Blockchain Development
Week: 8
Title: Consensus Engineering

Dr Ian Mitchell

Middlesex University,
Dept. of Computer Science,
London

September 26, 2019

Lecture

Aims

To critically appraise consensus algorithms

Objectives

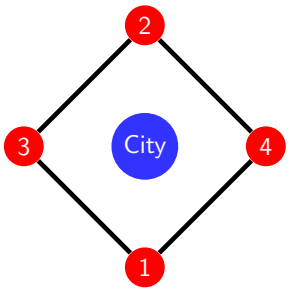
Knowledge

To explain different consensus algorithms [8, 5]

- Consensus?
- PoW: Proof-of-Work
- PoS: Proof-of-Stake
- PoET: Proof-of-Elapsed-Time
- PBFT: Byzantium Algorithms
- PoA: Proof-of-Authority

Consensus

- Byzantine Generals Problem [4]
- Reliable Complex System must cope with failure of one or more of its components
- A failed component may send conflicting information
- Each division commanded by its own General
- The Generals can communicate with each other
- Need a common plan of action
- Trust: Traitor in their midst preventing a consensus



Consensus

Conditions

1. All loyal Generals decide upon the same plan of action
2. A small number of traitors cannot cause the loyal Generals to adopt a bad plan

- Traitors can do what they wish
- Loyal Generals will all do what they are told
- $v(i)$ be information communicated by the i^{th} General. So, you have $v(1), v(2), \ldots, v(n)$, where there are n Generals

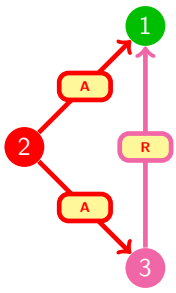
Byzantine Generals Problem

3-node

- 3 nodes, messages ‘A’ and ‘R’ for Attack and Retreat, respectively

Byzantine Generals Problem

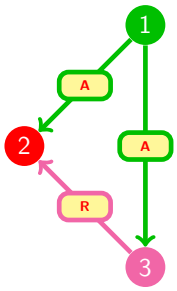
3-node



- 3 nodes, messages ‘A’ and ‘R’ for Attack and Retreat, respectively
- 1 and 2 are loyal; 3 is disloyal.
- 2 sends the same message to 3 and 1
- 3 changes the message and sends to 1
- 1 receives conflicting messages

Byzantine Generals Problem

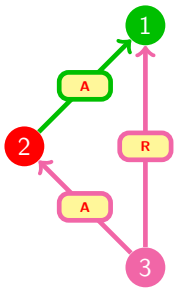
3-node



- 1 and 2 are loyal; 3 is disloyal.
- 1 sends the same message
- 3 changes the messages and sends to 2
- 2 receives conflicting messages

Byzantine Generals Problem

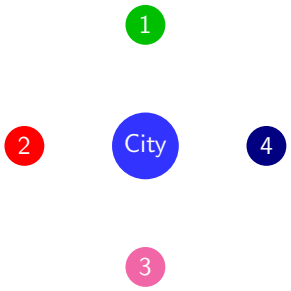
3-node



- 1 and 2 are loyal; 3 is disloyal.
- 3 sends different messages to 1 and 2
- 2 forwards the message to 1
- 1 receives conflicting messages
- needs to be $3m + 1$, where there are m traitors
- informal proof, formal proof [6]

Byzantine Generals Problem

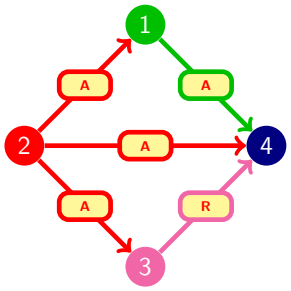
4-node



- 3 nodes, messages ‘A’ and ‘R’ for Attack and Retreat, respectively

Byzantine Generals Problem

4-node



- 3 nodes, messages ‘A’ and ‘R’ for Attack and Retreat, respectively
- 1, 2 and 4 are loyal; 3 is disloyal.
- 2 sends the same message to all nodes
- 3 changes the message and sends to 4
- 4 receives conflicting messages
- 4 acts on majority of messages ’A’

Practical Byzantium Fault Tolerance (PBFT) [2]

- PBFT was first Byzantium fault tolerant algorithm
- PBFT worked in practical and asynchronous environments
- leader-based and non-forking
- does not support open-enrollment
- nodes added by aministrator
- requires full P2P
- Fault tolerance - preserves liveness and safety
- Even when parts of the network are:
  - misbehaving
  - experiencing problems
  - unconnected
  - not working properly
  - behaving dishonestly
- Minimum percentage of node in PBFT are?

PBFT? [2]

- cited alot, least understood
- [4] 5500+
- [2] 2500+
- n number of nodes in network
- $[0, 1, 2, 3, ..., n-2, n-1]$
- f Max. bad nodes the network can tolerate

PBFT Equations

$$n = nodes \tag{8.1}$$

$$f = \frac{n-1}{3} \tag{8.2}$$

$$n = 3f + 1 \tag{8.3}$$

Voting-based adpated from [3]

- Only a single node can commit
- One or more nodes has complete view of network
- Adding or removing nodes from the network is difficult
- There are many P2P (complete)

PBFT [2]

1. A client send a request to invoke a service operation to the primary
2. The primary multicasts the request to the backups
3. Replicas execute the request and send a reply to the client
4. The client waits for $f + 1$ replies from different replicas with the same result; this is the result of the operation

Problem [3]

1. user sends a message to all the nodes
2. a series of messages is sent between the nodes to determine if the message is valid
3. once a number of nodes agree that the request is valid, then the instructions in the request are executed and result is return to client
4. the client waits for a number of replies that match, then accepts the result

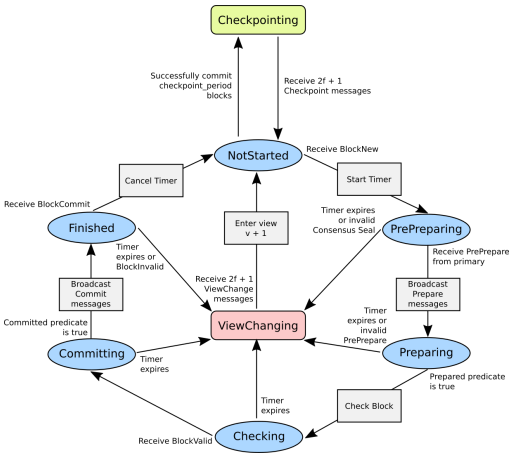
Definitions

Term	Definition
Node	'Machine' running components necessary to wring a blockchain
Server	synonym for Node
Validator	Component of a node for interactions with the blockchain
Primary	Node in charge
Secondary	Auxillary node used for consensus
Client	'Machine' that sends request and receives replies.
Checkpoint	Point when logs can get garbage collection
cp period	How many client requests in between each cp
block duration	block latency
Message	block, with additional information
Working block	block currently being committed, initialised but not finalised
Low water mark	sequence number of the last stable checkpoint

Definitions (cont'd)

Term	Definition
Low water mark	sequence number of the last stable checkpoint
High water mark	low water mark plus the desired maximum size of nodes' message logs
View	scope of PBFT when primary in charge
n	number of nodes
f	max. faulty nodes
v	the current view numner
p	primary server number $p = v \mod n$

State Transitions



Proof-of-Work

- Challenge to solve a very difficult puzzle
- Extremely hard to solve
- Very easy to verify correctness of solution
- Combination lock
- Use of a nonce

PoW

$$H_a(d+n) < h \qquad (8.4)$$

where H is hashing function; a is hashing algorithm (e.g. SHA256); $d$is data; n is nonce; and h is a result of a hashing function usually starting with 4 zeroes.

PoW: Example

- $d = 0$
- $H(0) =$ 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
- target = **0**feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
- $n = 1$
- while $(H(d+n) < target)$
  - n++
- $H(00x1) =$ 6fbc24c863cad03d71238d38f725383eb79804b1adf05b05511470f18ac66129
- $H(00x2) =$ 9eb14f1909e80b0005ea1531e91a315401e5f788e0c5e7f1b7c24f3d2c92e5a4
- $H(00x3) =$ 5e847f40960c2fe8fcaf2bf7b11df0cc012f73c59d52cd2ee8f5ee44b2711e85

⋮

- $H(00x48) =$ 0529f9d44d1ec54ce86601d63aac3a094ac90577b175e024058190a6ec062873
- target= **000**ceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
- $H(00x80) =$ 00021397ccc9e4e75258c17ac7d651674999ea72c6d3f6dfdae55ca8a2174420

PoW

- Waste of Energy
- Application-Specific integrated circuit - ASIC
  - 1kH/s - 1,000 hashes per second
  - 1MH/s - 1,000,000 hashes per second
  - 1GH/s - 1,000,000,000 hashes per second
  - ASIC chip around 30GH/s
- Bitcoin rewards miners
- ₿50 for firsts 210,000 blocks
- ₿25 for second 210,000 blocks
- ₿12 for third 210,000 blocks
- No reward?
  - Rely on transaction fees
  - Less miners and open to 51% attacks
  - Change in consensus algorithm?
- High latency of TX validation

Proof of Stake

- Nodes are validators, not miners
- Validate a TX, to earn TX fee
- Each node has a stake value
- Usually, stake cannot be spent
- Nodes are selected proportionately to the stake value
- Randomness where stakes are equal
- Example:
  - Node A has 200 MDXCoins
  - Node B has 100 MDXCoins
  - Node A is twice as likely to be selected to validate the TX
  - Upon doing so Node A receives the transaction fee
- Many variations on this, Proof-of-Deposit

PoS

Random Selection

- Ratio between stake:all cryptocurrency
- 1% stake of the entire blockchain results in being selected 1% of the time
- 51% stake results in 51% selection

Multi-round Voting

- Byzantine Fault Tolerance PoS [1]
- Select several staked nodes
- Staked users cast a vote
- Elected creates block

Coin Age

- Older stakes are more likely to get selected than younger stakes
- Age is reset after selection
- Fatigue

Delegate Systems

- users vote for nodes to become publishing nodes
- voting power is proportionate to stake
- incentivised to not act maliciously
- rewards and reputation

PoS

- Less energy spent
- No miners
- Does mean bigger stakes, have more probability of being selected
- low latency of TX validation
- Speeds up block creation

Proof-of-Elapsed-Time, PoET

- All nodes are validators
- Random allocation of wait time
- The node with the shortest wait times validates the TX
- Permissioned blockchain
- Low latency of TX validation
- Speeds up block creation [7]
- Does depend on size of block and data in transaction
- Scalability is still an issue (50M transactions per second)

Conclusion

Name	Goals	Advantages	Disadvantages	Domain	Implem

Summary

Criteria	PoW	PoS	Hybrid PoW/S	PoET
Efficiency	No	Yes	No	Yes
H/w	Very Important	None	Important	None
Speed	Poor	Good	Poor	Good
Example	BitCoin	NextCoin	BlackCoin	HyperLedger

References I

[1] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. "Making bft protocols really adaptive". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 904–913.

[2] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[3] Sawtooth Hylperledger. *PBFT consensus - a pending Sawtooth RFC*. https://github.com/hyperledger/sawtooth-rfcs. Version 0.1. [accessed: 04-AUG-19].

[4] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[5] Giang-Truong Nguyen and Kyungbaek Kim. "A Survey about Consensus Algorithms Used in Blockchain.". In: *Journal of Information processing systems* 14.1 (2018).

References II

[6] Marshall Pease, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults". In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

[7] Parth Thakkar, Senthil Nathan, and Balaji Vishwanathan. "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform". In: *arXiv preprint arXiv:1805.11390* (2018).

[8] Dylan Yaga et al. *Blockchain technology overview*. Tech. rep. National Institute of Standards and Technology, 2018.