Hey everyone,

Ismael from Lagrange Labs here. I wanted to share a design that our team put together for a verifiable and trust-minimized RetroPGF reward distribution flow.

We'd love any feedback from the OP Collective and OP Delegates! Our team is committed to support with making this a reality, if the Collective feels that this direction makes sense long-term.

Introduction

The Optimism Collective's Retroactive Public Goods Funding (RetroPGF) provides a powerful mechanism to incentivize the creation of public goods that support the development and usage of Optimism. With the third season of RetroPGF, the rewards are stratified across four categories: OP Stack

, Collective Governance

, Developer Ecosystem

and End User Experience & Adoption

.

The primary data for the categories Developer Ecosystem

and Collective Governance

will likely be derived from objective assessments of on-chain activity. We believe that there is an opportunity for the Optimism Collective to progressively decentralize how these tokens are distributed for future seasons.

In the below post, we'll outline a potential mechanism for a verifiable token distribution based on two steps:

1. The OP Collective would commit to an anonymized reward function at the beginning of a season.
2. At the end of a season, the OP collective would release funding recipients and a proof of the reward function being correctly computed across all of the Optimism blocks for that season.

Alternatively, the OP collective could simply open source the function and a circuit to generate a proof of eligibility, thereby amortizing the cost of computation across the funding recipients.

To handle generating proofs across this scale of data, we propose the use of Lagrange Labs' Zero-Knowledge MapReduce proof construction, which supports verifiable distributed computation at big data scale.

Understanding Lagrange's Zero-Knowledge MapReduce:

The Lagrange Labs ZK Big Data stack is a proof construction optimized for generating large scale batch storage proofs concurrently

with arbitrary dynamic distributed computation. In short, a proof can be generated from a single block header, proving both the data in an array of historical storage slots and the results of a distributed computation executed on that data. Each proof combines both the verification of storage inclusion and

a proof of the result of a verifiable distributed computation, within a single step.

As opposed to a ZKVMs that proves the result of a sequential execution

(e.g. LLVM, Risc5, WASM, etc), the ZK Big Data construction is optimized for proving large scale distributed computation

(MapReduce, RDD/Spark, Distributed SQL, etc) in zero-knowledge.

How is this approach different from other storage proof solutions? The answer is scale.

For a RetroPGF season, a single proof of reward eligibility would have to show on-chain activity across 3 months of Optimism blocks or ~19,440,000 blocks

. Consider a standard depth of 8 summed across both the state trie and storage trie. To access a single storage slot in a single contract over this period, a proof will have to process ~124.4 million Keccak hash functions and RLP encodings.

Intuitively, as the sophistication of the rewards function increases and more data is required per proof, the scale of computation quickly becomes unmanageable with current tooling.

Building A Verifiable RetroPGF:

A Predefined Rewards Function:

To decentralize the RetroPGF process, the OP Collective would first have to commit to a rewards function at the start of a season. A simple commitment could be made through deploying a verifier contract on Bedrock that is capable of releasing OP tokens from a vault, based on a verification of a correct proof of the reward function's execution.

In order to minimize gaming of the rewards process, the function would be kept anonymous until the end of the season. With the Lagrange ZKMR stack, a rewards function would have access to any data available within historical Bedrock blockheaders. This data would include all contract storage (state trie), transaction history (transaction trie) and event emissions (logs bloom & receipt trie).

As mentioned previously, a computation run across data from 3 months of Optimism blocks (~19.5m blocks) would require significant computation resources. To effectively process this type of data at scale with the Lagrange ZKMR stack, the rewards function would be defined based on a parallelizable execution model. Examples of these execution models include MapReduce, Spark / RDD and distributed SQL.

Consider the example of a simple rewards function that distributes rewards to the contracts that have the greatest number of users over a 3 month period. Below is an example of how to specify the function in SQL, though it could also easily be written in either MapReduce or Spark.

SELECT recipient, COUNT (DISTINCT sender) AS "uniqueusers" FROM transactions WHERE blocknumber < x AND blocknumber > x + 19440000 GROUP BY recipient ORDER BY uniqueusers LIMIT 50;

Generating a Proof of a Distributed Computation:

Once a distributed computation is specified in a supported language, a proof of the result of the computation can be generated through Lagrange's ZKMR stack. The ZKMR stack can be thought of as an extension of MapReduce that leverages recursive proofs to prove the correctness of the distributed computation over large amounts of on-chain state data.

This is achieved by generating proofs of correctness for each given worker during either the map

or reduce

steps of a distributed computation job. These proofs can be composed recursively to construct a single proof for the validity of the entire distributed workflow. In other words, the proofs of the smaller computations can be combined to create a proof of the entire computation.

The ability to compose multiple sub-proofs of worker computation into a single ZKMR proof enables it to scale more efficiently for complex computations on large data sets, as would be required for a verifiable RetroPGF reward function.

Consider the function above where rewards are distributed to the contracts that have the greatest number of users over a 3 month period. In this example, the SELECT

and WHERE

operators would be translated into Map steps, while the LIMIT, GROUP BY, COUNT

and ORDER BY

operators would be translated into Reduce steps.

[

image

6285×3603 921 KB

](https://global.discourse-cdn.com/business7/uploads/bc41dd/original/2X/4/4e2c0aa462a487c6e5970700ee0aa31c86ac3b2f.png)

While the above simple distribution strategy would likely be insufficient to properly incentivize public goods creation, it nevertheless provides a starting point of how RetroPGF rewards can be distributed in a verifiable manner. In reality, the Optimism Collective has full agency to develop and specify functions that correctly incentivize its users and developers based on on-chain activity.

Next Steps

We believe that decentralization of the RetroPGF reward distribution process is an essential long-term step for ensuring that the OP Collective can continue its mission to build a sustainable future for Ethereum. Given the impact that public good

funding has on incentivizing protocol participation, ensuring that reward distribution remains both honest

and verifiable

is an imperative.

Lagrange Labs will be soon announcing Version 1 of its proof construction, set to launch on testnet in the Summer of 2023. While the journey to full decentralization on public goods funding may be a while away, we are committed to working with teams to support it becoming a reality.

A big thank you to Mark Tyneway for his support in coming up with this design.

Our main asks for the OP Collective and the OP Delegates are:

1. Would a trust minimized and verifiable RetroPGF be interesting to the OP Collective long-term? If so, we'd love to work with everyone here to build towards it together.
2. What are the distributed programming frameworks that members of the OP Collective would most likely want to write computation in (MapReduce, SQL, Spark, etc)?
3. What would be an interesting proof-of-concept for a verifiable RetroPGF reward function?
4. Is access to data from other OP Stack based chains something that would be a priority within a RetroPGF reward function (Base, Zora, etc)?