This is my approach to a liquidator bot architecture and some considerations

Liquidator execution frequency

I understand that asset prices (markets) are updated once every 24 hours.
So I presume the following reasons affect (reduce) the user's margin ratio:

1. Asset price updates every 24 hours.
2. User trades that may cause the margin ratio to fall below the required margin (if allowed by the app).
3. Other unspecified reasons that I may not be aware of.

If my assumptions are correct, a continuous liquidator loop might not be necessary until such frequent updates (like every second) are implemented.

Liquidator architecture

off-chain

The fetching and filtering of margin accounts based on their risk status can be initiated only off-chain due to the constraints of Solana's execution environment. So it's ok as it is now.

on-chain

Trying to do as much as possible on-chain, the only thinking that I can come up with is to calculate if the account is eligible to liquidate on-chain.
Current data architecture means that is needed to provide the transaction with the 13 price_feeds (market price from pyth) and the 13 market addresses (market settings).
Of course this approach will incur in higher transaction costs. Thing is a new data structure, like 1 account with all price feeds stored and may be market settings too, could be used by the liquidator transactions and decrease compute units and execution time, as well as code complexity.

So the steps on-chain will be:

(Discuss about this approach feasibility)

1. Read margin account positions
2. Initialize price feeds
3. Initialize Market settings
4. Calculate required margin and available margin
5. Liquidation: Positions will be liquidated if the margin requirement is unmet

** on-chain architecture will be different if considering the bundles approach stated in the next page.*

CONSIDERATIONS

About solana congestion
High traffic and spam bots seem to be overwhelming some QUIC protocol implementations, affecting the network traffic but not the validators' runtime. This issue won't stop immediately; however, fortunately, almost 90% of validators have installed a new release (v1.17.31) that aims to resolve some of these issues until version 1.18 is ready and deployed.

Transaction execution
If there is another episode of big congestion, we will need to implement a firehose strategy. I would suggest adding a new member flag in the MarginAccount to be set to liquidation_check (true or false) so when the liquidation transaction succeeds it is set to true, and not modified till the owner adds more margin positions.
Subsequent launched transactions will check this parameter the first, so can be finalized earlier, resulting in a low compute units consumption.
As Account state is blocked sending many transactions would cause any data inconsistency, and this strategy will warranty that the first to be executed will totally liquidate the margin account.

Bundles approach
As this is the only known fast-lane to get a transaction to be prioritized, specially because we can add tips to the validator here, higher than the transaction priority fees, its a well known practice. Especially when the liquidation feasibility is calculated in real time using an oracle feed. All transactions are set in order inside a bundle to proceed to the liquidation at once when conditions are met.
It's probably the best approach if cost is not a problem, and if prices are quoted in real time.