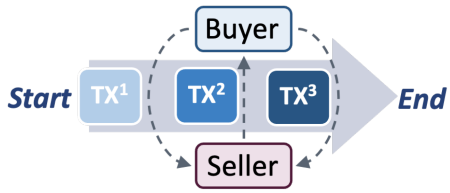
FLUID: Towards Efficient Continuous Transaction Processing in DAG-Based Blockchains

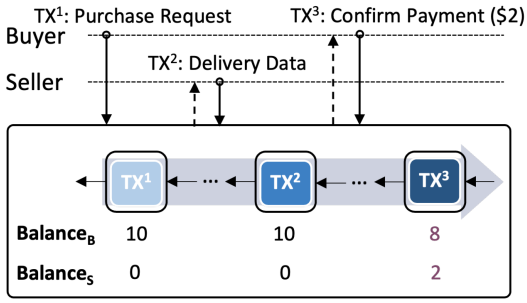
Junpei Ni, *Student Member, IEEE*, Jiang Xiao, *Member, IEEE*, Shijie Zhang, *Student Member, IEEE*, Bo Li, *Fellow, IEEE*, Baochun Li, *Fellow, IEEE*, and Hai Jin, *Fellow, IEEE*

***Abstract*—In most blockchain-based application scenarios, a complete application logic consists of multiple continuous transactions, in which the initiation of one transaction depends on the confirmation result of the previous one. This mandates that continuous transactions must be processed in the correct order. Unfortunately, existing chain-based blockchains fail to effectively support continuous transaction processing due to considerable latency in confirming continuous transactions. Recent studies shifted from chain-based blockchains to *Directed Acyclic Graph* (DAG) based blockchains, which reduced transaction confirmation latencies. However, DAG-based blockchains store transactions in an out-of-order manner that leads to unordered transaction processing. To address this challenge, we propose FLUID, a new DAG-based blockchain that supports continuous transaction processing while delivering high performance. The fundamental idea of FLUID is to design a transaction dependency tracking structure to ensure that continuous transactions can be processed in the correct order. FLUID utilizes a conflict resolution mechanism to provide instant confirmation and to support concurrent transaction processing with lower latencies. In addition, FLUID builds a checkpoint-based verification mechanism to achieve deterministic consensus on transaction processing results in the DAG. Extensive experiments demonstrate that our proposed FLUID can improve the throughput over state-of-the-art OHIE by 66% with two orders of magnitude lower latencies.**

***Index Terms*—Blockchain, continuous transaction processing, DAG, data trading, storage model.**



(a) Interactive process of continuous transactions in the application



(b) Processing of continuous transactions in the blockchain

Fig. 1. Transaction processing for data trading scenario. $Balance_B$ indicates the buyer's ledger balance. $Balance_S$ indicates the seller's ledger balance.

I. Introduction

DUE to the decentralized, tamper-proof and highly robust nature, the blockchain technology can reach agreements on the transaction processing results from different parties without the need of any trusted intermediaries, thereby supporting trusted applications built on top of it [1]. Many distributed applications such as data trading [2], [3], supply chain management [4], [5], and healthcare [6] have attempted to adopt the blockchain technology to support more robust application logic.

A complete application logic of a blockchain-based application consists of multiple continuous transactions that are clearly ordered, interdependent, and committed interactively by participants. For instance, in a data trading application, a complete application logic involving two parties includes three continuous transactions (i.e., TX^1, TX^2, TX^3), as shown in Fig. 1(a), the interaction between the buyer and the seller goes through the following steps: (1) TX^1: the buyer submits a purchase request; (2) TX^2: the seller delivers the data; (3) TX^3: the buyer confirms payment. These continuous transactions must be processed in the correct order as depicted in Fig. 1(b), otherwise, any transaction loss or error will destroy the correctness and completeness of the entire data trading. Compared with cryptocurrency transfer transactions, continuous transactions have two unique characteristics: (1) Transaction processing is done interactively

Manuscript received 15 November 2022; revised 12 March 2023; accepted 20 April 2023. Date of publication 2 May 2023; date of current version 8 November 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB2700700, in part by the Key Research and Development Program of Hubei Province under Grant 2021BEA164, in part by the National Natural Science Foundation of China under Grant 62072197, in part by the Knowledge Innovation Program of Wuhan-Shuguang, a RGC RIF under Grant R6021-20, and in part by RGC GRF under Grants 16209120, 16200221 and 16207922. Recommended for acceptance by Y. Tong. *(Corresponding author: Jiang Xiao.)*

Junpei Ni, Jiang Xiao, Shijie Zhang, and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China (e-mail: junpei@hust.edu.cn; jiangxiao@hust.edu.cn; shijiezhang@hust.edu.cn; hjin@hust.edu.cn).

Bo Li is with the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (e-mail: bli@cse.ust.hk).

Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5R 0A3, Canada (e-mail: bli@ece.toronto.edu).

Digital Object Identifier 10.1109/TKDE.2023.3272312

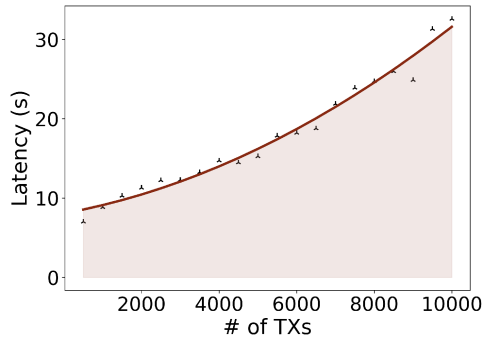


Fig. 2. Processing latency of continuous transactions in the chain-based blockchain storage model.

by multiple participants, and transaction execution has a definite sequential relationship because there are dependencies between transactions, e.g., TX^2 is committed by the seller based on the buyer's TX^1. (2) There is an intermediate state, which means that the application logic is not yet complete and does not change the state of the ledger. For example, TX^1, TX^2 are intermediate states, and their execution results in no modification to the ledger. Intermediate state may have an impact on atomic property during the initiation and interactive processing phases of continuous transactions, as peer nodes are unable to identify conflicts between intermediate state transactions, resulting in the possibility of transactions being recorded in the ledger that cannot be executed or should not be executed. To ensure the correct and complete application logic, blockchain systems must be able to support continuous transaction processing.

However, existing chain-based blockchain systems fail to effectively support continuous transaction processing. Since the initiation of a transaction needs to wait for the confirmation of the previous transaction when processing continuous transactions, multiple continuous transactions within the same application logic will be stored in separate blocks in the chain-based model. This, however, leads to substantial latency that significantly affects the efficiency of completing the application logic. We evaluated the latency of the data trading process in Fig. 1(a) in a chain-based blockchain (average block confirmation interval 3 s, up to 3000 transactions per block). As shown in Fig. 2, the average processing latency of continuous transactions grows substantially with the increasing number of transactions, which is not affordable in real-world blockchain applications.

Recent studies shifted from chain-based blockchains toward *Directed Acyclic Graph* (DAG) based blockchains [7], [8], [9], which reduces the transaction confirmation latency. Specifically, transactions in DAG-based blockchains can be directly appended to the DAG without the need for packaging into a block, showing greater promise than chain-based blockchains in supporting efficient transaction processing.

However, DAG-based blockchains still face several key challenges. First, how can sequential and interactive properties of continuous transactions be achieved? DAG transactions have no sequential relationships as they are arbitrarily appended to the blockchain and are out of order. Second, how can the atomic property be achieved, where the system is required to have a uniquely determined record of the intermediate state? Since the intermediate state does not change the ledger state, it can lead to conflicting transactions being committed in a weak consistency environment. Third, how do we guarantee the ultimate certainty of the transaction set to ensure consistency among DAG nodes?

To address these challenges, in this paper, we propose FLUID, a DAG-based blockchain that enables efficient continuous transaction processing. The main idea is to design a transaction dependency tracking structure in a DAG-based blockchain that forces continuous transactions to satisfy sequential relationships according to dependencies. To avoid conflicting intermediate states, we first set up a conflict resolution mechanism that uses the trusted miners of the chain-based blockchain to perform conflict resolution before transactions are submitted. Meanwhile, intermediate state transactions are not verified by nodes, conflicting transactions are finalized by participants, and invalid transactions will not be accepted by the system. To obtain a consistent set of transactions, we design a checkpoint-based consistency verification mechanism using deterministic consensus in chain-based blockchain systems. In particular, although we extend the DAG vertices from individual transactions to a dependency tracking structure, since we do not change the logical structure of the DAG ledger, they are still one transaction, which allows different application transactions to coexist in one system.

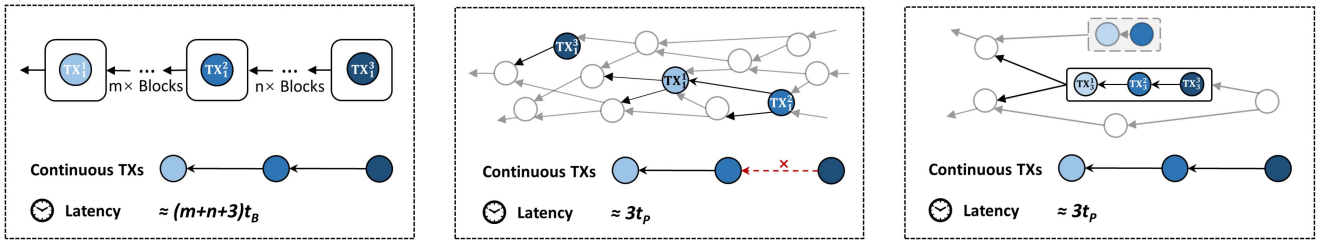
In summary, our main contributions in this paper are as follows:

- We are the first to formally define and characterize continuous transactions in blockchain systems. We design a dependency tracking structure that forces continuous transactions to be committed, executed, and stored in the correct order in DAG-based blockchains.
- We propose a conflict resolution mechanism that resolves potential conflicts and designs a checkpoint-based consistency verification mechanism to guarantee data consistency across nodes.
- We implement and evaluate FLUID, whose experimental results demonstrate its feasibility and superior efficiency. Extensive experiments demonstrate that our proposed FLUID can improve the throughput over the state-of-the-art OHIE by 66% with two orders of magnitude lower latency.

The remainder of the paper is organized as follows. We present the background and related work in Section II. Section III gives a system overview of FLUID. Section IV describes the detailed design of the FLUID. Section V presents the implementation of FLUID and the experimental evaluations. Finally, we conclude in Section VI.

II. Background and Related Work

In this section, we first introduce existing blockchain storage models and how they relate to transaction processing. Next, we briefly describe some related works on blockchain applications and system optimizations for chain-based and DAG-based storage models. Finally, we compare FLUID with existing approaches to demonstrate FLUID's superiority in terms of processing continuous transactions.



(a) Chain-based blockchain storage model (b) DAG-based blockchain storage model (c) FLUID storage model

Fig. 3. Processing of continuous transactions in different storage models. The t_B indicates the latency of a block from packing to committing to the blockchain. The t_P indicates the latency for a transaction to be transmitted and processed. t_B is much larger than t_P.

A. Blockchain Storage Model and Transaction Processing

The blockchain storage model not only defines the data structure of transactions but also determines the entire blockchain data flow. The blockchain data flow generally includes the phase of transaction data being appended to the chain via verification and consensus, as well as the phase of transactions being processed by each node. The order in which transactions are appended to the chain determines the order in which they are processed by each node.

Chain-based storage models: The chain-based storage model groups transaction data into separate blocks connected by hash pointers. The blockchain maintains the updating state generated by executing a collection of transactions in each block. As shown in Fig. 3(a), after a peer initiates a transaction based on the current state, the transaction waits to be packed into a block. Once the block containing the transaction is committed, the transaction can be processed according to its storage order in the block.

Although the chain-based storage model achieves highly robust data storage, it also has some drawbacks in terms of supporting efficient transaction processing:

Unacceptable transaction confirmation latency: In the chain-based storage model, transactions are confirmed in blocks, and the confirmation latency of a transaction is approximately equal to that of a block t_B, as shown in Fig. 3. Each block needs to be verified by the consensus algorithm, which requires complex computations or multiple rounds of communication, leading to a high block confirmation latency. For instance, Bitcoin [10] that adopts *Proof-of-Work* (PoW) has an average block confirmation interval of up to 10 minutes. Some blockchain systems using *Byzantine Fault Tolerance* (BFT) consensus algorithms have relatively small latencies yet still reach the second level, such as EOS [11], with a block confirmation interval of three seconds.

The crux of the problem is that the chain-based blockchain storage model commits and processes transactions on a block-by-block basis, which indicates that continuous transactions must go through several block cycles.

DAG-based storage models: The DAG-based storage model serves as an alternative to enable fast transaction confirmation. The classical DAG-based blockchain storage model employs a directed acyclic graph structure based on the tangle protocol [7], where transactions constitute the set of vertices. When a node submits a new transaction, it must select the two previous transactions for validation and reference them by building directed edges. If a vertex v has a path to a vertex u through the directed edges, it means that v indirectly verifies the validity of u. Once a node finds a transaction that conflicts with the tangle history, the node will not approve the conflicting transaction in a direct or indirect manner and invalidate it. The submitted transactions can be processed instantly without waiting to be packed into blocks, thereby reducing transaction confirmation latency t_P. This makes the t_P in Fig. 3 significantly smaller than t_B. This storage model thus shows promise in supporting the processing of highly concurrent transactions.

However, the DAG-based storage model that appends transactions in parallel also has the following drawback:

Unordered transaction storage: Some transactions without reference relationships in the DAG do not have a strict sequence in terms of storage. This may lead to two issues when processing these transactions: 1) they are processed in different orders at different nodes, and 2) they are mutually unverifiable since there is no reference relationship between them. As shown in Fig. 3(b), TX_1^3 is issued after TX_1^1, yet there is no storage order between TX_1^3 and TX_1^1 in the DAG since they have no reference relationship. If TX_1^1 and TX_1^3 belong to the same application logic, TX_1^3 should be processed after TX_1^1. However, their unordered storage causes nodes to process them in an unfixed order, which breaches the correct application logic. Besides, when processing TX_1^3, nodes have no means to locate TX_1^1 via references for verification.

Even though the DAG-based blockchain storage model is not yet capable of reliably handling continuous transactions, it is naturally low-latency, which provides good prerequisites for supporting a wider range of application scenarios. Its unordered transaction storage issue can be corrected by additional means.

B. Related Work

Many researchers employ blockchain to reinvent applications that lack a foundation of trust, such as data trading [2], [3], [17], [18], [19], supply chain [4], [5], privacy protection [6], [20], [21], and new computing paradigm [22], [23], [24]. A wide range of application scenarios containing continuous transactions are the focus of this paper, and they all include a complete application logic involving multiple parties. Moreover, improving the overall efficiency of completing application logic is a critical issue in most scenarios.

The above works are implemented based on a general single-chain system, and we have discussed the shortcomings of this

TABLE I
COMPARISON OF FLUID WITH EXISTING BLOCKCHAIN SYSTEMS

System	Continuous Transaction Features		Basic Performance	
	Continuous Dependency	Atomicity	High Throughput	Low Latency
Monoxide [11]	×	×	✓	×
Conflux [12]	×	×	✓	×
OHIE [13]	×	×	✓	×
Occam [14]	×	×	✓	×
EOS [10]	×	×	✓	×
PartitionChain [15]	×	×	✓	×
CAPER [3]	×	×	✓	✓
IOTA [6]	×	×	✓	✓
FLUID	✓	✓	✓	✓

scheme before. In recent years, many efforts have been made to optimize the performance of chain-based storage models. Some works have been done to obtain better system performance and balance security by designing more advanced consensus algorithms [12], [25], [26], [27]. Meanwhile, some works [16], [28], [29], [30], [31] propose lightweight storage solutions to effectively reduce storage overhead and improve scalability. To enable more efficient transaction processing, some works [32], [33], [34], [35], [36] utilize modern multi-core advantages to improve the concurrency of execution to accelerate transaction processing. However, they ignore the application logic implied in transaction processing. Limited by the serial structure of the chain-based storage model, these existing efforts face unacceptable efficiency problems when processing continuous transactions.

To resolve the inefficiency of chain-based storage models, many researchers have focused on DAG-based blockchain storage models in recent years. Typical DAG-based storage model defines each vertex in a DAG as a transaction (e.g., IOTA [7], ByteBall [8], Nano [9]). This has the advantage of enhancing the parallelism of transaction processing, thereby yielding lower processing latency compared to chain-based blockchains. However, such a scheme makes the blockchain data storage out of full order, which makes linear data validation difficult. To ensure the ordered data storage, some works utilize blocks as each vertex in the DAG and extend the conventional consensus algorithms adopted by chain-based blockchains. Conflux [13] and OHIE [14] extend the chain-based storage model and process multiple concurrent blocks to improve the system throughput. They can confirm the full order of data through a unified ordering algorithm. However, their transaction commitment still needs to wait for block packing and a complex mining process, which still suffers from high confirmation latency. DAG-Rider [37] proposes an asynchronous Byzantine atomic broadcast protocol in two layers, where the second layer performs a full ordering of the proposals in the DAG. Occam [15] proposes a scheme that adaptively changes the DAG concurrency according to the transaction demand in the network. However, as discussed before, current DAG-based storage models could not directly apply to application scenarios that contain continuous transactions. Nezha [38] optimizes transaction processing towards DAG-based storage models. It only improves the performance of concurrent transaction processing yet ignores the support for continuous transaction processing.

Table I presents a comparison of some representative blockchain systems with our work. To sum up, existing research efforts neglect the continuous transactions that may be included in many application logic and the relationship between the blockchain storage model and transaction processing. By contrast, this paper presents FLUID, enabling efficient continuous transaction processing in DAG-based blockchains. Compared to OHIE systems, it maintains the dependencies between continuous transactions and can process them with millisecond latency in terms of continuous transaction processing.

III. FLUID OVERVIEW

A. Design Goals

We design FLUID with the following design goals:

- *Supporting sequential, interactive processing of continuous transactions:* In order to reliably handle application logic that contains continuous transactions, we need to ensure in the new design that continuous transactions can be committed and processed sequentially and interactively, make it satisfy the sequential and dependency properties.
- *Ensure atomicity of continuous transaction processing:* We need to avoid conflicting intermediate states being finalized in the new design.
- *Supporting low latency continuous transaction processing:* While meeting the need to reliably process continuous transactions, we need to ensure that the new design would provide superior continuous transaction processing performance. It should be a significant performance improvement over existing chain-based solutions.
- *Improving data consistency:* Data consistency in DAG-based blockchain is weak, nodes in the system should be able to make fast and accurate verification of the consistency of local data with other peers.

B. Notations

To introduce the FLUID solution design in a more concrete and visual way, we place FLUID in the application scenario of data trading.

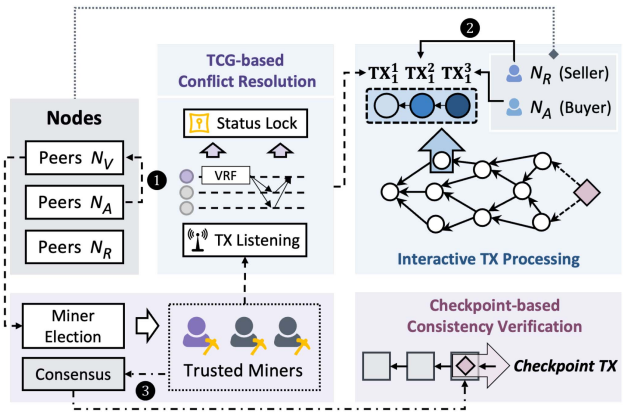


Fig. 4. FLUID's architecture.

Participants: The main participants in the system can be grouped into three categories, which contain the inherent roles of the underlying system and the participants of the data trading application:

- *Access node (N_A),* the buyer in a data trading. It is the initiator of data trading by submitting data requests to other resource nodes.
- *Resource node (N_R),* the seller in a data trading. It keeps some data resources and will respond to access requests submitted by N_A.
- *Verification node (N_V),* peer node in the blockchain system. It is not a participant in data trading, but supervises the data trading process of N_A and N_R to guarantee the consistency of the data.

Note that, for an entity node, it may belong to more than one of the above node types at the same time.

Transaction symbols: In the data trading scenario, we assume that a data trading contains three transactions T_{Req}, T_{Resp}, T_{ACK}, which denote data request, data reply, and transaction confirmation, respectively. Meanwhile, we use TX_m^n to denote a specific transaction, and subscripts are the label of the continuous transaction, having the same subscript means they belong to the same continuous transaction. The superscript indicates the sequential relationship of the transactions. For example, TX_1^2 denotes T_{Resp} in a data trading with number 1.

C. System Overview

To achieve the design goals, FLUID re-structured the DAG-based blockchain ledger so that it could provide more efficient continuous transaction processing while retaining the original blockchain features. Fig. 4 shows an overview of FLUID, which contains the core modules designed for the key challenges, as well as an illustration of the workflow in a data trading scenario.

In order to achieve reliable and efficient processing of continuous transactions in a disordered DAG model and to avoid conflicts as much as possible, a dependency tracking structure is designed. It redefines the structure of vertices in the DAG, where interdependent continuous transactions replace individual blockchain transactions. Based on this structure, the acknowledgment of the intermediate state during interaction is delayed, and nodes do not perform immediate acknowledgment of the intermediate state transactions. To avoid conflicts between transactions, we design a TCG-based conflict resolution mechanisms that listen for new transactions and maintain a state lock to validate and block potentially conflicting transactions for submission to the network. To compensate for the weak data consistency of the DAG system, FLUID synchronizes data through checkpoint transactions. Checkpoints are constructed based on a local checksum at a specific moment in time, through which nodes can lightly and quickly discover differences between replicas.

Continuous transaction processing workflow: As shown in Fig. 4, the workflow of FLUID in data trading scenarios can be summarized in the following steps:

- ❶ *Transaction initiation:* A node N_A first constructs a transaction T_{Req} to obtain the data owned by N_R and forward it to the trusted consensus group. The miners will verify the validity of the transaction and forward the valid transaction to the peers.
- ❷ *Transaction processing:* N_R and N_A will process transactions based on the T_{Req} and T_{Resp} they received. The peers will update the state based on the results of these transactions.
- ❸ *Consistency verification:* The leading miners will periodically initiate checkpoint transactions, and N_V will perform deterministic consensus on the DAG data digests in the checkpoint transactions based on the consensus algorithm.

IV. FLUID Design

In this section, we expand on the design ideas and details of the solution for FLUID and how it addresses the three challenges in Section I.

A. Dependency Tracking Structure

As discussed in the previous section, the obvious problem of DAG-based blockchain is that the transaction data structure cannot manifest the dependencies between continuous transactions, and the storage of transactions in the DAG is out-of-order. To enable continuous transaction processing, we need a mechanism to keep track of dependencies between continuous transactions.

Strawman design: Although the commit of transactions in DAG-based is unordered, we can record the relative order of continuous transactions with dependency tracking. Specifically, if any transaction includes the information of which transaction it depends on, any node processing the transactions in the blockchain would buffer it until its dependencies are satisfied.

However, the strawman design may introduce other problems, potential intermediate state conflicts may undermine the atomicity of continuous transactions:

Example 1: As shown in Fig. 5, after the buyer submits the purchase information TX_1^1, the seller might submit conflicting TX_1^2 and $TX_1^{2'}$. Since these two transactions do not affect the ledger state at the current moment, there is no conflict in transaction content between TX_1^2 and $TX_1^{2'}$, and they can be acknowledged by nodes in the ledger. However, the buyer will

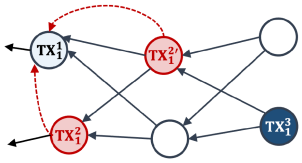


Fig. 5. An example of a conflicting intermediate state.

only process one of them. If the buyer chooses to confirm TX_1^2, $TX_1^{2'}$ submitted to the ledger will not be confirmed and also cannot be discarded. Hence, the submission of $TX_1^{2'}$ breaks atomicity.

In addition, the absence of a single record that points to a continuous transaction as a whole makes data records trivial, which brings additional challenges for data auditing and retrieval. The above problems motivate us to explore better solutions.

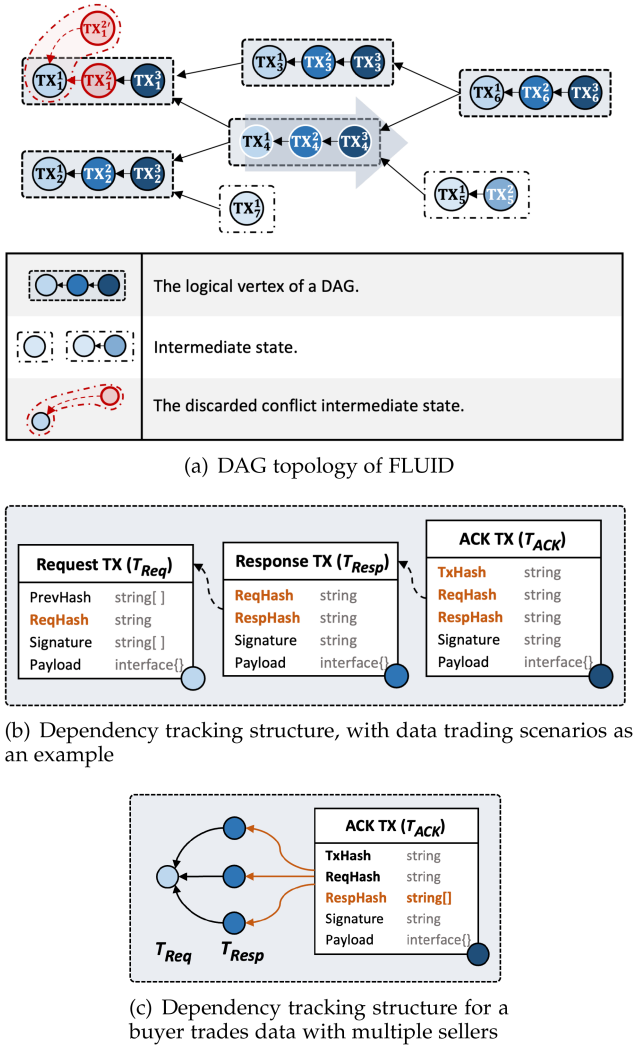
Based on the above analysis, we get two insights: (1) Intermediate state transactions can be suspended for an acknowledgment after they are committed to the system. (2) In addition to introducing basic dependency tracking information, the storage continuity of continuous transactions should be ensured, i.e., keeping interdependent continuous transactions concatenated.

Since dependency tracking information is usually represented by the hash of the dependent transaction, and the vertices of the DAG are also linked by hashes (i.e., edges in the DAG), we could reorganize the structure of the DAG by directly using dependency tracking information as the edges between vertices. However, we cannot rely on only one type of edge to connect vertices since a newly committed transaction does not depend on other transactions in the DAG. Thus, we follow the edges in the original DAG-based blockchain to verify the vertex (i.e., committed transaction) being connected. Besides, according to the first insight, we stipulate that any vertex in an intermediate state cannot be chosen to connect with until the continuous transactions are completed. By combining the above ideas, we design a novel DAG topology as shown in Fig. 6(a). The DAG topology of FLUID is composed as follows:

Vertex: One vertex consists of complete continuous transactions, where the continuous transactions are forcibly linked together by dependency tracking information. It is a logical-level vertex that may contain multiple transactions rather than a single transaction in existing DAG-based blockchains. Taking data trading as an example, a DAG vertex consists of T_{Req}, T_{Resp}, T_{ACK}, which are concatenated together through dependent hashes, such as TX_4^1, TX_4^2, TX_4^3 in Fig. 6(a). Besides, continuous transactions that have not yet satisfied atomicity cannot be regarded as vertices, such as TX_7^1 and TX_5^1, TX_5^2 in the figure.

Edge: There are two types of edges in FLUID, one links continuous transactions and another links DAG vertices. The edges between DAG vertices are created by nodes randomly selecting two vertices to verify when a new transaction is committed. The edges of continuous transactions are deterministic, and they represent the dependencies between continuous transactions.

Notice that the DAG vertices in FLUID are formed by the participants during the interaction rather than being submitted



(a) DAG topology of FLUID

(b) Dependency tracking structure, with data trading scenarios as an example

(c) Dependency tracking structure for a buyer trades data with multiple sellers

Fig. 6. FLUID's dependency tracking structure.

together at some point. Besides, since continuous transactions in intermediate states do not constitute DAG vertices, they are not acknowledged by peers. Therefore, one important difference between our design and the DAG topology that does not incorporate a dependency tracking structure is the timing of the acknowledgement of intermediate states. As shown in Fig. 6(a), TX_1^2 will be acknowledged after TX_1^3 is committed. $TX_1^{2'}$ that conflict with it will be discarded without affecting the atomicity of continuous transactions.

In Fig. 6(b), we give an example of a dependency tracking structure in a data trading scenario. T_{Req} denotes a data trading request submitted by the buyer. T_{Resp} denotes the response from the seller. T_{ACK} denotes the trade confirmation submitted by the buyer. They express the dependencies between transactions via hashes. Note that after the final confirmation of continuous transactions, the summary of continuous transactions needs to be concatenated to obtain a summary of the entire transaction, as in Fig. 6(b) with $TxHash = \mathcal{H}(T_{ACK}) + ReqHash + RespHash$.

It is important to note that the dependency tracking structure only determines the edges between continuous transactions

within a vertex and does not affect the edges between vertices. The dependency tracking structure can be customized based on the application logic to meet the transaction dependencies in various scenarios. The principle of the dependency tracking structure is that there must be a starting transaction (e.g., a data request in data trading) and an ending transaction (e.g., a transaction confirmation in data trading) that summarizes the dependency tracking structure so that it is consistent with other continuous transactions in the DAG ledger.

For example, in a scenario with one buyer and multiple sellers [23], [24], user purchase data from different sellers for federated processing. The following dependency tracking structure described in Fig. 6(c) can be designed based on this application logic, which matches the transaction dependencies in the one-to-many trading scenario can satisfy the order of interactions between nodes. Thus, buyer and sellers can submit request, return data, and confirm trading based on this dependency tracking structure.

Therefore, different DAG vertices can be designed according to the application requirements as a way to support multiple application logic.

B. Continuous Transaction Processing in FLUID

After introducing FLUID's dependency tracking structure, this subsection will focus on how FLUID can resolve conflicts and interactively complete continuous transaction processing.

1) TCG-Based Conflict Resolution Mechanism: Conflicts caused by intermediate state can occur during the initial commit phase of a continuous transaction or during interaction processing. The dependency tracking structure resolves potential intermediate state conflicts during interaction processing because it avoids immediate validation of intermediate state transactions. However, the current design still has the potential for conflicts at the beginning of a transaction, which affects atomicity. We give the following example to explain this problem:

Example 2: Take the example of a data trading where a user with a balance of 10 initiates two simultaneous purchase requests with an overhead of 10. For the blockchain node, the purchase request does not reduce the user's balance until it has been executed. Therefore, both requests are valid when verifying the transaction. However, only one of these two requests should be submitted. Therefore, under the influence of the intermediate state, the potentially conflicting transaction is confirmed on the ledger. However, the potential conflict will eventually occur. For the above example, both sellers of the trade may have provided resources, yet the user can only complete one trade. From a transaction processing point of view, one of the trading actions cannot be completed, but the seller has executed the transaction as it should have, which breaks the atomicity.

Therefore, we need a mechanism to resolve this potential conflict in transaction initiation phase.

The dependency tracking structure cleverly avoids the potential conflict of intermediate state transactions during the interaction. Notice that we do not actually set out to resolve the conflict after it occurs, but rather to avoid it. We would like to continue this line of thinking. The point is that once

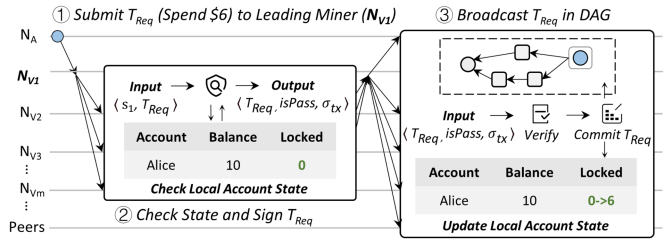


Fig. 7. Transaction initiation and submission.

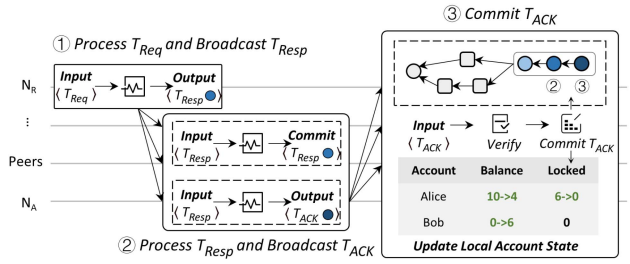
the potentially conflicting transaction described above is committed, the outcome of the conflict seems inevitable. Therefore, it occurred to us that we could listen for and discover these potentially conflicting transactions before they are committed to the DAG system, and prevent them from being committed to the respective system nodes. The remaining question is who should perform this task. Obviously, it is difficult for all peer nodes to do this job together, which would add an additional data synchronization challenge to the system. However, having a node or set of nodes specified in advance would undermine the reliability of the system. It occurs to us that the miner selection algorithm gives us a third option, where we can plug part of the chain-based blockchain's capabilities into FLUID in the form of a plug-in and use its miner selection algorithm to obtain a *Trusted Consensus Group* (TCG) of trusted miners who can perform this listening task. We have chosen the *Delegated Proof of Stake* (DPoS) miner election algorithm [11], [39] in FLUID to obtain a rotating group of trusted miners by node voting. All nodes in the system can participate in miner voting and election. The set of nodes that receive the most votes will become the miners.

Specifically, the process of submitting a data trading request to FLUID can be divided into two phases, transaction initiation and transaction submission. The specific scheme is as follows:

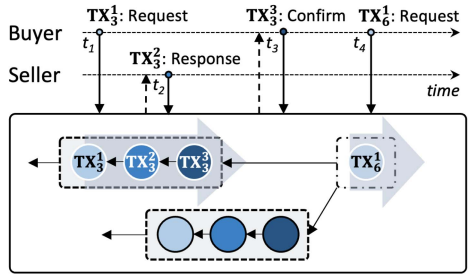
Transaction initiation: As described in Section II, when N_A initiates a new transaction T_{Req}, N_A fills the payload with the transaction content and randomly selects two complete transactions from the DAG for validation, and finally signs the T_{Req}.

Transaction submission: Transactions initiated by N_A are not immediately broadcast to the DAG network, but are listened to by the trusted consensus group and verified for potential conflicts. In order to verify transactions, the miner must know the current available balance of each node in the network, so a method is needed to keep track of transactions that have currently been submitted by the miner but have not yet completed. It occurs to us that the balance occupied by these outstanding transactions can be locked, i.e., a status lock can be cached to record the balance occupation. The value of the locked balance represents the unavailable portion of the current account balance. Therefore, the actual available balance equals the account balance minus the locked balance. With the above conditions in place, all that remains is to get the miner nodes to agree on the validity of the submitted transaction and then submit this transaction.

As shown in Fig. 7, after Alice constructs a transaction T_{Req}, she submits the transaction to the leading miner N_{V_1} in the trusted consensus group. N_{V_1} will verify if the transaction expense exceeds the available account balance and sign the



(a) Flow of continuous transaction interactive processing



(b) A running example of interactive processing

Fig. 8. Interactive continuous transaction processing.

verification result (valid or not), then forward the transaction and account state to other miners (N_{V2} to N_{Vm} shown in Fig. 7). Other miners will calculate the local account state based on the transaction content and compare it with the data sent from N_{V1}. If the local state meets the transaction submission condition and is consistent with the leading miner's state, the transaction will be signed as a valid transaction. Otherwise, it will be signed as an invalid transaction. Each miner returns the verification result to N_{V1}, and N_{V1} will count the verification results received from other miners. When at least 2/3 of the miners regard the transaction as valid, N_{V1} packages the signed verification result and the transaction and then broadcasts it to the DAG network. The other miners will update the state of the local locked accounts accordingly. If the transaction is verified as invalid, N_{V1} will roll back the local account state and return a submission failure message to N_A with the signed result from other miners.

2) Interactive Continuous Transaction Processing: An example of node interaction based on the dependency tracking structure of data trading is given in Fig. 8(b): (1) At time t1, the buyer submits a request $TX_3^1(T_{Req})$ to the trusted consensus group in the FLUID, and the transaction is broadcast in the FLUID after verification. (2) At time t2, the seller submits the reply message $TX_3^2(T_{Resp})$ of the data trading based on the received request TX_3^1. (3) At time t3, the buyer verifies the reply TX_3^2 based on the received reply TX_3^2 and submits the confirmation message $TX_3^3(T_{ACK})$ of the trading. (4) At time t4, the buyer submits another trading request TX_6^1. Although TX_6^1 is submitted to the DAG ledger, it has not yet received a response from the seller. This continuous transaction has not been finally confirmed, so other nodes will not select it for validation approval when initiating new transaction requests. Notice that all nodes can quickly verify the validity of transactions during node interactions based on the dependency tracking structure. Transactions that do not satisfy the interaction order will be discarded.

To ensure the liveness of transaction processing, a timeout mechanism needs to be set. For T_{Req} or T_{Resp} that are not processed within the time limit, the subsequent transactions will be constructed and committed by the node that initiates T_{Req} or T_{Resp} to ensure the integrity of transaction processing.

Due to the interactive nature of continuous transactions, the atomic property of continuous transactions is not broken even if there is a conflicting intermediate state of the transaction. In the case of a conflicting T_{Resp}, N_A will select a valid record to reply to based on the information received. Because these conflicting intermediate states are not verified by the peer nodes as blockchain transactions, only the continuous transactions confirmed by T_{ACK} are valid, and the other intermediate states will be discarded. So these intermediate states do not break the atomic property of continuous transactions.

C. Checkpoint-Based Consistency Verification

A significant problem with transaction processing based on the DAG-based blockchain is the weak data consistency. As discussed earlier, the validity of transaction submission is determined by the joint verification of miners. However, if the data consistency between miners is weak, the verification results will inevitably be affected. Therefore, a means to enhance the data consistency is required. To reach system-wide data agreement, we need two key methods: (1) A fast way to check the dissimilarity of two pieces of data. (2) A deterministic consensus method between multiple nodes for one piece of data. The first of these is to provide a basic consistency verification method for the second point.

First, we need a method to verify the data consistency of two DAG ledgers. Compared to chain-based ledgers, verifying the data consistency of DAG ledgers has a tricky problem: it has many branching paths that need to be verified. A one-by-one validation approach would be contrary to the original design intent of the DAG-based blockchain. The approach should be as efficient and lightweight as possible. Our first thought is to compress the information of DAG vertices and edges. A simple idea is that we can organize the vertices without edges pointing to it at a given moment in a Merkle-tree. But the problem is that any vertex that does not have an edge pointing to it may have many paths to the initial vertex of the DAG, so how to specify the path information in the compressed information? And, unlike a chain-based ledger, we cannot find a fixed frequency to encompass all DAG vertices. We note that vertices have one or two directed edges pointing to other vertices. Based on the idea of recursion, as long as the information contained in the vertices pointed by the directed edges of any vertex is consistent, then we only need to compare the consistency of the current node to get the consistency of a subset of data in the DAG. We only need to perform the above verification for all vertices that do not have edges pointing to them at some point to get the difference between the two copies of the DAG. This is like an accumulation of information, where the later submitted vertices continuously

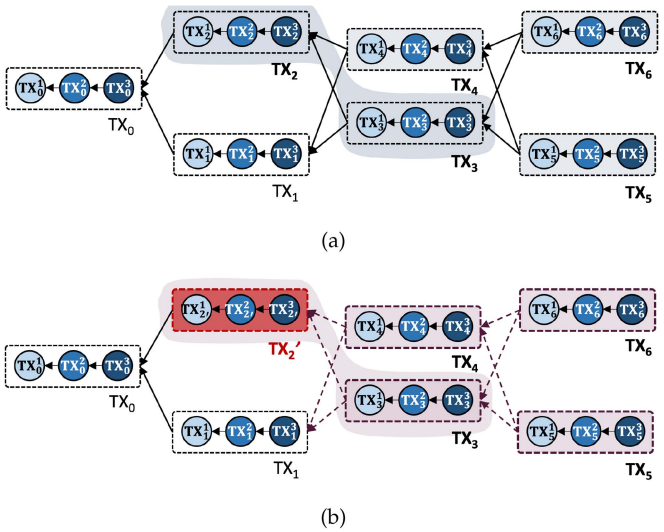


Fig. 9. An example of the checksum-based data consistency verification.

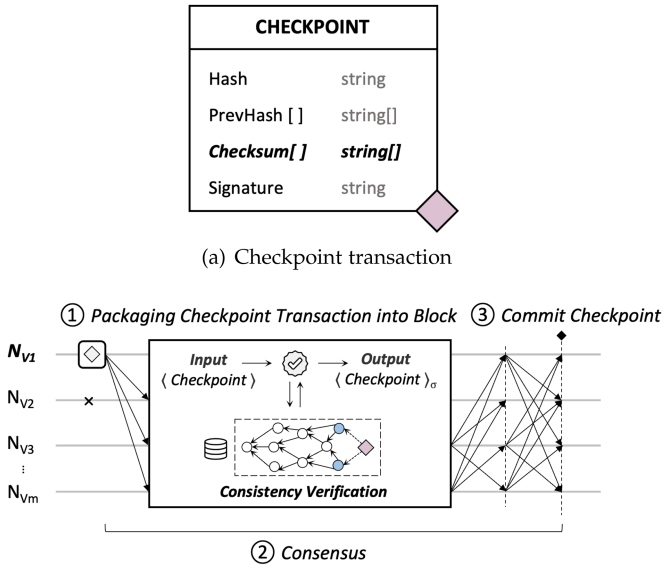


Fig. 10. Consistency verification based on checkpoint transactions.

contain the information of the previously submitted vertices, and we call this method a *checksum*.

Checksum: Note that we define $PrevHash$ in the data structure of T_{Req} as an array of length two. We distinguish two members of this array by defining $PrevHash[0]$ as the logical left node (denoted as L_N) and $PrevHash[1]$ as the logical right node (denoted as R_N). This allows us to separate the paths of the parent transactions clearly. When T_{ACK} is committed, the peers locally compute and maintain the checksum of this transaction (denoted as TX_i):

$$\mathcal{C}(TX_i) = \mathcal{H}(\mathcal{C}_R(TX_i) + \mathcal{C}_L(TX_i) + \mathcal{H}(TX_i)) \quad (1)$$

where $\mathcal{C}_L(TX)$ and $\mathcal{C}_R(TX)$ denote the checksum of L_N and R_N, respectively. $\mathcal{C}(TX)$ denotes the checksum of TX. $\mathcal{H}(TX)$ denotes the hash of TX

$$\mathcal{C}_L(TX_i) = \begin{cases} \mathcal{C}(L_N) & L_N \neq \emptyset \\ 0 & L_N = \emptyset \end{cases}$$
$$\mathcal{C}_R(TX_i) = \begin{cases} \mathcal{C}(R_N) & R_N \neq \emptyset \\ 0 & R_N = \emptyset \end{cases} \quad (2)$$

For each confirmed transaction, peers need to compute and maintain the checksums of L_N and R_N connected to the current transaction as well as its own checksum. The checksum of the current transaction represents a summary of the data on its preorder path.

Example 3: Suppose there is a difference in the DAG data of two nodes, as shown in Fig. 9 . Transactions TX_2 and TX_2' in Fig. 9 and (b) are different for some reasons. The checksum of TX_3 calculated in Fig. 9(a) is

$$\begin{aligned}\mathcal{C}(TX_3) &= \mathcal{H}(\mathcal{C}_L(TX_3) + \mathcal{C}_R(TX_3) + \mathcal{H}(TX_3)) \\ &= \mathcal{H}(\mathcal{C}(TX_1) + \mathcal{C}(TX_2) + \mathcal{H}(TX_3))\end{aligned}$$

while the checksum of TX_3 calculated in Fig. 9(b) is

$$\begin{aligned}\mathcal{C}(TX_3) &= \mathcal{H}(\mathcal{C}_L(TX_3) + \mathcal{C}_R(TX_3) + \mathcal{H}(TX_3)) \\ &= \mathcal{H}(\mathcal{C}(TX_1) + \mathcal{C}(TX_2') + \mathcal{H}(TX_3))\end{aligned}$$

The color in Fig. 9 indicates the range of data inconsistency. It can be observed that the checksums of transactions directly or indirectly referencing TX_2 on the DAG path are different between Fig. 9(a) and (b). The checksums on the left or right side lead to the path's direction where the data inconsistency occurs.

Based on the checksum, for each vertex of the DAG ledger, each peer node has a piece of information about its path. Combining this with the initial idea of fusing the checksums of vertices that do not have edges pointing to them at a certain moment, and comparing this information, one can quickly get whether the two ledgers agree or not.

The remaining question is how to reach deterministic consensus on a single DAG data across multiple nodes. We introduce the miner selection algorithm in chain-based blockchain as a way to obtain trusted miners when dealing with potential intermediate state conflicts during continuous transaction submission. Clearly, this miner selection algorithm acts as a plug-in to FLUID. Moreover, we can still leverage the miner nodes and consensus algorithms to facilitate the data consistency verification process. What they agree on is a blockchain transaction that incorporates the vertex checksum information, which we call a checkpoint transaction as shown in Fig. 10(a). We apply *(Practical Byzantine Fault Tolerance)* (PBFT) consensus algorithm [40] in FLUID among the set of miners. The submission and consensus process of a checkpoint transaction is as follows:

Checkpoint transaction submission process: Miners in the trusted consensus group will periodically initiate checkpoint transactions as shown in Fig. 10(b). The checkpoint transaction contains the hash of unverified transactions in the current DAG, the checksum of those transactions computed by the miner, the hash of the checkpoint transaction, and the signature of the miner. When a checkpoint transaction is proposed, the miner checks the unverified transactions in its DAG view and packs

the hashes of those transactions along with their checksums into a checkpoint transaction. The miner broadcasts the checkpoint transaction to other miners and completes a three-stage PBFT consensus against the checkpoint transaction. Peers compare the checksums of transactions in the checkpoint with their local computation to ensure that the DAG data to be determined is consistent with their local data. If a block containing checkpoint transactions passes the verification of the consensus algorithm, all transactions contained in the checkpoint transaction and their previous transactions are determined.

Discussion 1: security. In checkpoint-based consistency verification mechanism, whether the checksum is secure or not is vital to the effectiveness of the mechanism. The design of FLUID does not change the consensus algorithm underlying the blockchain system. Therefore, FLUID inherits the security properties of blockchain. Potential security issues with checksums can occur in two parts of the process, i.e., checksum generation and verification.

In the process of checksum generation, the checksum is calculated locally by each node in FLUID based on the transaction data appended in the DAG ledger. Therefore, the checksum generation process is tied to the generation process of the DAG ledger data. According to the design of the tangle protocol, it is impossible for a malicious node to manipulate the generation of the DAG ledger as long as it cannot outperform the sum of the capabilities of other nodes in the network.

In the process of checksum validation, a set of trusted miners validate against checkpoint transactions. The set of miners and the miners responsible for initiating the verification proposal at a given time slot are generated by the DPoS algorithm, which dynamically maintains a set of miners through voting elections. The verification of the checksum in the checkpoint is done by the set of miners running the PBFT consensus algorithm in the current time period. It is known that checksums in honest nodes cannot be manipulated by malicious nodes. The security of the checksum validation process is tied to the DPoS and PBFT algorithms. As long as the nodes performing consensus satisfy the security assumptions of the PBFT protocol, i.e., the number of honest miner nodes is greater than 2/3 of the total number of miners, the checksum verification process cannot be manipulated by malicious nodes.

Discussion 2: potential overhead. Referring to the analysis in tangle [7], the expected value of the number of new unvalidated transactions tends to zero when a node proposes a transaction, with the typical value of unvalidated transactions (L_0) in the current system

$$L_0 \approx \frac{\lambda h(L_0, N)}{ln2} \approx 1.44\lambda h(L_0, N) \tag{3}$$

where λ is the rate of the transaction input stream and $h(L_0, N)$ is the average time required for a node to send a transaction with a total of N transactions and L_0 unverified transactions in the system. This means that the storage and validation overhead of checkpoints does not grow indefinitely.

In summary, to ensure the consistency of DAG data, the peers in FLUID will compute and maintain the checksum of transactions, and complete the verification of checkpoint transactions based on the checksum, using the deterministic consensus of the chain-based blockchain system to guarantee the deterministic view of transaction records.

V. Experimental Evaluations

In this section, we evaluate FLUID through several sets of experiments to understand its continuous transaction processing performance and verification latency. We mainly focus on the following questions.

- How does FLUID perform in terms of throughput and latency when dealing with workloads that contain a large number of continuous transactions?
- How does FLUID perform in terms of checkpoint-based consistency verification?

Implementation: We implement the prototype of FLUID in Golang, as shown in Fig. 4. We implement a dependency tracking structure for continuous transactions, and we employ the random algorithm for the selection of preorder transactions. We extend the TCG-based conflict resolution mechanism on top of EOS [11], which consists of PBFT and DPoS protocols. We specify the miners in the initial state through a configuration file, where the leading miner is responsible for block packing and committing, as well as verifying transactions. We simulate the requests initiated by N_A based on *Apache Bench*[1] scripts, and automate the transaction process. In the above system, nodes communicate with each other through a web interface based on the HTTP protocol.

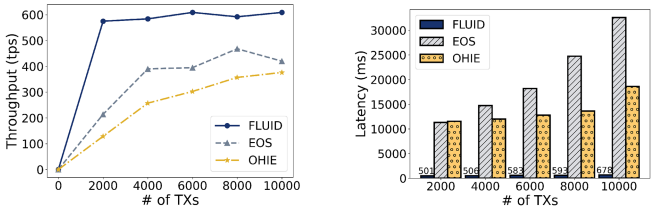
Baseline: We evaluate the performance of FLUID against two baselines: EOS [11], which is one of the representative chain-based blockchain systems and employs a hybrid consensus algorithm (i.e., DPoS + PBFT) to achieve deterministic block generation time and total order; OHIE [14], a state-of-the-art DAG-based blockchain, runs multiple Nakamoto consensus instances to enable miners to propose blocks in parallel based on the PoW algorithm. For fairness, both EOS and OHIE are implemented using the same program language and libraries, and they maintain the same transaction data structure as FLUID. Furthermore, we use the same workloads to evaluate them.

A. Experimental Setup

Testbed and setup: We conduct our experiments on Aliyun ECS. We use four instances of ecs.s6-c1m2.large[2], each of which contains 2 vCPUs and 4 GB memory, with a network bandwidth of 0.2 Gbit/s between nodes. For the FLUID and the EOS, we deploy 6 consensus nodes and 6 storage nodes. In FLUID, each instance runs two DAG-based model programs in an additional process. For OHIE, we deploy 12 nodes, who perform block mining on three chains in parallel.

Workloads: We simulate trading between nodes based on randomly generated trading requests and responses. The size of the workload is 10 K transactions, with an average data request field size of 457 B and an average response field size of 680 B. We use *Apache Bench* to stress test the system. It simulates

[1] https://httpd.apache.org/docs/2.4/programs/ab.html
[2] https://www.alibabacloud.com/help/en/elastic-compute-service



(a) Continuous transaction throughput (b) Continuous transaction processing latency

Fig. 11. Overall performance compared to baseline systems.

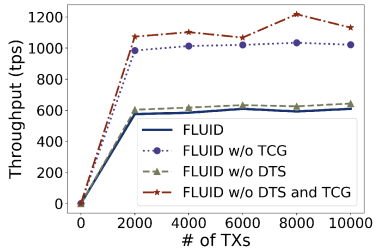


Fig. 12. Impact of dependency tracking structure (DTS) and TCG-based conflict resolution (TCG) on throughput.

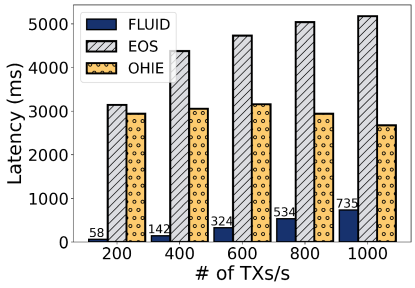


Fig. 13. Transaction submission latency compared to baseline systems.

access nodes N_A to launch data trading requests to the system's web interface without interruption based on script parameters. To standardize the test conditions, we utilize the same workloads to test FLUID and the two baselines. The interactive process of the workloads: N_A submits T_{Req} to the system. N_R receives T_{Req}, signs it and returns T_{Resp}. N_A verifies the signature and returns T_{ACK}.

Metrics: We evaluate the performance of FLUID and baselines by using the following metrics: (i) *Throughput*. We measure the average number of continuous transactions that the system can process per second. (ii) *Latency*. We measure a) the latency of T_{Req} from being submitted to being received by nodes, b) the latency of continuous transaction processing, i.e., the average latency between T_{Req} and T_{ACK}, and c) the latency of verifying checksums in checkpoint transactions. (iii) *Storage overhead.* We evaluate the storage overhead of checkpoint-based mechanism.

B. Experimental Results

1) Overall Performance: Fig. 11 shows the system throughput and latency of FLUID and the baselines when processing continuous transactions. We can see that FLUID is able to achieve a throughput improvement of more than 1.5x and a latency reduction of two orders of magnitude compared to the baselines. However, some of the key design aspects of FLUID also result in some performance compromises. The performance of FLUID will be measured and discussed in detail.

2) Transaction Throughput: Fig. 11(a) illustrates the continuous transaction throughput of FLUID and baselines under different transaction volumes.

Baselines: In the current experimental configuration, the throughput of EOS is basically stable at 400 tx/s as the transaction volume increases to 4000. Note that the throughput refers to the number of continuous transactions, and each transaction contains a T_{Req}, a T_{Resp} and a T_{ACK}. When we deploy 12 nodes and let them perform block mining on 3 chains in parallel, the throughput of OHIE will stabilize at 300-400 tx/s as the transaction volume grows. We observe that there is no significant difference between EOS and OHIE in terms of throughput performance.

FLUID: FLUID has a significant improvement in throughput compared to OHIE and EOS when dealing with continuous transaction workloads. FLUID's throughput is basically stable at 600 tx/s when the transaction volume reaches 2000. When the transaction volume in the system reaches 10000, FLUID's throughput can reach 1.5x and 2x compared with EOS and OHIE in the same environment, respectively. This is due to the fact that FLUID eliminates the resource overhead of block packing and the overhead of multiple rounds of network broadcasts.

Further, we tested the impact of key aspects of FLUID on system throughput. We tested the system throughput of FLUID without enabling the dependency tracking structure and TCG-based conflict resolution mechanism, respectively. Fig. 12 illustrates the impact of these designs on system throughput. It can be seen that for the same amount of data, the throughput of FLUID is almost equal to that of the system with only the dependency tracking structure disabled. This indicates that the introduction of the dependency tracking structure does not have a significant impact on the system throughput. This is because the dependency tracking structure does not change the basic transaction processing of the DAG blockchain system, but only imposes constraints on the relationships between transactions. In addition, the system without TCG-based conflict resolution has a 1.8x throughput improvement compared to FLUID. TCG-based conflict resolution limits the throughput performance. This is because transactions need to be validated by a trusted consensus group when they are submitted. Although the overhead of validating individual transactions is small, the overall performance degradation due to congestion is inevitable as the volume of transactions increases.

3) Transaction Latency: We measured the latency performance of FLUID and the baseline when processing continuous transactions at different transaction volumes.

EOS: As shown in Fig. 13, when 1000 transactions are submitted per second, the transaction submission latency in EOS exceeds 5 s, which is 1.7x that of OHIE and 7x that of FLUID. Moreover, the processing latency of continuous transactions in EOS appears to increase approximately linearly. As depicted

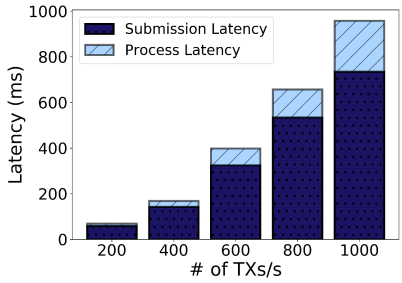


Fig. 14. Latency of continuous transactions from initiation to execution completion.

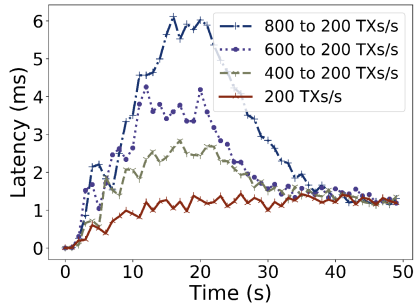


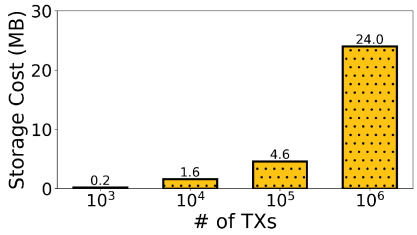
Fig. 15. Checkpoint verification latency under varying access pressures.

in Fig. 11(b), when the volume of transactions reaches 10,000, the processing latency of each continuous transaction in EOS exceeds 30 s, which is 1.9x that of OHIE and 48x that of FLUID. Although the throughput of EOS is not much worse than FLUID, its considerable transaction latency is unacceptable.

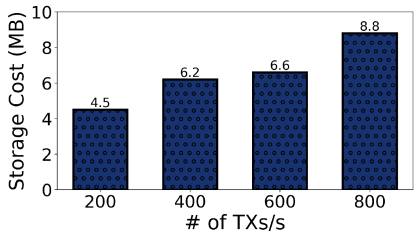
OHIE: It is worth noting that, when the volume of transactions grows, OHIE yields much lower transaction submission latency and transaction processing latency than EOS, as shown in Fig. 13. This is due to the fact that blocks can be packed in parallel, and fewer transactions are queued for packing under the same condition. However, parallel packing results in blocks containing duplicate transactions, which in turn affects the system throughput. As depicted in Fig. 11(b), although the latency of OHIE is reduced, its continuous transaction processing latency is still 30x than FLUID at a transaction volume of 10,000, with an average latency of 18 seconds per continuous transaction, which is still inefficient.

FLUID: By contrast, FLUID has a huge advantage over EOS and OHIE in terms of latency performance. When submitting transactions to the system at a frequency of 1000 transactions per second, FLUID's transaction submission latency is only 735.4 ms, which is an order of magnitude lower than EOS and OHIE. The continuous transaction processing latency advantage is more significant, as shown in Fig. 11(b), when the transaction volume reaches 10,000, FLUID is two orders of magnitude lower compared to EOS and OHIE. This is due to the transaction-based granularity of transaction processing in the FLUID model. Combining the latency overhead of FLUID transactions from submission to execution completion, as shown in the Fig. 14, we find that the transaction submission phase accounts for more than 75% of the latency overhead. This is because the TCG-based conflict resolution mechanism needs to go through the validation and network transmission of the transaction, which introduces additional overhead. However, this is necessary to ensure the reliable execution of continuous transactions.

4) Verification Latency: In this series of experiments, we evaluate the latency of nodes to validate checkpoint transactions under different access pressures. According to the analysis results in Section IV-C, the number of unvalidated transactions in the DAG will reach a relatively stable state when the data flow rate is stable. In FLUID, to ensure the consistency of DAG data, we need to submit checkpoint transactions periodically, which requires nodes to pack the unvalidated transactions in the DAG at the current moment and verify their checksums. If the capacity of unvalidated transactions in the DAG is uncontrollable, the checkpoint validation overhead will be unaffordable. As shown in Fig. 15, we submit access requests to FLUID at a rate of 200 to 800 transactions per second (*TXs/s*). We observe that the latency becomes relatively stable after a certain time, and the size of this stable value is related to the transaction rate.



(a) Storage overhead for checksums



(b) At a frequency of once a minute for checkpoint synchronisation, the system runs 24 hours of checkpoint storage overhead

Fig. 16. Checksum and checkpoint transaction storage overheads.

To further observe the stable state of the checkpoint verification latency, we reduce the transaction rate to 200 *TXs/s* when the stabilization is reached at different transaction rates. As shown in Fig. 15, it can be observed that the latency gradually decreases and eventually reaches the same steady state as in the case of maintaining a rate of 200 *TXs/s*. In summary, the checkpoint verification latency tends to stabilize at different transaction rates, hence, such a performance is within an acceptable range and does not affect the overall system performance in processing continuous transactions.

5) Storage Overhead: While a checkpoint-based data consistency verification mechanism can address the weak consistency of DAG-based blockchain data, the system sacrifices more storage space due to the introduction of checksums and

checkpoint transactions. We evaluated the storage overhead of the checkpoint-based data consistency validation mechanism.

First, we tested the storage overhead of the checksum mechanism. We tested the storage space occupied by the checksum when there are 10^3 to 10^6 transactions are completed in FLUID. The experimental results show that the checksum mechanism for 10^6 transactions will occupy about 24 MB of storage space. This includes, in addition to the checksum data, the indexes we set up during the experiment for fast retrieval.

In addition, we evaluated the overhead of checkpoint transactions. We set FLUID to perform a checkpoint synchronization every 1 minute and submit requests to the system at a rate of 200 to 800 TXs/s, respectively. We test the storage space occupied by the checkpoint-based mechanism every 24 hours under the above operation state, and the experimental results are shown in Fig. 16(b). Even with a transaction rate of 800 TXs/s and a synchronization frequency of once per minute, the additional overhead generated per day is only 8.8 MB, which is almost negligible compared to the transaction data generated.

VI. Conclusion

In this paper, we design FLUID, a DAG-based blockchain system capable of handling continuous transactions efficiently. Specifically, it builds a transaction dependency tracking structure on top of an efficient DAG-based blockchain storage model, allowing continuous transactions in complex applications to be processed sequentially based on dependencies interactively. To prevent intermediate states of continuous transactions from affecting the atomicity of transactions, we design a conflict resolution mechanism with the help of a trusted miners group. To compensate for the weak consistency of the DAG-based blockchain, we design a checkpoint-based consistency verification method to enhance the data consistency guarantee among nodes. Our extensive experiments have demonstrated that FLUID improves the throughput by 66% over the state-of-the-art, with lower latencies by two orders of magnitude.

References

[1] H. Jin and J. Xiao, "Towards trustworthy blockchain systems in the era of "internet of value": Development, challenges, and future trends," *Sci. China Inf. Sci.*, vol. 65, no. 5, pp. 1–11, 2022.

[2] F. Chen, J. Wang, C. Jiang, T. Xiang, and Y. Yang, "Blockchain based non-repudiable IoT data trading: Simpler, faster, and cheaper," in *Proc. IEEE 41st Conf. Comput. Commun.*, 2022, pp. 1958–1967.

[3] Q. Lin et al., "Demonstration of dealer: An end-to-end model marketplace with differential privacy," in *Proc. Int. Conf. Very Large Data Bases*, 2021, pp. 2747–2750.

[4] M. J. Amiri, D. Agrawal, and A. E. Abbadi, "Caper: A cross-application permissioned blockchain," in *Proc. Int. Conf. Very Large Data Bases*, 2019, pp. 1385–1398.

[5] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang, "Fine-grained, secure and efficient data provenance on blockchain systems," in *Proc. Int. Conf. Very Large Data Bases*, 2019, pp. 975–988.

[6] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: Toward a secure data-sharing system from decentralized trust," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 851–877.

[7] S. Popov, "The tangle," 2018. [Online]. Available: http://cryptoverze.s3.us-east-2.amazonaws.com/wp-content/uploads/2018/11/10012054/IOTA-MIOTA-Whitepaper.pdf

[8] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," 2016. [Online]. Available: https://byteball.org/Byteball.pdf

[9] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," 2018. [Online]. Available: https://nano.org/en/whitepaper

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[11] I. Grigg, "EOS - an introduction," 2017. [Online]. Available: https://whitepaperdatabase.com/eos-whitepaper

[12] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 95–112.

[13] C. Li et al., "A decentralized blockchain with high throughput and fast confirmation," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 515–528.

[14] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "OHIE: Blockchain scaling made simple," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 90–105.

[15] J. Xu, Y. Cheng, C. Wang, and X. Jia, "Occam: A secure and adaptive scaling scheme for permissionless blockchain," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 618–628.

[16] Z. Du, H.-F. Qian, and X. Pang, "Partitionchain: A scalable and reliable data storage strategy for permissioned blockchain," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4124–4136, 2023.

[17] G. Bu, T. S. L. Nguyen, M. P. Butucaru, and K. L. Thai, "HyperPub-Sub: Blockchain based publish/subscribe," in *Proc. IEEE 38th Int. Symp. Reliable Distrib. Syst.*, 2019, pp. 366–3662.

[18] J. Pei, "A survey on data pricing: From economics to data science," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 10, pp. 4586–4608, Oct. 2022.

[19] B. An, M. Xiao, A. Liu, Y. Xu, X. Zhang, and Q. Li, "Secure crowdsensed data trading based on blockchain," *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1763–1778, Mar. 2023.

[20] M. Yan, J. Xu, T. G. Marbach, H. Li, G. Wang, and X. Liu, "Audinet: A decentralized auditing system for cloud storage," in *Proc. IEEE 39th Int. Symp. Reliable Distrib. Syst.*, 2020, pp. 215–224.

[21] H. Shafagh, L. Burkhalter, S. Ratnasamy, and A. Hithnawi, "Droplet: Decentralized authorization and access control for encrypted data streams," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2469–2486.

[22] S. Han, Z. Xu, Y. Zeng, and L. Chen, "Fluid: A blockchain based framework for crowdsourcing," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 1921–1924.

[23] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 418–430, Jan. 2021.

[24] Y. Tong et al., "Hu-Fu: Efficient and secure spatial queries over data federation," in *Proc. Int. Conf. Very Large Data Bases*, 2022, pp. 1159–1172.

[25] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th ACM Symp. Operating Syst. Princ.*, 2017, pp. 51–68.

[26] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *Proc. 13th USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 45–59.

[27] Z. Xu, S. Han, and L. Chen, "PAS: Enable partial consensus in the blockchain," in *Proc. 26th Database Syst. Adv. Appl.*, Springer, 2021, pp. 375–392.

[28] C. Xu, C. Zhang, J. Xu, and J. Pei, "SlimChain: Scaling blockchain transactions through off-chain storage and parallel processing," in *Proc. Int. Conf. Very Large Data Bases*, 2021, pp. 2314–2326.

[29] Z. Xu, S. Han, and L. Chen, "CUB, a consensus unit-based storage scheme for blockchain system," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 173–184.

[30] X. Qi, Z. Zhang, C. Jin, and A. Zhou, "A reliable storage partition for permissioned blockchain," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 1, pp. 14–27, Jan. 2021.

[31] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy, "BlockchainDB: A shared database on blockchains," in *Proc. Int. Conf. Very Large Data Bases*, 2019, pp. 1597–1609.

[32] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2017, pp. 303–312.

[33] Y. Chen et al., "Forerunner: Constraint-based speculative transaction execution for ethereum," in *Proc. 28th ACM Symp. Oper. Syst. Princ.*, 2021, pp. 570–587.

[34] Y. Li et al., "FASTBLOCK: Accelerating blockchains via hardware transactional memory," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 250–260.

[35] P. Garamvölgyi, Y. Liu, D. Zhou, F. Long, and M. Wu, "Utilizing parallelism in smart contracts on decentralized blockchains by taming application-inherent conflicts," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 2315–2326.

[36] C. Jin, S. Pang, X. Qi, Z. Zhang, and A. Zhou, "A high performance concurrency protocol for smart contracts of permissioned blockchain," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5070–5083, Nov. 2022.
[37] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2021, pp. 165–175.
[38] J. Xiao, S. Zhang, Z. Zhang, B. Li, X. Dai, and H. Jin, "Nezha: Exploiting concurrency for transaction processing in dag-based blockchains," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst.*, 2022, pp. 269–279.
[39] S. Zhang and J.-H. Lee, "Analysis of the main consensus protocols of blockchain," *ICT Exp.*, vol. 6, no. 2, pp. 93–97, 2020.
[40] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 1999, pp. 173–186.

Junpei Ni (Student Member, IEEE) received the bachelor's degree from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2020. He is currently working toward the master's degree supervised by prof. Jiang Xiao. His research interests mainly include blockchain and distributed systems.

Jiang Xiao (Member, IEEE) received the BSc and PhD degrees from the Hong Kong University of Science and Technology, in 2009 and 2014, respectively. She is currently a professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. Her research interests include blockchain, and distributed computing. Her awards include CCF-Intel Young Faculty Research Program 2017, Hubei Downlight Program 2018, ACM Wuhan Rising Star Award 2019, Knowledge Innovation Program of Wuhan-Shuguang 2022, and Best Paper Awards from IEEE ICPADS/GLOBECOM/GPC/BLOCKCHAIN.

Shijie Zhang (Student Member, IEEE) received the MS degree from the Department of Software, Sangmyung University, Republic of Korea, in 2019. He is currently working toward the PhD degree with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), supervised by prof. Jiang Xiao. His current research mainly interests include blockchain and distributed systems.

Bo Li (Fellow, IEEE) received the BEng (summa cum laude) degree in computer science from Tsinghua University, Beijing, China, and the PhD degree from ECE Department, University of Massachusetts, Amherst. He is a chair professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He was a Cheung Kong Scholar visiting chair professor in Shanghai Jiao Tong University (2010-2016), and was the chief technical advisor for ChinaCache Corp. (NASDAQ:CCIH), a leading CDN provider. He made pioneering contributions in multimedia communications and the Internet video broadcast, which attracted significant investment from industry and received the Test-of-Time Best Paper Award from IEEE INFOCOM (2015). He received 6 Best Paper Awards from IEEE including INFOCOM (2021). He was the Co-TPC chair for IEEE INFOCOM (2004).

Baochun Li (Fellow, IEEE) received the BEngr degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995, and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a professor. He holds the Bell Canada Endowed chair in computer engineering since August 2005. His research interests include cloud computing, distributed systems, datacenter networking, and wireless systems. He has co-authored more than 420 research papers, with a total of more than 22000 citations, an H-index of 84 and an i10-index of 286, according to Google Scholar Citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems, in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM.

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, in 1994. He is a chair professor of computer science and engineering with the Huazhong University of Science and Technology (HUST), China. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked with the University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China, in 2001. He is a fellow of CCF, and a life member of the ACM. He has co-authored more than 20 books and published more than 900 research papers. His research interests include computer architecture, parallel and distributed computing, Big Data processing, data storage, and system security.