# 区块链记录

## 1:难度计算

diff(genesis) = 2^32

diff(block) = diff.block.parent +floor(diff.block.parent / 1024) \*

1 ifblock.timestamp - block.parent.timestamp < 9 else

-1 ifblock.timestamp - block.parent.timestamp >= 9

难度更新规则的设计目标如下：

• 快速更新：区块间的时间应该随着hash算力的增减而快速调整；

• 低波动性：如果Hash算力恒定，那么难度不应剧烈波动；

• 简单：算法的实现应相对简单；

• 低内存：算法不应依赖于过多的历史区块，要尽可能少的使用”内存变量“。假设有最新的十个区块，将存储在这十个区块头部的内存变量相加，这些区块都可用于算法的计算；

• 非开发性：算法不应让矿工有过多篡改时间戳或者矿池、反复添加或删除算力的能力，以使他们的收益最大化。

我们当前的算法在低波动性和非开发性上并不理想，至少我们计划切换时间戳比较父区块和祖父区块，所以矿工只有在连续挖2个区块时，才有动力去修改时间戳。另一个更强大的模拟公式：https://github.com/ethereum/economic-modeling/blob/master/diffadjust/blkdiff.py

## 2：GAS和费用

比特币中所有交易大体相同，因此它们的网络成本可以建成一个模型。以太坊中的交易要更复杂，所以交易费用需要考虑到账户的许多方面，包括宽带费用，存储费用和计算费用。尤其重要的是，以太坊编程语言是图灵完备的，所以交易会使用任意数量的宽带、存储和计算成本。这就可能会导致在计算成本过程中，突遭停电而计算被迫中止。

以太坊交易费用的基本机制如下：

• 每笔交易必须指明一定数量的gas（即指定startgas的值），以及支付每单元gas所需费用（即gasprice），在交易执行开始时，startgas \* gasprice 价值的以太币会从发送者账户中扣除；

• 交易执行期间的所有操作，包括读写数据库、发送消息以及每一步的计算都会消耗一定数量的gas；

• 如果交易执行完毕，消耗的gas值小于指定的限制值，则交易执行正常，并将剩余的gas值赋予变量gas\_rem ; 在交易完成后，发送者会收到返回的gas\_rem \* gasprice 价值的以太币，而给矿工的奖励是（startgas -gas\_rem）\* gasprice价值的以太币；

• 如果交易执行中，gas消耗殆尽，则所有的执行恢复原样，但交易仍然有效，只是交易的唯一结果是将startgas \* gasprice 价值的以太币支付给矿工，其他不变；

• 当一个合约发送消息给另一个合约，可以对这个消息引起的子执行设置一个gas限制。如果子执行耗尽了gas，则子执行恢复原样，但gas仍然消耗。

上述提到的几点都是必须满足的，例如：

• 如果交易没有指定gas限制，那么恶意用户就会发送一个有数十亿步循环的交易。没有人能够处理这样的交易，因为处理这样的交易花的时间可能很长很长，从而无法预先告知网络上的矿工，这会导致拒绝服务的漏洞产生。

• 替代严格的gas计数、时间限制等机制的方案不起作用，因为它们太主观了；

• startgas \* gasprice 的整个值，在开始时就应该设置好，这样不至于在交易执行中因gas不够而造成交易终止。注意，仅仅检查账户余额是不够的，因为账户可以在其他地方发送余额。

• 如果在gas不够的情况下，交易执行没有恢复操作（回滚），合约必须采用强有力的安全措施来防止合约发生变化。

• 如果子限制不存在，则恶意账户会通过与其他账户达成协议来对它们采取拒绝服务攻击。在计算开始时插入一个大循环，那么发送消息给受害合约或者受害合约的任何补救尝试，都会使整个交易死锁。

• 要求交易发送者而不是合约来支付gas，这样大大增加了开发人员的可操作性。以太坊早期的版本是由合约来支付gas的，这导致了一个相当严重的问题：每个合约必须实现“守护”代码，确保每个传入的消息有足够的以太币供其消耗。

gas消耗计算有以下特点：

• 对于任何交易，都将收取21000gas的基本费用。这些费用可用于支付运行椭圆曲线算法所需的费用。该算法旨在从签名中恢复发送者的地址以及存储交易所花费的硬盘和带宽空间。

• 交易可以包括无限量的“数据”。虚拟机中的某些操作码，可以让合约允许交易对这些数据的访问。数据的固定消耗计算是：每个零字节4gas，非零字节68gas。这个公式的产生是因为合约中大部分的交易数据由一些列的32字节的参数组成，其中多数参数具有许多前导零字节。该结构看起来似乎效率不高，但由于压缩算法的存在，实际上还是很有效率的。我们希望此结构能够代替其他更复杂的机制：这些机制根据预期字节数严格包装参数，从而导致编译阶段复杂性大增。这是三明治复杂模型的一个例外，但由于成本效益比，这也是合理的模型。

• 用于设置账户存储器的操作码SSTORE的消耗是：

1.将零值改为非零值时，消耗20000gas；

2.将零值变成零值，或非零值变非零值，消耗5000gas；

3.将非零值变成零值，消耗5000gas，加上交易执行成功后退回的20000gas。

退款金额上限是交易消耗gas总额的50%。这样设置会激励人们清除存储器。我们注意到，正因为缺乏这样的激励，许多合约造成了存储空间没有被有效使用，从而导致了存储快速膨胀；为存储收取费用提供了很多好处，同时不会失去合约一旦确立就可以永久存在的保证。延迟退款机制是必要的，因为可以阻止拒绝服务攻击。攻击者发送一笔含有少量gas的交易，循环清除大量的存储，直到用光gas，这样消耗了大量的验证算力，但实际并没有真正清除存储或消耗大量gas。50%的上限的是为了确保获得了一定交易gas的旷工依然能够确定执行交易的计算时间的上限。

• 合约提供的消息的数据是没有成本的。因为在消息调用期间不需要实质复制任何数据，调用数据可以简单地视为指向父合约内存的指针，该指针在子进程执行时不会改变。

• 内存是一个可以无限扩展的数组，然而，每扩展32字节的内存就会消耗1gas的成本，不足32字节以32字节计。

• 某些操作码的计算时间极度依赖参数，gas开销计算是动态变化的。例如，EXP的的开销是指数级别的；复制操作码（如：CALLDATACOPY,CODECOPY, EXTCODECOPY）的开销是1+1(每复制32字节)。内存扩展的开销不包含在这里，因为它触发了二次攻击。

• 如果值不是零，操作码CALL会额外消耗9000gas。这是因为任何值传输都会引起归档节点的历史存储显著增大。请注意，实际消耗是6700，在此基础上，我们强制增加了一个自动给予接受者的gas值，这个值最小2300。这样做是为了让接受交易的钱包至少有足够的gas来记录交易。

gas机制的另一个重要部分是gas价格本身体现出的经济学原理。比特币中，默认的方法是采取纯粹自愿的收费方式，矿工扮演守门人的角色并且动态设置收费的最小值。以太坊中允许交易发送者设置任意数目的gas。这种方式在比特币社区非常受欢迎，因为它是“市场经济”的体现：允许矿工和交易者之间依据供需关系来决定价格。然而，这种方式的问题是，交易处理并不遵循市场原则。尽管可以将交易处理看作是矿工向发送者提供的服务（这听起来很直观），但实际上矿工所处理的每个交易都必须由网络中的每个节点处理，所以交易处理的大部分成本都由第三方机构承担，而不是决定是否处理它的矿工。

当前，因为缺乏矿工在实际中的行为的明确信息，所以我们将采取一个非常简单公平的方法：投票系统，来设定gas限定值。矿工有权将当前区块的gas限定值设定在最后区块的gas限定值的0.0975% (1/1024)内。所以最终的gas限定值应该是矿工们设置的中间值。我们希望将来能够采用软分叉的方法来使用更加精确的算法。