Filecoin-Gas calculation

gas (https://learnblockchain.cn/tags/gas) FileCoin (https://learnblockchain.cn/tags/FileCoin)

> Filecoin's Gas model introduces BaseFee to adjust transaction congestion. BaseFee, in the case of block congestion or insufficient block transactions, will be adjusted accordingly at 12.5%. The fee calculation formula for each transaction: (Gas Premium + Base Fee) * Gas Limit. Among them, the BaseFee part will be burned, and Gas Premium will be used as the miner's handling fee. Special attention should be paid to the GasLimit not to be arbitrarily set, the excess Gas Limit will be burned.

Filecoin started Space Race. The strength of the miners are beginning to show. The official code also has many problems. In a few days, the version was upgraded from 0.5.1 to 0.5.6. I haven't looked at the logic related to the go language for a long time. I recently looked at the logic related to Filecoin's Gas calculation. Share it. This article analyzes the logic of 0.5.6, the last submission information of the Lotus code is as follows:

```
commit 606a58bc6bc035ec0b90c6b50488e29e90f4238f
Author: Aayush Rajasekaran
Date:   Sat Aug 29 00:56:24 2020 -0400

 Lotus version 0.5.6
```

The CHANGELOG of the Lotus code clearly records the changes in the Gas fee model: from the original limit/price to limit/premium/feecap. The new Gas model refers to EIP-1559: Sending transactions, the transaction fee does not exceed the "feecap *limit*". *The transaction fee earned by miners is the* "*premium* limit". Simply put, the feecap *limit is the upper limit of the gas fee, and the fee miners can earn is the premium* limit. (Feecap-premium)*limit Gas fee will be burned.

How is feecap set up? Is the limit set as large as possible?

The related logic of Gas cost calculation is implemented in the GasEstimateMessageGas function in node/impl/full/gas.go. Next, we will introduce Base/Limit/Premium/FeeCap in detail.

1. Base Fee

Each block will set a baseFee. All transactions in this block need to burn the corresponding baseFee. Note that baseFee, although the name looks fee, is actually price, and the specific burned fee is baseFee*limit. The settings related to baseFee are defined in build/params_shared_vals.go:

```
1    const BlockGasLimit = 10_000_000_000
2    const BlockGasTarget = BlockGasLimit / 2
3    const BaseFeeMaxChangeDenom = 8 // 12.5%
4    const InitialBaseFee = 100e6
5    const MinimumBaseFee = 100
6    const PackingEfficiencyNum = 4
7    const PackingEfficiencyDenom = 5
```

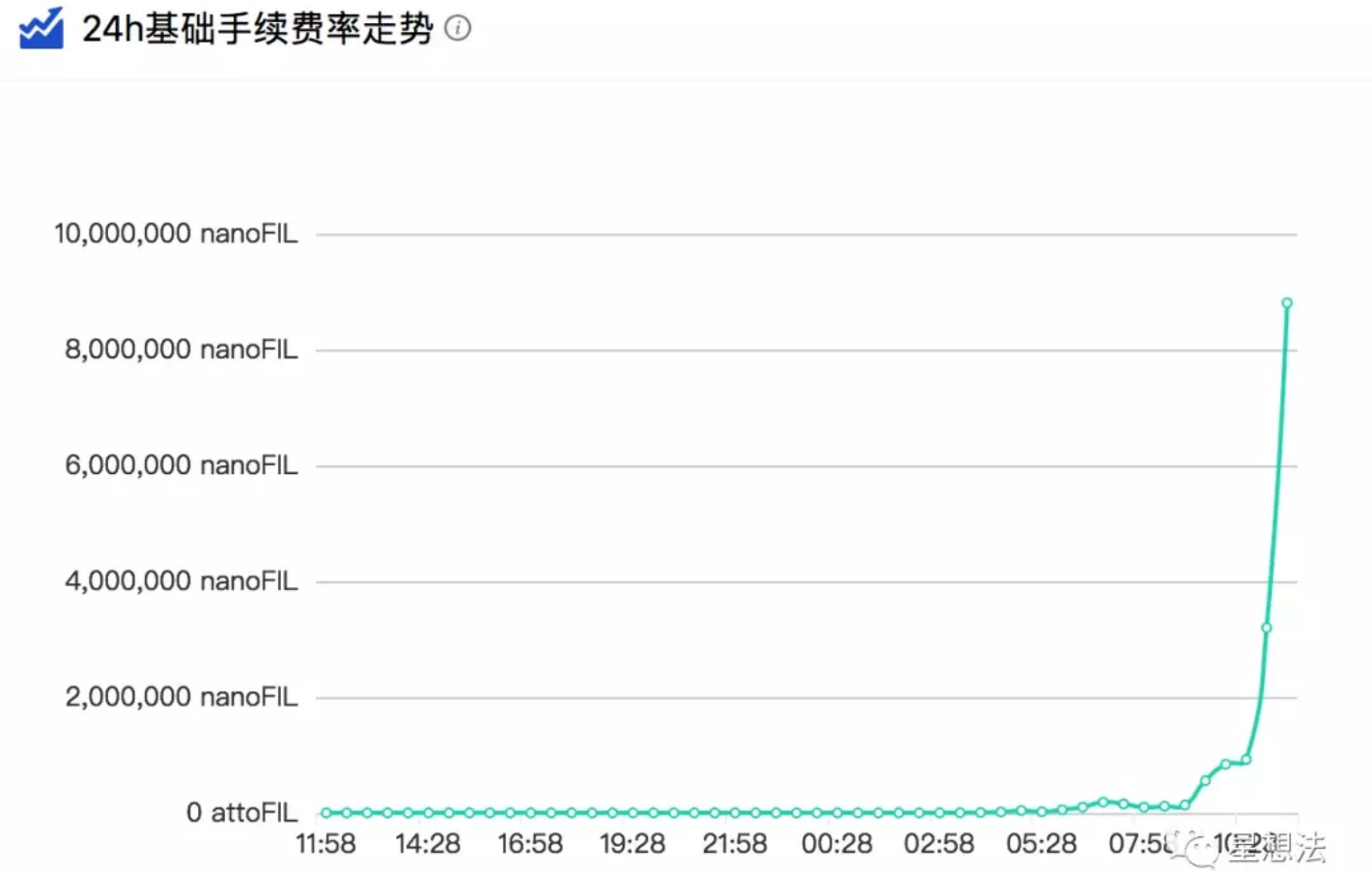
In the initial block, baseFee is set to InitialBaseFee (10^8). When generating the next block from the current block, it needs to be determined according to the total limit of the current block. For the specific logic, please check the ComputeBaseFee and computeNextBaseFee functions of chain/store/basefee.go.

- Minimum baseFee-MinimumBaseFee (100)
- Block Gas Limit-The sum of the Gas Limit of all transactions in the block. When calculating baseFee, it is 10% off (PackingEfficiencyNum/PackingEfficiencyDenom)
- The "exceeding" part of the block Gas Limit-each block has the target size of the Gas Limit-BlockGasTarget. The part that exceeds the BlockGasTarget is regarded as the excess part. Note that the excess can be positive or negative.
- Updated baseFee-the baseFee of the next block, the excess 12.5% (BaseFeeMaxChangeDenom) is added to the baseFee of the current block. The relevant calculation logic is as follows:
-

```
1    change := big.Mul(baseFee, big.NewInt(delta))
2    change = big.Div(change, big.NewInt(build.BlockGasTarget))
3    change = big.Div(change, big.NewInt(build.BaseFeeMaxChangeDenom))
```

Simply put, if the Gas Limit consumption in the current block exceeds the BlockGasTarget, the base fee increases by 12.5% of the excess. Under this logic, you will find that when there are many transactions, the base fee will increase and decrease rapidly.

The latest 24-hour base fee can be viewed on the Feihu browser (https://filfox.info/zh (https://filfox.info/zh)):



2. GasLimit

Gas Limit refers to the amount of gas that a transaction is willing to pay for the execution of the transaction. The Gas Limit consumed by a transaction is almost fixed. Please refer to the GasEstimateGasLimit function for the calculation process. Simply put, when a transaction needs to obtain the Gas Limit, at the current block height, "execute" the transaction:

```
1   res, err := a.Stmgr.CallWithGas(ctx, &msg, priorMsgs, ts)
```

CallWithGas is just to get the gas consumed by the execution process, and does not really change the current state.

3. GasPremium

Gas Premium refers to a transaction that is willing to pay the "oil price" for the execution of the transaction. The GAS fee is the result of multiplying the amount of oil by the price of oil. The amount of oil is related to the transaction itself and is almost

fixed. Obviously, with high oil prices and high GAS fees, the income of miners will be high, and they are more willing to prioritize the packaging of transactions. From the perspective of the transaction sender, the lower the oil price, the better. The Lotus code gives a method to calculate Gas Premium. Please check the GasEstimateGasPremium function, which is divided into several steps:

- View all transactions in the previous block (4 = 2*2) and sort them in order of Gas Premium from high to low
- Calculate the "average" Gas Premium for all transactions. The average means to find out the Gas Premium that consumes half of the fuel:

```
1        at := build.BlockGasTarget * int64(blocks) / 2
2        prev1, prev2 := big.Zero(), big.Zero()
3        for _, price := range prices {
4        prev1, prev2 = price.price, prev1
5        at -= price.limit
6        if at > 0 {
7        continue
8        }
9        }
```

- Plus five-thousandths of randomness

```
1        // mean 1, stddev 0.005 => 95% within +-1%
2        noise := 1 + rand.NormFloat64()*0.005
3        premium = types.BigMul(premium, types.NewInt(uint64(noise*(1
4        premium = types.BigDiv(premium, types.NewInt(1
```

4. GasFeeCap

In addition to Gas Premium, the transaction also needs to pay Base Fee. In other words, under normal circumstances, the transaction fee that needs to be paid is: (Gas Premium + Base Fee) * Gas Limit. The problem is that the Base Fee is variable, it may be too large, and the transaction sender is unwilling to pay. Gas Fee Cap (upper limit), is to set the upper limit of the payment fee. For related calculation logic, please see the GasEstimateFeeCap function.

- It is very likely that the transaction will not be packaged immediately in the next block. It is necessary to consider the change of Base Fee when the next multiple blocks (10) are packaged:

```
parentBaseFee := ts.Blocks()[0].ParentBaseFee
increaseFactor := math.Pow(1.+1./float64(build.BaseFeeMaxChangeDenom), float64(maxqueueblks))

feeInFuture := types.BigMul(parentBaseFee, types.NewInt(uint64(increaseFactor*(1
feeInFuture = types.BigDiv(feeInFuture, types.NewInt(1
```

```
每个区块Base Fee按照12.5%的涨幅计算。
```

- One percent of the current account balance is used as the upper limit of payment fees:
-

```
maxAccepted := types.BigDiv(act.Balance, types.NewInt(MaxSpendOnFeeDenom))
```

- The minimum value in the above two cases is regarded as the Gas Fee Cap

5. GasLimit set penalty

As we all know, the Gas Limit in Ethereum can be set very large. Under normal circumstances, the excess gas fee will be refunded in full. It is important to note that Filecoin is not exactly like this. Because Gas Limit participates in the calculation of Base Fee and Gas Premium, the actual Gas Limit is very important. If an unreasonable gas limit is set for a transaction, Filecoin adopts a penalty mechanism. The penalty gas cost is also burned. The calculation logic is based on the ComputeGasOverestimationBurn function.

- Allow a certain amount of error to calculate the excess part of the Gas limit

```
const (
gasOveruseNum   = 11
gasOveruseDenom = 10
)

over := gasLimit - (gasOveruseNum*gasUsed)/gasOveruseDenom
```

```
Gas消耗的1.1倍以内认为是合理设置。
```

```
if over > gasUsed {
 over = gasUsed
}
```

The upper limit of the excess is the amount of Gas used.

- Determine the amount of fuel penalized according to the proportion of over:

```
gasToBurn := big.NewInt(gasLimit - gasUsed)
gasToBurn = big.Mul(gasToBurn, big.NewInt(over))
gasToBurn = big.Div(gasToBurn, big.NewInt(gasUsed))
```

Simply put, the penalty is the ratio of over/gasUsed. If over exceeds gasUsed, it is all penalty lights.

to sum up:

Filecoin's Gas model introduces BaseFee to adjust transaction congestion. BaseFee, in the case of block congestion or insufficient block transactions, will be adjusted accordingly at 12.5%. The fee calculation formula for each transaction: (Gas Premium + Base Fee) * Gas Limit. Among them, the BaseFee part will be burned, and Gas Premium will be used as the miner's handling fee. Special attention should be paid to the GasLimit not to be arbitrarily set, the excess Gas Limit will be burned.

Published on 2020-09-04 09:25 Reading (2052) Credits (67)
Category: FileCoin (https://learnblockchain.cn/categories/FileCoin)

Articles you may be interested in

Why is NFT different? How does Filecoin's distributed storage solution empower NFT? (https://learnblockchain.cn/article/2495) 21 views