Enhanced Transaction Sequencing for Modular Distributed Ledgers

Can Umut Ileri, *IOTA Foundation*, canumut.ileri@iota.org
Andrew Cullen, *IOTA Foundation*, andrew.cullen@iota.org
Olivia Saa, *IOTA Foundation*, olivia.saa.cullen@iota.org
Roman Overko, *IOTA Foundation*, roman.overko@iota.org
Luigi Vigneri, *IOTA Foundation*, luigi.vigneri@iota.org

***Abstract*—We introduce an improved congestion control algorithm for managing transactions that access shared objects in distributed ledgers utilizing the Move Virtual Machine, such as Sui and IOTA Rebased. The current congestion control algorithm prioritizes fairness by favoring high-paying transactions, but this often comes at the expense of throughput due to an unnecessarily high number of transaction deferrals. To address this, we propose a more rigorous definition of fairness that reduces excessive deferrals without compromising the execution efficiency of non-concurrent transactions. Building on this refined fairness property, our algorithm can achieve higher throughput and lower latency while preserving this new notion of fairness and ensuring that the load on the busiest execution worker remains unchanged.**

***Index Terms*—transaction sequencing, distributed ledgers, congestion control**

I. Introduction

In modern distributed ledgers [1]–[5], consensus protocols tend to determine the set of transactions to be processed without enforcing an order on their execution. While this design improves the speed and scalability of consensus, it necessitates several post-consensus steps, which typically include sequencing (ordering transactions and managing congestion), scheduling (assigning transactions to execution workers), and executing transactions. Among these, sequencing is particularly important as it directly impacts system performance while trying to ensure fairness among transactions.

A key challenge in sequencing arises from the concept of shared objects, which are data entities that can be accessed and modified by anyone. This powerful flexibility enabled by shared objects, however, creates a critical challenge: ordering transactions that access the same shared object, since execution outcome of non-concurrent transactions depend on the order they are processed. Efficient and fair sequencing of such transactions is vital for overall system performance.

The current sequencing algorithm in Sui[1] and IOTA[2] prioritizes fairness by strictly favoring high-paying transactions. However, as explained in Section II, this approach often leads to an unnecessarily high number of deferrals, which in turn reduces throughput and increases latency. This trade-off stems from a simplistic fairness definition that overlooks the efficient handling of non-concurrent transactions.

[1]https://github.com/MystenLabs/sui
[2]https://github.com/iotaledger/iota

To address this limitation, we propose a refined definition of fairness that minimizes transaction deferrals without compromising execution efficiency in Section III. Building on this improved definition, we introduce an enhanced sequencing algorithm that defers fewer transactions without changing the load of the busiest execution worker.

II. Current Sequencing Algorithm

In the current implementation of Sui mainnet and IOTA testnet, the sequencing phase begins by ordering transactions based on their gas price after they have been approved by the fault-tolerant consensus algorithm [1]. These ordered transactions are then evaluated sequentially using AlgSeq (Algorithm 1), which determines whether each transaction should proceed to execution or be delayed.[3] Delayed transactions are incorporated into the subsequent consensus round's transaction batch, with the risk of cancellation if the batch size exceeds system limits. Importantly, AlgSeq preserves the gas price ordering established in the initial phase.

Algorithm 1 Current sequencing algorithm

procedure AlgSeq($\mathcal{T}$: tx set, L: congestion limit)
  $c_o \leftarrow 0$ for all object $o \in \mathcal{O}$
  $\mathcal{T}^* \leftarrow \mathrm{sort}(\mathcal{T})$ by gas price
  for $t \in \mathcal{T}^*$ **do**
    $c^* \leftarrow \max_{o \in \mathcal{O}_t} c_o$ ▷ $\mathcal{O}_t$ is t's input objects
    if $c^* + 1 > L$ **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{t\}$ ▷ Set of deferred txs
    else
      $c_o \leftarrow c^* + 1, \quad \forall o \in \mathcal{O}_t$
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{t\}$ ▷ Set of accepted txs
  return $\{\mathcal{A}, \mathcal{D}\}$

The primary function of AlgSeq is to selectively filter transactions to maintain a manageable load on the execution phase. The algorithm is designed to satisfy two properties:

1) **Critical sequential load (*PropSeqLoad*)**: For any accepted transaction t, given sufficient parallel processing capacity, execution must start before time $L-1$, where L represents a protocol-defined configuration parameter. This property ensures that the load on the busiest execution worker is limited by L.

[3]For simplicity, and without loss of generality, we assume in this paper that the estimated execution cost for all transactions is 1.

2) **Gas price strong fairness (*PropStrongFairness*)**: Given a transaction sequence ordered by gas price, for any pair of transactions that touch[4] a common shared object, the transaction with higher gas price must execute first.

III. Improvement Algorithm

We begin by demonstrating that AlgSeq's deferral decisions can lead to violations of *PropStrongFairness* across multiple sequencing rounds, even though this property is satisfied within the round.

Example 1: Consider a system where $L = 2$ with four transactions ordered by descending gas prices: tx_1 accessing only object a, tx_2 accessing both objects a and b, tx_3 accessing both objects b and c, and tx_4 accessing object c. AlgSeq will schedule tx_1, tx_2 and tx_4, while deferring tx_3, despite tx_3 paying more than tx_4. Notably, all transactions could be scheduled without violating Property *PropSeqLoad*. This scenario demonstrates that the possible scheduling of tx_3 in a later round violates *PropStrongFairness*.

We propose that by refining *PropStrongFairness*, we can develop an algorithm that schedules more transactions while maintaining Property *PropSeqLoad*'s guarantees.

- **Gas price weak fairness (*PropWeakFairness*)**: Given a transaction sequence ordered by gas price, for any pair of transactions tx_i and tx_j that touch a common shared object, if tx_i pays more than tx_j and is scheduled later than tx_j, then tx_i must be accessing at least one object not accessed by tx_j.

We propose the following algorithm for sequencing.

Algorithm 2 Improved sequencing algorithm

```
1:  procedure AlgImprSeq(T: tx set, L: congestion limit)
2:      s_{o,l} ← 0 for all object o ∈ O and l ∈ [1, L]
3:      for each transaction t, in the order provided, do
4:          for l ∈ [1, L] do
5:              if s_{o,l} = 0   ∀ o ∈ O_t then
6:                  s_{o,l} ← 1   ∀ o ∈ O_t
7:                  A ← A ∪ {t}              ▷ Set of accepted txs
8:                  break
9:          if t ∉ A then
10:             D ← D ∪ {t}                  ▷ Set of deferred txs
        return {A, D}
```

Lemma 1. AlgImprSeq *satisfies PropWeakFairness.*

Proof. Let $\mathcal{O}_t$, p_t and g_t represent the set of input objects, the scheduling position, and the gas price of transaction, respectively. *PropWeakFairness* states that for any two transactions i and j, if $p_j < p_i$ and $g_j < g_i$, then $\mathcal{O}_i \not\subseteq \mathcal{O}_j$. To establish lemma, we will prove $p_j < p_i \wedge g_j < g_i \implies \mathcal{O}_i \not\subseteq \mathcal{O}_j$.

As transactions are processed in descending gas price order, i is processed before j. Transaction i is assigned the minimum execution slot l where $s_{o,l} = 0$ for all $o \in \mathcal{O}_i$, after which $s_{o,l} \leftarrow 1$ for all $o \in \mathcal{O}_i$. Suppose j is then assigned to an earlier slot $p_j < p_i$. We claim $\mathcal{O}_i \not\subseteq \mathcal{O}_j$ must hold. If $\mathcal{O}_i \subseteq \mathcal{O}_j$, then the condition $s_{o,l} = 0$ for all $o \in \mathcal{O}_i$ would be at least as restrictive as for j. Since j is processed after i (as $g_j < g_i$), it would necessarily be assigned a later slot, contradicting $p_j < p_i$. Thus, $p_j < p_i \wedge g_j < g_i \implies \mathcal{O}_i \not\subseteq \mathcal{O}_j$. □

Lemma 2. *Any transaction accepted by* AlgSeq *will also be accepted by* AlgImprSeq.

Proof. The only condition to defer a transaction in AlgSeq is in Line 6, and depends on c^* and L. Since L is a constant, deferment of a transaction in AlgSeq depends solely on c^*. When evaluating a transaction t in AlgImprSeq, the value of c^* used in AlgSeq can be expressed as: $c^* = \max\left(\operatorname{argmax}_{l\in[1,L]} s_{o,l} \quad \forall o \in \mathcal{O}_t\right)$. If t is accepted by AlgImprSeq at a slot $l < c^*$, then c^* remains unchanged in the subsequent evaluation. An accepted transaction can only modify c^* when $l = c^*$. Now consider the case where t is deferred by AlgImprSeq. Then, there must exist at least one object $o \in \mathcal{O}_t$ such that $s_{o,L} = 1$, meaning $c^* = L$. In this case t would also be deferred by AlgSeq. Therefore, any transaction deferred by AlgImprSeq would also be deferred by AlgSeq. Hence, any transaction accepted by AlgSeq will also be accepted by AlgImprSeq. □

From Lemma 2, we can establish:

Lemma 3. AlgImprSeq *achieves throughput at least as high as* AlgSeq *for any given sequence of transactions.*

Theorem 4. *Given the same inputs and same post-consensus processing of transactions,* AlgImprSeq *improves throughput over* AlgSeq*, without violating fairness.*

Proof. By Property *PropSeqLoad*, AlgImprSeq does not worsen the expected execution time, while possibly increasing the number of sequenced transactions, as stated in Lemma 3. Fairness is preserved in two key ways: all transactions accepted by AlgSeq are guaranteed to be accepted as shown in Lemma 2, and transactions with overlapping objects are still executed in the order of gas price, as ensured by Lemma 1. □

To illustrate Theorem 4 we use Example 1, where AlgImprSeq successfully executes all four transactions while ensuring that the highest sequential load remains within the allowed limit, assuming sufficient parallelization. It is worth noting that AlgImprSeq does not necessarily preserve the transaction ordering enforced by AlgSeq, which may result in different execution outcomes between the two algorithms.

IV. Conclusion

We presented an improved sequencing algorithm for the current implementation of Sui mainnet and IOTA testnet. The algorithm achieves higher throughput through refined fairness requirements while maintaining system guarantees.

[4]We define *touch* as any access operation on an object that is subject to at least one *write* operation within the same transaction batch.

References

[1] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Mysticeti: Reaching the limits of latency with uncertified dags," *arXiv preprint arXiv:2310.14821*, 2023.

[2] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[3] S. Müller, A. Penzkofer, N. Polyanskii, J. Theis, W. Sanders, and H. Moog, "Tangle 2.0 leaderless nakamoto consensus on the heaviest dag," *IEEE Access*, vol. 10, pp. 105 807–105 842, 2022.

[4] S. Müller, A. Penzkofer, N. Polyanskii, J. Theis, W. Sanders, and H. Moog, "Reality-based utxo ledger," *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 3, pp. 1–33, 2023.

[5] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. R. Rain, S. Sezer *et al.*, "Move: A language with programmable resources," *Libra Assoc*, p. 1, 2019.