Phala Tokenomic Design

Samuel Häfner — samuelhaefner.github.io — samuel.haefner@proton.me
Hang Yin — github.com/h4x3rotab — hangyin@phala.network

August 17, 2023

Phala is a trustless cloud computing solution operating in the Polkadot ecosystem. The network allows end-users to connect via so-called contract clusters to secure workers that utilize specific TEE (Trusted Execution Environment) hardware. TEEs ensure confidentiality, security, and performance.

Phala tokens (PHA) serve a three-fold purpose. First, contract clusters need to stake PHA to obtain computing power from the network. Second, contract clusters charge their users PHA or some contract cluster-specific token for their services. And third, PHA are paid out by the protocol to workes (henceforth called miners) for providing their computing infrastructure.

This paper describes the specific tokenomic design ensuring that all participants have the correct incentives to sustain the network in the long run.

1 The General Idea

The main tokenomic problems in the Phala network are the following:

1. How is computing power allocated among contract clusters and their end-users?
2. How are contract clusters matched to miners?
3. How are miners incentivized to provide their computing power?

1.1 Demand-End Tokenomics

The demand side of Phala consists of individual *developers* who develop *contract clusters*. Contract clusters are the runtime environments for specific contracts, which may be deployed and called by *end-users*.

The network's computing power is allocated through a *two-layered mechanism*. The first layer allocates the network's computing resources among the different contract clusters. Here, Phala employs a *staking-based approach*. On the second layer, the contract clusters allocate their computing power to the jobs that they receive from their end-users. The particular allocation mechanism at this stage is mostly left to the contract clusters.

The Phala network itself may reserve a fraction of the computing power for end-users that want to run general fat contracts directly.

1.2 Cluster-Miner Matching

Once the share of the network's computing power that a contract cluster receives is determined, the contract clusters are matched with specific miners. Here, Phala employs a simple matching algorithm that allows contract clusters to state their preferences over miners and then prioritizes contract clusters with proportionally higher stakes to those with lower stakes.

1.3 Supply-End Tokenomics

Miners are remunerated for the time they devote to the network. We additionally reward miners that are in high demand. This ensures that miners with characteristics that are in high demand have incentives to increase their capacities, which increases the efficiency of the network.

2 Demand-End Tokenomics

Phala employs a two-layered mechanism to allocate computing resources to contract clusters.

The first layer allocates computing resources to the contract clusters in the network through a staking-based approach. End-users and developers are allowed to stake PHA into the contract clusters of their choice. The network then allocates the computing resources among the contract clusters according to their shares of staked PHAs.

The second layer allocates the computing resources of a contract cluster among its end-users. The contract clusters are free to employ their own solutions. Designated contract cluster tokens are possible. In any case, the mechanism on the second layer should ensure that first-layer stakers earn some revenue from their investment.

2.1 First Layer: Allocating Resources among the contract clusters

For the following, let P be the *total computing power* in the network that may be distributed. We obtain the number P by adding up the performance scores across all miners in the network, where the performance score of a miner is proportional to the number of computations that miner can handle per minute.

Time is divided in eras, which are indexed by $t = 1, 2, ...$ Phala network reserves a fraction $0 \leq \alpha \leq 1$ of the total computing power P for end-users to deploy general fat contracts (see next subsection for more details). Hence, if, in era t, there are n_t different contract clusters, and contract cluster $i \in \{1, ..., n_t\}$ has stakes amounting to s_{it}, then the amount P_{it} of computing power reserved for contract cluster i in era t is

$$P_{it} = \frac{s_{it}}{\sum_{j=1}^{n_t} s_{jt}}(1-\alpha)P.$$

The reserved computing power P_i for contract cluster i then translates into a list of specific miners to which contract cluster i can upload its code. The mechanism to do so is described in *Section 3: Cluster-Miner Matching* below.

Staking happens through a dedicated staking module to which all PHA holders can contribute. The module might have a cap on the total stake above which no further stake is accepted. The developer can dynamically adjust this cap.

A contract cluster can remain funded forever, but users can take out their stakes at any time (possibly subject to an unbonding period). This ensures that once users stop wanting to use the contract cluster or deem the contract cluster as not staking-worthy anymore, there won't be any resources reserved for the contract cluster anymore, either.

When initiating a contract cluster for the first time, the Phala protocol may charge a one-off permanent storage fee. This fee is meant to cover costs accruing from storing and synchronizing the corresponding states on chain.

2.2 Second Layer: Allocating a Contract Cluster's Resources to its Users

Once the individual contract clusters are allocated with their miners, the contract clusters need to allocate the jobs from their end-users to the respective miners.

The job allocation is part of the contract cluster's individual tokenomics, over which they have full discretion. In particular, a contract cluster i may require their users to pay PHA, fiat, or their own *contract cluster token CTO[i]* to use their services.[1]

Irrespective of the payment method, Phala suggests a *reverse-fee model* on the second layer.[2] Under a *reverse-fee model*, the end-user pays some PHA, fiat, or CTO[i] when deploying the contract. The end-user is free to choose the amount, and the amount determines the weight that the miner attaches to a call of the contract when scheduling the incoming jobs.[3]

The user calling the contract may be required to pay something to the contract for it to be executed. This can happen in PHA or CTO[i]. Alternatively, users could be asked to prepay in fiat for credits in the cluster. Also, the cluster may keep lists of

[1] Especially larger contract clusters might want to have their own CTO[i] both for community building and to decentralize decision-making. Contract clusters likely distribute CTO[i]s among their stakers proportionally to their stake, possibly recurrently, to incentivize staking (see also Section 2.3 below).

[2] The general fat contract environment hosted by Phala network will employ a reverse-fee model. The PHA thus raised will go to treasury and to miners. The exact division is subject to governance.

[3] There are several queuing algorithms to manage the incoming queries. For easy implementation, Phala suggests to use a SFQ-based Weighted Fair Queue scheduler.

authorized users that can call contracts for free. In any case, the cluster developer will be able to build a set of controlling contracts to approve or deny the contract deployment and execution requests based on their individual rules.

2.3 Incetivizing Contract-Cluster Development and Staking

The PHA or CTO[i]s that a contract cluster takes in from its end-users could go directly to the developer and the stakers, where the developer is free to determine the share that goes to herself or himself and the share that goes to stakers. Among the stakers, the proceeds are likely distributed proportionally to their stake.

If the contract cluster has its own designated CTO[i], then the deployer and the stakers can either keep their earned CTO[i]s or sell them on a secondary market. Moreover, the CTO[i]s can be used for a contract cluster's governance, which in turn might determine how the earned tokens are distributed.

3 Cluster-Miner Matching

Once the contract clusters $i = 1, ..., n_t$ are allocated with their respective computing power P_{it}, the miners need to be allocated to the individual contract clusters.

The Theoretical Problem Let the miners in era t be indexed by $j = 1, ..., m_t$. Let Q_{jt} be the computing power of miner j in era t. Define $Q_t = (Q_{1t}, ..., Q_{m_t t})$. Let A_t be an $(n_t + 1) \times m_t$-matrix whose entries are either zero or one. An entry $a_{ij} = 1$ means that contract cluster i and miner j are matched and an entry $a_{ij} = 0$ means that they are not. The row of entries with $i = n_t + 1$ corresponds to the allocation to the Phala contract cluster that allows users to submit general fat contracts.

For an allocation A_t to be *feasible*, each miner must be matched with exactly one contract cluster,

$$\sum_{i=1}^{n_t+1} a_{ij} = 1, \ \forall j = 1, ..., m_t. \tag{F}$$

The Phala Cluster-to-Miner matching algorithm seeks to find among the feasible allocations an allocation that maximizes the allocated computing power to the different contract clusters subject to their budget constraints. Letting $B_t = A_t \times Q_t$, these budget constraints can be written as

$$(B_t)_i \leq P_{it}, \ \forall i = 1, ..., n_t. \tag{B}$$

Ignoring for the moment that contract clusters might have preferences over miners, the problem that a Cluster-to-Miner matching algorithm solves can be stated as

$$\max_{A_t} \sum_{i=1}^{n_t} (B_t)_i \text{ subject to (F) and (B).} \tag{P}$$

If the budget constraint (B) does not bind for at least one contract cluster — which will happen if the computing powers of the miners do not exactly add up the budget of that contract cluster — then the general Phala contract cluster for fat contracts receives the residual computing power on top of αP.

The Algorithm Phala implements an algorithm which provides a solution that is close to that of (P) and, in addition, takes into account that the contract clusters might have preferences over the allocations satisfying (F) and (B).

The algorithm proceeds as follows. At the beginning of each era, each contract cluster is presented with an ordered list of all available miners, $L_{it} = \{\ell_{it1}, ..., \ell_{itm_t}\}$. That is each element in L_{it} is unique and satisfies $\ell_{itk} \in \{1, ..., m_t\}$. For every contract cluster, the order of the list is drawn randomly yet the contract clusters are free to rearrange the list as they please.

The algorithm, first, rearranges the different contract clusters so that their stakes are ordered; i.e., $s_{1t} \geq s_{2t} \geq s_{3t} \geq ... \geq s_{n_t t}$, where contract clusters with identical stakes are ordered randomly. Then, the algorithm goes through the contract clusters in decreasing order of their stakes and assigns them with their most preferred miners that have not been assigned before and that are still in their budget set.

Because every contract cluster has all miners on its list, all contract clusters will be matched to some miners and the algorithm stops as soon as the least-staked contract cluster is assigned its miners. The miners that remain unmatched after this last step are then matched to the Phala contract cluster for general fat contracts.

The algorithm is presented in Appendix A together with a proof of its approximate optimality when the number of miners is large.

Discussion The resulting allocation will be the same from era to era, if the set of miners, the set of clusters, and their stakes do not change, and the developers submit the same preferences. If another cluster comes in at some point, then we want to reallocate miners without changing the overall allocation too dramatically. The design proposed above ensures that the more you stake the less you are affected by such a new entry. In particular, if the new entrant at $t+1$ has a stake of size $\hat{s}$, then the budget of all remaining clusters are scaled, yet this rescaling is the less dramatic the more stake a cluster holds and the remaining clusters i for which $s_{it} > \hat{s}$ holds still get their most preferred miners (up to their new budget).

One might also be worried about miners trying the manipulate their popularity measure (cf. next section) by registering fake clusters. The algorithm prevents this by giving preference to higher-staked clusters over lower-staked ones, making such manipulation attempts rather costly. In particular, it is the function $g(.)$, discussed in the next section, that can be used to steer how much the popularity of a miner affects its pay. So, if there is reason to worry that manipulations occur, $g(.)$ can in principle be changed at any time by governance to a function that does not change very much in its argument.

4 Miner Side

Remuneration of a miner depends on the security of the TEE, the computing power that it contributes to the network and its general popularity. The former is objectively measurable, the latter is determined based on the lists that the contract clusters submit to the matching algorithm.

Measuring Popularity To measure the popularity of a miner in an era t, we collect the lists $\{L_{it}\}_{i=1}^{n_t}$ that the clusters submit to the cluster-miner matching algorithm. Then, we assign points to a miner according to the ranks that he takes in these lists. Specifically, we give m_t points to the first miner on a list, $m_t - 1$ to the second, and so on.

Then, we add up all the points of a miner and divide the miners in $k \geq 1$ brackets $b = 1, ..., k$. of equal size according to their points. That is, each bracket contains $100/k$% of all miners; the bracket $b = k$ contains the miners with the highest number of points and the bracket $b = 1$ contains the miners with the lowest number of points. For further reference, we write b_{it} for the bracket of miner i in era t.

Security and Computing Power We measure the security of a miner from the Remote Attestation report from Intel and denote it by $C \geq 0$ in the following, meaning that a higher C corresponds to a higher confidence in the miner. The computing power of a miner is measured by the variable $P \geq 0$, meaning that a higher P corresponds to a better performance. The specific details of how these variables are measured is subject to governance.

Stake Miners that want to become active have to put down a stake $S \geq 0$. The stake may depend on the security and the computing power of a miner.

Remuneration Miner remuneration is tracked via a so-called value promise, $V_t \geq 0$. When a miner becomes active, in round $t = \underline{T}$, its initial value is zero.

Then, in every round $t \geq \underline{T}$ in which the miner is still active, the value promise is increased by $f(C, P) + g(b_{it})$, where $f(.,.)$ and $g(.)$ are increasing in their respective arguments. The specific functional forms of f and g are subject to governance.

Slashing In case a miner misbehaves, the miner gets slashed. Depending on the severity of misbehavior, the slash may take on different sizes. The specifics of the slashing scheme are subject to governance. Let $s_{it} \geq 0$ be the slash that is applied to worker i in era t.

Intermediate Payouts A worker can always request an intermediate payout. Let $w_{it} \geq 0$ be the intermediate payout of bidder i in round t. The intermediate payout may never be greater than

the value promise in that round net of the slashes and payouts in earlier rounds,

$$w_{it} \leq \sum_{\tau=\underline{T}}^{t} [f(C,P) + g(b_{i\tau}) - s_{i\tau}] - \sum_{\tau=\underline{T}}^{t-1} w_{i\tau}.$$

Because $w_{it} \geq 0$, above condition implies that, as soon as the slashes exceed the value promise net of previous payouts, no new payouts can be requested.

Final Payout A miner can stay in the mining pool either until his stake plus value promise net of slashes and payouts becomes negative, or until he decides to leave.

That is, a miner will be excluded from the mining pool in time $\overline{T}$ if and only if

$$S + \sum_{t=\underline{T}}^{\overline{T}} [f(C,P) + g(b_{it}) - s_{it} - w_{it}] \leq 0.$$

If, on the other hand, a miner decides to leave the set of active miners, he receives a final payout, W. If a winer leaves in period $t = \overline{T}$, then the final payout amounts to

$$W = S + \sum_{t=\underline{T}}^{\overline{T}} [f(C,P) + g(b_{it}) - s_{it} - w_{it}].$$

A The Algorithm to Solve (P)

Let M_t be the set of miners in era t. Q_t is the computing power vector of the miners as defined in the text, $\{L_{it}\}_{i=1}^{n_t}$ is the lists submitted by the clusters, and $\{P_{it}\}_{i=1}^{n_t}$ is the list of computing power budgets of the clusters.

The following result characterizes the solution of Algorithm 1, $\hat{A}_t^*$. It shows that it is always in the feasible set of the original problem P. Moreover, it bounds the average distance of the contract cluster's allocated mining power to the maximally feasible allocated mining power.

Theorem 1. *The result of Algorithm 1, $\hat{A}_t$, satisfies* (F) *and* (B). *Letting* $\bar{q} = \max\{Q_{1t}, ..., Q_{m_t t}\} \times \left[\sum_{j=1}^{m_t} Q_{jt}\right]^{-1}$, *it also holds*

$$\frac{1}{n_t}\left[\frac{\sum_{i=1}^{n_t} P_i - \sum_{i=1}^{n_t}(\hat{A}_t \times Q_t)_i}{\sum_{j=1}^{m_t} Q_{jt}}\right] \leq \bar{q}.$$

Algorithm 1 Cluster-Miner Matching

Require: M_t, Q_t, $\{L_{it}\}_{i=1}^{n_t}$, $\{P_{it}\}_{i=1}^{n_t}$.

1: $i \leftarrow 1$, $\mathcal{M}_i \leftarrow M_t$, $A_t \leftarrow 0$.
2: **for** $i \leq n_t$ **do**
3: $\quad j \leftarrow 1$.
4: $\quad$ **while** $j \leq m_t$ **do**
5: $\quad\quad$ **if** $\ell_{itj} \in \mathcal{M}_t$ **then**
6: $\quad\quad\quad$ **if** $(A_t \times Q_t)_i + Q_{\ell_{itj}} \leq P_{it}$ **then**
7: $\quad\quad\quad\quad a_{i\ell_{itj}} \leftarrow 1$.
8: $\quad\quad\quad\quad \mathcal{M}_t = \mathcal{M}_t \setminus \ell_{itj}$.
9: $\quad\quad\quad\quad j = j + 1$.
10: $\quad\quad\quad$ **else**
11: $\quad\quad\quad\quad$ **break**
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **else**
14: $\quad\quad\quad j = j + 1$.
15: $\quad\quad$ **end if**
16: $\quad$ **end while**
17: $\quad i = i + 1$.
18: **end for**

The result shows that the algorithm approximately solves problem (P) in the following sense. If we fix the total computing power in the network, $\sum_{j=1}^{m_t} Q_{jt}$, and let the size of the largest miner vanish (which amounts to increasing the number of miners), then the algorithm's solution approaches that of (P).

Proof of Theorem 1. Because the set of miners that are still available, $\mathcal{M}_t$, is adjusted whenever a miner gets assigned to an contract cluster (cf. line 8 in Algorithm 1), no miner can be assigned to two or more contract clusters, giving us that $\hat{A}_t$ satisfies (F).

The constraint (B) holds, because an additional miner on the list of a contract cluster is assigned to the contract cluster only if that does not violate the budget of the contract cluster (cf. line 6).

Finally, that the differene between the budget and the final allocation for a bidder i is bounded by $\max\{Q_{1t}, ..., Q_{m_t t}\}$,

$$P_i - (\hat{A}_t \times Q_t)_i \leq \max\{Q_{1t}, ..., Q_{m_t t}\}.$$

Adding over all clusters $i \in \{1, ..., n_t\}$ and then dividing both sides by $\sum_{j=1}^{m_t} Q_{jt}$ then gives the claim. □