OPTIMISM

Rollups, Shards & Fractals

The Dream of Atomically Composable Horizontal Scaling

@norswap

Devcon VI
October 13th, 2022

Goals

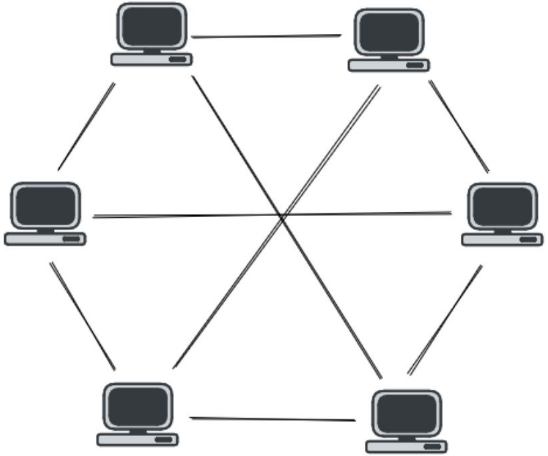
1. Quickly contextualize (decentralization-preserving) scaling schemes
2. Propose a new sharding scheme with atomic cross-shard transactions
3. Demonstrate research process (problem/solution)
4. Foster interest, research & collaboration in the area!

(Yes, this is a nerdsnipe!)

Vertical vs Horizontal Scalability

vertical

(rollups)



horizontal
(sharding)

Rollups Enable Vertical Scaling

- Because they can be validated by validating Ethereum
  - extra assumption: one honest validator / data supplier
- But what if we need extra scalability?
  - not just the sequencers: we still want a healthy number of validators
- What if we want many different rollups?
  - Different security assumptions (data availability)
  - Different parameters (fees, fee token, throughput, block time, validity rules)

Horizontal Scalability

Two main approaches

- parallelization
  - Big blockchain with load spread the load between machines
  - Option 1: optimistic parallelization
    - Can't increase throughput: can't charge more if not parallelizable
  - Option 2: strict access lists (all touched contracts or storage slots)
  - Still imposes high costs on validators
- sharding
  - Validators can validate a single shard
  - Shard choice becomes an explicit choice for apps
  - Enables heterogeneous rollups
  - **Without a way to effectively communicate between shards, this is a bad solution**

Cross-Shard Communication Example

1. Shard A: swap BTC for ETH
2. Bridge ETH from shard A to shard B
3. Shard B: buy NFT for ETH

This is entirely **specified by a single transaction**.

Atomicity

1. Shard A: swap BTC for ETH
2. Bridge ETH from shard A to shard B
3. Shard B: buy NFT for ETH

Desirable property: **atomicity**

If any part reverts, everything reverts.

e.g. if I can't buy the NFT, I don't swap BTC for ETH

Application-Level Atomicity

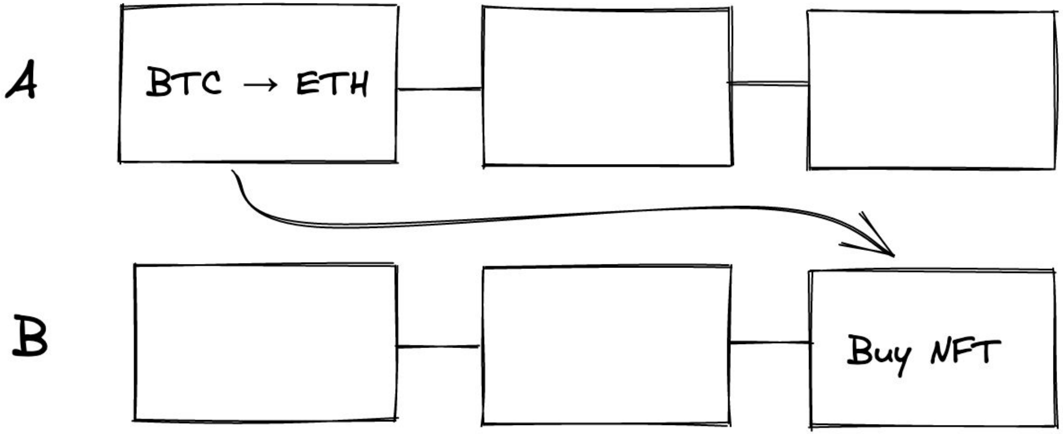
1. Shard A: swap BTC for ETH
2. Bridge ETH from shard A to shard B
3. Shard B: buy NFT for ETH

Not really feasible: pay swap fees twice / exposure to ETH/BTC volatility.

Sometimes feasible for some applications,
if the A part is reversible within a certain delay.

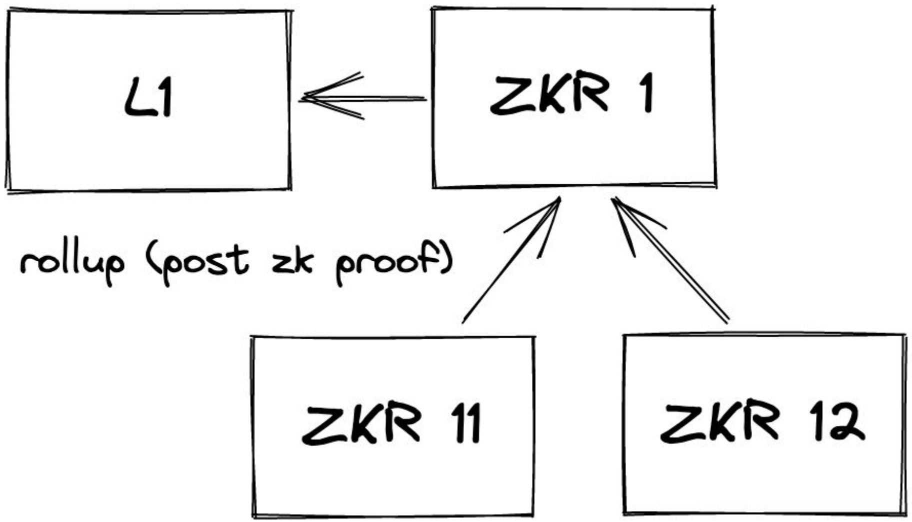
Today: Eventual Delivery

- for cross-chain communication in general
- A part done → B part will eventually be attempted
- not atomic: B part could revert
- implementation: light-clients or zk-proofs



Bounded Delivery with Fractal ZK-Rollups

- improve eventual delivery: minimize time between A part and B part
- hierarchical/recursive/fractal ZK-rollup
- one zk-proof in ZKR 1 per child rollup can guarantee latency
- … but not atomicity!
- expensive today



Cross-Shard Atomicity

- one "blockchain block" = one block for every shard
- atomic cross-shard transactions
  → shards must be able to exchange and answer "messages"

- most naive idea: eager inter-shard blocking
  - somewhat equivalent to strict access lists
  - at worst same throughput to synchronous blockchain, but can charge fees
  - no need to specify access lists
  - but: **cross-shard latency**

Inter-Shard Message Exchange

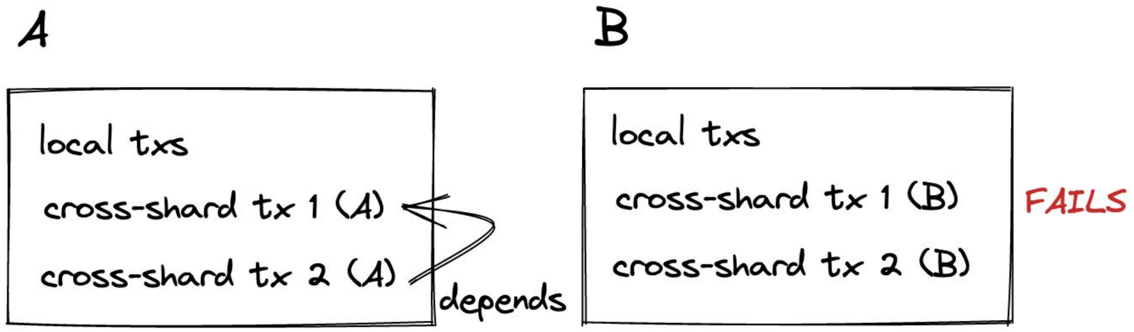
- divide blockchain block time into multiple slots
- first slot: each shard executes transactions...
  and collects messages to send to other shards
- second slot: each shard executes its received messages...
  and (optional) collects messages to send to other shards
- (optional) keep going

→ This is **bounded message delivery**: can't revert A part if B part fails,
because other txs rely on result of A part.

Naive Inter-Shard Message Exchange is Not Atomic

- assume tx 1 and tx 2, two "swap / bridge / buy NFT" transactions
- the swap (A) part of tx 2 depends on the swap part of tx 1

  → the swap part of tx 1 cannot be reverted

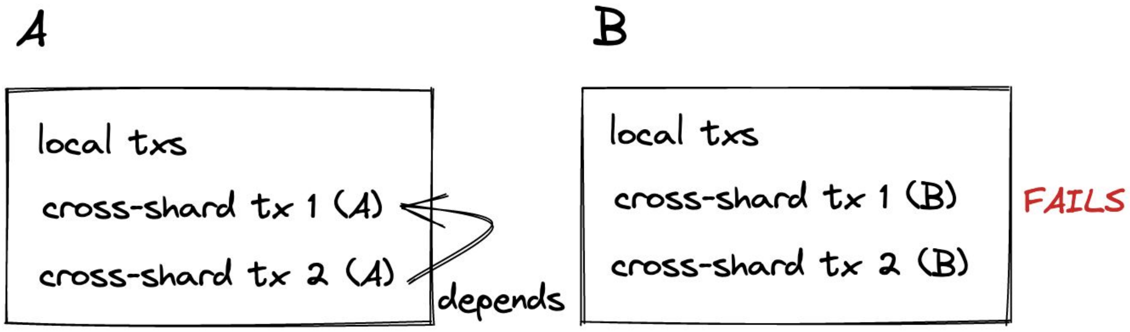


Atomicity Requires Synchronicity

- shards must act "as one" to execute cross-chain transactions
- local transactions can still be processed separately
- problem 1: cross-shard latency is poison
- solution 1: task a special shard to execute cross-shard transactions
- problem 2: forcing the special "atomic" shard to have all shards' state
    - this removes one of the big benefits of sharding!
- solution 2: make the atomic shard execution stateless
    - shards must supply all state accessed by cross-chain transaction parts

The Poop 💩: Transaction Simulation

- In the EVM, given uncertain shard state, it's impossible (in general) to collect all storage slots accessed by a transaction.
- Approximation is possible: run against the state assuming non-reversion.
- Hinting is possible: explicitly instruct the shard on which slots to includes, or how to guess the slots.
    - via in-contract code, in-protocol information, or out-of-protocol information



Transaction Simulation Illustrated

```
// cross-shard tx (A part)

x = compute(state)
y = send(B, msg(x))
compute(y)
z = send(C, msg(y))
compute(z)
```

Atomic Inter-Shard Message Exchange

Phase 1:

- shards execute local txs
- shard simulate cross-shard txs
  - collect cross-shard messages
  - collect accessed storage slots

Phase 2:

- shards exchange messages
- shards simulate messages
  - collect accessed storage slots

Phase 3:

- shards send cross-shard txs & messages to the atomic shard
  - including collected messages and initial storage slot values
- atomic shard executes cross-shard txs atomically

Transaction Simulation Restricts Expressivity

- We need to simulate transactions for stateless execution
- Hence, we can only safely express cross-chain transaction where simulation will always fetch all the the required storage slots.
- We could take the risk that the transaction won't work...
  but it's not always possible for another reason.

- **Note**: This is the same problem as building strict access lists!!

Transaction Simulation & Cross-Shard Messages

- The problem is actually worse:
  cross-shard messages may depend on uncertain state!
- We need to derive messages during simulation!
  - This is why we need to restrict ourselves to deterministic simulation.

- Another problem: what if we want an answer from the other shard?
- Execution may depend on this answer, and so **it must be hinted** for simulation to proceed.

Transaction Simulation Illustrated

```
// cross-shard tx (A part)

x = compute(state)
y = send(B, msg(x))
compute(y)
z = send(C, msg(y))
compute(z)
```

Transaction Simulation Constraints, Illustrated

```
// cross-shard tx (A part)

x = compute(state)
y = send(B, msg(x))
compute(y)
z = send(C, msg(y))
compute(z)
```

- `compute(state)` — accessed storage slots must be deterministic
- `msg(x)` — computation must be deterministic
- `compute(y)` — must be hinted approximately (get correct storage slots)
- `msg(y)` — must be hinted accurately (used as message argument)

Open Questions

- Are these restrictions reasonable?
  Do they still enable a powerful rich model?
- Can we statically guard against non-deterministic execution?
  Or do we make it the users/tools' responsibility?
  - Especially relevant for data passed as message.
  - Is the footgun worth the new possibilities?
- What is the correct abstraction level?
  - If we don't do checks, we can simply add a "cross-shard call" opcode to the EVM.
  - We can change the implementation (how shards handle this) later!
  - But some of the "semantically valid" tx won't be executable because of simulation.

Conclusion

- Sharding horizontally requires either parallelization with strict access lists or sharding.
- Sharding with atomic cross-shard transactions is awkward.
- Atomic cross-shard transactions are feasible…
  at least at the cost of expressivity restrictions.
  (which is ~ similar to the problem of building strict access lists)
- cf. open questions

The Call to Adventure

Is this interesting to you?
Do you want to work on stuff like this?

Let's talk!

- Optimism is hiring
- Other forms of collaboration welcome

For any question/discussion/collab, feel free to hit me up @norswap (tg, twitter, @optimism.io)

@norswap