Written in collaboration with @Mikerah.

Overview

We present a practical scheme for on-chain non-interactive data availability proofs. It can be implemented as a pre-compile on Ethereum. Moreover, we provide a definition for a standardized content-addressable filestore, a s

tandardized f

iles

ystem (SFS), that may be of independent interest.

Background

Prerequisite Reading

- Towards on-chain non-interactive data availability proofs
- Minimal Viable Merged Consensus
- A data availability blockchain with sub-linear full block validation (i.e.

, LazyLedger)

Extra Reading

- Validity Proofs vs. Fraud Proofs
- Cryptographic proof of custody for incentivized file-sharing

Data Availability Proofs Using Erasure Codes

- A note on data availability and erasure coding
- Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

The essence of sub-linear data availability proofs is client-side random sampling of erasure codes. Erasure codes

allow for the reconstruction of some data (of size M

encoded as N

chunks) using any M

of N

chunks. While this may seem linear in cost, we can do better by splitting up the data in two dimensions, requiring only \sqrt{M}

chunks to be probabilistically sampled.

Unfortunately, this scheme is subjective and cannot be readily executed on-chain at present. An earlier attempt at turning this protocol non-interactive and running it on-chain is extremely complex, requires a Sybil-free set of providers, and makes strong assumptions.

Data Availability In Eth 2.0

The data availability proof method described above is central to the viability of a sharded blockchain. Indeed, it is trivial to see that if all nodes were required to process data on all shards, the resultant sharded blockchain would be isomorphic in performance and resource requirements to a blockchain with a larger blocksize.

If a shard chain block is withheld, "clients with data availability verification can detect the fault and reject it" by using data availability checking as described above. Under a synchrony assumption and an uncoordinated-majority assumption, this allows arbitrarily high levels of confidence that all data across all shards is available.

We note that blocks in a blockchain aren't inherently more accessible to clients than any other data. It is only that nodes

have access to block data through a standardized API by default

Definitions

We define the standardized filesystem

(SFS) as a hash-indexed (i.e.

, content-addressed) data store accessible to a node. Potential examples include, but are not limited to: chain dataIPFS, Swarm, or Filecoin. Swarm is of especial interest as it “is designed to deeply integrate with the devp2p multiprotocol network layer of Ethereum,” i.e.

, it is a SFS by design.

On-Chain Non-Interactive Data Availability Proofs

Assumptions

This scheme makes no stronger assumptions than are necessary for Eth 2.0 to be viable. Namely, an honest majority of main chain block producers assumption for liveness. When used in conjunction with fraud proofs, a (full) synchrony assumption (which requires the main chain to be censorship-resistant i.e.

, the majority of main chain block producers aren’t actively censoring transactions/blocks) is also needed for safety.

Scheme

We propose a scheme that involves a new precompiled contract, C

. The contract has two methods, check(bytes32 hash) -> (void)

and isAvailable(bytes32 hash) -> (bool)

.

check

starts a client-side data availability check on a hash

in the SFS. This may take some time to actually finish executing, so the task can be dispatched in parallel. Additionally, it saves the hash of the pending check along with the current block number in storage (e.g.

, in a map).

isAvailable

returns the results of a previously-requested data availability check of hash

in the SFS. If fewer than D

blocks (D

is a system parameter) have passed since the request, or if a data availability check of hash

has not been requested, this method revert

s. If not, the method returns true

if the data availability check was positive or false

if the data availability check was negative. Successful completion of this method call also clears the request (hash and block number) from storage.

These methods should have significant gas costs, but are by construction reasonably cheap to execute client-side—otherwise performing data availability checks in Eth 2.0 would make the whole system intractably expensive. The gas cost of check

should be proportional to the size of data,specifically $O(\sqrt{blocksize} + \log(\sqrt{blocksize}))$

.

Note that hash

addresses data in the SFS, not just chain or side chain data. This scheme can be used for any arbitrary data in the SFS (i.e.

, data accessible in a standardized manner to clients). This is in fact identical to the core idea of LazyLedger.

Also note that the precompiled contract interface here is agnostic to the underlying data availability technique used. If a new one superior to the one used here is discovered, it can be used instead by simply changing the precompile's functionality.

Example Usage

Suppose we want to deploy a side chain with optimal state safety guarantees. It borrows security from the main chain by committing side chain block headers on-chain, but funds can still be stolen if the side chain operators withhold an invalid block. To do this naively would require all transaction data to be posted on-chain to guarantee data availability (so that potential fraud proofs can always be computed). This is already a big step up in terms of scalability, as the Ethereum chain now only needs to come to consensus on ordered data rather than execution as well, but we want to do even better!

We can instead not

post the side chain block data on-chain, and use on-chain non-interactive data availability proofs. If the side chain block producer(s) withhold data for side chain block with header hash h

, a data availability check is requested C.check(h)

. D

blocks later, the result of this request is acquired C.isAvailable(h)

. If the data isn't available, the side chain can be halted, or the block producer(s) penalized, etc.

Safety Analysis

Just as block producers cannot forge digital signatures or create coins out of thin air, they cannot cause safety violations, even with a dishonest majority

. Block producers that evaluate the results of an isAvailable

call incorrectly are in fact attempting to cause a safety violation on the main chain. Under an honest majority assumption this will never happen. Should the majority of main chain block producers attack the main chain to cause a safety violation, social governance can be used to mitigate this (this is fundamental to Eth 2.0 as well, though doing this requires an additional weak subjectivity assumption).

Applications

The scheme we present here for non-interactive on-chain data availability proofs has a wide range of applications:

- Trust-minimized side chains
- Optimistic (forkful) merged consensus, potentially opens the door for stake-based merged consensus
- Permissionless Plasma
- Rollups without having to post all non-witness data on-chain all the time
- A two-way bridge to/from Eth 2.0