We present a scheme for enabling Ethereum 1.x to process data availability in parallel, potentially allowing the chain to indirectly process and secure many more transactions per second with minimal changes and without adverse effects on state growth. This is accomplished by deferring execution to secondary layers, so that Ethereum is mostly responsible for ordering and authenticating data, and processing fraud or validity proofs.

Background

Prerequisite Reading

- On-chain scaling to potentially ~500 tx/sec through mass tx validation
- Layer 2 state schemes
- Minimal Viable Merged Consensus
- Compact Fraud Proofs for UTXO Chains Without Intermediate State Serialization

Suggested Watching

- The phase 2 design space (by @vbuterin)

Using Ethereum as a Data Availability Layer

Before discussing Ethereum as a data availability layer, we must first understand how side chains work (or rather, don't work). A side chain is a blockchain that validates another blockchain—in plain English, a side chain "understands" the existence of a canonical parent chain. The converse is not true: the parent chain does not know that a canonical side chain exists. As such, one-way bridges of funds are trivially feasible (parent -> side) but not two-way bridges (parent <-> side). An example of a side chain is the Eth 2 beacon chain, with its parent chain being the Eth 1 chain.

In order to bridge funds back to the parent chain, the naive solution is to simply commit to each side chain block header on the Ethereum chain (essentially running a side chain light client within the EVM)—known as a relay—and allow withdrawal of funds with an inclusion proof against the side chain's transactions or state. If all the side chain's block data is available, then a fraud proof can be issued on-chain to block fraudulent withdrawals.

But what if the side chain's block data is withheld? Then a fraudulent withdrawal can be done, since a fraud proof can't be produced without the data! Using validity proofs does not solve this problem, as inclusion proofs for withdrawals are against the transactions or state of the side chain, which are withheld in this scenario.

While data availability proofs with random sampling can be done, those are subjective and can't be run on-chain (well, at least not easily). Herein lies the paradigm shift around what a base blockchain needs to provide

: the Ethereum chain can be used for provable data availability.

In this paradigm, the Ethereum chain only needs to check that a blob of data authenticates to a given Merkle root, and process fraud/validity proofs. So long as this happens, layer-2 systems can be responsible for executing the data. Essentially, this separates consensus on data from consensus on execution, and allows Ethereum to only perform minimal execution.

Tradeoffs

This scheme is of course not without any

tradeoffs, but the tradeoffs have been almost universally approved of:

Cons

Pros

Increased disk storage

Greatly reduced random accesses to large, mutable, shard data structure (state)

Increase network bandwidth

Incentive to not use state excessively, removing the need for complex state rent

Different side chain features (eWASM, Move, UTXO, privacy, etc.)

A small price to pay for salvation.

Multi-Threaded Data Availability

We note a cute property of the above scheme: computing the Merkle root of the data blobs (either through full Merkleization, sparse Merklelization, or going up a Merkle branch), is a pure reducing function

. As such, there is no reason for reducing data blobs to be executed in sequence, or even be executed within the EVM at all!

By simply multi-threading this process, we can allow Ethereum to process its data availability in parallel, with the majority of execution deferred to layer-2. The sequential work that must be done by Ethereum then reduces to basically comparing the computed root with the given root, and if true

saving the side chain's block header in state for later use until explicitly discarded, and processing fraud proofs or validity proofs.

Proposed Implementation

We propose the following high-level implementation: a new flag for transaction data that indicates a data blob somewhere else in the payload, and the pure reducing function that will be applied to it. This can be EVM code, eWASM code, or one from a set of pre-defined hard-coded reducing functions.

Clients can then go through transactions in a block (or even in their mempool!) and compute the reduction, inlining the result into the appropriate transaction's calldata

.

Conclusion

We present a scheme that enables multi-threaded data availability for Ethereum 1, borrowing inspiration from execution environments in phase 2 of Ethereum 2.

Edit: a potential issue with this scheme is that gas must be paid for the reducing function, which may make it non-pure if implemented naively. This shouldn't be a problem, as this whole scheme can be thought of as a form of speculative execution.