共識演算法 - 工作量證明 Proof of Work（PoW）

張育銘

2021/03/21

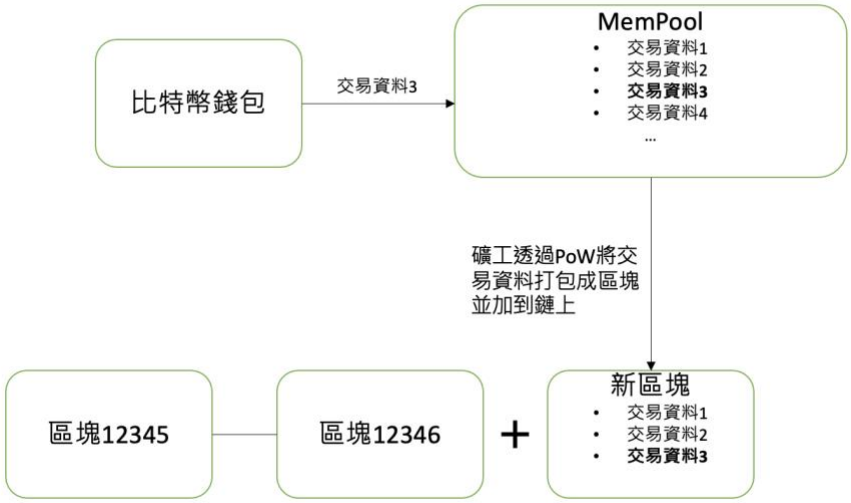
簡介

在分散式的架構下，任何人(node)都可能是增加區塊的角色，倘若沒有一個機制去限制與驗證增加區塊時的正確與公平性，則會出現像是雙重支付(Double Spending)或是區塊資料遭到竄改等問題，這些問題都會發生在沒有 Proof of Work 之前，區塊產生容易且速度快。為確保區塊鏈上交易數據的正確性與公平性，以比特幣為例，使用了 Proof of Work(PoW) 作為區塊產生的共識演算法。此演算法目的在於維持區塊產生的時間區間與大小，以及使任何想要將“交易數據打包成區塊並將區塊加入鏈上”的角色(比特幣上稱之為礦工 miner)都需付出一定的代價(以比特幣為例，代價則是電腦算力)，進而達到資料的正確性與增加區塊的公平性。

工作量證明 **Proof of Work**

當我們將交易資訊發佈至比特幣的網路上時，交易資料會先被傳至一個暫存區，稱之為 Mempool(Memory Pool)，此時的交易資料狀態為 Pending 或稱作 Unconfirmed transaction。這些交易資訊需要被打包成區塊並加入到鏈上，這則為礦工的主要工作，圖一。為達上述目的，礦工們透過 PoW 來證明自己花費了算力與時間將資料打包並加入區塊鏈上，誰先透過加密演算法找到符合規則的 hash 碼則誰先能將區塊資料加到區塊鏈上。

圖一：



區塊鏈上的每一區塊都會對應到一組加密過的 Hash 碼，每個區塊鏈所使用的加密演算法不同，以比特幣為例使用 SHA-256。打包交易資料、產生區塊並產生出 Hash 碼這一步驟被稱為挖礦，而這中間的過程則是透過完成 PoW。

以下透過簡單的 Python 程式碼作為範例講解 PoW 原理。

圖二

blockchain.py

```python

import hashlib
import json

class Block():
    def __init__(self,nonce,timestamp,transactions,previous_hash=''):
        self.nonce=nonce
        self.timestamp=timestamp
        self.transactions=transactions
        self.previous_hash=previous_hash
        self.hash=self.calculate_hash()

    def calculate_hash(self):
        block_data = json.dumps({
            "nonce":self.nonce,
            "timestamp":self.timestamp,
            "transactions":self.transactions,
            "previous_hash":self.previous_hash
            })
        return hashlib.sha256(block_data).hexdigest()

    def mine_block(self,difficulty):
        """
        這邊規則定義為hash碼若開頭達到六個位數的0
        才是符合規則的hash碼
        """
        while(self.hash[:difficulty]!= str('').zfill(difficulty)):
            """
            範例中只將nonce做遞增
            但實際上可根據需求對nonce的產生做變化
            """
            self.nonce += 1
            self.hash=self.calculate_hash()


class BlockChain():
    def __init__(self):
        self.chain=[self.create_genesis_block(),]
        self.difficulty=6
        #範例將difficulty定義為6，但根據區塊鏈的不同此數值可依需求變更

    def create_genesis_block(self):
        return Block(0,'2021/03/21','Genesis')

    def get_last_block(self):
        return self.chain[-1]

    def add_block(self,new_block):
        new_block.previous_hash=self.get_last_block().hash
        new_block.mine_block(self.difficulty)
        self.chain.append(new_block)

```

PoW 主要透過礦工提供算力，根據區塊鏈的定義規則去找出符合該規則之 hash 碼，例如 hash 碼前五位需為 0。根據規則所帶參數之不同以及網路上存有的 node 數量不同，找出符合規則的 hash 碼所需時間也有所不同，因此區塊鏈上需要透過參數 difficulty 來根據情況調整產生區塊之速度。

以圖二為例，在 calculate_hash 這個 method 內，將交易資料打包並加密產生區塊的 hash 碼，然而我們需要一個機制來確保區塊產生速度不會太快進而防止前面提到的問題，因此必須在產生區塊與產生 hash 碼這一環節增加一些規則。

我們可以看到在區塊鏈(BlockChain())物件上我們定義了 difficulty 這個變數，這個變數則會影響到區塊產生的速度，越大，找到符合規則的 hash 碼越困難。

我們將 difficulty 在產生區塊時帶入區塊的 mine_block method 裡(這一步驟則是俗稱的挖礦)，mine_block 這個 method 裡，每迴圈一次會去變更 nonce(number only used once)這個變數，範例中只有遞增，但實際情況可以透過不同演算法去變更 nonce 變數達到困難度的不同。這個變數實質上沒有什麼意義，但因當我們將一樣的資料丟到單向加密演算法做加密時，所產生的 hash 碼都會一樣，這樣則無法達到找尋符合規則之 hash 的需求。因此我們需要有一個變數能夠讓我們在每次加密時，在不更動區塊內的交易資料下還能產生不同的 hash 碼。當找到符合需求(範例為，hash 碼前六碼為 0)的 hash 碼後，則將區塊加入到區塊鏈中。

總結

PoW 還是存在著一定的缺陷，比如說需要消耗大量的算力，又或者，雖然是分散式區塊鏈網絡，但當 51%以上的算力都集中到了某些大型的礦池時，區塊鏈的安全性還是會受到質疑，因為算力佔的百分比越大，也代表著能夠更快的找到 hash 碼且將區塊加入區塊鏈上，這樣就容易讓有心駭客利用，竄改交易資料。因此在比特幣之後，出現許多其他的區塊鏈，並也個別提出更多更複雜的共識演算法，如 Proof of Stake, Proof of Governance 等。以 Proof of stake(PoS)為例，除去了礦工之間使用大量算力來競爭誰最先找出 hash 碼打包區塊的問題，將打包區塊的角色從 miner 變成 validator。成為 validator 則需要做到 stake 的動作，也就是將一部分資金鎖上，區塊鏈會根據規則定義，一段時間內選出下一個 validator 來做區塊加密的動作，完成後則會得到報酬，所謂的 block reward。

然而，在一個運作許久的區塊鏈上不管是 PoW 還是 PoS，需要佔有 51%以上的算力或是資金都是非常困難的，因此雖然有上述之可能的問題，但發生的可能性都極低。