Fundamental Properties of Cryptocurrency in Distributed Systems

Lewis Tseng
Boston College
1 Bitcoin equals 32,650.00 United States Dollar

Jan 21, 7:45 PM UTC · Disclaimer

Google search, 01/21/2021

Data provided by Morningstar for Currency and Coinbase for Cryptocurrency
Decentralized
Fault-tolerant
Secure

ways of sharing and storing information
Decentralized
Fault-tolerant
Secure
… many other

ways of sharing and storing information
Fundamental Questions

Blockchains and the Future of Distributed Computing
[Herlihy PODC 17]

- No formal abstraction of these objects has been proposed

Formalizing and Implementing Distributed Ledger Objects
[Anta et al. NETYS 18]

- What is the service that must be provided by a distributed ledger?
- What properties a distributed ledger must satisfy?
- What are the assumptions made by the protocols and algorithms on the underlying system?
- Does a distributed ledger require a linked cryptocurrency?
Fundamental Questions

Blockchains and the Future of Distributed Computing

[Herlihy PODC 17]

● No formal abstraction of these objects has been proposed

Formalizing and Implementing Distributed Ledger Objects

[Anta et al. NETYS 18]

● What is the service that must be provided by a distributed ledger?
● **What properties a distributed ledger must satisfy?**
● What are the assumptions made by the protocols and algorithms on the underlying system?
● Does a distributed ledger require a linked cryptocurrency?
Why Blockchain is Hard?
Blockchain?

Block 3
Header
Hash Previous (3)

Body
Transaction
Transaction
Transaction

Block 2
Header
Hash Previous (2)

Body
Transaction
Transaction
Transaction

Block 1
Header
Hash Previous (1)

Body
Transaction
Transaction
Transaction
Blockchain?

Block 3
Header
Hash Previous (3)

Body
Transaction
Transaction
Transaction

Block 2
Header
Hash Previous (2)

Body
Transaction
Transaction
Transaction

Block 1
Header
Hash Previous (1)

Body
Transaction
Transaction
Transaction
Blockchain? Block + Chain!
In this talk, no crypto detail

- What Block?
- How to link?
- How secure?
- How to mine?
Bitcoin on a high-level:
In each round,
- Nodes exchange blocks (mining)
- Nodes “agree on” a block
Block + Chain!  [in Distributed Computing]

Bitcoin on a high-level:
In each round,
- Nodes exchange blocks (mining)
- Nodes “agree on” a block
Block + Chain! [in Distributed Computing]
disagreement $\Rightarrow$ double-spending attack
Block + Chain!

disagreement $\Rightarrow$ double-spending attack

FLP Result
[JACM 85]: Fault + Async. + Consensus = Impossible
Most common approach: Proof-of-XXX
Is Consensus necessary?
Is Consensus necessary?
No
The Consensus Number of a Cryptocurrency

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi [PODC 2019]

Model

Asynchronous network: arbitrary message delay

Permissioned, static system: a fixed set of nodes \([1, \ldots, n]\)

Crash fault: up to \(f\) fail-stop failures
Cryptocurrency: Working Definition

A cryptocurrency is a virtual asset that relies on cryptography tools to prevent counterfeit or double-spend.
Cryptocurrency: Abstraction

A cryptocurrency is a virtual asset that relies on cryptography tools to prevent counterfeit or double-spend.
Cryptocurrency: Counterfeit

A cryptocurrency is a virtual asset that relies on cryptography tools to prevent counterfeit or double-spend.

Node A: $\text{Init}(A) = \{c\}$

Node B: $\text{Init}(B) = \{}$

Node C: $\text{Init}(C) = \{}$

Transactions:
- $\text{Tran}(A,B,\{c'\})$
- $\text{Recv}(A,B,\{c'\})$
Cryptocurrency: Double-spend

A cryptocurrency is a virtual asset that relies on cryptography tools to prevent counterfeit or **double-spend**.

**Node A**
- Init(A) = \{c\}
- Tran(A,C,\{c\})
- Tran(A,B,\{c\})

**Node B**
- Init(B) = {} 
- Tran(A,B,\{c\})
- Revc(A,B,\{c\})

**Node C**
- Init(C) = {} 
- Tran(A,C,\{c\})
- Revc(A,B,\{c\})
What is the (concurrent) data structure?

Asset transfer object [PODC 2019]

- Each node has an account
- ATO state: balance of each account
- Transfer(A,B,x):
  - Decrease A’s account by x
  - Increase B’s account by x
- Read(A): return A’s balance
What is the (concurrent) data structure?

Asset transfer object [PODC 2019]

- Each node has an account
- ATO state: balance of each account
- Transfer(A,B,x):
  - Decrease A’s account by x
  - Increase B’s account by x
- Read(A): return A’s balance

Valid transfer/transaction: Transfer(A,B,x)

- Invoked by A
- \( \text{balance}(A) \geq x \)
What is the (concurrent) data structure?

Asset transfer object [PODC 2019]

- Each node has an account
- ATO state: balance of each account
- Transfer(A,B,x):
  - Decrease A’s account by x
  - Increase B’s account by x
- Read(A): return A’s balance

Valid transfer/transaction: \( \text{Transfer}(A,B,x) \)

- Invoked by A
- \( \text{balance}(A) \geq x \)  \( \text{No overdraft} \)
Atomic Snapshot Object [Afek et al. JACM 93]

Object partitioned into n segments

Each segment is “owned” by a node (single-writer)

**Update**: write a value to own segment

**Scan**: read values from all segments -- take a snapshot

Operations **linearizable** -- a total order that follows the real-time order
Atomic Snapshot Object [Afek et al. JACM 93]

Object partitioned into n segments

Each segment is “owned” by a node (single-writer)

**Update**: write a value to own segment

**Scan**: read values from all segments -- take a snapshot

Operations **linearizable** -- a total order that follows the real-time order

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

**Intuition:**
- i’s entry = all outgoing transfers at node i

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>{(1, 2, 100}}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

Read(A):

- $S \leftarrow \text{AS.scan}$
- Return A’s balance in $S$

**Intuition:**

- $i$’s entry = all outgoing transfers at node $i$

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>${(1, 2, 100)}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

Read(A):

- S ← AS.scan
- Return A’s balance in S

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>{(1, 2, 100)}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Intuition:

- i’s entry = all outgoing transfers at node i
ASO-based Cryptocurrency [PODC 2019]

Read(A):
- \( S \leftarrow \text{AS.scan} \)
- Return A’s balance in \( S \)

**Intuition:**
- \( i \)'s entry = **all outgoing transfers** at node \( i \)

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>{(1, 2, 100)}</td>
<td>{(2, 3, 50)}</td>
<td></td>
</tr>
<tr>
<td>Balance</td>
<td>0</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

Read(A):
- $S \leftarrow AS$.scan
- Return A’s balance in $S$

Transfer(A,B,x):
- $S \leftarrow AS$.scan
- If valid transaction:
  - $OP[A] \leftarrow OP[A] \cup \{(A,B,x)\}$
  - AS.update($OP[A]$)

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>${(1, 2, 100)}$</td>
<td>${(2, 3, 50)}$</td>
<td></td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

Read(A):
- $S \leftarrow AS\text{.scan}$
- Return A’s balance in $S$

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{(1, 2, 100)}</td>
</tr>
<tr>
<td>2</td>
<td>{(2, 3, 50)}</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Balance

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>

Transfer(A,B,x):
- $S \leftarrow AS\text{.scan}$
- If valid transaction:
  - $OP[A] \leftarrow OP[A] \cup \{(A,B,x)\}$
  - $AS\text{.update}(OP[A])$

learn new incoming tx’s
Example

Node 1

Tran(1,2,100)

Node 2

Read(2) → Read(2) → Read(2)

Node 3
Example

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Node 1

Node 2

Node 3
Example

Node 1

Tran(1,2,100)

Node 2

Read(2)

Read(2)

Read(2)

Node 3

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
**ASO-based Cryptocurrency [PODC 2019]**

**Read(A):**
- S ← AS.scan
- Return A’s balance in S

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{(1, 2, 100)}</td>
</tr>
<tr>
<td>2</td>
<td>{(2, 3, 50)}</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Transfer(A,B,x):**
- S ← AS.scan
- If valid transaction:
  - OP[A] ← OP[A] U {(A,B,x)}
  - AS.update(OP[A])

**Balance**

<table>
<thead>
<tr>
<th>ID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>
ASO-based Cryptocurrency [PODC 2019]

Read(A):
- \( S \leftarrow AS\text{-}scan \)
- Return A’s balance in \( S \)

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{(1, 2, 100)}</td>
</tr>
<tr>
<td>2</td>
<td>{(2, 3, 50)}</td>
</tr>
</tbody>
</table>

Transfer(A,B,x):
- \( S \leftarrow AS\text{-}scan \)
- If valid transaction:
  - \( \text{OP[A]} \leftarrow \text{OP[A]} \cup \{(A,B,x)\} \)
  - \( \text{AS}\text{-}update(\text{OP[A]}) \)
  - Update A’s outgoing tx’s

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>{(1, 2, 100)}</td>
<td>{(2, 3, 50)}</td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Node 1: Tran(1, 2, 100)

Node 2: Read(2)

Node 3

---

Lewis Tseng (BC)  
AtheCrypt 2021  
Jan. 2021
Example

<table>
<thead>
<tr>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Node 1: \texttt{Tran(1,2,100)}

Node 2: \texttt{Read(2)} \rightarrow \texttt{Tran(2,3,50)}

Node 3: \texttt{Read(3)}
Is this possible?

Node 1

Tran(1,2,100)

Node 2

Read(2)

Node 3

Read(2)

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Linearizability:
Total order + **Real-time order**

<table>
<thead>
<tr>
<th>Balance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Linearizability: 
Total order + Real-time order

<table>
<thead>
<tr>
<th>Balance</th>
</tr>
</thead>
</table>
| 1       | 0  
| 2       | 50 |
| 3       | 50 |

Node 1: **Tran(1,2,100)**

Node 2: **Read(2)** → **Tran(2,3,50)**

Node 3: **Read(*)**
So what?
ASO can be implemented in asynchronous systems!
It’s all good, but ...
Is it scalable and highly available?
CAP Theorem  [Brewer PODC 00, Gilbert/Lynch 02]

Consistency:  right response to each request

Availability:  termination eventually

Partition tolerance:  
unreliable comm. network
CAP Theorem  [Brewer PODC 00, Gilbert/Lynch 02]

Consistency: right response to each request

Availability: termination eventually

Partition tolerance: unreliable communication network

Impossible to have all three!

When there is a partition, choose consistency or availability
ASO-Crypto: Incorrect

<table>
<thead>
<tr>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Node 1: Tran(1,2,100)

Node 2: Read(2)
ASO-Crypto: Slow

Node 1: Tran(1,2,100)

Node 2: Read(2)

Balance

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Cryptocurrency in a Partitioned Network? (under submission)
Key observation:
Pending transactions
(Delivered but not applied)
Modern Bank

Node 1: Tran(1,2,100)

Node 2: Read(*)

Node 3

Balance

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Modern Bank

Node 1: Tran(1,2,100)

Node 2: Read(*)

Node 3: Read(*)
Modern Bank

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Tran(1,2,100)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 2</th>
<th>Read(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 3</th>
<th>Read(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Key tool: abstracting consistency guarantees
Operations

Transfer

Read

**Audit**: return the “validity proof” of all the outgoing transactions

**Valid transaction**: no double-spend, no counterfeit, no overdraft
Properties

**Eventual delivery:** Tx from the same partition is eventually applied

**Local operation:** No communication needed to complete an operation

**Read-my-write:** Read reflects the effect of all the prior outgoing transfers

**Auditability:** One is able to present validity proof

**Validity:** All applied transactions are valid
Properties under CAP framework

**Eventual delivery:** Tx from the same partition is eventually applied

**Local operation:** No communication needed to complete an operation

**Read-my-write:** Read reflects the effect of all the prior outgoing transfers

**Auditability:** One is able to present validity proof

**Validity:** All applied transactions are valid
Properties under CAP framework

**Eventual delivery:** Tx from the same partition is eventually applied

**Local operation:** No communication needed to complete an operation

**Read-my-write:** Read reflects the effect of all the prior outgoing transfers

**Auditability:** One is able to present validity proof

**Validity:** All applied transactions are valid
Properties under CAP framework

Eventual delivery: Tx from the same partition is eventually applied

Local operation: No communication needed to complete an operation

Read-my-write: Read reflects the effect of all the prior outgoing transfers

Auditability: One is able to present validity proof

Validity: All applied transactions are valid
Thm1: Byzantine node + Eventual delivery + Partition-tolerance = impossible
Thm1: Byzantine node + Eventual delivery + Partition-tolerance = impossible
Thm1: Byzantine node + Eventual delivery + Partition-tolerance = impossible
Thm1: Byzantine node + Eventual delivery + Partition-tolerance = impossible

Don’t apply: violating eventual delivery

Apply: double-spend
Causal consistency

On a high-level, causal consistency lets each node observe the entire causal history (happens-before relation)

Node A → Write1 → Read1 → Node C

[Ahamad et al. DC 95]
Causal consistency

On a high-level, causal consistency lets each node observe the entire causal history (happens-before relation)

- Node A: Write1
- Node B: Read1, Write2
- Node C: Read2

Write1 → Read1 → Write2 → Read2
⇒ Write1 → Read2
Thm2: Causal consistency is necessary
Thm2: Causal consistency is necessary

Audit() does NOT contain Tran(A,B,1)

⇒ Node C violates auditability!
Positive Results

CCC: Causal Cryptocurrency under Crash faults

- Similar to ASO-Crypto, but use causal memory underneath

CCB: Causal Cryptocurrency under Byzantine faults

- Byzantine causal memory [Tseng et al. NCA 19]
- Reliable broadcast [Bracha and Toueg JACM 85]
- Sequence number to stop double-spending
- PKI and digital signature
- A weaker form of eventual delivery: one needs to be able to talk to n-f correct nodes
Summary

ASO-Crypto: consensus not necessary

Our work: total order and strong consistency not necessary

causal consistency necessary

an inherent challenge of Byzantine crypto in partition

two implementations
Future Works

Implementation and evaluation

Permission-less systems

Probabilistic guarantees
Advice

Know fundamentals

- FLP
- CAP

Reach out to other communities

Be comfortable with formalism
Thanks!
Questions?

lewis.tseng@bc.edu