Blockstore in a Solana Validator

After a block reaches finality, all blocks from that one on down to the genesis block form a linear chain with the familiar name blockchain. Until that point, however, the validator must maintain all potentially valid chains, calledforks . The process by which forks naturally form as a result of leader rotation is described infork generation . Theblockstore data structure described here is how a validator copes with those forks until blocks are finalized.

The blockstore allows a validator to record every shred it observes on the network, in any order, as long as the shred is signed by the expected leader for a given slot.

Shreds are moved to a fork-able key space the tuple ofleader slot +shred index (within the slot). This permits the skip-list structure of the Solana protocol to be stored in its entirety, without a-priori choosing which fork to follow, which Entries to persist or when to persist them.

Repair requests for recent shreds are served out of RAM or recent files and out of deeper storage for less recent shreds, as implemented by the store backing Blockstore.

Functionalities of Blockstore

1. Persistence: the Blockstore lives in the front of the nodes verification
2. pipeline, right behind network receive and signature verification. If the
3. shred received is consistent with the leader schedule(i.e. was signed by the
4. leader for the indicated slot), it is immediately stored.
5. Repair: repair is the same as window repair above, but able to serve any
6. shred that's been received. Blockstore stores shreds with signatures,
7. preserving the chain of origination.
8. Forks: Blockstore supports random access of shreds, so can support a
9. validator's need to rollback and replay from a Bank checkpoint.
10. Restart: with proper pruning/culling, the Blockstore can be replayed by
11. ordered enumeration of entries from slot 0. The logic of the replay stage
12. (i.e. dealing with forks)will have to be used for the most recent entries in
13. the Blockstore.

Blockstore Design

1. Entries in the Blockstore are stored as key-value pairs, where the key is the concatenated slot index and shred index for an entry, and the value is the entry data. Note shred indexes are zero-based for each slot(i.e. they're slot-relative).
2. The Blockstore maintains metadata for each slot, in theSlotMeta
3. struct containing:
4.
    - slot_index
5.
    -
        - The index of this slot
6.
    - num_blocks
7.
    -
        - The number of blocks in the slot(used for chaining to a previous slot)
8.
    - consumed
9.
    -
        - The highest shred indexn
10.
    - , such that for allm < n
11.
    - , there exists a shred in this slot with shred index equal ton
12.
    - (i.e. the highest consecutive shred index).
13.
    - received
14.
    -
        - The highest received shred index for the slot
15.
    - next_slots

16.
    -
        - A list of future slots this slot could chain to. Used when rebuilding
17.
    - the ledger to find possible fork points.
18.
    - last_index
19.
    -
        - The index of the shred that is flagged as the last shred for this slot. This flag on a shred will be set by the leader for a slot when they are transmitting the last shred for a slot.
20.
    - is_connected
21.
    -
        - True iff every block from 0...slot forms a full sequence without any holes. We can derive is_connected for each slot with the following rules. Let slot(n)be the slot with indexn
22.
    - , and slot(n).is_full()is true if the slot with indexn
23.
    - has all the ticks expected for that slot. Let is_connected(n)be the statement that "the slot(n).is_connected is true". Then:
24.
    - is_connected(0)is_connected(n+1)iff(is_connected(n)and slot(n).is_full()
25. Chaining - When a shred for a new slotx
26. arrives, we check the number of blocks(num_blocks
27.)for that new slot(this information is encoded in the shred). We then know that this new slot chains to slotx - num_blocks
28. .
29. Subscriptions - The Blockstore records a set of slots that have been "subscribed" to. This means entries that chain to these slots will be sent on the Blockstore channel for consumption by the ReplayStage. See theBlockstore APIs
30. for details.
31. Update notifications - The Blockstore notifies listeners when slot(n).is_connected is flipped from false to true for anyn
32. .

Blockstore APIs

The Blockstore offers a subscription based API that ReplayStage uses to ask for entries it's interested in. These subscription API's are as follows:

1. fn get_slots_since(slots: &[u64]) -> Result<>
2. : Returns slots that are connected to any of the elements ofslots
3. . This method enables the discovery of new children slots.
4. fn get_slot_entries(slot: Slot, shred_start_index: u64) -> Result<
5. : For the specifiedslot
6. , return a vector of the available, contiguous entries starting fromshred_start_index
7. . Shreds are fragments of serialized entries so the conversion from entry index to shred index is not one-to-one. However, there is a similar functionget_slot_entries_with_shred_info()
8. that returns the number of shreds that comprise the returned entry vector. This allows a caller to track progress through the slot.

Note: Cumulatively, this means that the replay stage will now have to know when a slot is finished, and subscribe to the next slot it's interested in to get the next set of entries. Previously, the burden of chaining slots fell on the Blockstore.

Interfacing with Bank

The bank exposes to replay stage:

1. prev_hash
2. : which PoH chain it's working on as indicated by the hash of the last entry it processed
3. tick_height
4. : the ticks in the PoH chain currently being verified by this bank
5. votes
6. : a stack of records that contains:
7.
    - prev_hashes
8.
    - : what anything after this vote must chain to in PoH

9.
    - tick_height
10.
    - : the tick height at which this vote was cast
11.
    - lockout period
12.
    - : how long a chain must be observed to be in the ledger to be able to be chained below this vote

Replay stage uses Blockstore APIs to find the longest chain of entries it can hang off a previous vote. If that chain of entries does not hang off the latest vote, the replay stage rolls back the bank to that vote and replays the chain from there.

Pruning Blockstore

Once Blockstore entries are old enough, representing all the possible forks becomes less useful, perhaps even problematic for replay upon restart. Once a validator's votes have reached max lockout, however, any Blockstore contents that are not on the PoH chain for that vote for can be pruned, expunged. Previous Validator's Transaction Processing Unit (TPU) Next Validator's Transaction Validation Unit (TVU)