I long had this idea about a multi-rollup design that aims to solve some of the problems rollups face today. For about a year and a half I though someone would build it and never really dug into it or considered the specifics of such a system.

It's been a while now and it seems that there isn't a design that solves the problems I will describe in this post. So, I will try to flesh out as much details as I can about the system in the hope that someone takes inspiration from this or even existing rollups borrow a few ideas.

Introduction

One of the problems rollups face today isuser experience. In many designs, rollups come as isolate ecosystems with very different traits. There are some ways to interoperate but with multiple heterogeneous systems wiring them up it's quite a challenge. Moreover, getting users to sign up for all these rollups is hard. They have to learn about each one of them individually, assess the relevant smart contracts, connect their wallet to a new RPC endpoint, bridge assets to the chain etc.

What if there was a rollup design that can provide a unified experience across the rollups? How would it look like?

I was asking this question to myself and came to five realizations:

1. It should

provide a unified RPC

to query and call smart contracts across different rollups. Smart contracts should have a unique address that does not depend on the rollup they are part of.

1. It should

allow scaling up and down

based on demand. More transactions should mean more rollups to process them and uneven load between rollups should be balanced out.

1. It should

incentived sequencers

in different rollups to be live. The system should encourage other sequencer to replace sequencers that go offline.

1. It should

support instant cross-rollup transactions

. Transactions should settle fast enough that cross chain operations make sense.

1. It should

maintain light client and block explorer

functionality across rollups. Block explorers should provide a unified view of the blockchain and light clients should allow to cheaply validate it.

With all this in mind, I came up with a design that involves a rollup hub and a variable number of child rollups. The rollup hub acts as both the registry

and the load balancer

of all the child rollups but does not do any smart contract processing. Smart contracts are processed in the child rollups instead.

In the following sections I'll go through a draft design that deals with 5 considerations I mentioned above.

Design Overview

The system has two main components: the rollup hub and the child rollups. The rollup hub is a rollup that contains a registry of all smart contracts

for all child rollups and is the one that decides which rollups are responsible for which smart contracts. Moreover, the rollup hub contains another registry of all sequencers

for the child rollups. Child chains are responsible forexecuting transactions for the smart contracts the rollup hub assigned them in the smart contract registry. The sequencer registry contains each sequencer RPC endpoint and DA address.

Sequencers registry:

The sequencers registry acts as a mapping from the global smart contract addresses to the smart contract addresses. This is used to route RPC calls to the specific sequencer RPCs that correspond to the smart contract that is queried or updated.

Smart contract registry:

The smart contract registry acts as a mapping from the global smart contract addresses to the smart contract addresses.

Rollup chains:

The child chains have a state root as they normally do. This state route can be update by calling the smart contract directly but can also be updated when the rollup hub assigns the smart contract to a different rollup. In that case, the smart contract should be removed and should be added to the other smart contract.

A Unified RPC

Goal:

To not have to connect

to a new chain for every rollup and have transparent to the user cross-rollup transactions

A unified RPC restores the user experience of a single chain in a multi-rollup world. Users don't have to connect to a different network to use a different rollup.

The system uses the registry of therollup sequencers from the rollup hub to find out the RPC endpoint of a sequencer that corresponds to a specific smart contract. The request is then submitted directly to that sequencer. Multiple transactions can be done by submitting requests to different rollups. Check the section below for more details.

How it works:

1. The rollup hub keeps a registry of sequencers for all child chains.
2. When a user wants to submit a new transaction, the user wallet queries the smart contract registry to get the rollup id of that smart contract and the sequencer registry to get the RPC endpoint of a sequencer in that same rollup.
3. The transaction is then submitted to the RPC endpoint of the sequencer.

Load balancing

Goal:

To have balanced out fees

across all rollups

Load balancing allows to balances the load across the rollups. When the system is jammed, new rollups can be spawned to handle the load. When there's no much usage rollups can be removed to save resources. Moreover the system could avoid fee spikes by moving smart contracts with high demand in transactions to rollups that more available capacity.

How it works:

1. Every epoch the rollup hub evaluates the load of all the rollups in the system. Epochs should last a few hours (possibly 6 to 24) to avoid extensive smart contracts reassignments.
2. The rollup hub can decide which smart contracts to reassign and when to spawn or remove rollups with governance or autonomously using the gas consumption history of different smart contracts.
3. The rollup hub checks if any rollup has above average

transaction load and (ie. very high fees) or has below average

transaction load (ie. very low fees).

- If one rollup has above average

load, the rollup hub evaluates which smart contracts areconsuming the most gas and reassigns them to a different rollup that can afford the extra load. Smart contracts are then dropped from their initial host rollup state.

- If the average load is above average

in all rollups, the rollup hub creates a new rollup and assigns a few smart contract to the new rollup. Similarly, if the average load in all rollups is below average

, the rollup hub drops one rollup and reassigns its smart contracts to the other rollups.

1. Rollup chains should look into the rollup hub at every epoch and download the storage of any new smart contracts they are assigned to and drop any smart contracts they are not responsible for anymore.

Note:

Downloading the storage of some smart contracts might not be a trivial task. At first the state is not available on the DA layer and is quite large in size. This puts bounds on the minimum epoch time that would need a grace period to prepare that smart contract storage.

Sequencer incentives

Goal:

Having standby sequencers

that are incentivize

with partial rewards in the native token

Most rollups today are built on a single chain and have a single or very few sequencers to manage with the goal of maximizing rollup uptime. In contrast, in a multi rollup system, there multiple standalone child rollups that each have to be online to have liveness across the whole system.

Sequencers are naturally incentivized to join rollups to collect MEV but it's better to have proper rewards for these sequencers since they are more consistent and don't miss-aligned incentive as MEV does. These rewards should come from the rollup hub monetary policy.

Moreover, it's better to have a few sequncers on standby and ready to be enter. These sequencers can join when there's an increase in transaction demand and leave the system when there's not to save computational resources.

The standby sequencers will stay in a sequencer queue and get a small amount of rewards for there availability commitment. When they are swapped in in a rollup they will get the full amount. Rewards will come from a fee burning mechanism in the rollup hub.

How it works:

1. Sequencers can join the sequencer queue in the rollup hub by committing to afinancial bond (similarly to current rollup systems).
2. The sequencers in the queue are required to provideDA proofs that they have the rollup hub state and are read to join a rollup anytime.
3. When they submit proofs they will get partial rewards in the native token of the system. That token is handle on the rollup hub.
4. If the rollup hub decides that a new rollup is needed they will get assigned and get paid the full reward. That reward is determined by the amount of total fees burned in the system.

Cross-rollup transactions

Goal:

Cross-rollup transactions should be instant and transparent

to the user

A cross-rollup transaction between rollup A and rollup B needs to have 2 parts: 1) a transaction on rollup A and 2) a transaction on rollup B that only happens if the transaction on rollup A has succeeded and is final.

To get fast confirmations the user wallet can check whether the transaction is both submitted to the underlyingDA layer and is valid using ZK proofs. If a transaction is both included and valid then the sequencers have to reach thesame conclusion about that particular transaction.

Credits to Mustafa Al-Bassam and Sovereign Labs for the idea.

How it works:

1. A user submits a transaction that involves 3 rollups, say rollup A, B and C.
2. Let's consider a specific example. Rollup A has a stablecoin smart contract, rollup B has a DEX and rollup C has a lending protocol. In this example, the user wants to convert their stablecoins to a different coin to deposit it to the lending protocol.
3. The user has to first submit the rollup A transaction that transfers the stablecoins to the DEX on rollup B.
4. Then they can submit the rollup B DEX transaction that takes the stablecoins and converts them to the desired token on rollup B.
5. That token should then be transferred to the rollup C and so the user submits a 3rd transaction that does exactly that.
6. Finally the user submits a 4th and final transaction that deposits the token to the lending protocol.

Light nodes and block explorers

Goal:

Light nodes should be able to validate a smart contract

across rollups and block explorers should provide a unified view of the chain

A blockchain system should allow anyone to run a node and verify the chain themselves. In this multi-rollup design, smart contracts are constantly reassigned to different child rollups and there should be a way to follow these specific smart contracts across. That's a mental shift from validating one or more chains to validating one or more smart contracts. Light nodes can utilize ZK proofs to cheaply validate all child rollups.

How it works (light clients):

1. Rollup nodes should support a verifying mode along sequencer mode.
2. Verifying mode validates the state of a single smart contract and does not submit transaction batches to the DA layer as the sequencer mode does.
3. If the smart contract changes child rollup, verifying node have to only update the child rollup they listen to for changes. That's because they already have the smart contract storage up to the point before the reassignment.
4. Smart contracts are meant to be processed in one rollup at a time. Since they are confined in one rollup verifying nodes with the same specs should be able to follow and verify them.
5. Light node can cheaply verify the state of the chain with ZK proofs.

Block explorers are an integral part of a blockchain system. They facilitate balance queries for the native asset, smart contract queries and maintain transaction history from the first block to the current one. In this multi-rollup system, a block explorer should provide a unified view across all child rollups.

How it works (block explorers):

1. Block explorers should support to query balances (for the native asset) on the rollup hub and transaction history for all child rollups.
2. Similarly to single-rollup systems, block explorers enable this using an index. Multi-rollup systems have to index all rollups to be prepared to serve queries for any smart contract in the system.
3. If the rollup hub decides to to scale up the number of child rollups, block explorers should be prepared to handle it. They should either provision for more child rollup capacity or have a container orchestration system (eg. Kubernetes) to automatically scaling up the child rollups.
4. They should use the block number from the DA layer to provide consistency across all rollups.

Conclusion

Right now this design is just an idea and I might never pursue it further, yet, I hope you found it interesting. If the design check out, I'm looking forward to rollup project utilize it and approach EIP-4844, Celestia or Avail capacity.

If you think there's a flaw somewhere or the design is problematic feel free to let me know.