预挖总量19.5亿:18亿作为冰河基金封存、1.5亿流通。0.5亿Pow奖励，初期每月50万，每年600万，每四年奖励减半。总量控制在20亿枚

一、预挖19.5亿：

方案：在第二块时一次奖励19.5亿到矿工地址。

二、奖励

0.5亿pow奖励，初期每月50万，每年600万：

每年秒数为365天×24小时×60分钟×60秒 = 31536000秒

每秒钟的奖励为6000000/31536000个。

假设平均15秒出一个块，那么一个块的奖励为15\*6000000/31536000 = 2.853881279 个。

以上为不考虑叔块的情况。

但由于存在叔块奖励和保留叔块奖励，每个叔块奖励为主块奖励的7/8，保留叔块奖励为1 /*32，按以太坊来算，每天大约会产生叔块500个，一年则为500\*365=182500个叔块，按以太坊平均15秒出一个主块来算，一年大约出块31536000*/15=2102400个主块

假设每个主块奖励为x，则2102400x + 182500\*7/8x = 6000000，求得x = 2.652417292

加上保留叔块也有奖励，则每个主块奖励约为2.5个。

奖励部分代码：SealEngine.cpp的74行

原为：

u256 SealEngineBase::blockReward(u256 **const**& \_blockNumber) **const**

{

EVMSchedule **const**& schedule{**evmSchedule**(\_blockNumber)};

**return chainParams**().**blockReward**(schedule);

}

修改后为：

u256 SealEngineBase::blockReward(u256 **const**& \_blockNumber) **const**

{

**if**(**chainParams**().morningbookType){

**return chainParams**().**customBlockReward**(\_blockNumber);

}

EVMSchedule **const**& schedule{**evmSchedule**(\_blockNumber)};

**return chainParams**().**blockReward**(schedule);

}

ChainOperationParams.cpp的87行：

u256 ChainOperationParams::customBlockReward(u256 **const**& \_blockNumber) **const**

{

u256 perYear = 2120000; // 每年块数，按15秒一个的平均数算出的

boost::optional<u256> baseReward = {5 \* **ether**};

boost::optional<u256> subsidyReward = {1950000000 \* **ether**};

**if**(\_blockNumber == 2){ // 第2个块的矿工奖励为19.5亿

**return** \*subsidyReward;

}

**unsigned** periodCount = boost::**lexical\_cast**<**unsigned**>(((\_blockNumber) / (4 \* perYear)).**str**());

**if**(periodCount >= 2){ //2个周期8年后达到约0.5亿则不再有奖励

**return** 0;

}

**return** u256(\*baseReward / ((1 << (periodCount + 1)))); //每隔4年减半

}

三、难度

以太坊难度 = 上个块难度值+难度调整+难度炸弹

计算一个区块的难度时，需要以下输入：

parent\_timestamp:上一个区块产生的时间

parent\_diff:上一个区块的难度

block\_timestamp:当前区块产生的时间

block\_number:当前区块号

难度调整 = parent\_diff // 2048 \* MAX(1 - (block\_timestamp - parent\_timestamp) // 10, -99)

难度炸弹 = INT(2\*\*((block\_number // 100000) – 2))

block\_diff = parent\_diff + 难度调整 + 难度炸弹

另外，区块难度不能低于以太坊的创世区块，创世区块的难度为131072，这是以太坊难度的下限。

上面涉及到几种运算符：

整数除法，符号//

计算a//b时，先计算a/b，然后取不大于a/b的最大整数。

例如：

-11.3 // 5 = -3

11.3 // 5 = 2

取整，符号INT

计算INT(a)时，仅仅取整数部分，丢弃小数。

例如：

INT(3.7) = 3

INT(-3.7) = -3

最大值，符号MAX

计算MAX(a,b)时，结果为a和b中较大的那一个。

例如：

MAX(-1,0) = 0

MAX(7,10) = 10

## **以太坊难度的特点**

以太坊的区块难度以单个区块为单位进行调整，可以非常迅速的适应算力的变化，正是这种机制，使以太坊在硬分叉出以太坊经典(ETC)以后没有出现比特币分叉出比特币现金(BCC)后的算力“暴击”问题。同时，以太坊的新区块难度在老区块的基础上有限调整的机制也使区块难度不会出现非常大的跳变

## **关于难度炸弹**

难度炸弹每100000个区块就会翻倍，目前只有1T左右。但是由于是指数级递增，指数递增是很迅速的，用炸弹来形容倒是非常贴切。到5400000区块时，难度将达到4500T，会使挖矿变得极为困难，甚至不能收回电费成本。

冰河这里不需要难度炸弹，去掉难度炸弹。

难度相关代码：Ethash.cpp的193行calculateDifficulty

。

在其中插入了以下代码：

**if**(**chainParams**().morningbookType){

o = **max**<bigint>(minimumDifficulty, o);

**return** u256(**min**<bigint>(o, std::numeric\_limits<u256>::**max**()));

}

未把难度炸弹加入就返回难度值，这样区块难度=上个区块难度+难度调整。

四、测试

命令：

1、启动节点

./code/cpp-ethereum/build/eth/eth --mining on --db-path test/privateNet/ --verbosity 9 --address 0xe11d362750ba290157f295582bf6d213459eec43 --config test/config.json

2、启动ethconsole

ethconsole test/privateNet/geth.ipc

进入ethconsole命令行后

a、获取当前区块数

web3.eth.getBlockNumber(function(err, number)

{

if (err)

console.error("Could not connect to node. Please start an Ethereum node first.");

else

{

console.log("Connection successful.");

console.log("Current block number: " + number);

}

});

b、获取指定高度区块信息

web3.eth.getBlock(2, function(err, info)

{

if (err)

console.error("Could not getBlock.");

else

{

console.log(info);

}

});

c、获取指定账户在指定高度时的余额，查看第二个区块的余额即可看到19.5亿的预挖

web3.eth.getBalance("0xe11d362750ba290157f295582bf6d213459eec43", 2, function(err, info)

{

if (err)

console.error("Could not getBalance.");

else

console.log(info);

}

);

附：config.json文件

{

"sealEngine": "Ethash",

"params": {

"accountStartNonce": "0x00",

"homesteadForkBlock": "0x118c30",

"daoHardforkBlock": "0x1d4c00",

"EIP150ForkBlock": "0x259518",

"EIP158ForkBlock": "0x28d138",

"byzantiumForkBlock": "0x42ae50",

"constantinopleForkBlock": "0x500000",

"networkID" : "0x01",

"chainID": "0x01",

"maximumExtraDataSize": "0x20",

"tieBreakingGas": false,

"minGasLimit": "0x1388",

"maxGasLimit": "7fffffffffffffff",

"gasLimitBoundDivisor": "0x0400",

"minimumDifficulty": "0x020000",

"difficultyBoundDivisor": "0x0800",

"durationLimit": "0x0d",

"blockReward": "0x4563918244F40000",

"allowFutureBlocks":true

},

"genesis": {

"nonce": "0x0000000000000042",

"difficulty": "0x40000",

"mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",

"author": "0x0000000000000000000000000000000000000000",

"timestamp": "0x00",

"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",

"extraData": "0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",

"gasLimit": "0x1388"

},

"accounts": {

"0000000000000000000000000000000000000001": { "precompiled": { "name": "ecrecover", "linear": { "base": 3000, "word": 0 } } },

"0000000000000000000000000000000000000002": { "precompiled": { "name": "sha256", "linear": { "base": 60, "word": 12 } } },

"0000000000000000000000000000000000000003": { "precompiled": { "name": "ripemd160", "linear": { "base": 600, "word": 120 } } },

"0000000000000000000000000000000000000004": { "precompiled": { "name": "identity", "linear": { "base": 15, "word": 3 } } },

"0000000000000000000000000000000000000005": { "precompiled": { "name": "modexp", "startingBlock" : "0x2dc6c0" } },

"0000000000000000000000000000000000000006": { "precompiled": { "name": "alt\_bn128\_G1\_add", "startingBlock" : "0x2dc6c0", "linear": { "base": 500, "word": 0 } } },

"0000000000000000000000000000000000000007": { "precompiled": { "name": "alt\_bn128\_G1\_mul", "startingBlock" : "0x2dc6c0", "linear": { "base": 40000, "word": 0 } } },

"0000000000000000000000000000000000000008": { "precompiled": { "name": "alt\_bn128\_pairing\_product", "startingBlock" : "0x2dc6c0" } }

}

}