Replay Attacks and Defenses Against Cross-shard Consensus in Sharded Distributed Ledgers

Alberto Sonnino
Acknowledgments

Authors

Alberto Sonnino  Bano Shehar  Mustafa Al-Bassam  George Danezis

University College London
A set of nodes
Byzantine Fault Tolerance

> $2/3$
Consensus
State Sharding
State Sharding
An example transaction

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
State Sharding

An example transaction

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
State Sharding
Only two acceptable final states
Cross-Shard Consensus
How do shards communicate with each other?
Chainspace: A Sharded Smart Contracts Platform

Mustafa Al-Bassam

Section I: Introduction

Chainspace is a sharded infrastructure that provides a smart contracts platform on top of a permissioned blockchain. It enables the execution of smart contracts on a decentralized network of validators, each managing a portion of the system's state. This approach allows for horizontal scalability and improved transaction processing capacity compared to single-shard or permissionless blockchains.

Section II: Background

Permissioned blockchains, such as Hyperledger Fabric or Corda, support smart contracts but are limited by their single-shard architecture and the need for full nodes to verify transactions. Chainspace overcomes these limitations by using a sharded design with permissioned validators.

Section III: Client-Facing Application Interface

The client-facing interface of Chainspace is designed to be user-friendly while ensuring secure and efficient transaction processing. It integrates a smart contract value system named CSCoin, which is used for transaction fees and execution costs.

Section IV: Internal Data Structures

Chainspace employs a stateless sharding approach to manage the state of the smart contracts. This design choice allows for scalability and decreases the load on the validators.

Section V: Security

Chainspace is designed to be secure against subsets of nodes trying to compromise its integrity or availability. It supports various security features, including certificate verification and extensibility through modern protocols to enhance privacy.

Section VI: Smart Contract Benchmarks

The performance of smart contracts in Chainspace is evaluated by measuring their execution time and transaction throughput. Benchmarks are conducted using various load scenarios to assess the system's scalability.

Section VII: Evaluation

The core protocols of Chainspace are evaluated to ensure their correctness and performance. An evaluation of the byzantine distributed commit protocol and its performance is presented, along with benchmark results.

Section VIII: Limitations

Various limitations of Chainspace are described, including challenges in implementing a secure sharding mechanism and the need for further optimization of consensus protocols.

Conclusion

Chainspace presents a promising approach to scaling smart contracts on a decentralized network. With its unique sharding design and permissioned network, it offers a viable solution for managing smart contracts at a large scale.

This paper makes the following contributions:

- A sharded smart contracts platform that scales arbitrarily.
- A system design that supports privacy-friendly smart contracts.
- An evaluation of the core protocols and smart contracts performance.

Outline:

- Introduction
- Background
- Client-Facing Application Interface
- Internal Data Structures
- Security
- Smart Contract Benchmarks
- Evaluation
- Limitations
- Conclusion

Chainspace offers an innovative solution for managing decentralized ledgers, providing a secure and scalable platform for executing smart contracts.
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

client

shard 1

shard 2

shard 3
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

delete \( X_1, X_2 \); create \( Y_1, Y_2 \)
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Insecure under parallel composition
Attacks

Double spend any object

• Does not need to collude with any node
• Acts as client or passive observer
• Re-orders network messages (not always needed)
Attack against S-BAC

Double-spend $X_1$

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$
Attack against S-BAC

Double-spend $X_1$

$$T'(\overline{x_1, x_2}) \rightarrow (y_1, y_2, y_3)$$
Attack against S-BAC

Double-spend $X_1$

\[ T'(\overline{x_1}, x_2) \rightarrow (y_1, y_2, y_3) \]
Attack against S-BAC
Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$

$\text{lock } X_2$
Attack against S-BAC

Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$

$c$  

$s_1$  

$s_2$  

$s_3$  

$\text{lock } X_2$

pre-accept($T'$)  

pre-abort($T'$)  

$T'$ is a transaction that maps $\tilde{x}_1$ and $x_2$ to $y_1$, $y_2$, and $y_3$. The diagram illustrates the steps involved in the attack and the actions taken by the parties involved.
Attack against S-BAC
Double-spend $X_1$

\[ T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3) \]

- $c$ → pre-abort($T'$)
- $s_1$ → BFT
- $s_2$ → BFT
- $s_3$ → pre-accept($T'$)

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

- $c$ → pre-accept($T$)
- $s_1$ → BFT
- $s_2$ → BFT
- $s_3$ → BFT

lock $X_2$
Attack against S-BAC
Double-spend X₁

T′(x₁, x₂) → (y₁, y₂, y₃)

pre-abort(T')

pre-accept(T)

from shard 1

lock X₂

T(x₁, x₂) → (y₁, y₂, y₃)

pre-accept(T)

abort(T)

pre-abort(T)
Attack against S-BAC

Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$

lock $X_2$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

unlock $X_2$
Attack against S-BAC

Double-spend $X_1$

$$T^*(x_1) \rightarrow (y_*)$$

client

BFT

shard 1
Attack against S-BAC

Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC

Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC
Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC
Double-spend $X_1$

Before attack

- $X_1$: 10
- $X_2$: 5

After attack

- $Y^*$: 10
- $Y_2$: 4
- $Y_3$: 10
