)$.

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

9: **event** $r_2 \in C_2$ receives $w$ from a replica in $C_1$ **do**
10: Broadcast $w$ to all replicas in $C_2$.
11: **event** $r_2 \in C_2$ receives $w$ from a replica in $C_2$ **do**
12: $r_2$ considers $w$ **RECEIVED**.
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, \( n_1 = 7, n_2 = 4, f_1 = 3, f_2 = 1, \sigma = 6 \)

\[
\begin{align*}
C_1 & \quad \{ R_{1,1}, R_{1,2}, R_{1,3}, R_{1,4}, R_{1,5}, R_{1,6}, R_{1,7} \} \\
C_2 & \quad \{ R_{2,1}, R_{2,2}, R_{2,3}, R_{2,4} \}
\end{align*}
\]

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

Crash failures, $n_1 = 7$, $n_2 = 4$, $f_1 = 3$, $f_2 = 1$, $\sigma = 6$
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).
Role Specialization: Reliable read-only updates of the blockchain

System (All Data)

Requests (Reads, Updates)

\[ \begin{array}{cccc} 
R_1 & 
\rightarrow & 
R_2 \\
\downarrow & & \downarrow \\
R_3 & \leftarrow & R_4 
\end{array} \]

Storage System (All Data)

Requests (Updates)

\[ \begin{array}{cccc} 
R_1 & 
\rightarrow & 
R_2 \\
\downarrow & & \downarrow \\
R_3 & \leftarrow & R_4 
\end{array} \]

Compute Systems (Copy of Relevant Data)

Analytics

Machine Learning

Visualization

(Read-only Workloads)

Byzantine Learning (updates)

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 \textit{receive all} transactions in $\mathcal{L}$; and
- the learner $l$ will \textit{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 $L$ to learner $L$

1. Partition the ledger $L$ 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 $L$.
4. Learner $L$ receives at least $n - f$ distinct and valid pieces and decodes $S$.

Observation ($n > 2f$)

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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: $C_5(S) = [h_4, h_{67}, h_{0123}, h_{01234567}]$. 
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 \((n > 2f)\)

**Theorem**

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

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

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

Concluding remarks
Conclusion

We need an extensive toolbox!

- Consensus (permissioned) (permissionless)
  PBF, 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/.