At the Eth crypto workshop in Palo Alto we had a few really good discussions about locking, sharding, load balancing and account lookup. One interesting idea I posited was a solution to avoid the problem of the root node having a huge dictionary of which account lives where, and every intermediate node below it also needing a full and exact set of all accounts on the nodes below it.

TL;DR By mapping Accounts to the closest Shard in a Ring, re-balancing happens by adding more Shard positions to the Ring. Shards each have 1+ positions. Then account lookup can be done with only the shard information, instead of needing the whole dictionary, e.g. finding closest shard is $f : a \to { \text{abs}(a - s_i) : \forall s_i \in S}$

With a mapping of $\text{account}_i \to \text{node}_n$

which propagates upwards any modification to the map which moves an account from one subtree or another requires communication with all common parent nodes. Re-mappings of an account to a node within the same subtree require no further upward communication.

i.e each parent needs to keep a list of all accounts and the sub-node the account can be reached via a routing algorithm which doesn't store the full destination (as that would propagate all changes upwards to the root) - only the direction (reduces communication overhead - e.g. the root node will only be updated whenever an account is transferred from anywhere in the left half to anywhere in the right half.).

Any kind of re-balancing algorithm should minimise the amount of upward communication, e.g. in the worst-case the re-balancing scheduler makes every account migration to the other half of the tree, e.g. imagine a dumb rebalancer which first inverts all the bits of the node index, $Bad(i) \approx {\bar{i}}$

, then finds the nearest sub-node with enough capacity - it's likely that every migration would require the account mapping dictionary at the root to be updated.

By using a simple distance metric, e.g. $\text{abs}(a-s_i)$

, rather than $\text{BitCount}(a \oplus s_i)$

, between the Account ID and the Shard ID, along with the Shard knowing the IDs of its immediate neighbours each shard knows the range of accounts it's responsible for, and when new shard positions are introduces it will be quick to re-calculate the responsibility range and quickly identify the accounts that it needs to migrate or ignore etc.

A hierarchical organisation is required for the locking mechanisms and finality scheme, otherwise you end up with a potential n^2

communication between every node which doesn't scale.

Using this ring system with shard positions is compatible with hierarchical organisation, but instead of being a binary tree it is now an n-ary tree, where each level/ring can have two or more shard positions - this should reduce the depth of the tree and could provide another slight improvement.