PROOF OF STAKE

Uitleg van het proof of stake model gemaakt door S6 van periode: Februari t/m Juni 2018.
Geschreven door Merik Westerveld - S64.

De volgende stappen leggen in volgorde uit hoe het proof of stake consensus model in elkaar zit en werkt. De volgende stappen slaan we in detail over: Genereren / inloggen van een wallet - tijd gelijk stellen met het netwerk - binnenhalen van de up-to-date ledger tot wanneer de wallet een volledig geupdate node is. Vanaf dit moment gaat het proof of stake model in werking en moet het de volgende stappen uitvoeren.

1. Maak een nieuwe transactie

```
Transaction transaction = new Transaction();
SharedMessage<Transaction> newTransactionMessage = null;

if(input.substring(0, 1).toLowerCase().equals("c")) {
  transaction = new CoinTransaction(wallet.getPublicKey(),
        wallet.getPrivateKey(),
        HashUtil.decodePublicKey(publicKey),
        amount);
  newTransactionMessage = new SharedMessage(transaction, Global.getMyIp(),
MessageType.NEW_COIN_TRANSACTION);
} else if(input.substring(0, 1).toLowerCase().equals("s")){
  transaction = new StakeTransaction(wallet.getPublicKey(),
        wallet.getPrivateKey(),
        HashUtil.decodePublicKey(publicKey),
        amount);
  newTransactionMessage = new SharedMessage(transaction, Global.getMyIp(),
MessageType.NEW_STAKE_TRANSACTION);
}

WebSocketServer.handler.addToReceivedMessage(newTransactionMessage);
WebSocketClientWrapper.getWrapper().sendMessage(newTransactionMessage);
```

Er wordt gekeken of er in de console een nieuwe coin of nieuwe stake transactie is gemaakt. Op basis hiervan wordt er een nieuwe transactie gemaakt met de huidige wallet (deze transactie wordt dus ook gesigned met de private key van de huidige wallet). Vervolgens wordt de gemaakte transactie verstuurd naar de peers (P2P netwerk wordt hier in gang gebracht).

2. Ontvangen van de gemaakte transactie

```
TemporaryStorage.getInstance().addPendingCoinTransaction((CoinTransaction) message.getContent());
System.out.println("[INFO] Received new coin transaction: " + message.getContent().toString());
if(handler.shouldSendMessage(message)) {
  handler.forwardToPeers(message);
}
```

De ontvangers ontvangen de nieuwe transactie en zetten deze transactie in de pendingCoinTransaction lijst. Deze pendingCoinTransaction lijst in de temporary storage singleton wordt later gebruikt bij het genereren van een nieuw block.

3. Nieuw block maken

```
Thread createBlockThread = new Thread(() -> {
  String lastBlockHash = GlobalState.getInstance().getLastBlockHash();
  Block proposedBlock;
  if(!lastBlockHash.trim().equals("")){
     proposedBlock = new Block(lastBlockHash, validator,
TemporaryStorage.getInstance().getPendingCoinTransactions(),
           TemporaryStorage.getInstance().getPendingStakeTransactions());
  } else {
     proposedBlock = new Block("", validator, TemporaryStorage.getInstance().getPendingCoinTransactions(),
           TemporaryStorage.getInstance().getPendingStakeTransactions());
  }

  TemporaryStorage.getInstance().addProposedBlock(proposedBlock);

  SharedMessage sendMessage = new SharedMessage(proposedBlock, Global.getMyIp(), MessageType.NEW_BLOCK);
  WebSocketServer.handler.addToReceivedMessage(sendMessage);
  WebSocketClientWrapper.getWrapper().sendMessage(sendMessage);

  Timer timer = new Timer();
  timer.schedule(new LotteryTask(), 10000);
});
createBlockThread.start();
```

Vervolgens wordt er elke 5 minuten door elke node die aan staat een nieuw block gegenereerd. Elke node doet dit op precies dezelfde tijd omdat de node in het begin zijn tijd gesynct heeft met het netwerk. Bij het maken van het block worden alle pending transactions en de previous block hash in het nieuwe block gezet. Daarnaast zet de huidige wallet zichzelf als validator in het block. Vervolgens voegt hij zijn eigen gemaakte block toe in zijn lijst met proposedBlocks in de temporary storage singleton. Daarna wordt er 1 minuut gewacht tot dat de daadwerkelijke proof of stake loterij wordt afgetrapt. Deze tijd heeft de node nodig om al de overige proposed blocks te ontvangen.

4. Ontvangen van proposed blocks andere nodes

```
TemporaryStorage.getInstance().addProposedBlock((Block) message.getContent());
System.out.println("[INFO] Received new block");
if(handler.shouldSendMessage(message)) {
  handler.forwardToPeers(message);
}
```

Alle andere proposed blocks worden ontvangen door alle nodes en in hun eigen temporary storage singleton gezet.

5. Begin van loterij aftrappen

```
String previousHash = GlobalState.getInstance().getLastBlockHash();
Block chosenBlock = LotteryUtil.pickWinner(TemporaryStorage.getInstance().getProposedBlocks(), previousHash);

HashMap<String, Transaction> notValidatedCoinTransactions = new HashMap<>();
HashMap<String, Transaction> notValidatedStakeTransactions = new HashMap<>();

for(Block block : TemporaryStorage.getInstance().getProposedBlocks()){
  if(block.getAllTransactions().size() > chosenBlock.getAllTransactions().size()){
     notValidatedCoinTransactions = (HashMap<String, Transaction>) block.getCoinTransactions();
     notValidatedStakeTransactions = (HashMap<String, Transaction>) block.getStakeTransactions();
    for(Map.Entry<String, Transaction> transactionSet : chosenBlock.getAllTransactions().entrySet()){
       notValidatedCoinTransactions.remove(transactionSet.getKey());
       notValidatedStakeTransactions.remove(transactionSet.getKey());
    }
  }
}

TemporaryStorage.getInstance().clearProposedBlocks();

TemporaryStorage.getInstance().addPendingCoinTransactions(notValidatedCoinTransactions);
TemporaryStorage.getInstance().addPendingStakeTransactions(notValidatedStakeTransactions);

GlobalState.getInstance().addBlock(chosenBlock);
for(Map.Entry<String, Transaction> coinTransactionKeyValueSet : chosenBlock.getCoinTransactions().entrySet()){
  Transaction coinTransaction = coinTransactionKeyValueSet.getValue();
  GlobalState.getInstance().changeBalance(coinTransaction.getReceiver(), coinTransaction.getAmount(),
BalanceChanger.ADD, TransactionType.COIN);
  GlobalState.getInstance().changeBalance(coinTransaction.getSender(), coinTransaction.getAmount(),
BalanceChanger.SUBTRACT, TransactionType.COIN);
}
for(Map.Entry<String, Transaction> stakeTransactionKeyValueSet : chosenBlock.getStakeTransactions().entrySet()){
  Transaction stakeTransaction = stakeTransactionKeyValueSet.getValue();
  GlobalState.getInstance().changeBalance(stakeTransaction.getReceiver(), stakeTransaction.getAmount(),
BalanceChanger.ADD, TransactionType.STAKE);
}
```

Er wordt door elke node een block uitgekozen in de loterij (LotteryUtil.PickWinner). Als het gekozen block eruit is gekomen wordt er gekeken of er nog transactions zijn die nog niet in

het gekozen block zitten. Deze worden weer in de pending transactions terug gezet. Vervolgens wordt de global state bijgewerkt met de account balances.

6. Loterij task

```
public static Block pickWinner(List<Block> receivedBlocks, String previousBlockId){
  HashMap<String, Long> stakeHashMap = new HashMap<>();
  for(GlobalStateUser globalStateUser : GlobalState.getInstance().getUsers()){
     stakeHashMap.put(globalStateUser.getPublicKey(), globalStateUser.getStake());
  }
  HashMap<String, Long> candidatesHashMap = new HashMap<>();
  HashMap<String, Block> candidateBlocksHashMap = new HashMap<>();

  Long tickets = 0l;

  for(Block block : receivedBlocks) {
     if(stakeHashMap.containsKey(block.getValidator())) {
        tickets += stakeHashMap.get(block.getValidator());
        candidatesHashMap.put(block.getValidator(), stakeHashMap.get(block.getValidator()));
        candidateBlocksHashMap.put(block.getValidator(), block);
     }
  }

  TreeMap<String, Long> candidatesSortedMap = new TreeMap<>(candidatesHashMap);

  Random random = new Random();

  StringBuilder hashToLong = new StringBuilder();

  for(char c : previousBlockId.substring(0, 7).toCharArray()){
     hashToLong.append(String.valueOf((int) c));
  }

  random.setSeed(Long.parseLong(hashToLong.toString()));
  int randomNumber = random.nextInt(tickets.intValue()) + 1;

  int bottomMargin = 0;
  int upperMargin = 0;
  int candidatePointer = 0;
  while (upperMargin < randomNumber) {
     String publicKey = new ArrayList<>(candidatesSortedMap.keySet()).get(candidatePointer);
     Long stake = candidatesSortedMap.get(publicKey);
     upperMargin += stake;
     if (bottomMargin <= randomNumber && upperMargin >= randomNumber) {
        return candidateBlocksHashMap.get(publicKey);
     }
     bottomMargin += stake;
     ++candidatePointer;
  }

  return new Block();
}
```

Als aller eerste wordt er een stake hashmap gemaakt die bijhoudt hoeveel stake welke gebruiker / wallet heeft die een block heeft gestuurd. Vervolgens wordt er berekend hoeveel tickets er beschikbaar zijn. Dit wordt gedaan door alle stake de deelnemende nodes bij elkaar op te tellen. Vervolgens wordt er een treemap gemaakt met een gesorteerde kandidaten lijst. Er wordt een random nummer gegenereerd door middel van de previous block hash. Op deze manier is het een random getal maar wel bij elke node in het netwerk hetzelfde. Als laatste wordt er gekeken welke node gewonnen heeft aan de hand van welke tickets hij vast houdt en welk random nummer is gegenereerd (zie while loop). Het gekozen block wordt terug gestuurd en gaat verder bij stap 5.