Algorithm for Cross-shard Cross-EE Atomic User-level ETH Transfer in Ethereum

Crosschain Workshop 2021

Raghavendra Ramesh
Consensys Software R&D
Background
Ethereum 2

- 64 shards
  - Node space and Users-space are partitioned
- Execution Environments (EEs)
  - Many
Peter Robinson, 14 Mar 2020 ethresear.ch

- Similar to events in contracts, which is at application level
- Unforgeability - using Merkle Proof on it
Netted Balance Approach for EE level Transfers

Vitalik Buterin, 30 Dec 2019 ethresear.ch

- Distribute EE balance on shards
- Globally, for every EE, maintain a matrix
- A row in every shard
- Real balance on a shard
- Add up that column

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>-5</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Real</td>
<td>26</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Check realBalance of EE1 on shard1 > X
  - Needs Merkle proofs from all the other 63 shards
Update locally on the source shard
Atomic Asynchronous Cross-shard User-level Transfers
Core Ideas

- Use the netted shard state to communicate outstanding credits and outstanding reverts
- Make reverts as enshrined EE host functions instead of user transactions avoiding unnecessary gas expense
Shard State

E1

E2

E3

partBalance

credits

reverts

outstanding Credits

EETransferAmount
Transactions

- Cross-shard debit transfer
  - Signed by the sender. Signature is stored in the fields $v$, $r$ and $s$.
  - $a_i \xrightarrow{x_i} b_i$, cross-shard transfer of $x_i$ ETH from the user $a_i$ on $(s_1, E_1)$ to the user $b_i$ on $(s_2, E_2)$.
  - Submitted on sender shard.
  - Contains a unique transaction identifier.
  - Emits a **ToCredit** System event on success, which includes the block number and an index number.

- Cross-shard credit transfer
  - $a_i \xrightarrow{x_i} b_i$, credit transfer of $x_i$ ETH to $b_i$ on $(s_2, E_2)$, which is from $a_i$ on $(s_1, E_1)$.
  - Submitted on recipient shard.
  - Includes the **ToCredit** System Event and the Merkle Proof for it.
Block Proposer (BP) Algorithm Sketch

1. Select cross-shard transactions: debit / credit
2. Process each transactions
   a. If debit is success
      i. Emit ToCredit System Event
      ii. Do User-level Debit
      iii. Add credit info to shared state
      iv. Record EE-transfer amount
   b. If credit
      i. Success ⇒ Do User-level Credit
      ii. Fail ⇒ add revert info to shared state, record EE-transfer amount
3. Complete EE-level transfer
Corner case

Sender disappears by the time revert happens.

We end up in a state where there is ETH loss at user-level, but not at EE-level.

Is there a better solution?
Example walkthroughs
Optimistic Case

1. Update BitFieldMap with credits from previous slot
2. Kick out expired credits
Debit Fail

1. Update BitFieldMap with credits from previous slot
2. Kick out expired credits
Credit Fail

s1

\[ a_1 \rightarrow b_1, a_2 \rightarrow b_2, a_3 \rightarrow b_3, E_1 \rightarrow E_2 \]

next slot

process reverts

s2

1. Update BitFieldMap with credits from previous slot
2. Kick out expired credits

\[ b_1 \rightarrow a_2, a_1 \rightarrow b_1, a_2 \rightarrow b_2, \text{FAIL } a_3 \rightarrow b_3, E_2 \rightarrow E_1 \]
Benefits

- Atomic
- No locking and blocking
- No constraint on BP to pick or order specific transactions
- In every block, a BP has to get the outstanding credits and reverts from every other shard.
- Inherited from netted balance approach.
- However, in the EE-level netted-balance approach the querying is restricted to the sender EE’s that are derived from the user-level transactions included in the block. The problem is aggravated here, because we need to query from all EE’s, even for the EE’s not touched in this block.
Threat Analysis of Byzantine BP (BBP)

- show no or false
  - part EE-balances or
  - set of impending credits or
  - set of reverts
- not update or wrongly update outstandingCredits with impending credits
- not process or wrongly process impending reverts
- not emit or emit with incorrect data the ToCreditSystem Event
- not include a revert for a failed credit transaction
- not affect appropriate EE-level transfer

Block is invalidated by attesters assuming that #Byzantine nodes is within the limit posed by consensus algorithm.
Thank You