HyperAGI GPU 核心激励算法

本文档系统解释$HYPT 代币的挖矿奖励和 Gas 费算法，这些算法形成从代币发行和消费的闭环，是经济模型的基础算法，可以公开。

所有 HyperAGI GPU 节点将每 6.4 分钟广播一次当前状态协议，并在每个 Epoch 结束随机从上一个 Epoch 活跃的节点中选择一个来进行代币奖励。奖励包括基本奖励和 GPU 节点费：

基本奖励： 基本奖励 B 等于 E 除以难度系数，将在渲染/流或训练/推理期间的每个 Epoch 进行发送，公式如下：

$$B = \left(\frac{E}{\text{Difficulty}} + E\right) \Big/ 2$$

E 是在每个 Epoch 中发送给 GPU 节点的最大代币奖励。第一年用于 GPU 节点激励的代币总额为保留给 GPU 挖矿的 10%，每四年减半，公式如下

$$E = \frac{\text{MAX} * 57\% * 10\%}{365 * 225}$$

获得激励代币的方式是完成通过 GPU 进行 3D、AI 等计算任务。

“挖矿”难度系数 Difficulty 与当前用户 GPU 真实使用率成反比，最小值为 1，即可以按照当前最大出币量获得代币。

$$\text{Difficulty} = \frac{\text{Number of GPU}}{\text{Number of GPU in working}}$$

上述基本奖励公式确保在任何情况下全网节点都可以按照当前繁忙程度均衡得到代币激励，同时结合二级市场币价，形成开关机价格，从而形成可持续的激励机制，激励强度与实际业务使用率形成正相关，避免过高的通胀与泡沫。

工作状态意味着 GVM 正在工作以执行由超级图定义的任务。在该平台的早期版本中，它也提到了渲染和流媒体，人工智能培训/推理将在未来的版本中得到支持。基本奖励将在 12 个月内线性发放，10%将被保留作为保证金。渲染，AI 训练或者网络连接质量差的罚款将从保证费中扣除。在将来 PoUW 版本中这些 GPU 节点将自行证明（或验证）计算结果并获取代币激励，目前还是以中心化的数据采集方式将节点的工作状态喂进智能合约。所以需要中心化方式定期采集节点的工作状态以计算和更新难度系数。某时刻节点的工作状态是指该时刻节点（单 GPU）在进行渲染，AI 训练或推理等高强度计算。可以通过轮询或节点上报状态的方式采集节点工作状态，比如采集 GPU 的使用率来表示其工作状态，GPU 使用率低于 40%则认为是处于没有工作的状态。但是过高的采样

频率导致额外的计算开销以及数据开销，特别是还需要将状态数据通过预言机进行上链操作，当网络中有大量节点时这些开销将变得巨大而不可接受。

在当前版本中使用奈奎斯特采样方式采集节点的工作状态。即以二分之一 Epoch 为周期采集每个节点的工作状态， 按照奈奎斯特原理，这是保证准确获取节点工作状态的最小采集（采样）频率。

解释如下：
假设我们选取 GPU 使用率作为节点的状态值，GPU 使用率是一个 0-100%的连续变量，每个 Epoch 进行采样的数值曲线见图 1。

图 1 节点 GPU 使用率，T2-T1 = 1 Epoch

节点状态值是 0=没有工作或 1=在工作中（比如将 GPU 使用率大于 50%设为在工作状态），如果将状态值用笛卡尔坐标表示为一个函数，其中横轴是时间，纵轴为状态（0 或 1），可以看到这就是一个方波函数，**准确反应节点工作状态就等价于准确采集这个方波的波形，**参见图 2。

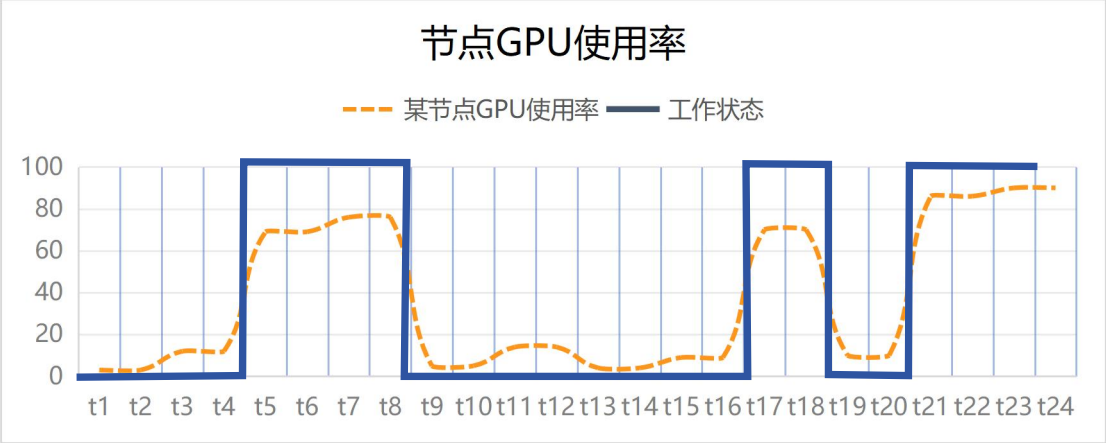

图 2 节点工作状态方波，t2-t1 = 1/2 Epoch

设连续信号为χ(t) ，周期为 Ts 的冲激序列为：

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s)$$

采样后的信号为χs (nTs)，由 χs (nTs)最后生成的离散时间序列为 χ(N)。采样过程中，χs (nTs)可看作χ(t) 与周期性冲激序列 P(t)相乘的结果。

以采样频率 fs (对应的采样周期为 Ts)对连续信号 χ(t)进行采样，可以得到采样序列χ(nTs)，n=… ，一 1，0，1，… ，若χ(t)是带限实信号，其频谱最高频率为 Fc，则当满足条件 fs≥2fc 时，可由采样序列χ(nTs)完全恢复出连续信号χ(t)，即χ(nTs)保留了χ(t)的全部信息，证明略。

基本思路

将倍频(1/2 Epoch，每 5 分钟)采样的数据上采样为低频(1 Epoch，每 10 分钟)的数据，用于准确表示全网节点的状态。由于两个采样频率存在倍数关系，所以高频采样数据中连续三次为高才判定为低频中的高，连续三次低才判定为低频中的低， 连续三次异常才判定为登出。只需要将每 Epoch 的所有节点状态传入智能合约就可以完成难度系数和 Gas 价格的计算。并且要考虑将数据进行压缩减少预言机和合约执行成本。

参考实现方法：

节点加入网络后得到一个唯一的无符号整数编号 uint256 NodeID，智能合约中注册时传入该节点 NodeID 与节点的挖矿钱包地址进行绑定（可能还需要进行网卡 MAC、GPU 序列号和 IP 地址绑定以确保该节点的唯一性和可识别性，智能合约可以提供更新函数，节点所有者可以用私钥进行节点信息更新，比如更换 IP 或显卡等配件）；当前节点工作状态 uint8 NodeStatusHalfEpoch[0,m]，当前只有 00 登出，01_b 异常，10_b 闲置和 11_b 活跃四种状态(由两位二进制数表示)。登出状态的节点不计入总节点中，可能已经关机退出，后面会详细讨论。

在链下合约中分别定义按照 Epoch 更新的节点状态码数组 uint256 NodeStatusEpoch[groupNumber]，由于每两位二进制数可以表示节点的状态（00_b,01_b,10_b ,11_b），所以一个 uint256 状态码可以表示 128 台节点的工作状态，称为一组。NodeStatusEpoch[0]表示第 1 组 128 台节点的状态码, NodeStatusEpoch[1]表示第 2 组 128 台节点的状态码，以此类推，一百个状态码可以表示 128x100=12800 台节点的状态。

假设以下掩码低位在右边，即第 1 和第 2 位是 10_b，表示 NodeID 为 0 的节点此刻状态=2 闲置，Node1 状态= 0 登出，NodeID 为 2 的节点状态=3 闲置。

Uint256 NodeStatusEpoch[0] =

Node127																		Node2		Node1		Node0	
0	0	0	0	0	0	0	0	0	0	.	.	1	1	0	0	0	0	1	1	0	0	1	0

一个 NodeStatusEpoch 例子

Uint256 NodeStatusEpoch[1] =

Node 255																		Node 131		Node 130		Node 129	
0	0	0	0	0	0	0	0	0	0	.	.	1	1	0	0	0	0	1	1	0	0	1	0

接下来我们需要获得节点的状态。NodeStatusHalfEpoch[0,n]是一个数组，索引是 NodeID。比如 NodeStatusHalfEpoch[0,2]表示 2 号节点的状态。现在以 1/2Epoch 为周期进行链外采样，比如 GPU 使用率大于一定数值（40%）就将 NodeStatusHalfEpoch[0,n]设为 1 否则为 0，遍历所有节点我们可以得到每个节点的状态 NodeStatusHalfEpoch[0,n]。最大间隔为 1/2 Epoch，也就是要在这个周期内尽可能地将完成所有节点的状态采集，如果在 1/2 Epoch 周期内没有采集到 n 节点的状态，就将 NodeStatusHalfEpoch[0,n]状态设为 2 表示超时。采集状态的代码要支持多线程接受来自大量节点的网络数据。

以下是伪代码，仅作参考：

更新节点状态

```
//链下伪代码

n = NodeID//最大节点数

//声明多维数组分别代表每 1/2Epoch 采样的当前状态，上一次和再上一次的节点状态，下标分别为[0,n], [1,n]，[2,n]
Uint256 NodeStatusHalfEpoch[2,n];

//将当前状态数组保持在上一次状态数组中，也许要保存多次状态数据，比如 7 天的状态，225x7x2 = 1575 个 Epoch，也就是 1575x2 = 3150 组状态(倍频)。这些数据只在链下保存。

while n then
{
    //移动上两次数据，为新的数据腾出空间，新采集的数据将存入
    //NodeStatusHalfEpoch[0,n]
    NodeStatusHalfEpoch[2,n] = NodeStatusHalfEpoch[1,n];
    NodeStatusHalfEpoch[1,n] = NodeStatusHalfEpoch[0,n];
```

```
    n--;
}
//将所有待采集的节点状态初值设为 00b，即登出状态
n = NodeID//最大节点数
While n then NodeStatusHalfEpoch[0, n--] = 00b；

While (1)
{
    //采样周期开始，1/2 个 Epoch = 5 分钟
    timer = Epoch/2;
    n = NodeID//最大节点数
    While(timer--)
        {
            //多线程异步调用,最大等待时间=Epoch/2
            NodeStatusHalfEpoch[0, n] = getNodeStatus(n);
            If n then n--;//遍历所有节点
        }
}


//汇总三次的状态如同都一致，则更新高频@1 Epoch 的状态
t = 3;
N = NodeID;//最大节点数
While n
{
    //每 5 分钟采样，连续三次登出，判定登出
    if (NodeStatusEpoch[0, n] == 00b) &
      (NodeStatusEpoch[1, n] == 00b) &
    (NodeStatusEpoch[2, n] == 00b) then NodeStatusEpoch[n] = 00b;


    //每 5 分钟采样，连续三次异常，判定登出
    if (NodeStatusEpoch[0, n] == 01b) &
      (NodeStatusEpoch[1, n] == 01b) &
    (NodeStatusEpoch[2, n] == 01b) then NodeStatusEpoch[n] = 00b;

    //每 5 分钟采样，连续三次闲置，判定闲置
    if (NodeStatusEpoch[0, n] == 10b) &
      (NodeStatusEpoch[1, n] == 10b) &
    (NodeStatusEpoch[2, n] == 10b) then NodeStatusEpoch[n] = 10b;

    //每 5 分钟采样，连续三次活跃，判定活跃
    if (NodeStatusEpoch[0, n] == 10b) &
      (NodeStatusEpoch[1, n] == 10b) &
```

```
    (NodeStatusEpoch[2, n] == 10b) then NodeStatusEpoch[n] = 10b;
}

//接下来将所有节点的状态@1 Epoch 汇总到状态码中，涉及比较多的二进制计算和位操作，可以回顾具体编程语言的用法
//已经注册最大节点数
n = NodeID
//节点分组数，128 台一组占用一个 uint256 数，取模加一
groupNumber =   Mod(NodeID/128) +1
//将当前所有分组节点的状态码设为 0，即默认所有节点处于登出状态
While (groupNumber)
{
    NodeStatusEpoch[groupNumber] = 0x0;
    groupNumber --;
}

While (n)
{
    //用位与操作提取节点状态的最低两位，因为登出，异常，闲置，工作四种状态最
    //少可用两位二进制数据表示
    Uint8 s = NodeStatusHalfEpoch[0, n] & 0b00000011;


    //将节点 n 的状态左移到状态码相应的位置，比如节点 0，不需要移动，节点 127
    //则需要左移 254 位，即移到最左边（最高位），参见前面关于状态码的说明
    Uint256 S = s << n*2;

    //将已经移到相应位置的状态与状态码进行位或操作，而且每一个节点的状态需要
    叠加到状态码中，位或操作可以不影响原有的数据（状态码其他位置的二进制数值）
    NodeStatusEpoch[0, n]   = NodeStatusEpoch[0, n] | S;
    n--;
}
```

标记僵尸节点

网络中的总节点数等于所有处于超时、闲置和活跃的节点数。因为超时可能是暂时的网络故障，还是可以将该节点算做网络节点。但是通过链外统计发现节点 n 连续 3 个 1/2 Epoch 周期都是超时异常，也就是 15 分钟离线，该 GPU 节点可能已经关机或者持续异常，就将其状态设为 3=登出，并且不计入节点总数中。因为累计可能会有很多节点注册，占用很多节点 ID，协议不能将 NodeID 强行收回，所以要剔除那些僵尸 ID。

现在我们已经通过链外操作，完成了每 1/2 Epoch 周期对所有节点的当前状态 NodeStatusHalfEpoch[0,n]的更新，并通过三次一致性检查更新了 NodeStatusEpoch[n]，通过预言机将 code 读入智能合约，就可以按照每 1Epoch 的频率计算并更新挖矿难度，发放奖励和更新 Gas 价格。

合约中计算节点总数的时候，注意剔除僵尸节点！

注意：节点通过智能合约注册后得到 NodeID（注册新的 NodeID 只能在上次注册的 NodeID 上递增 1）。