Stake Delegation and Rewards

Stakers are rewarded for helping to validate the ledger. They do this by delegating their stake to validator nodes. Those validators do the legwork of replaying the ledger and sending votes to a per-node vote account to which stakers can delegate their stakes. The rest of the cluster uses those stake-weighted votes to select a block when forks arise. Both the validator and staker need some economic incentive to play their part. The validator needs to be compensated for its hardware and the staker needs to be compensated for the risk of getting its stake slashed. The economics are covered in[staking rewards](#) . This section, on the other hand, describes the underlying mechanics of its implementation.

Basic Design

The general idea is that the validator owns a Vote account. The Vote account tracks validator votes, counts validator generated credits, and provides any additional validator specific state. The Vote account is not aware of any stakes delegated to it and has no staking weight.

A separate Stake account(created by a staker)names a Vote account to which the stake is delegated. Rewards generated are proportional to the amount of lamports staked. The Stake account is owned by the staker only. Some portion of the lamports stored in this account are the stake.

Passive Delegation

Any number of Stake accounts can delegate to a single Vote account without an interactive action from the identity controlling the Vote account or submitting votes to the account.

The total stake allocated to a Vote account can be calculated by the sum of all the Stake accounts that have the Vote account pubkey as theStakeStateV2::Stake::voter_pubkey .

Vote and Stake accounts

The rewards process is split into two on-chain programs. The Vote program solves the problem of making stakes slashable. The Stake program acts as custodian of the rewards pool and provides for passive delegation. The Stake program is responsible for paying rewards to staker and voter when shown that a staker's delegate has participated in validating the ledger.

VoteState

VoteState is the current state of all the votes the validator has submitted to the network. VoteState contains the following state information:

- votes
-
    - The submitted votes data structure.
- credits
-
    - The total number of rewards this Vote program has generated over
- its lifetime.
- root_slot
-
    - The last slot to reach the full lockout commitment necessary for
- rewards.
- commission
-
    - The commission taken by this VoteState for any rewards claimed
- by staker's Stake accounts. This is the percentage ceiling of the reward.
- Account::lamports - The accumulated lamports from the commission. These do not
- count as stakes.
- authorized_voter
-
    - Only this identity is authorized to submit votes. This
- field can only modified by this identity.
- node_pubkey
-
    - The Solana node that votes in this account.
- authorized_withdrawer
-
    - the identity of the entity in charge of the lamports

- of this account, separate from the account's address and the authorized vote
- signer.

VoteInstruction::Initialize(VoteInit)

- account[0]
-
    - RW - The VoteState.
- VoteInit
- carries the new vote account'snode_pubkey
- ,authorized_voter
- ,authorized_withdrawer
- , andcommission
- .
- other VoteState members defaulted.

VoteInstruction::Authorize(Pubkey, VoteAuthorize)

Updates the account with a new authorized voter or withdrawer, according to the VoteAuthorize parameter(Voter orWithdrawer). The transaction must be signed by the Vote account's currentauthorized_voter orauthorized_withdrawer .

- account[0]
-
    - RW - The VoteState.VoteState::authorized_voter
- orauthorized_withdrawer
- is set toPubkey
- .

VoteInstruction::AuthorizeWithSeed(VoteAuthorizeWithSeedArgs)

Updates the account with a new authorized voter or withdrawer, according to the VoteAuthorize parameter(Voter orWithdrawer). UnlikeVoteInstruction::Authorize this instruction is for use when the Vote account's currentauthorized_voter orauthorized_withdrawer is a derived key. The transaction must be signed by someone who can sign for the base key of that derived key.

- account[0]
-
    - RW - The VoteState.VoteState::authorized_voter
- orauthorized_withdrawer
- is set toPubkey
- .

VoteInstruction::Vote(Vote)

- account[0]
-
    - RW - The VoteState.VoteState::lockouts
- andVoteState::credits
- are updated according to voting lockout rules seeTower BFT
- .
- account[1]
-
    - RO -sysvar::slot_hashes
- A list of some N most recent slots
- and their hashes for the vote to be verified against.
- account[2]
-
    - RO -sysvar::clock
- The current network time, expressed in
- slots, epochs.

StakeStateV2

A StakeStateV2 takes one of four forms, StakeStateV2::Uninitialized, StakeStateV2::Initialized, StakeStateV2::Stake, and StakeStateV2::RewardsPool. Only the first three forms are used in staking, but only StakeStateV2::Stake is interesting. All RewardsPools are created at genesis.

StakeStateV2::Stake

StakeStateV2::Stake is the current delegation preference of thestaker and contains the following state information:

- Account::lamports - The lamports available for staking.
- stake
- the staked amount(subject to warmup and cooldown)for generating
- rewards, always less than or equal to Account::lamports.
- voter_pubkey
- The pubkey of the VoteState instance the lamports are
- delegated to.
- credits_observed
- The total credits claimed over the lifetime of the
- program.
- activated
- the epoch at which this stake was activated/delegated. The full
- stake will be counted after warmup.
- deactivated
- the epoch at which this stake was de-activated, some cooldown
- epochs are required before the account is fully deactivated, and the stake
- available for withdrawal.
- authorized_staker
- the pubkey of the entity that must sign delegation,
- activation, and deactivation transactions.
- authorized_withdrawer
- the identity of the entity in charge of the lamports
- of this account, separate from the account's address, and the authorized
- staker.

StakeStateV2::RewardsPool

To avoid a single network-wide lock or contention in redemption, 256 RewardsPools are part of genesis under pre-determined keys, each with std::u64::MAX credits to be able to satisfy redemptions according to point value.

The Stakes and the RewardsPool are accounts that are owned by the sameStake program.

StakeInstruction::DelegateStake

The Stake account is moved from Initialized to StakeStateV2::Stake form, or from a deactivated (i.e. fully cooled-down) StakeStateV2::Stake to activated StakeStateV2::Stake. This is how stakers choose the vote account and validator node to which their stake account lamports are delegated. The transaction must be signed by the stake'sauthorized_staker .

- account[0]
- RW - The StakeStateV2::Stake instance.StakeStateV2::Stake::credits_observed
- is initialized toVoteState::credits
- ,StakeStateV2::Stake::voter_pubkey
- is initialized toaccount[1]
- . If this is the initial delegation of stake,StakeStateV2::Stake::stake
- is initialized to the account's balance in
- lamports,StakeStateV2::Stake::activated
- is initialized to the current Bank
- epoch, andStakeStateV2::Stake::deactivated
- is initialized to std::u64::MAX
- account[1]
- R - The VoteState instance.
- account[2]
- R - sysvar::clock account, carries information about current
- Bank epoch.
- account[3]
- R - sysvar::stakehistory account, carries information about

- stake history.
- account[4]
-
    - R - stake::Config account, carries warmup, cooldown, and
- slashing configuration.

StakeInstruction::Authorize(Pubkey, StakeAuthorize)

Updates the account with a new authorized staker or withdrawer, according to the StakeAuthorize parameter(Staker orWithdrawer). The transaction must be by signed by the Stakee account's currentauthorized_staker orauthorized_withdrawer . Any stake lock-up must have expired, or the lock-up custodian must also sign the transaction.

- account[0]
-
    - RW - The StakeStateV2.
- StakeStateV2::authorized_staker
- orauthorized_withdrawer
- is set to toPubkey
- .

StakeInstruction::Deactivate

A staker may wish to withdraw from the network. To do so he must first deactivate his stake, and wait for cooldown. The transaction must be signed by the stake'sauthorized_staker .

- account[0]
-
    - RW - The StakeStateV2::Stake instance that is deactivating.
- account[1]
-
    - R - sysvar::clock account from the Bank that carries current
- epoch.

StakeStateV2::Stake::deactivated is set to the current epoch + cooldown. The account's stake will ramp down to zero by that epoch, and Account::lamports will be available for withdrawal.

StakeInstruction::Withdraw(u64)

Lamports build up over time in a Stake account and any excess over activated stake can be withdrawn. The transaction must be signed by the stake'sauthorized_withdrawer .

- account[0]
-
    - RW - The StakeStateV2::Stake from which to withdraw.
- account[1]
-
    - RW - Account that should be credited with the withdrawn
- lamports.
- account[2]
-
    - R - sysvar::clock account from the Bank that carries current
- epoch, to calculate stake.
- account[3]
-
    - R - sysvar::stake_history account from the Bank that carries
- stake warmup/cooldown history.

Benefits of the design

- Single vote for all the stakers.
- Clearing of the credit variable is not necessary for claiming rewards.
- Each delegated stake can claim its rewards independently.
- Commission for the work is deposited when a reward is claimed by the delegated
- stake.

Example Callflow

Staking Rewards

The specific mechanics and rules of the validator rewards regime is outlined here. Rewards are earned by delegating stake to a validator that is voting correctly. Voting incorrectly exposes that validator's stakes toslashing .

Basics

The network pays rewards from a portion of networkinflation . The number of lamports available to pay rewards for an epoch is fixed and must be evenly divided among all staked nodes according to their relative stake weight and participation. The weighting unit is called apoint .

Rewards for an epoch are not available until the end of that epoch.

At the end of each epoch, the total number of points earned during the epoch is summed and used to divide the rewards portion of epoch inflation to arrive at a point value. This value is recorded in the bank in asysvar that maps epochs to point values.

During redemption, the stake program counts the points earned by the stake for each epoch, multiplies that by the epoch's point value, and transfers lamports in that amount from a rewards account into the stake and vote accounts according to the vote account's commission setting.

Economics

Point value for an epoch depends on aggregate network participation. If participation in an epoch drops off, point values are higher for those that do participate.

Earning credits

Validators earn one vote credit for every correct vote that exceeds maximum lockout, i.e. every time the validator's vote account retires a slot from its lockout list, making that vote a root for the node.

Stakers who have delegated to that validator earn points in proportion to their stake. Points earned is the product of vote credits and stake.

Stake warmup, cooldown, withdrawal

Stakes, once delegated, do not become effective immediately. They must first pass through a warmup period. During this period some portion of the stake is considered "effective", the rest is considered "activating". Changes occur on epoch boundaries.

The stake program limits the rate of change to total network stake, reflected in the stake program'sconfig::warmup_rate (set to 25% per epoch in the current implementation).

The amount of stake that can be warmed up each epoch is a function of the previous epoch's total effective stake, total activating stake, and the stake program's configured warmup rate.

Cooldown works the same way. Once a stake is deactivated, some part of it is considered "effective", and also "deactivating". As the stake cools down, it continues to earn rewards and be exposed to slashing, but it also becomes available for withdrawal.

Bootstrap stakes are not subject to warmup.

Rewards are paid against the "effective" portion of the stake for that epoch.

Warmup example

Consider the situation of a single stake of 1,000 activated at epoch N, with network warmup rate of 20%, and a quiescent total network stake at epoch N of 2,000.

At epoch N+1, the amount available to be activated for the network is 400(20% of 2000), and at epoch N, this example stake is the only stake activating, and so is entitled to all of the warmup room available.

epoch effective activating total effective total activating N-1

2,000 0 N 0 1,000 2,000 1,000 N+1 400 600 2,400 600 N+2 880 120 2,880 120 N+3 1000 0 3,000 0 Were 2 stakes(X and Y)to activate at epoch N, they would be awarded a portion of the 20% in proportion to their stakes. At each epoch effective and activating for each stake is a function of the previous epoch's state.

epoch X eff X act Y eff Y act total effective total activating N-1

2,000 0 N 0 1,000 0 200 2,000 1,200 N+1 333 667 67 133 2,400 800 N+2 733 267 146 54 2,880 321 N+3 1000 0 200 0 3,200 0

Withdrawal

Only lamports in excess of effective+activating stake may be withdrawn at any time. This means that during warmup, effectively no stake can be withdrawn. During cooldown, any tokens in excess of effective stake may be withdrawn(activating == 0). Because earned rewards are automatically added to stake, withdrawal is generally only possible after deactivation.

Lock-up

Stake accounts support the notion of lock-up, wherein the stake account balance is unavailable for withdrawal until a specified time. Lock-up is specified as an epoch height, i.e. the minimum epoch height that must be reached by the network before the stake account balance is available for withdrawal, unless the transaction is also signed by a specified custodian. This information is gathered when the stake account is created, and stored in the Lockup field of the stake account's state. Changing the authorized staker or withdrawer is also subject to lock-up, as such an operation is effectively a transfer.

Previous Managing Forks Next Synchronization