Bitcoin and Nakamoto Consensus

Distributed Systems, Spring 2020
Nikita Borisov
Topics for Today

• Replicated State Machines and Log Consensus
• Bitcoin
  • Consensus approach
  • Transaction broadcast
• MP2 overview
Announcements

• Midterm grades are out: med 52, mean 52.7, STD 6.73 (out of 70)
  • Regrades are due by 11pm on Mar 25\textsuperscript{th}
  • Solution will be released today/tomorrow

• MP2 out today
  • Due on April 13

• HW3 \textbf{extended} till \textbf{Monday 16}

• HW4 out Friday, due \textbf{April 2}
  • \textbf{No extensions}

• Midterm 2 on \textbf{April 6}
State Machine

- A process with some state that responds to events

Set X=7

Set X=5

X: 5
Banks

• State: account balances
  • Alice: $100
  • Bob: $200
  • Charlie: $50

• Events: transactions
  • Alice pays Bob $20
  • Charlie pays Alice $50
  • Charlie pays Bob $50
Databases (e.g., enrollment)

• State: database tables
  • Classes:
    • Alice: CS425, CS438
    • Bob: CS425, CS411
    • Charlie: ECE428, ECE445
  • Rooms:
    • CS425: DCL1320
    • ECE445: ECEB3013

• Events: transactions
  • Alice drops CS425
  • Bob switches to 3 credits
  • Charlie signs up for CS438
  • ECE445 moves to ECEB1013
Filesystems

• State: all files on the system
  • Midterm.tex
  • HW2-solutions.tex
  • Assignments.html

• Events: updates
  • Save midterm solutions to midterm-solutions.tex
  • Append MP2 to Assignments.html
  • Delete exam-draft.tex
State machines

- State: complete state of a program
- Events: messages received
- Assumption: all state machines deterministic
Replicated state machines
Replicated State Machines

• A state machine can fail, taking the state with it

• Replicate for
  • Availability — can continue operation even if one SM fails
  • Durability — data is not lost

• Must ensure:
  • Consistency!
Consistency

X: 0
Set X=7
X: 7
Set X=5
X: 5
Set X=7
Set X=5
Consistency Requirement

All state machines must process

• The same set of events
  • R-multicast
• In the same order
  • Total ordering

Other requirements
• Same initial state
• Deterministic execution
Log Consensus

• Reliable, totally-ordered multicast == Consensus

• TO multicast can implement consensus (how?)

• Consensus can implement TO multicast (how?)

• Event ordering / log consensus: main application of consensus protocols!
Bitcoin

- Implement a distributed, replicated state machine that maintains an account ledger (\(=\) bank)
- Scale to thousands of replicas distributed across the world
- Allow old replicas to fail, new replicas to join seamlessly
- Withstand various types of attacks
Approaches that don’t work

• Totally ordered multicast (e.g., ISIS)
  • Quadratic communication overhead
  • Do not know who all replicas are a priori

• Leader election (e.g., Bully)
  • Quadratic communication overhead
  • Do not know who all replicas are a priori
  • *Nodes with highest IDs are leaders =>*
    • Bottleneck
    • Security
Lottery Leader Election

• Every node chooses a random number
• Leader = closest to 0
Hash Functions

• Cryptographic hash function: \( H(x) \rightarrow \{ 0, 1, ..., 2^{256}-1 \} \)
• Hard to invert:
  • Given \( y \), find \( x \) such that \( H(x) = y \)
• E.g., SHA256, SHA3, ...

• Every node picks random number \( x \) and computes \( H(x) \)
• Node with \( H(x) \) closest to 0 wins
Using a seed

• Every node picks x, computes $H(\text{seed} \ || \ x)$
  • Closest to 0 wins

What to use as a seed?

• Hash of:
  • Previous log
  • Node identifier
  • New messages to add to log

• Two remaining problems:
  • How to find closest to 0?
  • How to prevent nodes from trying multiple random numbers?
Iterated Hashing / Proof of work

• Repeat:
  • Pick random $x$, compute $y = H(seed || x)$
  • If $y < T$, you win!

• Set threshold $T$ so that on average, one winner every few minutes

• E.g.:
  • 1000 nodes
  • $10^{12}$ hash/second
  • Target interval: 10 minutes
  • $T = ?$

• Given a solution, $x$ such that $H(seed || x) < T$, anyone can verify the solution in constant time (microseconds)
Block

\[ H(B1) = H(\text{log entries} || \text{solution}) < T \]
Chaining

• Each block’s puzzle depends on the previous one
  • $L_n \rightarrow L_{n-1} \rightarrow \ldots \rightarrow L_1 \rightarrow L_0$
  • To add $m$ blocks, must solve $m$ puzzles

• Longest chain wins
Chain evolution
Incentives for Logging

• Security better if more people participated in logging
• Incentivize users to log others’ transactions
  • Transaction fees: pay me x% to log your data
  • Mining reward: each block creates bitcoins
    • Replace “Alice minted x” entries with “Alice logged line L_n”
• Payment protocol:
  • Alice->Bob: here’s coin x
  • Broadcast to everyone: Alice transfers x to Bob
  • Bob: wait until transfer appears in a new log line
    • Optionally wait until a few more lines follow it
Alice generated 50 BTC
Nonce: 1234

Bob generated 50 BTC
Nonce: 5678

Carol generated 50 BTC
Alice transferred 10 BTC to Bob + 1 BTC to Carol (fee)
Nonce: 9932

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>39 BTC</td>
</tr>
<tr>
<td>Bob</td>
<td>60 BTC</td>
</tr>
<tr>
<td>Carol</td>
<td>51 BTC</td>
</tr>
</tbody>
</table>
Logging Speed

• How to set $T$?
  • Too short: wasted effort due to broadcast delays & chain splits
  • Too long: slows down transactions

• Periodically adjust difficulty $T$ such that one block gets added every 10 minutes
  • Determined algorithmically based on timestamps of previous log entries

• Current difficulty
  • $7 \times 10^{22} \approx 2^{76}$ hashes to win

• Large number of participants: hard to revise history!
Bitcoin broadcast

• Need to broadcast:
  • Transactions to all nodes, so they can be included in a block
  • New blocks to all nodes, so that they can switch to longest chain

• Why not R-multicast?
  • Have to send $O(N)$ messages
  • Have to *know* which nodes to send to
Gossip / Viral propagation

• Each node connects to a small set of neighbors
  • 10–100

• Nodes propagate transactions and blocks to neighbors

• Push method: when you hear a new tx/block, resend them to all (some) of your neighbors (flooding)
• Pull method: periodically poll neighbors for list of blocks/tx’s, then request any you are missing
Push propagation
Pull propagation

What transactions do you know?

Node 1

Tx1, tx7, tx13, tx25, tx28

Node 2

Please send me tx13, tx28

Contents of tx13, tx28
Maintaining Neighbors

• A *seed* service
  • Gives out a list of random or well-connected nodes
  • E.g., seed.bitnodes.io

• Neighbor discovery
  • Ask neighbors about *their* neighbors
  • Randomly connect to some of them
Bitcoin summary

Foundations:
• Unreliable broadcast using gossip
• Probabilistic “leader” election for mining blocks (tx ordering)
• Longest chain rule to ensure long-term consistency / security

Compared with Paxos/Raft:
• Scales to thousands of participants, dynamic groups
• Tens of minutes to successfully log a transaction (vs. milliseconds)