Lottery-based decentralized verification for

Marcin Mielniczuk

September 12, 2017

Abstract

We first present a distributed blockchain-based algorithm for computation verification. First, we'll present the general idea, ignoring the costs of blockchain message insertion. Later, we'll show how to gather the results in an efficient way.

1 Assumptions

We assume that the task is easily distributable, i.e. that it can be easily divided into multiple subtasks, each of them to be computed separately.

Additionally, assume the following:

1. The provider outputs the percentage c of correct results (e.g. random garbage).
2. The percentage of good (honest) nodes in the network equals p, $p > \frac{1}{2}$.
3. The whole task is constituted of k subtasks

2 Algorithm

Choose N nodes from the network at random and assign exactly n subtasks to each. The task of each node is to recalculate all of the subtasks. Gathering the results consists of two phases

1. Each node publishes an encrypted and signed verdict about the correctness of all n subtasks (e.g. 1 when all n subtasks were correct, 0 when there was any incorrect subtask)

2. Each node publishes the decryption key for his own message

After all nodes have published their decryption keys (or a timeout was reached), the results are gathered and the final verdict announced (or just calculated). Precisely, we assume that the true verdict is the one the majority of peers has voted for.

This algorithm has an inherent risk of peers deciding not to reveal the decryption keys at some point in time. This will not be addressed since the blockchain-fee-optimal method doesn't suffer from this problem.

Notice that this approach gives us a great flexibility in the terms of computational redundancy. This is better than verification by replication whenever $N \cdot n < k$.

3 Probabilistic analysis

Let X be a random variable outputting the number of nodes who voted that the computation was carried out correctly. Let's consider two scenarios.

3.1 Bad provider trying to get away with an incorrect computation

With sufficiently large number of verifying nodes N, we can expect approximately pN honest and $(1-p)N$ dishonest nodes. Assume the worst case: all dishonest peers are colluding. This means that

$$\frac{\mathbb{E}X}{N} = (1-p) + pc^n \tag{1}$$

with $\lim_{n\to\infty} c^n = 0$ it's just the matter of adjusting the parameter n so that the dishonest peers cannot overthrow the verdict being in minority.

3.2 Good provider penalized by colluding dishonest peers

In this case all pN peers provide the truthful verdict, so the requirement $p > \frac{1}{2}$ is enough.

4 Game-theoretical analysis

Peers need to have a financial incentive to participate in the verification process. This can be solved very easily, by paying the participating peer a reward proportional to the fraction of the subtask computed, i.e. $\frac{n}{k}P$, where P is the price to be earned for computing the whole task. Only peers whose result is consistent with the majority vote (i.e. believed to be true) receive this reward.

This imposes the risk of peers colluding, whose alleviation is not much more difficult, though. Instead of rewarding only the peers with a fixed amount of GNT, we set a pool of $\frac{N \cdot n}{k}P$, distributed equally among the peers which are believed to have calculated correctly. If $p = 0$, i.e. all peers are honest, this is equivalent to the previous reward system.

The new system has this interesting property that it discourages collusion on a game-theoretic background, just as proposed in the solution for verification by replication. Precisely, if two peers decide to cooperate, compute the result once (by a single party) and share the result with the other, the computing party's winning strategy is to properly recompute the chunk and give a false result to the other one.

TODO What is the expected number of subtasks to be shared between two nodes? It should be small if N is large, but something more precise would be nice.

Now note, that due to the fact that whenever the parameters as such, that the expected result is the true one, the joining node's best decision is either to calculate the subtask honestly or give a random verdict, depending on the incentives.

This can be alleviated by requiring putting the subtask result on the blockchain instead of the sheer verdict. In such case, the node can either publish the real result or garbage. Since the expected total verdict is equal to the true verdict, the best choice is to honestly calculate the subtasks.

TODO more detailed analysis, this is too much of waving the arms

5 Cheap verdict collection

The simple *put it on blockchain* method is very inefficient when it comes to the Ethereum transaction fees. The number of blockchain writes can be easily minimized using the following protocol:

The gathering of results consists of two phases, as previously. Instead of everyone putting his verdict on his own, a random node collects the results from the whole network (this can be a problem with a very large number, where several nodes could be used instead).

All verdicts are, as previously encrypted and signed (in this order!). This means that the verdict of any node cannot be tampered with - if it's present in the report, it's exactly how the node voted. Thus, the only attack that can be carried out in this model is not to include the verdicts from a particular node in the voting summary, which is more likely in the second phase, when the collector can decrypt the votes prior to submitting them onto the blockchain and omit some of them to malform the voting results.

This can be alleviated by allowing an *objection* on the blockchain. Precisely, after a node puts the summary on the blockchain, there is a timeout until these results become valid. Within this timeout any node can raise an objection and announce that he is the next one to gather the results from any node possibly ignored by the previous collectors.

This way, every node will eventually submit his results onto the blockchain. What is the expected number of writes onto the blockchain, $\mathbb{E}W$? That's easy!

$$\begin{aligned}\mathbb{E}W = 1 \cdot p + 2 \cdot p(1-p) + 3 \cdot p(1-p)^2 + \cdots = \sum_{n=1}^{\infty} p(1-p)^{n-1} n = \\ p\sum_{n=1}^{\infty}(1-p)^{n-1} n = -p\sum_{n=1}^{\infty}\left((1-p)^n\right)' = -p\left(\sum_{n=1}^{\infty}(1-p)^n\right)' = \\ -p\left(\frac{1-p}{p}\right)' = -p\frac{-1}{p^2} = \frac{1}{p}\end{aligned} \quad (2)$$

6 Possible attacks

6.1 Sybil attack

This is one more place where a Sybil attack poses a major threat, due to the sheer importance of the parameter p.

7 Presentation on concrete parameter values

Run `qmlscene visualization.qml` in the `visualization` subfolder.