Storage Staking

When you deploy a smart contract to NEAR, you pay for the storage that this contract requires using a mechanism called storage staking.

In storage staking (sometimes calledstate staking), the account that owns a smart contract must stake (or lock) tokens according to the amount of data stored in that smart contract, effectively reducing the balance of the contract's account. Coming from Ethereum?

If you're familiar with Ethereum's pricing model, you may know that, like NEAR, the protocol charges a fee (called "gas") for each transaction. Unlike NEAR, Ethereum's gas fee accounts for the amount of data stored via that transaction. This essentially means that anyone can pay once to store permanent data on-chain. This is a poor economic design for at least two reasons: 1. The people running the network (miners, in the case of Ethereum 1) are not appropriately incentivized to store large amounts of data, since a gas fee far charged in the distant past can increase storage costs forever, and 2. The users of a smart contract are charged for the data they send to store in it, rather than charging the owner of the smart contract.

How does NEAR's design align incentives?

Storage-staked tokens are unavailable for other uses, such as validation staking. This increases the yield that validators will receive. Learn more inthe economics whitepaper .

When do tokens get staked?

On each incoming transaction that adds data.

Let's walk through an example:

1. You launcha guest book app
2. , deploying your app's smart contract to the accountexample.near
3. Visitors to your app can add messages to the guest book. This means your users willby default
4. , pay a small gas fee to send their message to your contract.
5. When such a call comes in, NEAR will check thatexample.near
6. has a large enough balance that it can stake an amount to cover the new storage needs. If it does not, the transaction will fail.

The "million cheap data additions" attack

Note that this can create an attack surface. To continue the example above, if sending data to your guest book costs users close to nothing while costing the contract owner significantly more, then a malicious user can exploit the imbalance to make maintaining the contract prohibitively expensive.

Take care, then, when designing your smart contracts to ensure that such attacks cost potential attackers more than it would be worth.

btw, you can remove data to unstake some tokens

People familiar with the "immutable data" narrative about blockchains find this surprising. While it's true that anindexing node will keep all data forever,validating nodes (that is, the nodes run by most validators in the network) do not. Smart contracts can provide ways to delete data, and this data will be purged from most nodes in the network within a fewepochs .

Note that a call to your smart contract to remove data has an associated gas fee. Given NEAR's gas limit, this creates an upper limit on how much data can be deleted in a single transaction.

How much does it cost?

Storage staking is priced in an amount set by the network, which is set to1E19 yoctoNEAR per byte , or100kb per NEAR token (Ⓝ) .1 2

NEAR's JSON RPC API providesa way to query this initial settingas well as aa way to query the live config / recent blocks.

Example cost breakdown

Let's walk through an example.

Anon-fungible token is unique, which means each token has its own ID. The contract must store a mapping from token IDs to owners' account ID.

If such an NFT is used to track1 million tokens, how much storage will be required for the token-ID-to-owner mapping? And how many tokens will need to be staked for that storage?

Let's calculate the storage needs when using aPersistentMap that stores data as UTF-8 strings.

Here's ourPersistentMap :

type

AccountId

=

string ; type

TokenId

= u64 ; const tokenToOwner =

new

PersistentMap < TokenId ,

AccountId

("t2o") ; Behind the scenes, all data stored on the NEAR blockchain is saved in a key-value database. That't2o' variable that's passed toPersistentMap helps it keep track of all its values. If your accountexample.near owns token with ID0 , then at the time of writing, here's the data that would get saved to the key-value database:

- key:t2o::0
- value:example.near

So for 1 million tokens, here are all the things we need to add up and multiply by 1 million:

1. The prefix,t2o
2. , will be serialized as three bytes in UTF-8, and the two colons will add another two. That's 5 bytes.
3. For an implementation whereTokenId
4. auto-increments, the values will be between0
5. and999999
6. , which makes the average length 5 bytes.
7. Let's assume well-formed NEARAccountId
8. s, and let's guess that NEAR Account IDs follow the approximate pattern of domain names, whichaverage about 10 characters
9. , plus a top-level name like.near
10. . So a reasonable average to expect might be about 15 characters; let's keep our estimate pessimistic and say 25. This will equal 25 bytes, since NEAR account IDs must use characters from the ASCII set.

So:

1_000_000 * (5 + 5 + 25)

35 million bytes. 350 times 100Kib, meaning Ⓝ350. To do the exact math: Multiplying by 1e19 yoctoNEAR per byte, we find that thetokenToOwner mapping with 35m bytes will require staking 3.5e26 yoctoNEAR, or Ⓝ350

Note that you can get this down to Ⓝ330 just by changing the prefix fromt2o to a single character. Or get rid of it entirely! You can have a zero-length prefix on onePersistentVector in your smart contract. If you did that with this one, you could get it down to Ⓝ250.

Calculate costs for your own contract

Doing manual byte math as shown above is difficult and error-prone. Good news: you don't have to!

You can test the storage used using theSDK environment and checkingenv.storage_usage()

Other ways to keep costs down

Storing data on-chain isn't cheap for the people running the network, and NEAR passes on this cost to developers. So, how do you, as a developer, keep your costs down? There are two popular approaches:

1. Use a binary serialization format, rather than JSON
2. Store data off-chain

Use a binary serialization format, rather than JSON

The core NEAR team maintains a library calledborsh , which is used automatically when you usenear-sdk-rs . Someday, it will probably also be used bynear-sdk-js .

Imagine that you want to store an array like[0, 1, 2, 3] . You could serialize it as a string and store it as UTF-8 bytes. This is whatnear-sdk-js does today. Cutting out spaces, you end up using 9 bytes.

Using borsh, this same array gets saved as 8 bytes:

\u0004\u0000\u0000\u0000\u0000\u0001\u0002\u0003

At first glance, saving 1 byte might not seem significant. But let's look closer.

The first four bytes here,\u0004\u0000\u0000\u0000 , tell the serializer that this is au32 array of length4 using little-endian encoding. The rest of the bytes are the literal numbers of the array –\u0000\u0001\u0002\u0003 . As you serialize more elements, each will add one byte to the data structure. With JSON, each new element requires adding two bytes, to represent both another comma and the number.

In general, Borsh is faster, uses less storage, and costs less gas. Use it if you can.

Store data off-chain

This is especially important if you are storing user-generated data!

Let's use thisGuest Book as an example. As implemented today, visitors to the app can sign in with NEAR and leave a message. Their message is stored on-chain.

Imagine this app got very popular, and that visitors started leaving unexpectedly long messages. The contract owner might run out of funding for storage very quickly!

A better strategy could be to store data off-chain. If you want to keep the app decentralized, a popular off-chain data storage solution isIPFS . With this, you can represent any set of data with a predictable content address such as:

QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG

Such a content address could represent a JSON structure or an image or any other type of data. Where does this data get physically stored? You could useFilecoin or run your own IPFS server to pin your app's data.

With this approach, each record you add to your contract will be a predictable size.

Summary

NEAR's structure incentivizes network operators while giving flexibility and predictability to contract developers. Managing storage is an important aspect of smart contract design, and NEAR's libraries make it easy to figure out how much storage will cost for your application.

Got a question? Ask it on StackOverflow!

Footnotes

Footnotes

1. Storage staking price
2. ↩
3. Lower storage cost 10x
4. ↩ Edit this page Last updatedonMar 25, 2024 bygagdiez Was this page helpful? Yes No

Previous Storage on NEAR Next Alternative Solutions