Improving Capital Efficiency in WALRUS

v1(based on testnet) - Feb 2025

walqua.xyz
walqua.xyz@gmail.com

1 INTRODUCTION

WALRUS leverages contracts on the SUI blockchain to facilitate payment for Blob storage. When creating a blob, a storage object with capacity at least equal to the size of the blob—and secured for a minimum duration of one epoch—must be provisioned. Each epoch, the committee determines both the available capacity and the cost per unit of storage; the WAL tokens paid as fees are then distributed as rewards to the committee's nodes. In the case of deletable blobs, deleting a blob returns remaining storage, which can then be reused.

WALRUS requires a system that aims to maximize capital efficiency. The storage price is continuously adjusted by the committee, and the price of WAL itself is subject to market fluctuations. Rather than leaving reserved storage idle, holders can sell unused capacity to recover WAL or rent it to other users, thus creating multiple opportunities for capital optimization. Furthermore, if the current market price for storage exceeds the initial purchase price, the sale of reserved capacity may yield a profit.

2 DESIGN & IMPLEMENTATION

Since storage can be purchased through the Walrus contract, its price in the secondary market is expected to be lower than the original cost unless capacity is fully utilized. In the current epoch, purchasing storage for one epoch through the Walrus contract requires paying the full cost of that epoch, regardless of the remaining duration. This pricing mechanism may be perceived as inequitable. Therefore, the secondary market should be incentivized to apply time-based discounts, ensuring that prices decrease in proportion to the elapsed time.

The price of storage for the next epoch is determined once the current committee's voting process is completed. During the voting period, it is possible to predict the upcoming storage price by analyzing the parameters of the participating nodes.

Let P represent the storage price in the current epoch, P' the storage price in the next epoch, and $p_e()$ the regular price that users believe will be the price of storage for $current\_epoch + e$, with E representing the epoch duration of one week and t denoting the current time.

$$p_0(t) = P - \frac{P}{E}(t\%E) \tag{1}$$

$$p_e(t) = min(P, P')(e \geq 1) \tag{2}$$

In the case of p_0, its value depreciates over time. When creating a new blob in the current epoch, if acquiring it from the Walrus contract is more expensive than purchasing from an existing holder, it becomes a rational choice for the buyer, ensuring a guaranteed economic advantage. Simultaneously, for the seller, selling the asset before its value fully decays is beneficial, optimizing their returns.

Each epoch maintains a dedicated storage bucket, with a corresponding orderbook facilitating efficient trading. This structure streamlines transactions, allowing DApps to seamlessly utilize storage through programmable transaction blocks, enhancing usability and integration within the system.

Algorithm 1: selling a storage

```
Input: storage, price
for $i \leftarrow start\_epoch$ to $end\_epoch - 1$ do
    buckets[i].merge(storage.split(i, i+1))
    orderbooks[i].place(LIMIT, ASK, storage.size, price)
```

Algorithm 2: buying a storage

```
Input: storage epoch, size, price
Output: storage
size = orderbooks[storage_epoch].place(IOC, BID, size,
  price)
return buckets[i].split(size)
```

Algorithm 3: epoch change

```
orderbooks.pop_front()
orderbooks.append(new orderbook)
buckets.pop_front()
buckets.append(new bucket)
```

Algorithm 4: example of usage in dapp

```
Input: size, duration
storage = buy(0, size, reasonable price)
blob = register_blob(storage,size)
for $i \leftarrow 1$ to $duration - 1$ do
    if can_buy(i, size) then
        storage = buy(i, size, reasonable price)
        extend_blob_with_resource(blob, storage)
    else
        extend_blob(blob, 1, regular price)
```

3 LIMITATION

The implementation optimized for minimizing gas costs and the actual demand from users and DApps have not yet been fully validated.