Thanks to Li Xuanji, Ben Jones, David Knott, and vi for coming up with this. Also thanks to Justin Drake for the DBML construction <3.

TL;DR

We can use double-batched Merkle log accumulators (DBMLs) to make Plasma commitments cheaper.

Background

Plasma block commitments are expensive. Without rent, if we're making one commitment per Ethereum block, then we're paying in the order of 100000s USD/yr just for gas. A lot of this is due to the cost of adding storage to the root chain (20k gas/word). This post describes one way to reduce cost for each commitment.

Fancy Merkle Stuff

Double Batched Merkle Log Accumulators

Edit:

"the entire bottom buffer is unnecessary, god bless Dan Robinson"

Dan Robinson pointed out to me that the bottom buffer here is unnecessary. We can instead just go full Merkle Mountain Range and reuse the same memory space repeatedly. Basically, we Merklize as we go and only add an additional memory slot once we overflow.

This comes at the cost of some minor complexity, and increased long-term proof size (log(n) with the number of commitments). However, storage is expensive and calldata is cheap, so the trade-off seems to be worth it. There's also a minor annoyance if users want to access recent blocks and new block additions cause the tree to be Merklized and the proof to be therefore invalidated.

Basically, this is a lot like the "Amortizing Cost" section described below, except that we use much less storage space (only n

slots to make 2^n

commitments). It's still possible to use the same "top buffer" used in DBMLs to cap the max proof size and create some immutable roots every once in a while.

We can use DBMLs to make this more efficient. Both structures (sort of) basically work the same way. When the operator commits blocks, the headers are stored in a special data structure. Every 2^n

commitments (plasma contract decides n

), the commitments are Merklized into a single root, and the root is written to storage. Then, the next commitments "roll over" and start overwriting the old commitments. Once we reach 2^n

commitments again, we Merklize and store, rinse and repeat.

Writing over a previously-written word only costs 5k gas, so we're getting lots of gas savings (minus some new costs for Merklizing). Note that this requires users store all of the block roots themselves instead of relying on the root chain to have them available, but we're pretty much expecting them to do so anyway.

The main downside to this is that Merkle proofs become a little more complex. When a block's root is still in the "reusable storage" space, the proof has to point to the temporary index. Once the "reusable storage" has been Merklized, the proof has to point to the storage index and becomes log(n) * 32 bytes longer.

Amortizing Cost

The cost to Merklize 2^n

commitments might be very high. We can amortize the cost over the entire set of commitments by Merklizing as we go. This ends up looking a lot like Merkle Mountain Ranges, except that we reach some maximum height and restart.

Shared Commitments

Another cool way (and unrelated way) to save space is for multiple Plasma chain operators to agree to submit roots together. This requires off-chain agreement, but it means that we can represent multiple Plasma chains in a single 32 byte commitment. Unavailability on one Plasma chain doesn't impact any of the other chains. Some set of operators could refuse to participate, but this just means that the other operators will commit separately. We could even use a shared DBML/MMR contract.