Demystifying Bitcoin

Prof R. Guerraoui EPFL
Demystifying

- Bitcoin
- Blockchain
- Ethereum
- Proof of work
- Smart contracts
- Leader
- Consensus
- Broadcast
- Snapshot
Perspectives

(1) The journalist
(2) The user
(3) The participant
(4) The engineer
(5) The scientist
(1) The Journalist

2008: Financial crisis – Nakamoto (1/21m)
- From 1c to 10000$ through 20000$ (16,000$)

From trading hardware to general trading

2014: Ethereum (CH) - Now 555 $
Perspectives

(1) The journalist

(2) The user

(3) The participant

(4) The engineer

(5) The scientist
(2) The User
(2) The User

- The wallet: 1 private key + several public keys

- Transaction validation
  - Signing + gossiping + mining + chaining

- Transaction commitment
  - After time $t$: thousands of users have seen it
(3) The Participant

Honey, I'm home!
I found a block today!

Miner Jack
P vs NP (Nash/GV 50 – Ford 70)

\[ ? \times ? = 91 \]

\[ 7 \times 13 = ? \]
(3) The Participant

<table>
<thead>
<tr>
<th>Block:</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonce:</td>
<td>2790</td>
<td></td>
</tr>
<tr>
<td>Data:</td>
<td>NCore</td>
<td></td>
</tr>
<tr>
<td>Hash:</td>
<td>0000c5f693ac77a18ae73ace5df932457fc62e8dfa23c2f3c6d8ebb125ba7843</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mine</td>
<td></td>
</tr>
</tbody>
</table>
(3) The Participant

To validate a transaction, a miner has to solve a puzzle including it
- Fairness and cooperation

Incentive: 6.25 bitcoins / puzzle
- 50 bitcoins 3 years ago

Total: 21 millions bitcoins
- Now: 18 millions
(4) The Engineer

- Joinning (a P2P network)
- Signing (a transaction)
- Gossiping (the transaction)
- Gathering (a block)
- Mining (proof of work - nonce)
- Chaining (hash)
- Gossiping (the block)
- Committing/Aborting
Hashing

- Input Data
- Hashing Algo
- Output Hash
The Big Picture

Bitcoin block

Mining: find nonce such that This < d

How? By trying different nonces (brute force)
Smart Contracts

Option contract written as code into a blockchain.

Contract is part of the public blockchain.

Parties involved in the contract are anonymous.

Contract executes itself when the conditions are met.

Regulators use blockchain to keep an eye on contracts.

Happy Hustlin’
Perspectives

(1) The journalist

(2) The user

(3) The participant

(4) The engineer

(5) The scientist
(5) The Scientist

State Machine Replication (78)

Basic consensus

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>X</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z</td>
<td>7</td>
</tr>
</tbody>
</table>
```
Consensus Universality (78)

**Safety:** No two nodes must choose different values. The chosen value must have been proposed by a node.

**Liveness:** Each node must eventually choose a value.

Every service can be implemented in a highly available manner using Consensus
Consensus Impossibility (84)

Consensus is impossible in an asynchronous system
Computing’s central challenge is how not to make a mess of it ...» E. Dijkstra
Can we implement a payment system asynchronously?
P vs NP

7 * 13 = ?

? * ? = 91

Asynchronous vs Synchronous

Is payment an asynchronous problem?

« To understand a distributed computing problem: bring it to shared memory » T. Lannister
The infinitely big

The infinitely small
Message Passing

p1 ➤ Send ➤ p2 ➤ Receive ➤ p3

 示意图说明了进程间的消息传递过程。进程 p1 发送消息到进程 p2，然后 p2 发送消息到 p3。
Shared Memory

p1

Write()

1

p2

Read()

Registers

Message Passing
Atomic Shared Memory

write(1) - ok

read() - 1

read() - 1

read() - 1
write(1) - ok

read() - 1

read() - 0
Message Passing $\iff$ Shared Memory

Quorums (asynchrony)
« To understand a distributed computing problem: bring it to shared memory » T. Lannister

« Optimization is the source of all evil » D. Knuth
P vs NP

7 * 13 = ?

? * ? = 91

Asynchronous vs Synchronous
Can we implement a payment system asynchronously?
Counter: Specification

A counter has two operations \textit{inc()} and \textit{read()}; it maintains an integer \textit{x init to 0}

\textbf{read()}:  
\> return(x)

\textbf{inc()}:  
\> x := x + 1;  
\> return(ok)
The processes share an array of registers \( \text{Reg}[1,..,N] \)

**inc():**
- \( \text{Reg}[i].write(\text{Reg}[i].read() + 1); \)
- \( \text{return}(\text{ok}) \)

**read():**
- \( \text{sum} := 0; \)
- \( \text{for } j = 1 \text{ to } N \text{ do} \)
  - \( \text{sum} := \text{sum} + \text{Reg}[j].read(); \)
- \( \text{return}(\text{sum}) \)
Counter*: Specification

Counter* has, in addition, operation dec()

dec():
  if x > 0 then x := x - 1; return(ok)
  else return(no)

Can we implement Counter* asynchronously?
2-Consensus with Counter*

- Registers R0 and R1 and Counter* C - initialized to 1

- Process pI:
  - `propose(vI)`
  - `RI.write(vI)`
  - `res := C.dec()`
  - `if(res = ok) then`
    - `return(vI)`
    - `else return(R{1-I}.read())`
Impossibility [FLP85,LA87]

- **Theorem:** no *asynchronous* algorithm implements *consensus* among two processes using *registers*

- **Corollary:** no asynchronous algorithm implements Counter* among two processes using *registers*
Theorem: no asynchronous algorithm implements set-agreement using registers

Sperner’s Lemma
The **consensus number** of an object is the maximum number of processes than can solve consensus with it.
Payment Object (PO): Specification

*Pay*(a,b,x): transfer amount x from a to b if a > x (return ok; else return no)*

NB. Only the owner of a invokes Pay(a,*,*)

- **Questions:** can PO be implemented asynchronously? what is the consensus number of PO?
A snapshot has operations update() and scan(); it maintains an array \( x \) of size \( N \)

```plaintext
scan():
    return(x)

update(i,v):
    x[i] := v;
    return(ok)
```
Algorithm?

The processes share one array of N registers \( \text{Reg}[1,\ldots,N] \)

- **scan()**: 
  - for \( j = 1 \) to \( N \) do 
    - \( x[j] := \text{Reg}[j].\text{read}() \)
  - return(\( x \))

- **update\((i,v)\)**: 
  - \( \text{Reg}[i].\text{write}(v) \); return(ok)
Atomicity?

update(1,1) - ok

update(3,2) - ok

scan() - [1,0,2]
**Atomicity?**

- `update(1,1) - ok`
- `scan() - [1,0,2]`
- `update(3,2) - ok`
Atomicity?

\[ \text{scan()} \quad - \quad [0,0,10] \]

\[ \text{update}(2,1) \quad - \quad \text{ok} \]

\[ \text{update}(3,10) \quad - \quad \text{ok} \]
Key idea for atomicity

To *scan*, a process keeps reading the entire snapshot (i.e., *collecting*), until two arrays are the same.

Key idea for wait-freedom

To update, scan then write the value and the scan.

To *scan*, a process keeps collecting and returns a collect if it did not change, or some collect returned by a concurrent *scan*. 
The Payment Object: Algorithm

Every process stores the sequence of its outgoing payments in its snapshot location.

To **pay**, the process scans, computes its current balance: if bigger than the transfer, updates and returns ok, otherwise returns no.

To **read**, scan and return the current balance.
PO can be implemented Asynchronously

Consensus number of PO is 1

Consensus number of PO(k) is k
Payment System (AT2)

- AT2_S
- AT2_D
- AT2_R

- Number of lines of code: one order of magnitude less
- Latency: seconds (at most)
References

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi: The Consensus Number of a Cryptocurrency. PODC 2019: 307-316

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi: Scalable Byzantine Reliable Broadcast. DISC 2019: 1-16 (Best Paper Award)

Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, Athanasios Xygkis: Online Payments by Merely Broadcasting Messages. DSN 2020: 26-38 (Runner for the Best Paper Award)
References