This post describes a mechanism to implement Storage Rent in a way that improves predictability of storage costs and any user who wants to use a deleted contract can just pay and revive the data needed without reviving the complete contract storage.

TLDR: The overview of this approach is that a user deploys a contract for a certain time period, everyone who is storing some data on the contract will pay for the data for that period. It will be possible to extend the time period of the contract but that involves paying for everything stored on the contract for the extension period. Once the contract expires, anyone can send Merkle proof of data and resurrect the part of the contract state that is needed. The first user to resurrect partly or complete contract storage also has to pay for the contract code resurrection.

Additional Requirements

For the proposed model, the following are the prerequisites needed.

1. Rent_Valid_Till

and Base_Revival_Period

parameter is to be added to the account information that is part of the state.

1. Revive

transaction that helps in reviving a part of a contract or complete contract.

1. Repurposing SELFDESTRUCT

opcode to delete the contract data and return the storage costs for rest of Rent_Period

to the specified address.

1. EXTEND

opcode which helps extend the Rent_Valid_Till

of the contract.

How is contract deployed?

Two additional parametersRent_Period

and Base_Revival_Period

are to be specified during contract deployment. Rent_Period

is the time(in blocks) for which rent has to be paid while deployment. Any data that is added to the contract at a later point will only be paid for and valid till Rent_Valid_Till

block. Base_Revival_period

is the default time for which rent has to be paid when reviving the contract.

When a contract deployment transaction is added to the blockchain, the account data in the state has Rent_Valid_Till

field and Base_Revival_Period

which are respectively initialized with the Rent_Period

and Base_Revival_Period

specified in the transaction. Pay_Rent

transaction is used to extend the time for which the contract is alive.

About Rent

There are 2 types of rent in this mechanism

Extending lease

This type of rent is paid in case the rent period of the contract is to be extended. This is a function of the storage consumed until then.

Paying for the lease

This type of rent is paid to store additional data(via a Tx to contract) for the remaining rent period. This is a function of extra storage due to a tx and charged for the rental period of the contract.

How is rent paid?

The rent will be paid as gas but the gas used for the storage rent will not be considered towards the gas limit. When a contract is deployed additional gas is sent along with the transaction to account for storage rent based on specified Rent_Period

. Thereafter any transaction to the contract should have enough gas to account for the storage costs for the additional data due to the transaction for the remainder of Rent_Period

of the contract. Similar to gas price

there will be a storage price

which is used in calculating the storage rent. storage price

is to be decided periodically based on the decrease in storage costs over time. The Pay Rent

transaction takes contract address and Rent_Period

as arguments and increases the Rent_Valid_Till

of any contract if sufficient gas is received to cover the storage costs of the contract for Rent_Period

at the time Pay Rent

transaction is executed.

Most of the ether paid for storage rent should be burnt and a small reward is given to the miner.

How is a contract deleted?

There are 2 ways in which contract data can be deleted.

1. Using the SELFDESTRUCT

opcode in the contract.

1. If the Rent_Period

is completed and the Rent_Valid_Till

block is reached for the contract. The account information and the storage data of the contract will be deleted. Any other transaction to the contract that is expired(but before deleting contract data) will result in failure and contract data gets deleted.

In the first case, SELFDESTRUCT

is used to delete the contracts subject to the minimum Rent_Period

restriction(explained below) and the remaining storage costs are refunded to the user. As the rent that is paid when data is added is being burnt, so the simplest way to provide refund is minting the reward.

In the second case to identify the contract that is to be deleted, we set up a small reward for anyone who identifies that a contract is about to expire. This small reward should be less than the base cost needed to deploy a contract. The main issue with this is that it is prone to front-running. So a miner can see the pending deletion transaction and then create a new transaction with the same data and claim the reward. This can be solved using a slightly modified commit-reveal approach. The commit phase consists of sending a deletion transaction where hash(claimerAddress, contractToBeDeleted, deletionBlock)

is to be submitted. This commit phase can start 50 blocks before the contract deletion. Reveal phase starts right at the block at which the contract should get deleted. In the reveal phase, the committer can reveal the address of the contract to be deleted and get the reward. This reveal phase lasts for next 50 blocks. If the deletion didn't happen in the last commit - reveal cycle. A new commit-reveal cycle starts again which lasts for the next 100 blocks and so on till the data gets deleted. Note that in this approach, no one needs to track this commit-reveal process before it is revealed that the contract is to be deleted.

How to revive a contract after getting deleted?

We can revive the contract after it is deleted by submitting the Merkle proof for the account information of the contract as well as state data of the contract for the last block at which Rent was valid. We also have to make sure that the Merkle proof submitted corresponds to the state when the contract was last deleted and does not accept any stale information. This can be achieved if we solve 2 problems

1. How do you make sure that provided proof is the last state of the contract before deleting?

2. How do you make sure that it has not already been revived?

The first problem can be solved by providing the Merkle proof of the account information which contains Rent_Valid_Till

field. Now that we know the block at which deletion happened, we can match the state root at that block(which is part of the proof) with the Rent_Valid_Till

block.

To solve the second problem we can use an approach suggested in aprevious post by Vitalik. The contract can be revived by submitting Merkle proof of the account information of the contract, state data of the contract and a series of Merkle branches showing that the data was not revived after proof was provided. The minimum Rent_Period

can be about 1 week. Then a series of Merkle proofs can be provided with 1-week gaps in between proving that the contract was never revived in between.

There may be contracts which are too huge to be revived in a single block. It is possible to revive such contracts in multiple transactions in different blocks. The deleted contracts are always revived for Base_Revival_Period

and then can be further extended. The extension of the contract is to be controlled using the EXTEND

opcode as extending contract unconditionally will result in no one being able to use the contract.

These partial revivals improve the UX as every user has to just pay for data that they use. So every time they use some data, it stays for Base_Revival_Period

unless the contract time is not extended and then someone revives the data needed for them to use for Base_Revival_Period

time. The first user has to pay for the fee for the code. As the code is bounded by the gas limit, fee user has to pay is also bounded.

Questions:

- Is there a need for restriction on the maximum rent period?
- How is the storage gas price decided?