Special thanks to @jvluso for inspiration

.

Prerequisites:

- Improving the UX of rent with a sleeping+waking mechanism
- A minimal state execution proposal

The general process for a cross-shard transaction (for an example, we'll use transferring 5 ETH) is:

- On shard A, destroy 5 ETH, creating a receipt (ie. a Merkle branch with a root committed into the state root of that block) containing (i) the destination shard, (ii) the destination address, (iii) the value (5 ETH), (iv) a unique ID.
- Once shard B becomes aware of the state roots of shard A up until that point, submit a Merkle branch proving that receipt into shard B. If the Merkle branch verifies and that receipt has not already been spent, generate the 5 ETH and give it to the recipient.

To prevent double-spends, we need to keep track in storage which receipts have already been claimed. To make this efficient, receipts need to be assigned sequential IDs. Specifically, inside each source shard, we store a next-sequence-number for each destination shard, and when a new receipt is created with source shard A and destination shard B, its sequence number is the next-sequence-number for shard B in shard A (this next-sequence-number gets incremented so it does not get reused). This means that in each destination shard we only need to keep track of SHARD_COUNT bitfields, one for each source shard, to prevent double spends, which means a cost of only one bit of storage per cross-shard tx.

Notice that there is an opportunity for protocol simplification here by unifying two mechanics

: we can make a contract hibernation/waking process simply be a compulsory cross-shard transaction, except with the source and destination being the same. The second half of the cross-shard transaction (the receipt publication) would simply happen whenever anyone wishes to wake the contract.

Now, we still need to deal with one problem with this scheme: it creates permanent storage for each cross-shard transaction (and for each hibernation/waking). It is only one bit of permanent storage, but still permanent storage nonetheless. Assuming 10 cross-shard transactions plus hibernations per second coming into each shard, that's 315M bits ~= 39 MB per year of storage

.

We could

simply grin and bear it, but there is another alternative. We break up time into N-block periods (eg. N blocks = 1 year). If the current period is k

, we store in the state a list of sequence IDs that have been consumed during periods k-1

and k

. When the period increments to k+1

, we remove the list for period k-1

from the state, and store the Merkle root of this list as a receipt (eg. in aDBMA).

If someone wishes to claim in period k

a receipt that was produced in period j < k-1

, then for every period in j.....k-2

they would need to provide a Merkle proof showing that that receipt has not yet been consumed.

This scheme has the following properties:

- Max storage requirements: a few hundred MB per shard (39 MB in the best case if everyone is consuming receipts within one period, but it could become larger if consuming very old receipts happens frequently, as the total entropy of the ~400 million ID numbers being consumed within some period would be higher)
- Max length of a Merkle proof: ~1 kb per year

There is a natural linear tradeoff here: we can have 10x less storage in exchange for 10x longer Merkle proofs for resurrecting old contracts by simply shortening the length of a period.