TD-ORDERER: Transaction Dependency Orderer

Kyuin Jhi
dept. of Computer Science and Engineering
Incheon National University
Incheon, Republic of Korea
hnn991206@inu.ac.kr

Gi Seok Park
dept. of Computer Science and Engineering
Incheon National University
Incheon, Republic of Korea
gspark@inu.ac.kr

***Abstract*— As blockchain networks scale, ensuring efficient and fair transaction processing becomes increasingly challenging. Existing systems like Hyperledger Fabric rely on a centralized orderer that lacks full knowledge of transaction intent, leading to failures and unfair execution. We propose a transaction ordering system that replaces or complements the orderer in Hyperledger Fabric. Our system analyzes the internal intent of transactions to minimize conflicts, provides early feedback on likely failures, and considers user fairness during ordering. This approach reduces transaction failure rates and mitigates strategic behaviors, improving both system efficiency and user experience in permissioned blockchain environments.**

Keywords—Blockchain, Hyperledger fabric, Transaction.

I. INTRODUCTION

Blockchain systems operate in a distributed environment where each node maintains a consistent copy of the ledger. As the number of nodes increases, the time required for consensus and synchronization grows, leading to scalability challenges. To address this, various blockchain platforms have introduced architectural innovations to improve scalability.

Hyperledger Fabric [1] takes a different approach compared to traditional public blockchains by introducing an orderer component that determines the sequence of transactions. Transactions sent by users are received by the orderer, ordered by arrival time, and then packaged into blocks. These blocks are distributed to peer nodes, which independently validate each transaction for correctness. This architecture follows an order-execute-validate model, which improves throughput and supports scalability across large networks.

However, this structure also presents several issues. In particular, the actual effects of a transaction—such as which keys are read or written—are not fully known until the validation phase at the peer nodes. As a result, the orderer organizes transactions without complete knowledge of their intent or potential conflicts. Although this design improves performance, it opens the door to front-running attacks [2], where malicious users may infer the purpose of a transaction—based on metadata or endorsement requests—and submit conflicting transactions that reach the orderer first. In such cases, the original transaction may be invalidated, leading to reduced user satisfaction and concerns about fairness.

To address these challenges, we propose a transaction ordering system designed to replace the existing orderer in Hyperledger Fabric. First, it analyzes the internal intent of each transaction to minimize conflicts and reduce the likelihood of failures. Second, it improves user experience by providing early feedback for transactions that are predicted to fail. Third, to ensure fairness among users, the system monitors transaction outcomes and adjusts the ordering process to prevent any participant from consistently gaining an advantage due to network conditions or behavioral patterns. This approach aims to enhance both the efficiency and fairness of transaction processing.

II. PROPOSED SYSTEM

Fig.1 illustrates the overall system architecture, which consists of three main components: Kafka [3], Watchdog [4], and TD-Orderer. These components collaboratively provide a conflict-aware and fairness-preserving ordering service for the Hyperledger Fabric network. Kafka serves as a distributed message queue that receives continuous streams of transactions from Fabric peers. It decouples transaction submission from ordering by first collecting transactions in its input queue, ensuring smooth ingestion without blocking the ordering logic. The TD-Orderer replaces Fabric's default ordering service and extends the RWSet structure with additional metadata such as the transaction invoker, transaction type, and urgency level. This urgency field enables prioritization of time-critical operations, such as certain financial transactions.

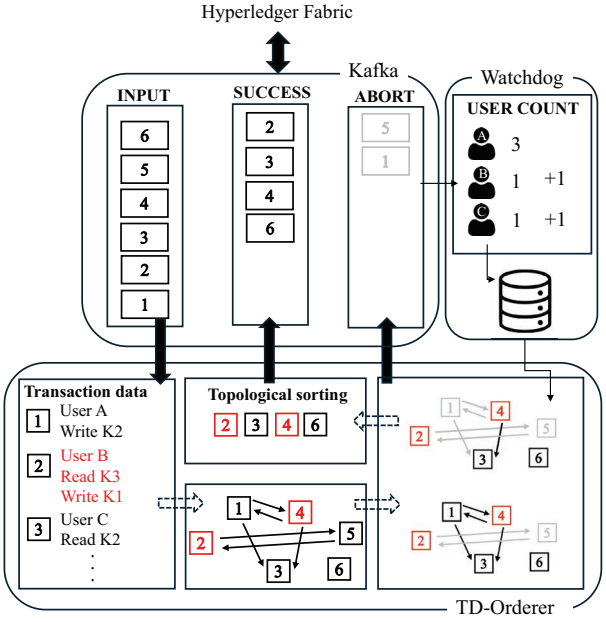

Fig1. System Architecture

Upon receiving batches of transactions from Kafka, the TD-Orderer analyzes overlaps in read and write sets to construct a dependency graph. A directed edge is added whenever one transaction writes a key that another reads, indicating a potential read-after-write conflict. The graph is then processed using topological sorting to produce an execution order that minimizes conflicts. This conflict-aware ordering ensures that no transaction within a block reads keys modified by earlier transactions in the same block, thereby reducing abort rates.

To illustrate the benefit of conflict-aware ordering, consider three transactions: T1 reads key A and writes key B; T2 reads key B and writes key C; T3 reads key C and writes key A. Executing them in the original sequence (T1 → T2 → T3) creates a cyclic dependency that leads to aborts. However, by reordering them as (T3 → T1 → T2), the system breaks this cycle, ensuring conflict-free execution within the block. This example highlights how the proposed ordering method effectively reduces abort rates and enhances transaction throughput.

Despite these measures, some transactions may still fail due to dynamic conflicts. Such transactions are moved to the abort queue, where the Watchdog component monitors them continuously. Watchdog records user-specific failure rates and stores this information in a distributed, tamper-resistant manner to uphold fairness. High abort rates influence future ordering decisions by deprioritizing transactions from certain users or avoiding scheduling conflicting transactions together. These fairness controls prevent resource monopolization and ensure equitable network access.

III. Experimental Results

A. Setup

The system is partially deployed in a Docker environment. Kafka is configured with three brokers and three ZooKeeper nodes, while Watchdog consists of validation nodes and consensus nodes responsible for recording transaction failures and maintaining consistency through distributed consensus. In contrast, the core component, TD-Orderer, runs locally alongside a modified Hyperledger Fabric, replacing the default ordering service. System performance is evaluated using Hyperledger Caliper [5], focusing on metrics such as latency, transaction success rate. The Smallbank scenario was used for testing, with a total of 100 keys. Zipf's law was applied with a parameter of 1.15 to simulate key access distribution.

B. Results

The experiment lasted for 30 seconds, comparing the existing Fabric system with the proposed system. Five users with varying network conditions each submitted 200 transactions per second. Among these, 10% were read-only transactions, while the remaining 90% were designed to potentially cause conflicts.

Fig. 2 shows the transaction abort rates per user. While Fabric exhibits varying abort rates depending on users' network conditions, the proposed system achieves fairer transaction processing by considering user failure rates. This approach helps prevent intentional attacks on specific users' transactions or prioritization of one's own transactions. Additionally, Fig. 3 presents the standard deviation and

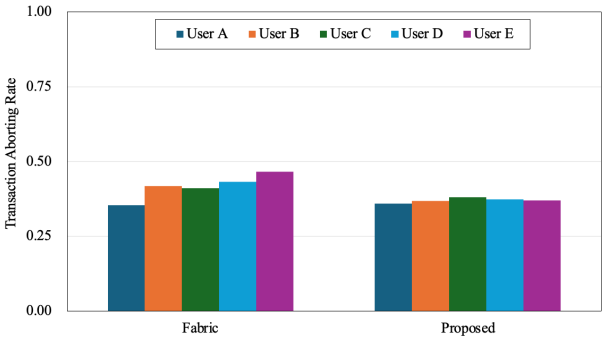

Fig2. Transaction aborting rate per user

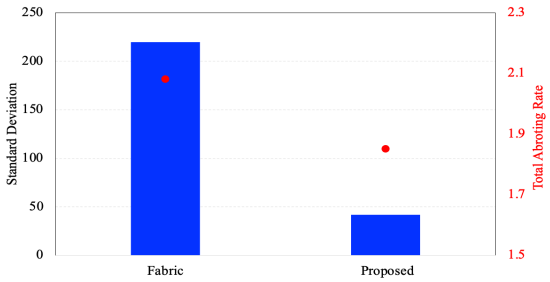

Fig3. Standard deviation and Total Aborting rate

overall transaction abort rates. Fabric's standard deviation was 219, whereas the proposed system achieved a much lower value of 41, demonstrating significantly improved fairness. By analyzing dependencies between transactions, the proposed system also reduces the overall abort rate, as confirmed by the experimental results.

IV. Conclusion

This paper proposes a conflict-aware and fairness-preserving ordering service for Hyperledger Fabric by combining Kafka, TD-Orderer, and Watchdog. The system reduces transaction aborts through dependency analysis and prioritizes fairness by monitoring user failure rates. Experiments using the Smallbank scenario show improved throughput, fairness, and lower abort rates compared to Fabric's default ordering. This architecture enhances transaction validity and equitable access in permissioned blockchains, providing a practical improvement without sacrificing performance.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant RS-2023-00212931

References

[1] Hyperledger Fabric Documentation. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html

[2] Misra and A. D. Kshemkalyani, "Towards Stronger Blockchains: Security Against Front-Running Attacks," in *Networked Systems. NETYS 2024*, A. Bouajjani and A. Mostefaoui, Eds., LNCS, vol. 14484, pp. 171–187, Springer, Cham, 2024.

[3] Apache Kafka (2024). Available: https://kafka.apache.org

[4] G. S. Park and H. Song, "Watchdog: Network-aware consensus protocol for enhancing scalability of public blockchains," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 20138–20151, June 2024.

[5] Hyperledger Caliper [Online]. Available: https://github.com/hyperledger-caliper/caliper