

PoW区块链共识协议仿真

姓名：湛翔 学号：517021910947 电话：18621813160 邮箱：cascades@sjtu.edu.cn

PoW区块链共识协议仿真

作业要求

仿真过程

数据结构

轮次同步

结果分析

区块链增长

变量一：节点数量

变量二：出块几率

恶意节点攻击

攻击一：分叉攻击

攻击二：自私挖矿

总结

遇到的问题

反思

附录

作业要求

- 利用Python实现一个PoW的仿真程序，模拟一定数量的节点生成区块链的状态。
 - 设置参数包括：节点数量、每个轮次出块的成功率
 - 测量区块链的增长速度
- 设置一定数量的恶意节点实施攻击
 - 测量不同恶意节点比例（10%-40%）条件下，统计分叉攻击成功的长度
 - 测量不同恶意节点比例条件下，自私挖矿收益比例

仿真过程

数据结构

- **Block**: 模拟区块结构
 - 主要成员包括：
 - timestamp: 时间戳，通过time.time()函数获得
 - index: 当前区块的指针
 - transactions: 交易信息，本仿真实验中用当前区块信息来代替
 - nonce: 用于进行PoW的随机数，初始值为0
 - hash: 当前区块的hash值，生成之后不能改变
 - prev_hash: 前一个区块的hash值，用于验证区块链的合法性
 - 包含了挖矿函数mine和通过SHA-256算法计算哈希值的函数calc_hash
- **Blockchain**: 模拟区块链结构，主要成员为blocks，包括验证区块链合法性的validate函数和添加新区块的add_newblock函数。
- **Node**: 模拟节点，通过address成员来标识，分为恶意节点和诚实节点，主要成员为Blockchain。

- **Blockchains**：全局的数据结构，以列表形式存储所有节点当前的区块链，**和对应节点的Blockchain成员指向同一块Blockchain对象的地址**。同步操作时，对该数据结构读操作；更新区块链时，进行写操作。由于不存在临界资源，所以不需要互斥访问。

轮次同步

- **主线程**

主线程的轮次由for循环控制，通过**信号量机制**，**对每个线程都设置初始值为0的信号量**来实现主线程和每个节点的从线程之间的**半同步**，从而让主线程能够控制各个节点（也就是各个线程）的轮次。其中共识操作和挖矿操作各对应一个信号量。

- **各节点对应的子线程**

每次一轮次开始时，等待主线程释放信号量。先进行共识操作，再尝试生成新的区块，随机扩散到其他节点。

结果分析

区块链增长

共进行50轮仿真，观察仿真结束后的区块链长度。

变量一：节点数量

先后尝试了10，20，30，40，50个节点的情况，取q=3，difficulty=1000的情况。考虑到挖矿的随机性，各情况进行3次平行实验。结果如下：

变量\节点数	10	20	30	40	50
平均长度	37.7	41.3	43.3	44.7	46

可以看到，随着节点数的增加，总体的算力增加，每一轮中挖出区块的比例也增加。

变量二：出块几率

根据PoW协议，出块几率由difficulty和q两个参数决定。根据初步实验，发现difficulty的改变对区块链生成的影响较大，所以主要用q值来调整出块几率，尝试的变量组合如下：

- q（括号内为difficulty=3时对应的大致生成几率）：500（10%），1000（20%），1500（30%），2000（40%），3000（50%）
- difficulty：3，4，5

difficulty\q	500	1000	1500	2000	3000
3	33	37	39	40	43
4	16	17	19	21	23
5	5	6	6	6	7

可以看出，difficulty对区块链增长的影响较大，q影响较小。

恶意节点攻击

设置轮数上限为200，若攻击成功则停止。

攻击一：分叉攻击

默认一共20个节点，从第一轮开始攻击，变量为恶意节点比例。先后尝试了一共10，20，30，40，50个节点的情况，并分别尝试了10%，20%，30%，40%为恶意节点的情况，统计分叉攻击成功的长度。结合真实场景，我们采用[比特币"六块确认原则"](#)，即当攻击长度为6（比特币的长度），认为攻击成功。

通过在Node数据结构中添加is_evil变量，如果为恶意节点，便参与到恶意区块链的挖矿过程中。所有恶意节点共享同一条恶意区块链，在每轮操作中对该恶意区块链进行挖矿，当其长度满足攻击条件时，即比诚实节点的最大长度高6时，则视为攻击成功。考虑到恶意节点的数目在50%以下，为其分配更多的算力，通过增大q值来实现。攻击结果如下，表格内为攻击成功需要的轮次，∞代表攻击失败：

总节点数\恶意节点比例	10%	20%	30%	40%
10	∞	197	163	129
20	199	180	165	134
30	∞	197	150	140
40	190	188	170	120
50	∞	160	180	119


可以看出，攻击节点越多，需要的攻击轮次越少，攻击长度和节点总数关系不太大。

攻击二：自私挖矿

以攻击长度（即区块个数）来衡量自私挖矿的攻击效益。先后尝试了一共10，20，30，40，50个节点的情况，并分别尝试了10%，20%，30%，40%为恶意节点的情况，统计自私挖矿的收益。

自私挖矿攻击是指在正常节点参与区块链的共识的情况下，自己另外开启一条区块链进行不同步地挖矿。其攻击结果鉴定如下：

自私挖矿攻击



竞争情况	描述	收益结果	发生概率
1	恶意矿工在诚实节点出块前生成一个新块	2	a
2	恶意节点在诚实新块之前没有生成一个区块但是竞争成功	1	(1-a)/2
3	恶意节点在诚实新块之前没有生成一个区块但是竞争失败	0	(1-a)/2

自私挖矿的攻击结果如下，表格内为自私挖矿的收益：

总节点数\恶意节点比例	10%	20%	30%	40%
10	2	3	5	6
20	3	4	7	10
30	5	7	13	15
40	6	10	15	19
50	7	197	20	26

可以看出，随着恶意节点增加，自私挖矿的收益增加，攻击收益随着节点总数增加而增加。

总结

在本次大作业中，遇到一些问题，也有一些收获。

遇到的问题

- **对python中变量的命名与存储机制不够了解**

python作为一种动态语言，它的变量赋值是将对象对应的地址赋给变量，其实有点像C++中引用的概念。如果直接在对象间赋值，会让两个变量指向同一个结构体，从而导致一次同步后，所有的挖矿都在同一条区块链上进行，从而区块链长度上限变成了轮次的五倍。

- **对信号量机制不熟悉**

上学期的《操作系统》课学习了用信号量来进行互斥和同步。在同步问题中，分为全同步和半同步两种模式。考虑到每个线程执行时间的不确定性，如果采用全同步，则依然不能做到同时开始。所以应采用半同步，由主线程统一释放信号量。

反思

- **掌握了vscode中python调试器的使用方法**

python作为动态语言，常见的调试方法就是通过print输出来调试，但不太适合多线程的情况。常见的调试器由pydbg这样的库，像pycharm，vscode也集成了python调试器。加上现在正在学习逆向工程，又重温了代码调试的方法，并通过分析变量地址检查出了bug。

- **作业进度安排不合适**

本次大作业虽然开始得比较早，但在一开始遇到问题以后一直在debug，加上中途有其他的事情，所以debug进展缓慢，本来想利用graphviz库（一个适用于画节点图的库）做区块链可视化的工作也来不及了。以后要提高自己的debug能力，也要提前安排好进度。

- **可以尝试使用web服务来模拟节点之间的通信**

在github看到了一些区块链仿真的代码，它们都是用python的flask框架，为每个节点开设一个端口，这样更加贴近真实环境，也避免了多线程的麻烦。

附录

本次大作业，涉及到的参考资料：

- backbone协议：<https://eprint.iacr.org/2014/765>
- 使用 Python 从零开始开发区块链应用程序：<https://www.ibm.com/developerworks/cn/cloud/library/cl-develop-blockchain-app-in-python/index.html>
- <https://bigishdata.com/2017/10/17/write-your-own-blockchain-part-1-creating-storing-syncing-displaying-mining-and-proving-work/>

本次大作业的github地址: <https://github.com/cascades-sjtu/myblockchain>