区块链实验二实验报告

PB19071405 王昊元

2022 年 04 月 22 日

1 实验目的及要求

1.1 实验目的

- 实现区块链上的 POW 证明算法
- 理解区块链上的难度调整的作用

1.2 实验要求

- 完成 `proofofwork.go`，包括 `Run()` 和 `Validate()` 两个函数
- 完成比较 `targetBits` 的修改带来的计算次数上的变化情况，从 5-15 的变化

2 实验原理

2.1 POW

工作量证明（Proof-of-Work，PoW）是一种对应服务与资源滥用、或是拒绝服务攻击的经济对策。

一般要求用户进行一些耗时适当的复杂运算，并且答案能被服务方快速验算，以此耗用的时间、设备与能源做为担保成本，以确保服务与资源是被真正的需求所使用。

2.2 区块链哈希

比特币采用了哈希现金 (hashcash) 的工作量证明机制，对应流程如下：

1. 首先构建当前区块头，区块头包含上一个区块哈希值（32 位），当前区块数据对应哈希（32 位，即区块数据的 merkle 根），时间戳，区块难度，计数器（nonce）。
2. 添加计数器，作为随机数。计算器从 0 开始基础，每个回合加 1。
3. 对于上述的数据来进行一个哈希的操作。
4. 判断结果是否满足计算的条件：如果符合，则得到了满足结果；如果没有符合，从 2 开始重新直接 2、3、4 步骤。

3 实验平台

- 操作系统：macOS Big Sur 11.2.3
- Go version: go1.17.8 darwin/amd64

4 实验步骤

4.1 proofofwork.go

- `Run()` 按照第2.2部分的算法，在 `nonce < maxNonce` 时不断循环计算哈希，直到满足要求。代码实现如下：

```
func (pow *ProofOfWork) Run() (int, []byte) {
        nonce := 0
        var hash_int big.Int
        var hash [32]byte

        for nonce < maxNonce {
                // data waiting for hash
                data := bytes.Join(
                        [][]byte{
                                pow.block.PrevBlockHash,
                                pow.block.HashData(),
                                IntToHex(pow.block.Timestamp),
                                IntToHex(int64(pow.block.Bits)),
                                IntToHex(int64(nonce)),
                        },
                        []byte{},
```

```
                )
                // compute hash and the corrsponding big int
                hash = sha256.Sum256(data)
                hash_int.SetBytes(hash[:])
                // compute the target int (a number begin with pow.block.
                    Bits zero in binary)
                target := big.NewInt(1)
                target.Lsh(target, uint(256 - pow.block.Bits))
                // validate
                if hash_int.Cmp(target) == -1 {
                        break
                } else {
                        nonce += 1
                }
        }
        // return nonce, pow.block.Hash
        return nonce, hash[:]
}
```

- `Validate()` 直接判断区块头的哈希是否满足要求。

```
func (pow *ProofOfWork) Validate() bool {
        var hash_int big.Int
        // compute the big int corresponding block hash
        hash_int.SetBytes(pow.block.Hash[:])
        // compute the target int
        target := big.NewInt(1)
        target.Lsh(target, uint(256 - pow.block.Bits))
        // validate
        return (hash_int.Cmp(target) == -1)
}
```

5 实验结果

向区块链中插入数据，结果如下：

从图1中可以看到，数据成功的被加密，`TargetBits` 随着链长的增加逐渐增加，整体上 `Nonce` 的大小也随着 `TargetBits` 的增加而增加，`Pow` 部分变为了 `True`。

`Nonce` 随 `TargetBits` 的变化如下图所示：从图2中能看出由于 `TargetBits` 比较

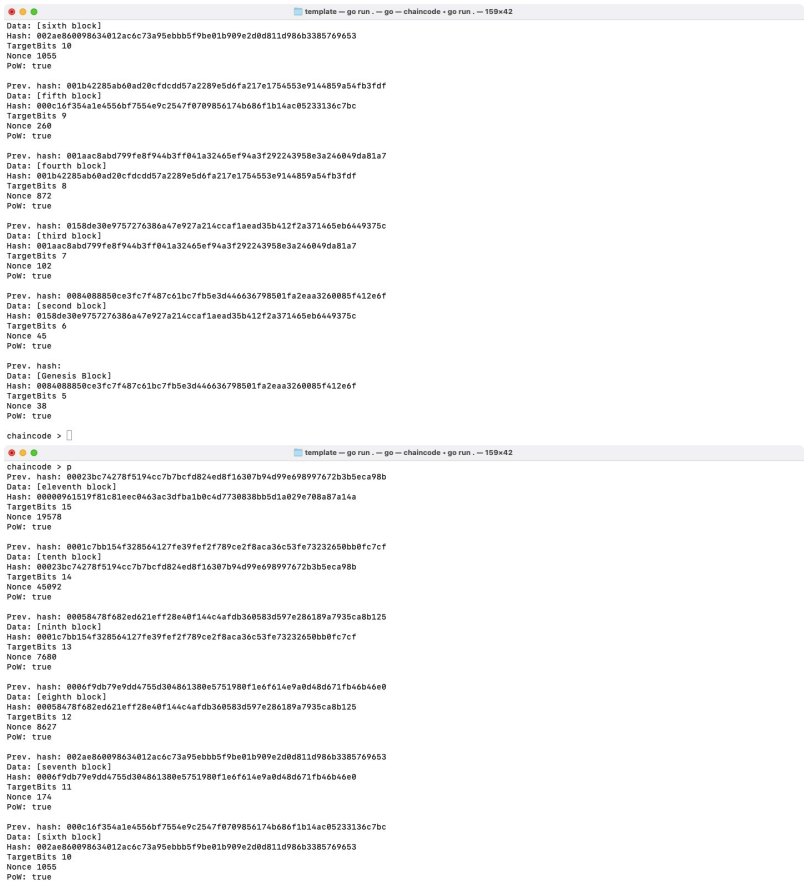
template — go run . — go — chaincode • go run . — 159×42
Data: [sixth block]
Hash: 002ae8600986340l2ac6c73a95ebbb5f9be01b909e2d0d811d986b3385769653
TargetBits 10
Nonce 1055
PoW: true
Prev. hash: 001b42285ab60ad20cfdcdd57a2289e5d6fa217e1754553e9144859a54fb3fdf
Data: [fifth block]
Hash: 000c16f354a1e4556bf7554e9c2547f0709856174b686f1b14ac05233136c7bc
TargetBits 9
Nonce 260
PoW: true
Prev. hash: 001aac8abd799fe8f944b3ff041a32465ef94a3f292243958e3a246049da81a7
Data: [fourth block]
Hash: 001b42285ab60ad20cfdcdd57a2289e5d6fa217e1754553e9144859a54fb3fdf
TargetBits 8
Nonce 872
PoW: true
Prev. hash: 0158de30e975727638 6a47e927a214ccaf1aead35b412f2a371465eb6449375c
Data: [third block]
Hash: 001aac8abd799fe8f944b3ff041a32465ef94a3f292243958e3a246049da81a7
TargetBits 7
Nonce 102
PoW: true
Prev. hash: 0084088850ce3fc7f487c61bc7fb5e3d446636798501fa2eaa3260085f412e6f
Data: [second block]
Hash: 0158de30e975727638 6a47e927a214ccaf1aead35b412f2a371465eb6449375c
TargetBits 6
Nonce 45
PoW: true
Prev. hash:
Data: [Genesis Block]
Hash: 0084088850ce3fc7f487c61bc7fb5e3d446636798501fa2eaa3260085f412e6f
TargetBits 5
Nonce 38
PoW: true
chaincode >
template — go run . — go — chaincode • go run . — 159×42
chaincode > p
Prev. hash: 00023bc74278f5194cc7b7bcfd824ed8f16307b94d99e698997672b3b5eca98b
Data: [eleventh block]
Hash: 0000096l619f81c81eec0463ac3dfba1b0c4d7730838bb5d1a029e708a87a14a
TargetBits 15
Nonce 19578
PoW: true
Prev. hash: 0001c7bb154f328564127fe39fef2f789ce2f8aca36c53fe73232650bb0fc7cf
Data: [tenth block]
Hash: 00023bc74278f5194cc7b7bcfd824ed8f16307b94d99e698997672b3b5eca98b
TargetBits 14
Nonce 45092
PoW: true
Prev. hash: 00058478f682ed621eff28e40f144c4afdb360583d597e286189a7935ca8b125
Data: [ninth block]
Hash: 0001c7bb154f328564127fe39fef2f789ce2f8aca36c53fe73232650bb0fc7cf
TargetBits 13
Nonce 7680
PoW: true
Prev. hash: 0006f9db79e9dd4755d30486l380e5751980f1e6f614e9a0d48d671fb46b46e0
Data: [eighth block]
Hash: 00058478f682ed621eff28e40f144c4afdb360583d597e286189a7935ca8b125
TargetBits 12
Nonce 8627
PoW: true
Prev. hash: 002ae8600986340l2ac6c73a95ebbb5f9be01b909e2d0d811d986b3385769653
Data: [seventh block]
Hash: 0006f9db79e9dd4755d30486l380e5751980f1e6f614e9a0d48d671fb46b46e0
TargetBits 11
Nonce 174
PoW: true
Prev. hash: 000c16f354a1e4556bf7554e9c2547f0709856174b686f1b14ac05233136c7bc
Data: [sixth block]
Hash: 002ae8600986340l2ac6c73a95ebbb5f9be01b909e2d0d811d986b3385769653
TargetBits 10
Nonce 1055
PoW: true

图 1: 实验结果截图

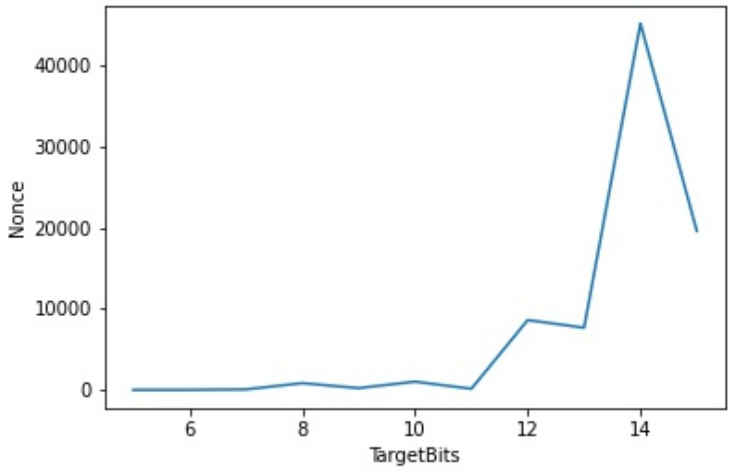

图 2: TargetBits - Nonce 图

小时，符合要求的数较多，导致 `Nonce` 会有波动，不过从整体上 `Nonce` 的大小还是随着 `TargetBits` 的增大而增大。