Mitigating Conflicting Transactions in Hyperledger Fabric-Permissioned Blockchain for Delay-Sensitive IoT Applications

Xiaoqiong Xu, Xiaonan Wang, Zonghang Li, Hongfang Yu, *Member, IEEE*, Gang Sun, *Member, IEEE*, Sabita Maharjan, *Senior Member, IEEE*, and Yan Zhang, *Fellow, IEEE*

***Abstract*—Blockchain is a promising emerging technology that is envisioned to play a key role in establishing secure and reliable Internet-of-Things (IoT) ecosystems without the involvement of any third party. Hyperledger Fabric, a permissioned blockchain system that can yield high throughput and low consensus delay, has shown its capability in enhancing security and privacy protection for delay-sensitive IoT services. The literature, however, has not considered the conflicting transaction problem which may substantially limit the system performance and degrade QoS for the end users. In this article, we propose CATP-Fabric, a new blockchain system to address the conflicting transaction problem by reducing the number of potentially conflicting transactions with less overhead. First, the transactions within a block are divided into different groups to facilitate parallel transaction processing. Then, CATP-Fabric filters stale transactions and prioritizes the read-only transactions in each group to eliminate unnecessary overhead. Finally, we formulate the selection of aborting transactions in CATP-Fabric as a binary integer-programming problem and develop a low-complexity optimization algorithm to minimize the number of aborted transactions. Illustrative results show that our proposed CATP-Fabric blockchain system achieves high throughput of successful transactions while maintaining a lower aborting transaction rate compared to the benchmark blockchain systems.**

***Index Terms*—Conflicting transaction, hyperledger fabric, Internet of Things (IoT), permissioned blockchain.**

Manuscript received October 12, 2020; revised December 2, 2020; accepted December 23, 2020. Date of publication January 8, 2021; date of current version June 23, 2021. This work was supported by the PCL Future Greater-Bay Area Network Facilities for Large-Scale Experiments and Applications under Grant PCL2018KP001. *(Corresponding authors: Hongfang Yu, Gang Sun.)*

Xiaoqiong Xu, Xiaonan Wang, and Zonghang Li are with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: xiaoqiongxu810@gmail.com; lrqnrsm66@126.com; lizhuestc@gmail.com).

Hongfang Yu is with the Key Laboratory of Optical Fiber Sensing and Communications (Ministry of Education), University of Electronic Science and Technology of China, Chengdu 610054, China, and also with the Networks and Communications Research Center, Peng Cheng Laboratory, Shenzhen 518055, China (e-mail: yuhf@uestc.edu.cn).

Gang Sun is with the Key Laboratory of Optical Fiber Sensing and Communications (Ministry of Education), University of Electronic Science and Technology of China, Chengdu 610054, China, and also with the Agile and Intelligent Computing Key Laboratory of Sichuan Province, Chengdu 611731, China (e-mail: gangsun@uestc.edu.cn).

Sabita Maharjan and Yan Zhang are with the Department of Informatics, University of Oslo, 0373 Oslo, Norway, and also with the Simula Metropolitan Center for Digital Engineering, 0167 Oslo, Norway (e-mail: sabita@ifi.uio.no; yanzhang@ieee.org).

Digital Object Identifier 10.1109/JIOT.2021.3050244

I. Introduction

THE RAPID development of 5G communication technologies promotes the widespread application of Internet-of-Things (IoT) services, such as smart home, smart healthcare, autonomous vehicles, and smart cities [1], [2]. The IoT network is connected to billions of heterogeneous devices that produce a vast amount of data and enable new services and products for consumers. However, IoT networks may face serious security and privacy threats, such as Distributed Denial of Service (DDoS) attack and privacy leakage attack [3], [4]. These threats hinder the widespread implementation of IoT services. Many works have attempted to address such issues [5]–[7], but the proposed solutions are of limited use due to the lack of trust among parties and lack of transparency in data processing.

Blockchain, a distributed tamper-proof ledger, is a promising technology to deal with the aforementioned security and privacy issues [8]–[12]. Considering the delay-sensitive features of IoT applications [13], the permissioned blockchain Fabric [14] is more suitable for supporting IoT interactions as it offers low transaction delay and high throughput. Fabric reaches consensus in a relatively shorter time by limiting the number of nodes participating in the consensus establishment process, achieves high throughput through a novel transaction processing paradigm named execute-order-validate, and overcomes the single point of failure problem with the use of a decentralized architecture.

Combining Fabric blockchain and IoT can lead to conflicting transactions, a new and unique challenge that needs to be addressed to fully realize the benefits of the integration. In the typical IoT scenarios, IoT devices are expected to collect and exchange data in real time. To ensure privacy protection, these data are stored in the Fabric blockchain in the form of transactions. When the number of IoT devices is large, the data collection will have a concurrent bottleneck, that is massive transactions are prone to conflicting on concurrent data [15]. These conflicting transactions are marked invalid and aborted only during the validation phase, thus wasting a large amount of network resources. Besides, IoT devices need to retransmit these aborted conflicting transactions, which can lead to additional energy consumption and delay. Therefore, a conflict mitigating transaction processing with low energy and time overhead is urgently needed, to efficiently process conflicting transactions in Fabric.

We observe some efforts to mitigate the effects of conflicting transactions in the Fabric blockchain. Xu *et al.* [16] used a locking mechanism to detect conflicting transactions at the beginning of the transaction flow. This method handles almost all conflicting transactions successfully in the case of high concurrency. However, a trusted distributed locking service is needed to synchronize access, which requires coordination time and costs excessive network resources, making the solution less suitable for the delay-sensitive IoT applications. Amiri *et al.* [17] and Sharma *et al.* [18] recorded a transaction dependency graph in each block to detect conflicting transactions, and reordered transactions to minimize the number of unnecessary conflicts. Unlike the locking mechanism, these reordering methods do not need additional service and extra coordination time. Moreover, it can increase the number of valid transactions in each block to partially avoid unnecessary retransmissions, thereby reducing the energy overhead of IoT devices. However, such reorderable transactions do not always exist, and a fairly high number of transactions are still aborted due to conflicts in high concurrency IoT applications. As a result, further mitigating the problem of conflicting transactions with a lower number of retransmissions is a prominent challenge for delay-sensitive IoT applications, and more work is needed in this direction.

In this article, we propose *CATP-Fabric*, a conflicting transaction-tolerant permissioned blockchain system. *CATP-Fabric* first divides transactions into different groups based on transaction keys. Then, it lets read-only transactions with high priority to schedule during the ordering phase and filters stale transactions as early as possible. Then, *CATP-Fabric* detects conflicting transactions by calculating the final balance for all accounts and aborts parts of transactions that should have conflicts with the remaining ones in the validation phase. Finally, we design an optimal aborting transaction selection algorithm to further minimize the number of aborted transactions.

The contributions of this article are summarized as follows.

1) We propose a permissioned blockchain system named *CATP-Fabric*, by introducing three plug-in modules, to effectively mitigate conflicting transactions with a lower number of retransmissions.
2) We propose a key-based transaction grouping method to enhance parallel transaction processing. We also design a transaction filtering mechanism and a priority mechanism to save network resources, thus further improving the system performance.
3) We formulate conflicting transaction selection problem as a binary integer-programming (BIP) problem and propose a new aborting transaction selection algorithm to obtain the optimal *transactions-aborting-set*. Numerical results based on the smallbank benchmark data set corroborate that our *CATP-Fabric* blockchain achieves a lower transaction aborting rate compared to existing solutions.

The remainder of this article is organized as follows. In Section III, we introduce the problem of conflicting transactions in the Fabric blockchain. In Section IV, we present our *CATP-Fabric* system in detail. In Section V, we formulate the aborting transaction selection problem and propose a low-complexity optimization algorithm to obtain the optimal solution to it. Extensive experiments are presented in Section VI. Finally, the conclusion follows in Section VII.

II. Related Works

A. Blockchain Adaptation for IoT

With the rapid development in IoT technology and the widespread deployment of IoT services, it has become not only more important but also more challenging to address security problems, such as the DDoS attack, Sybil attack, and privacy leakage [19]. The emergence of blockchain opens the door to mitigate such security issues in IoT systems by leveraging the tamper-proof feature and decentralized consensus mechanism. We note that efforts have been made to combine blockchain with IoT networks. For instance, Huang *et al.* [20] designed a blockchain with the credit-based consensus mechanism to guarantee the security of Industrial IoT, and proposed a data authority management method to protect the confidentiality of sensitive data. Debe *et al.* [21] also utilized the public Ethereum blockchain to enable decentralized and trustworthy service provisioning between IoT devices and public fog nodes. Considering scalability, Lei *et al.* [22] proposed *Groupchain*, a novel scalable public blockchain with a two-chain structure, that integrates the fog computing of IoT services with blockchain. These blockchains promote the implementation and deployment of the trusted, secure, and transparent IoT networks. However, all the above works are based on blockchains with computation-intensive consensus mechanisms, in which miners always compete with each other to create new blocks by solving a difficult Proof-of-Work (PoW) puzzle. These PoW-based blockchains result in high energy consumption while the performance is relatively poor. Such blockchains are, therefore, not suitable for energy-constrained and delay-sensitive IoT networks.

To address such issues, some studies proposed to utilize the Fabric blockchain with a deterministic consensus mechanism to replace the PoW-based blockchains. Fabric [14] is one of the prominent permissioned blockchains, which offers significantly higher throughput compared to Bitcoin and Ethereum, and achieves a reasonably fast consensus. Based on Fabric blockchain, Liu *et al.* [23] proposed an opensource access control system named Fabric-IoT to provide fine-grained and dynamic access control management for IoT networks. Liang *et al.* [24] used the Fabric blockchain-based dynamic secret sharing mechanism to ensure secure data transmission in industrial IoT networks. These solutions are, however, of limited applicability for IoT applications with a large number of conflicting transactions, that can consequently lead to the failure of a large number of transactions and considerably degrade the overall performance.

B. Conflicting Transactions in Fabric Blockchain

A few works have attempted to address the conflicting transaction problem in Fabric blockchain. Xu *et al.* [16] proposed a locking-based mechanism by detecting conflicting transactions at the early stage of Fabric's transaction flow. This locking-based method can mitigate the effect of conflicting transactions, but it is of limited practical value for energy-constrained IoT applications due to larger coordination overhead. Sharma *et al.* [18] proposed to use a dependency graph between transactions to detect possible conflicts between transactions and designed a reordering mechanism to alleviate conflicting transactions within a block. This reordering mechanism has been shown as a practical approach for mitigating conflicting transactions without a centralized coordinator. However, there is still a higher number of transaction retransmissions due to unnecessary aborts, leading to high energy and time overhead. Besides, Nasirifard *et al.* [25] proposed the FabricCRDT blockchain system to address conflicting transactions by integrating conflict-free replicated datatypes (CRDTs) into Fabric blockchain. FabricCRDT successfully merges all conflicting transactions without any failures, but it requires a specified data structure for CRDT transactions, thus not suitable for many IoT applications. In this article, we propose a conflicting transaction mitigating solution for Fabric blockchain, where the energy consumption and latency requirements of IoT applications are incorporated.

III. Background and Motivation

In this section, we first present the overview of the Fabric blockchain for IoT applications. Then, we define the problem of conflicting transactions in Fabric IoT systems, and explain with an example how existing approaches have attempted to address this problem.

A. Fabric IoT System Infrastructure

In the Fabric IoT systems, the massive data produced by IoT devices (such as sensors and actuators) are transmitted to Fabric blockchain in the form of transactions. Nodes in the Fabric blockchain start to validate and store these transactions under an execute-order-validate transaction flow. In terms of functional division, these nodes can be divided into three categories, i.e., client, peer, and ordering nodes. Client nodes are those energy-constrained IoT devices, they send new transactions and do not store blockchain ledger. Peer nodes are IoT devices with processing and storage capacity such as gateways, and their main duty is to maintain blockchain ledger and execute the chaincode. Ordering nodes are powerful devices like servers, which produce new blocks according to the consensus mechanism. All nodes in Fabric IoT systems are identified by a specific manager that also helps to manage (add/delete) the IoT devices and Fabric blockchain.

The transaction flow in Fabric blockchain consists of three phases: 1) simulation; 2) ordering; and 3) validation, as shown in Fig. 1. In the following, we describe those phases in more detail.

Simulation Phase: Client node sends a transaction proposal with its identifier to one or more peer nodes, called endorsers.

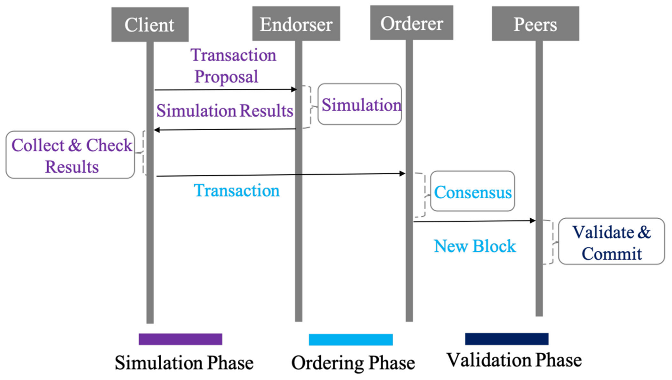


Fig. 1. Transaction flow in fabric involves three phases: simulation, ordering, and validation.

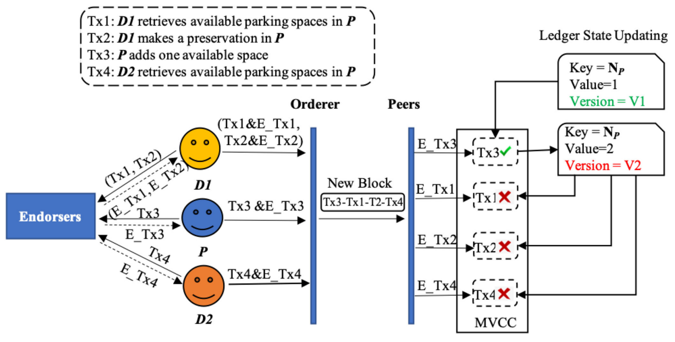
The endorsers verify the client's identifier and execute the transaction proposal based on the preinstalled chaincode. After that, each endorser builds a readset (RS) and a writeset (WS) as the simulation result. The RS contains the keys read during the execution and the corresponding versions, while the WS consists of the modified keys, versions, and updated values. Note that the simulation results do not directly modify the ledger state. After the execution, endorsers return the simulation results with their signatures to the client. When the client has collected enough simulation results, it checks the consistency of all results and assembles them into a transaction.

Ordering Phase: The client submits this assembled transaction to the ordering service (ordering node). The transaction contains a transaction proposal, a set of simulation results, and a channel ID. The ordering nodes maintain the order of all submitted transactions per channel without inspecting the contents of transactions. Then, the ordering nodes batch a certain number of transactions per channel into a new block and produce a hash-chained value for this block. Ordering nodes, then, deliver this block to all peers using a gossip protocol.

Validation Phase: The block is received by all peers and enters the validation phase. For all transactions within the block, peers first execute the validation system chaincode (VSCC) in parallel to validate the simulation results of the transaction. If the simulation results do not satisfy the requirements of the endorsement policy, the transaction is marked as invalid and aborted. Second, peers carry out a read–write conflict check for all transactions sequentially. The check includes whether the versions of the keys in the RS or WS field are the same as those in the current ledger state, that is, whether the transaction conflicts with any preceding transactions. If the versions are different, the transaction is also marked as invalid and aborted. Finally, the block is appended to the blockchain, and the ledger changes its state from the WS of all valid transactions and updates the version number of the modified keys.

B. Conflicting Transaction Problem in Fabric IoT Systems

In Fabric blockchain, only a few nodes (endorsers) execute transactions, thereby achieving a fast consensus and making it more suitable for the delay-sensitive IoT applications. However, IoT devices report data continuously, which



Simulation Results		
Transactions	Readset	Writeset
E_Tx1	$\{N_P, V1\}$	null
E_Tx2	null	$\{N_P, V1, 0\}$
E_Tx3	null	$\{N_P, V1, 2\}$
E_Tx4	$\{N_P, V1\}$	null

Fig. 2. Sample of conflicting transactions in Fabric IoT system: there are three clients (two drivers $\mathcal{D}1$ and $\mathcal{D}2$ and one parking lot $\mathcal{P}$ in the smart packing scenario) who propose four transactions (Tx1, Tx2, Tx3, Tx4) at the same time or in a short time interval. These four transactions are batched into a block at orderer and then validated by all peers. Finally, only Tx3 is valid to write into the ledger, others are invalid and aborted due to conflicting with Tx3, resulting in 75% aborting transactions.

need to be processed and stored on Fabric blockchain in real time, leading to high concurrency. When multiple transaction requests access the same entry, conflicts can occur in Fabric blockchain. These conflicting transactions are marked as invalid only during the validation phase and are completely aborted, resulting in inefficient use of resources. The mathematical definition of a conflicting transaction within a block is given as follows.

Definition 1 (Conflicting Transaction): Let Tx_i and Tx_j be two transactions. After being executed by endorsers, the simulation results of Tx_i are $\text{RS}(\text{Tx}_i)$ and $\text{WS}(\text{Tx}_i)$, where $\text{RS}(\text{Tx}_i)$ represents the readset and $\text{WS}(\text{Tx}_i)$ is the writeset. The simulation results of Tx_j are $\text{RS}(\text{Tx}_j)$ and $\text{WS}(\text{Tx}_j)$. Assume that Tx_i is ordered before Tx_j within a block, then Tx_j will be a conflicting transaction which will be marked invalid and discarded in the validation phase when one of the following conditions is true.

1) $\text{WS}(\text{Tx}_i) \cap \text{RS}(\text{Tx}_j) \neq \varnothing$.
2) $\text{WS}(\text{Tx}_i) \cap \text{WS}(\text{Tx}_j) \neq \varnothing$.

Next, we present an example to discuss the impact of conflicting transactions in detail.

Example: The smart parking system [26] is a typical IoT scenario, where each parking lot periodically collects the number of available spaces through IoT devices and sends these data to the Fabric blockchain. Drivers can send transactions to the Fabric blockchain to retrieve parking offers and make an online reservation. Let us assume there are two drivers ($\mathcal{D}1$, $\mathcal{D}2$) and one parking lot ($\mathcal{P}$), as shown in Fig. 2. In the initial ledger state, the key of the available spaces N_P in the parking lot is 1 and the version number is $V1$. There are four transactions (Tx1, Tx2 from $\mathcal{D}1$, Tx3 from $\mathcal{P}$, and Tx4 from $\mathcal{D}2$) that are submitted to the Fabric blockchain in a short time interval. According to the transaction flow of Fabric, these four transactions are sent to endorsers for execution to obtain simulation results (as shown in the Simulation Results Table in Fig. 2). Subsequently, these four transactions are sent to the ordering nodes to be batched into a new block with a consistent order Tx3⇒Tx1⇒Tx2⇒Tx4. Then, all peers receive this new block from the ordering nodes and perform VSCC and read–write conflict checks. Peers first compare the local ledger state with the simulation result of Tx3. The version corresponding to key N_P in the WS field is consistent with those in peers' local ledger state. Therefore, this transaction Tx3 is marked as valid to be stored in the ledger, while the value of N_P in the ledger is updated to 2 and the version of N_P is updated to $V2$ successfully. Afterward, transaction Tx1 comes to be validated. Because the version of N_P in the ledger state has been changed to $V2$, so Tx1 is marked as invalid and aborted. Likewise, peers find that Tx2 and Tx4 also have conflicts with preceding transactions. Thus, Tx1, Tx2, and Tx4 fail to be committed and require retransmissions resulting in extra resource and time overhead.

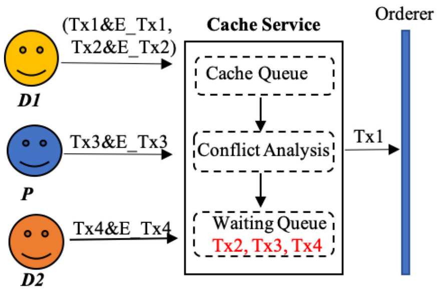


Fig. 3. How the *caching mechanism* [27] mitigates the conflicting transaction problem: there is a cache service between IoT clients and the Fabric system to remove conflicting transactions into the waiting queue by analyzing the relationship of different transactions RS & WS.

C. Existing Solutions

There are some existing solutions to this problem, which can be divided into two categories: 1) *caching mechanism* [27], a simple approach to mitigate the conflicting transaction problem with a cache service and 2) *reordering mechanism* [17], [18], where the main idea is to turn partial conflicting transactions into conflict-free transactions based on transaction dependency graph analysis.

The caching mechanism detects conflict at an early stage of the transaction flow. As shown in Fig. 3, after receiving enough simulation results from endorsers, clients assemble transactions and submit them to a cache service. This novel cache service first buffers transactions, then analyzes whether there are conflicts between different transactions. If one transaction (e.g., Tx1 in our example) is conflict-free with the preceding transactions, this transaction is sent to the ordering service directly. Otherwise, transactions (e.g., Tx2, Tx3, and Tx4 in our example) conflicting with the preceding ones (Tx1) are moved to a conflict queue. After conflict-free transactions (e.g., Tx1) are committed successfully, one transaction, e.g., Tx2, can be removed from the conflict queue and sent to endorsers for execution again. if this transaction (Tx2) is

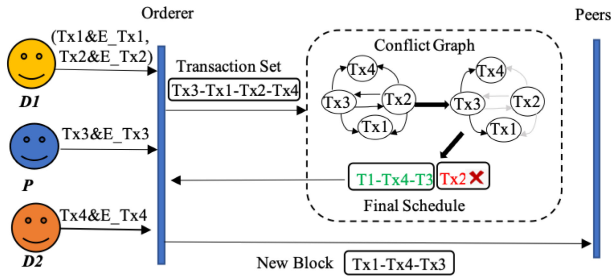


Fig. 4. How the *reordering mechanism* [18] mitigates the conflicting transaction problem: it introduces a transaction reordering mechanism that aims at minimizing the number of unnecessary conflicting transactions within each block.

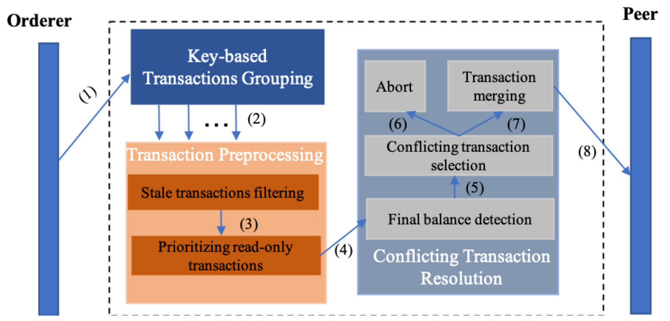

Orderer
Peer
Key-based Transactions Grouping
(1)
(2)
Transaction Preprocessing
Stale transactions filtering
(3)
Prioritizing read-only transactions
(4)
Abort
Transaction merging
(6)
(7)
Conflicting transaction selection
(5)
Final balance detection
Conflicting Transaction Resolution
(8)

Fig. 5. Overview of *CATP-Fabric*. After finishing the execution phase (same as vanilla Fabric, not presented), clients send the transactions and simulation results to the ordering service. This ordering service batches the receiving transactions into a new block and sent it to the key-based transaction grouping module (1). Transactions are divided into different groups to preprocess (2). Then, low version transactions filtered (3) and read-only transactions prioritized (4). After that, final balance detection for all keys within the block to decide whether to perform transaction aborting (5). If need, a conflicting transaction selection mechanism gets a transaction set to abort (6) and other transactions are merged (7). Finally, this block only with *final-transactions* is delivered to all peers for validation (8).

conflict-free in this round, it will be submitted to the ordering service and committed. Finally, other transactions, e.g., Tx3 and Tx4, are also removed from the conflict queue and committed sequentially after both Tx1 and Tx2 are committed.

Indeed, all four transactions in our example are committed successfully to the Fabric blockchain based on the caching mechanism. Unfortunately, simply caching conflicting transactions brings some challenges: 1) the cache service can be a bottleneck, large buffer space is required when the transaction arrival rate is high or when transactions are highly concurrent; 2) the average transaction latency is very high due to the waiting time of conflicting transactions; and 3) the additional resource overhead is unavoidable due to the need of execution and conflict analysis once again. Thus, the caching mechanism is not suitable for the energy-constrained and delay-sensitive IoT applications.

The *Reordering mechanism* is requested as a part of the ordering service. After establishing an order agreement and constructing a block for all pending transactions, the ordering service generates a dependency graph for transactions within this block based on the relationship of their RS and WS. Then, it forms a cycle-free graph by removing certain transactions and reorders the remaining transactions to create a serializable schedule. With this serializable schedule, all transactions can be marked as valid and committed successfully in the validation phase. Recalling the same example in Section III-B, all four transactions Tx1, Tx2, Tx3, and Tx4 are sent to the ordering service. Then, the ordering service constructs a conflict graph as shown in Fig. 4. Then, Tx2 is removed to get a cycle-free graph and a serializable schedule with Tx1⇒Tx4⇒Tx3 is provided by the ordering service.

Compared to the *caching mechanism*, the *reordering mechanism* increases the number of valid transactions within a block without rolling back the processed transactions. This can indeed improve resource utilization and decrease transaction latency. There are, however, some issues with the reordering mechanism. In our example, Tx2 is aborted in the ordering phase because it conflicts with other transactions. However, checking the value of the available spaces N_P in the parking lot indicates that it is available for offering parking service, that is, Tx2 is not a double-spending transaction. Tx2 can be committed successfully by retransmission. We call this type of transaction *final valid transaction* and provide its mathematical definition below.

Definition 2 (Final Valid Transaction): A client issues a transaction Tx to perform a write operation on a key x. If the value of the key x in the ledger state is sufficient to support the operation of Tx, then Tx is called a final valid transaction, i.e., transactions that are not double-spending for all keys. Certainly, all read-only transactions are final valid ones.

The final-valid transactions can be executed and committed to the ledger successfully, but have to go through at least two rounds of endorsing and ordering in the reordering mechanism, thus leading to an inefficient use of resources and high transaction delay. In some special scenarios with high concurrency, this problem is more prominent due to a considerable number of aborted transactions that require retransmissions, resulting in high latency and lower QoS for end users.

IV. *CATP-Fabric* Architecture

In this section, we present an overview of the proposed *CATP-Fabric* system. Conflicting transactions in *CATP-Fabric* are operated and processed in the ordering service by introducing three modules: 1) the key-based transaction grouping module; 2) transaction preprocessing module; and 3) conflicting transaction resolution module.

The flow of transactions in *CATP-Fabric* is depicted in Fig. 5: after finishing the simulation phase (same as vanilla Fabric), clients send transactions and simulation results to the ordering service. When the ordering service receives a certain number of transactions, it batches all pending transactions into a new block. Due to the high delay associated with sequential transaction execution within a block, transactions in this new block are divided into a set of independent groups so that they can be processed in parallel in the subsequent steps. Then, for transactions in each group, the preprocessing mechanism filters stale transactions (defined in Section IV-B) that may cause data inconsistency in the validation phase. After that, transactions in each group will be the read or write ones with a uniform version of RS and WS. Because the read-only transactions do not modify the value and version number of

Algorithm 1: Key-Based Transaction Grouping

```
Input: $Block = \{\text{Tx}_1, \text{Tx}_2, \ldots, \text{Tx}_N\}$: a new block;
Output: $S$: The grouped transaction set;
1  Initialize $S = \emptyset$;
2  for $\text{Tx}_i \in Block$ do
3      for $\langle key_m,\ version_m\rangle \in \text{Tx}_i\,.\,readset$ do
4          Creat a sub_transaction $\text{Tx}_i(m)$;
           $\text{Tx}_i(m)\,.\,readset = \langle key_m,\ version_m\rangle$;
           if $key_m \in S\,.\,groupIDs$ then
5              $S[key_m]\,.\,append(\text{Tx}_i(m))$;
6          end
7          else
8              Add a group with $key_m$ as group ID to S;
               $S[key_m] = \text{Tx}_i(m)$;
9          end
10     end
11     for $\langle key_n,\ version_n,\ value_n\rangle \in \text{Tx}_i\,.\,writeset$ do
12         Creat a sub_transaction $\text{Tx}_i(n)$;
           $\text{Tx}_i(n)\,.\,writeset = \langle key_n,\ version_n,\ value_n\rangle$;
           if $key_n \in S\,.\,groupIDs$ then
13             $S[key_m]\,.\,append(\text{Tx}_i(n))$;
14         end
15         else
16             Add a group with $key_n$ as group ID to S;
               $S[key_n] = \text{Tx}_i(n)$;
17         end
18     end
19 end
20 return $S$;
```

keys in the ledger state, these transactions are final valid ones based on Definition 2 in Section III-C. Thus, giving the highest priority to these transactions can reduce the number of conflicting transactions within a block. Finally, the conflicting transaction resolution module divides transactions in each group into nonfinal valid transactions and final-valid transactions. The nonfinal valid transactions are discarded directly and all final-valid transactions are sent to peers for validation.

Next, we elaborate on the details of the design of these three modules.

A. Key-Based Transaction Grouping

Peers in the Fabric blockchain maintain the ledger state in the form of versioned key value, and conflicting transactions occur due to the read or write operations on the same keys according to Definition 1 in Section III-B. Thus, to enable highly parallel processing of conflicting transactions, we design a grouping algorithm using the keys in RS or WS fields of transactions as group ID to divide transactions into different groups.

As shown in Algorithm 1, we iterate through all transactions within a block. For each transaction, we iterate through the key-value pairs of its RS (lines 3–10) and WS (lines 11–18). The algorithm first checks if a group ID for the key in RS or WS field already exists in the group set S. If it does not exist, the algorithm instantiates a new group with the key as a group ID and adds subtransaction to this new group. This subtransaction includes the same content with their parent transaction, and a RS or WS only with a single key. Afterward, the transactions are converted to multiple subtransactions with only one key-value pair and inserted into the corresponding groups. Finally, all transactions in each group are processed in parallel in the subsequent steps.

Let us consider an example here. A parent transaction Tx is: a driver $\mathcal{D}$ retrieves the available parking spaces in parking lots $P1$ and $P2$. The simulation results of Tx are $\{(N_{P1}$, version), $((N_{P2}$, version)$\}$. Then, these two subtransactions are split: Tx_1 with RS $\{(N_{P1}$, version$\}$ and Tx_2 with RS $\{(N_{P2}$, version$\}$. Finally, Tx_1 is added in to the group with group ID N_{P1} and Tx_2 to the group with group ID N_{P2}.

B. Transaction Preprocessing

After dividing transactions within a block into different groups, we run a lightweight transaction preprocessing mechanism for all groups in parallel. First, since transactions with outdated versions are aborted in the validation phase due to data inconsistency, the preprocessing mechanism filters these outdated transactions at an early stage to save resources and to reduce time overhead. We identify these outdated transactions as *stale transactions* and provide the mathematical definition of stale transactions as follows.

Definition 3 (Stale Transaction): A client issues transaction Tx to operate on a key x, and another transaction Tx$'$ also updates the value of the key x before Tx. After Tx is endorsed in the simulation phase, and Tx$'$ is committed to update the version and value of the key x in the ledger state. Since the simulation results of Tx have an older version of key x in the RS or WS, Tx fails validation. In this case, transaction T_x is called a stale transaction, i.e., transaction with the outdated version number in their RS or WS field.

In general, stale transactions occur due to additional latency between the simulation and validation phases, outdated ledger state of offline peers, and can also be a result of malicious behavior [25], [28]. If a transaction is stale, it is bound to be aborted in the validation phase. This untimely discarding results in unnecessary transmission overhead from ordering service to validation peers. Thus, filtering stale transactions at the early stage of the ordering phase can save resources and reduce the average transaction completion time. In *CATP-Fabric*, we sort transactions in each group based on their version number. Then, we filter out the transactions with lower version numbers, and only keep the transactions with the highest version.

Besides, after grouping, each group contains either read or write transactions for the same key. Since the read-only transactions only query the current ledger state and do not execute any operations on the value or version of the keys. Reordering all read-only transactions to handle them first can ensure that all read-only transactions can be committed successfully in the validation phase, thereby reducing the number of transaction retransmissions. Moreover, it also decreases the possibility of conflict with subsequent write transactions. For transactions in each group, the preprocessing mechanism performs reordering such that the read-only transactions are followed by write transactions.

C. Conflicting Transaction Resolution

From the observations of the example in Section III-B, reordering mechanism aborts the final valid transactions that have conflicts with other transactions in the current round of updates. However, this type of transaction can be processed successfully in the next round because the value (*or* balance) of the key in the WS field of the transaction is sufficient to support its payment. Discarding final valid transactions leads to nontrivial transaction latency and retransmission overhead. To avoid the failure of these final valid transactions, in *CATP-Fabric*, we design a novel conflicting transaction resolution module. We first apply the final balance detection method to check whether there are transactions in each group making the value of some key exceed its balance. If such transactions exist, a conflicting transaction selection mechanism divides transactions within a block into nonfinal valid transactions or final valid ones. Then, nonfinal valid transactions are aborted directly and all final valid write transactions in the same group are integrated into a total transaction with the uniform version to be sent to all peers for validation.

1) Final Balance Detection: For all keys in the group set S within a block, we implement the final balance detection concurrently. Assume that there are K write transactions $\{\mathrm{Tx}_1, \mathrm{Tx}_2, \ldots, \mathrm{Tx}_K\}$ in a group $S_m \in S$ with group ID m, and the WS of Tx_i ($i \in [1, K]$) is $\langle key_i, value_i, version_i \rangle$. Then, the final balance Bal_m of key m associated with group S_m is

$$\mathrm{Bal}_m = \mathrm{Bal}_m^{\mathrm{init}} + \sum_{i=1}^{K} \Delta(\mathrm{Tx}_i) \tag{1}$$

where $\mathrm{Bal}_m^{\mathrm{init}}$ is the value of key m in the ledger state and $\Delta(\mathrm{Tx}_i)$ is the updated value of transaction Tx_i, which can be calculated as follows:

$$\Delta(\mathrm{Tx}_i) = value_i - \mathrm{Bal}_m^{\mathrm{init}}. \tag{2}$$

If the final balance of keys associated with all groups are nonnegative, transactions within this new block are both final valid transactions that can be merged to avoid failure in the validation phase. In contrast, a part of transactions are selected to ensure the nonnegative final balance of all keys and are early aborted. The optimal conflicting transaction selection will be discussed in Section V.

2) Transaction Merging Mechanism: When the final balance of all keys within a block are nonnegative, a merging mechanism is executed for all groups in parallel to avoid the failure of transaction validation. First, it reorders all read-only transactions in front of the block queue. Afterward, for write transactions with the same key and version, the necessary merging operation is performed to integrate these write transactions into a total transaction. The new total transaction has the WS <*key*, *value*, *version*>, where the *value* is the final balance Bal based on the sum of all write value, and the *version* is the highest version number. The merging operation also holds a dependency list containing specific write transactions. Finally, this total write transaction is appended to the block queue after all read-only transactions.

V. Transaction Aborting Decision: Problem Formulation and Algorithm Design

When the final balance of some keys are not sufficient to support operations of all transactions within a block, we consider aborting parts of transactions to guarantee the nonnegative final balance for all keys. That is, given a block with group set $S = \{S_1, S_2, \ldots, S_N\}$, we use (1) to calculate the final balance for all keys associated with S, if $\mathrm{Bal}_m < 0$ ($m \in [1, N]$), we consider aborting some transactions.

A. Problem Formulation

Let us define the problem of transaction aborting formally. Assume that A is a transaction set in a block, the goal of the aborting transaction selection problem is to find a subset of transactions $A' \subset A$ to abort, such that after aborting all transactions in subset A', the remaining transaction set A/A' makes the final balance of all keys nonnegative. To formulate this problem, we model the transaction relations in a block with an associated edge- and node-weighted dependency graph $G = \{S, E, \mathcal{W}, \Theta\}$. In which, the node set $S = \{S_1, S_2, \ldots, S_N\}$ denotes the related N keys (group IDs) in this block.The directed edge set E includes write transactions in a block (since read transactions do not update any values and are prioritized in the preprocessing module, they do not have any conflict with other transactions, thus making it not necessary to insert edges for read-only transactions into the graph). Given a transaction that transfers value from key i to key j ($i, j \in V$), add an edge from node i to node j. $\mathcal{W}$ is an $N \times N$ weight matrix and $w_{ij} \in \mathcal{W} \to R^+$ represents the transferring value from i to j. If there is no transaction from key i to key j, $w_{ij} = 0$. A nonnegative weighted scalar $\Theta = [\theta_1, \theta_2, \ldots, \theta_N]$, where θ_i ($i \in [1, N]$) indicates the value of the key associated with group S_i in the current ledger state. Thus, the aborting transaction selection problem can be converted to searching an edge set $E' \in E$ in G.

We design an actual objective function to select the edge set (aborting transactions) based on delay-sensitive IoT applications. In our design, we consider an objective function that minimizes the size of aborting transactions set A', that also minimizes the retransmission overhead.

The optimization problem with this objective can be converted to a minimum edge set searching problem for graph G. By using a binary variable x_{ij} to indicate whether the edge $i \to j$ is involved in the solution, this problem can be formulated as follows:

$$\mathbf{P1:}\quad \max_{x_{ij}} \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}$$

$$\text{s.t.}\quad x_{ij} \in \{0, 1\} \quad \forall i, j \in V \tag{3a}$$

$$\sum_{j\in V} x_{ji}w_{ji} - \sum_{j\in V} x_{ij}w_{ij} \le \theta_i \quad \forall i, j \in V \tag{3b}$$

$$\sum_{i\in V}\sum_{j\in V} x_{ij} \ge 1 \tag{3c}$$

where $C = \{c_{ij}\}$ is an $N \times N$ binary coefficient matrix, if there is a transaction from i to j, $c_{ij} = 1$; otherwise, $c_{ij} = 0$. While the first item on the left-hand side of constraint (3b) is the sum weights of incoming edges of node i, i.e., the amount transferred to i. The second item is the sum weight of outgoing edges, i.e., the amount received. Constraint (3b) guarantees that the final balance of all keys in this block are nonnegative. Moreover, constraint (3c) ensures that at least one transaction is left behind.

B. Algorithm Design

The key challenge in solving $\boldsymbol{P1}$ is the integer constraint $x_{ij} \in \{0, 1\}$, which makes $P1$ a BIP problem. In general, BIP is NP-hard and cannot be solved without relaxing the integer constraint. Then, by solving the resulting LP, there will be an optimal LP solution that may contain variables with noninteger values. Finally, rounding the values of the LP solution up or down to get an integer-valued solution may lead to feasible solutions for BIP. Unfortunately, the number of possible roundings grows rapidly as the number of integer variables increases, i.e., when the size of the transaction set within a block is very high, it may take too long to find a good integer-valued solution. We, therefore, propose an efficient algorithm based on the branch-and-bound approximation that has been widely used for obtaining BIP solutions.

First, we create the LP relaxation of $\boldsymbol{P1}$ by removing the binary constraints as following:

$$\textbf{LP Relaxation:} \quad \max_{x_{ij}} \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}$$

$$\text{s.t.} \quad 0 \le x_{ij} \le 1 \quad \forall i \in V \tag{4a}$$

$$(3b),\ (3c). \tag{4b}$$

Then, we solve the LP relaxation to obtain a fractional optimal solution $\bar{\mathbf{x}}$ with the optimal objective value $\mathbf{z}_{\text{LP}}$. If the optimal solution $\bar{\mathbf{x}}$ is integer-valued, then it is also an optimal solution to $\boldsymbol{P1}$ and we can stop. Otherwise, the LP relaxation becomes the first unfathomed node of the branch-and-bound tree and $\mathbf{z}_{\text{LP}}$ is the bound on the first unfathomed node. Then, we should choose a variable $\bar{\mathbf{x}}_{ij}$ with fractional value to round up or down and create a new subproblem. To reduce the number of rounds, we use a greedy algorithm to find the optimal branching step.

First, we consider a new trimmed dependency graph $G' = \{S, E', \mathcal{W}, \Theta'\}$. Where the edge set E' only includes the edge for which $0 < \bar{\mathbf{x}}_{ij} < 1$. The weighted scalars Θ' are the updated values after calculating the original Θ with the weights of outgoing or incoming edges $\bar{\mathbf{x}}_{ij} = 1$ in LP relaxation.

Based on G', we calculate the new weighted final balance for all $S_m \in S$ as follows:

$$\text{Bal}'_m = \theta'_m + \sum_{j\in V} \bar{\mathbf{x}}_{ji} * w_{ji} - \sum_{j\in V} \bar{\mathbf{x}}_{ij} * w_{ij} \tag{5}$$

where $0 < \bar{\mathbf{x}}_{ji}, \bar{\mathbf{x}}_{ji} < 1$ form the optimal solution of the LP relaxation problem. From constraint (3b), we know that $\text{Bal}'_m \ge 0$ for all $S_m \in S$. We sort S_m based on their new weighted final balance Bal'_m in increasing order. Then, we choose an outgoing edge of the node with a bottom-ranked weighted final balance to remove. That is, suppose S_r has the lowest weighted final balance and there is an edge from node S_r to node S_k ($S_{r,k} \in S$), then we should add a new constraint $x_{rk} = 0$ to the LP relaxation, creating a new subproblem. Then, we add only one new node to the branch-and-bound tree that is associated with this subproblem, and find the bound for the new node of the branch-and-bound tree by solving these associated subproblems ($\mathbf{LP}'$)

$$\mathbf{LP}'\text{:} \quad \max_{x_{ij}} \quad \sum_{i\in V}\sum_{j\in V} c'_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{ji}w_{ji} - \sum_{j\in V} x_{ij}w_{ij} \le \theta'_i \quad \forall i \in V \tag{6a}$$

$$x_{rk} = 0 \quad r, k \in V \tag{6b}$$

$$(4a),\ (3c) \tag{6c}$$

where $C' = \{c'_{ij}\}$ is an updated coefficient matrix of C. If the solution of variable $\bar{\mathbf{x}}_{ij} = 0$ in LP relaxation, we update $c_{ij} = 0$. If there is an incumbent integer-valued solution, then the incumbent solution is optimal. Otherwise, a new solution is found iteratively by branching-and-bounding on the new remaining graph. The procedure of transaction aborting decision is shown in Algorithm 2.

Algorithm 2: Transaction Aborting Decision

```
Input: $G = \{S, E, \mathcal{W}, \Theta\}$
Output: $E'$: The aborting transactions set.
1  Initialize Set $E' = \emptyset$
2  Built the BIP problem $\boldsymbol{P1}$ based on $G$;
3  Let $\bar{\mathbf{x}}$ be an optimal solution of the LP relaxation;
4      $\mathbf{z}_{LP}$ be the optimal objective value of the LP
   relaxation;
5  if there are unfathomed nodes in the branch-and-bound
   tree then
6      Built the trimmed graph $G'$;
7      Calculate the new weighted final balance $\text{Bal}'$ for all
        node in $S$;
8      Rank the node in $S$ with increasing $\text{Bal}'$;
9      Choose the edge r,k from node with bottom-ranked
        $\text{Bal}'$ to remove;
10     $E' = E' \cup \{i, j\}$
11     Add constraint $\bar{\mathbf{x}}_{rk} = 0$ to LP relaxation;
12     Sovle the new subproblem $\mathbf{LP}'$;
13     $\bar{\mathbf{x}}$ be an optimal solution of the $\mathbf{LP}'$;
14     Go to 5;
15 end
16 for $(i, j) \in \{(1, 2, \ldots, N) \times (1, 2, \ldots, N)\}$ do
17     if $\bar{x}_{ij} = 0$ then
18         $E' = E' \cup \{i, j\}$
19     end
20 end
21 return $E'$
```

VI. Numerical Results

In this section, we present a comprehensive evaluation of our *CATP-Fabric*. We first describe the experimental setup for our prototype and the workload adopted in our experiments. Then, we evaluate the performance of *CATP-Fabric* against vanilla *Fabric* and *Fabric++*. As the caching mechanism is not a distributed mechanism, and as it usually yields high queuing delay, we do not compare CATP-Fabric with it.

A. Setup

Environment: Our experiments are conducted in a local Fabric network with one ordering node and two organizations. Each organization includes three peers and both of them join in a single channel. Each node is launched as a Docker container and then connected to the Fabric network using the Docker Swarm. Containers corresponding to each node (client, peer, or ordering node) are run on an independent physical server in a LAN. Each physical server has four vCPUs (Intel Xeon 2.2 GHz) with 12-GB RAM. Each machine runs Ubuntu 16.04 LTS with Fabric V1.4 installed. All physical servers are connected with a 1 Gb/s Ethernet switch.

Workloads: We use SmallBank [29] as the workload in our experiments. SmallBank is a standard benchmark available in BLOCKBENCH [30] and was used in several studies for evaluating the performance of distributed databases. There are six transaction types in the SmallBank workload for simulating the typical operations under banking applications. *CreatAccount* initializes each customer with a random balance in their checking account and savings account. *TransactSavings* and *DepositChecking* add an amount (can also be a negative number) to the customer's savings account and checking account, respectively. *WriteCheck* removes an amount from the customer's checking account, while *SendPayment* moves funds from one customer's checking account to another's. *Amalgamate* transfers the entire contents of one customer's savings account into another customer's checking account. In addition to the above five write transactions, there is also a read-only transaction *Query* that queries the balance of saving and checking account of a customer. For all experiments in this article, we initialize workload with 10 000 customers and 100 000 transactions. We set to send these six types of transactions randomly, and the probability of generating *Query* transactions (i.e., read transactions) is P_r, while the probability of one of the five writing transactions is $1 - P_r$. Furthermore, we use Zipfian distribution for the Smallbank workload to simulate contention for data access and configure the data skewness by setting the skew parameter. Note that setting the skew parameter with 0 corresponds to a uniform distribution.

We use the Hyperledger Caliper tool [31] to measure the performance under the Smallbank workload. By implementing a set of predefined use cases, Hyperledger Caliper can generate reports, including throughput, latency, and successful transaction numbers. In our experiments, the caliper runs on client nodes and broadcasts Smallbank workloads into the Fabric blockchain. Then, it listens to block events from peers to check for transaction confirmations on the blockchain ledger.

B. Results

We now evaluate the impact of different parameters on the performance of different blockchain systems. We parametrize:

TABLE I
FABRIC BLOCKCHAIN CONFIGURATION

Parameters	Values
Transaction submission rate	50 tps [32]
Number of transactions in a block (block size)	10, 20, 40, 80, 100
Interval time to batch a new block (block interval)	1 second [32]

TABLE II
SMALLBANK WORKLOAD CONFIGURATION

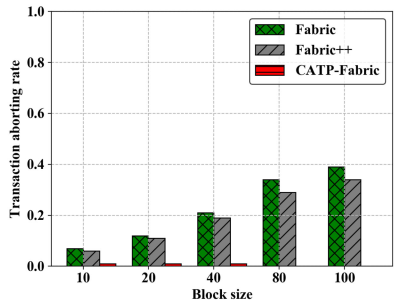
Parameters	Values
Total Number of transactions	$100,000$
Number of customers	$10,000$
Percentage of read transactions	10% - 90% in steps of 10% [25]
Skew parameter of Zipf distribution	0.2 - 2.0 in steps of 0.2 [18]

1) the block size; 2) the skew parameter of Zipfian distribution; and 3) the read transaction percentage. The performance metrics measured in our experiments include the transaction aborting rate to measure the number of transaction retransmissions, average transaction latency, and successful transaction throughput. For all experiments shown in this section, we conduct 500 runs for each parameter and the final results are the average of these sum runs.

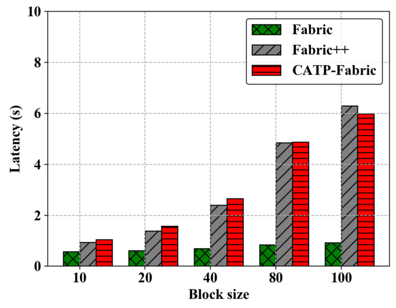
Impact of Block Size: First, to evaluate the impact of block size on performance, we increase the number of transactions in each block from 10 to 100. Similar to [18] and [25], we set the percentage of read transactions as 50% and the skew parameter as 1.0. For other parameters, refer to Tables I and II.

Fig. 6(a) shows the transaction aborting rate under three simulated blockchain systems. In this figure, we set the percentage of read transactions as 50% and the skew parameter as 1.0. For other parameters, refer to Tables I and II. The results show that all three systems only drop 0%–10% transactions with conflicts when the block size is set to 10. However, with an increase of block size from 20 to 100, we see that both *Fabric* and *Fabric++* exhibit a sharp increase in transaction aborting rate. The reason is that the larger block size causes a greater probability of increasing the number of conflicting transactions within a block. Our *CATP-Fabric* keeps the transaction aborting rate much less compared to *Fabric* and *Fabric++*. Moreover, we observe that the transaction aborting rate drops further in *CATP-Fabric* even for block size 80 and 100. The reason is that *CATP-Fabric* can resolve conflicting transactions and integrate most of them into conflict-free transactions. Moreover, the results also indicate that *CATP-Fabric* can achieve a transaction aborting rate close to 0.7%, that is, *CATP-Fabric* can successfully handle almost all transactions in the case of high concurrency conflicts. Thus, *CATP-Fabric* is more efficient for IoT applications with concurrency conflicts.

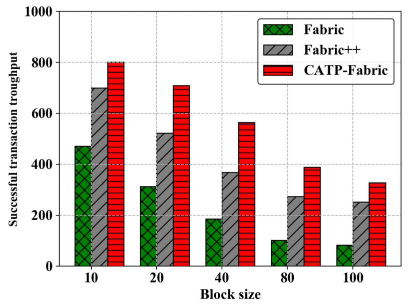
Fig. 6(b) shows the average transaction latency under different block size settings. From this Figure, we can see that *Fabric* always maintains a low latency (below 2 s) and outperforms all systems. While the average latency increases in both *Fabric++* and our *CATP-Fabric* as the block size increases. These results are expected since both *Fabric++*



(a)



(b)



(c)

Fig. 6. Impact of the block size on transaction (a) aborting rate, (b) average latency, and (c) throughput of *Fabric*, *Fabric++*, and our *CATP-Fabric*.

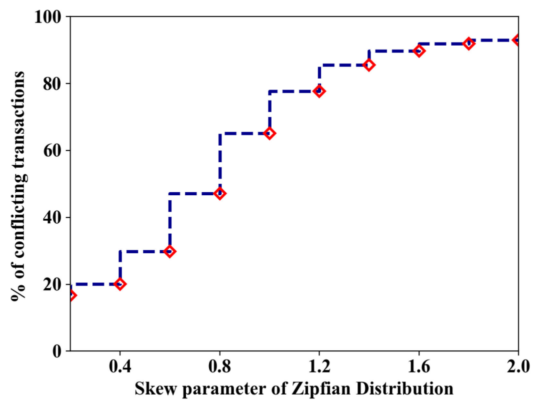


Fig. 7. Conflicting degree under various skew parameters.

and our *CATP-Fabric* require additional computation to process conflicting transactions. More transactions included in a block result in longer processing time thus leading to higher latency. But the latency in *Fabric++* is comparable with latency in *CATP-Fabric*. We show that the successful transaction throughput of the three systems under various block size values in Fig. 6(c). We observe that *CATP-Fabric* has a higher throughput than both *Fabric* and *Fabric++*. This further corroborates the effectiveness of our proposed system.

Impact of Transaction Contention: Next, we perform a detailed analysis of the impacts of transaction contention on the overall performance. We set different degrees of transaction contention by adjusting the skew parameter of the Zipf distribution in Smallbank workload from 0.2 to 2 with step size 0.2, as Fig. 7 shows, where the small skew parameter means low transaction contention. Then, we evaluate the transaction aborting rate, average latency and successful transaction throughput of the three systems under diverse skew parameters. In this experiment, we configure the block size as 50 and the percentage of read transactions as 50%. For other parameters, refer to Tables I and II.

Fig. 8(a) summarizes the results of transaction aborting rate under various skew parameters. We can see that for small skew (below 0.4), the transaction aborting rate of all three systems is relatively low because the number of potentially conflicting transactions is small by default. But, it can be seen that the transaction aborting rate in *Fabric* and *Fabric++* increases with the higher skew parameter (>0.4). The reason is that high data skew in Smalllbank workload leads to a large number of potential conflict transactions that cannot be resolved by *Fabric* and only small part can be resolved by *Fabric++*. In contrast, our *CATP-Fabric* allows the system to maintain the transaction aborting rate below 1% for all skew parameters. *CATP-Fabric* is not sensitive to the increase of contention degree due to merging all final valid transactions.

We compare the average latency in Fig. 8(b). We can observe that our *CATP-Fabric* and *Fabric++* experience a higher latency than *Fabric*. *Fabric++* shows a maximum increased 6.1× latency than *Fabric*, and our *CATP-Fabric* shows a maximum increased 5.5× latency. This is because *Fabric++* and *CATP-Fabric* might need extra processing time to resolve conflict transactions in the ordering phase. We can also see that, compared to *Fabric++*, our *CATP-Fabric* brings in additional latency less than 1 s when the skew parameter is lower than 1.2, but shows an improvement when the skew parameter is higher than 1.2. The reason is that increasing data contention would lead to more conflicting transactions, and thereby leading to high time complexity in *Fabric++*. But it has a relatively small impact on latency in our *CATP-Fabric* system which employs a key-based grouping method to process transactions in parallel.

Finally, we provide the results of successful transaction throughput under various data contention rates in Fig. 8(c). We find that successful transaction throughput decreases as the skew parameter increases in all three systems. This happens due to an increase in the number of potentially conflicting transactions that bring in a higher transaction aborting rate and larger transaction processing time. Moreover, Fig. 8(c) shows that *CATP-Fabric* significantly improves the successful transactions throughput compared to *Fabric* and *Fabric++*. When skew parameter is sets to 2.0, *CATP-Fabric* achieves over 400 successful transactions per second outperforming *Fabric++* by up to 2× and *Fabric* by up to 10×. That is because *CATP-Fabric* prefers to maintain more valid transactions within a block by the optimal conflicting transaction selection algorithm. Moreover, *CATP-Fabric* is able to convert the conflicting transactions, which would be *final-valid* to *conflict-free* ones to minimize the number of unnecessary aborts [as shown in Fig. 8(a)].

Impact of Read Transaction Percentage: In this part, we evaluate how the percentage of read transactions impacts the performance of the simulated blockchain systems. We set the

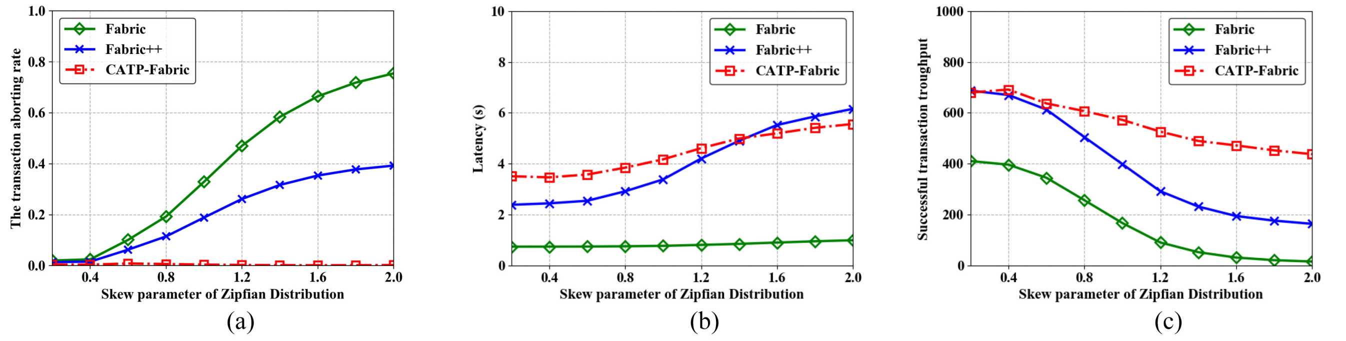


Fig. 8. Impact of the skew parameter of Zipfian distribution on transaction (a) aborting rate, (b) average latency, and (c) throughput of *Fabric*, *Fabric++* and our *CATP-Fabric*.

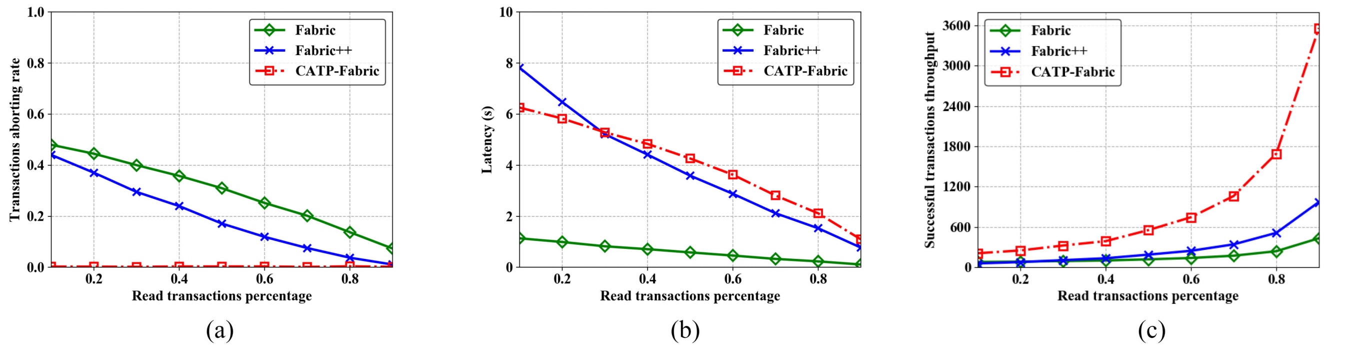


Fig. 9. Impact of Read transaction percentage on (a) aborting rate, (b) average latency, and (c) throughput of *Fabric*, *Fabric++*, and our *CATP-Fabric*.

skew parameter to 1.0, and the block size to 50. For other parameters, refer to Tables I and II. We vary the percentage of read transactions from 0.1 to 0.9.

From Fig. 9(a), we can see that the transaction aborting rate of *Fabric* and *Fabric++* decreases with the increase in the percentage of read transactions. A higher percentage of read transactions implies that transaction records will update by fewer transactions in a short timeslot, resulting in a lower number of potentially conflicting transactions. On the other hand, since *CATP-Fabric* optimizes over reordering of all read-only transactions first. It ensures that all read-only transactions can be processed successfully. Hence, the transaction aborting rate of *CATP-Fabric* is better than *Fabric* and *Fabric++*. Fig. 9(b) and (c) shows the average latency and the successful transaction throughput of the three systems. We see that the latency of *Fabric++* and *CATP-Fabric* are affected by the percentage of read transactions as expected. In comparison to *Fabric++*, *CATP-Fabric* shows a higher successful transaction throughput.

VII. Conclusion

In this article, we proposed *CATP-Fabric*, a permissioned blockchain system that supports conflicting transaction execution effectively in IoT ecosystems. We adopted a key-based transaction grouping method to improve system performance with parallel processing. A low-complexity stale transaction filtering mechanism was then developed to discard unnecessary transactions as early as possible to save network resources. Finally, an optimal conflicting transaction selection algorithm was proposed by optimizing selection decision and computation resources. Experimental evaluations based on a real workload demonstrated that the *CATP-Fabric* achieves significantly better performance compared to both Fabric and Fabric++ permissioned blockchain systems when incorporating conflicting transactions.

References

[1] Y. Liu, C. Yang, L. Jiang, S. Xie, and Y. Zhang, "Intelligent edge computing for IoT-based energy management in smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 111–117, Mar./Apr. 2019.

[2] L. Chettri and R. Bera, "A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 16–32, Jan. 2020.

[3] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2702–2733, 3rd Quart., 2019.

[4] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8182–8201, Oct. 2019.

[5] A. M. Zarca, J. B. Bernabe, A. Skarmeta, and J. M. A. Calero, "Virtual IoT honeynets to mitigate cyberattacks in SDN/NFV-enabled IoT networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1262–1277, Jun. 2020.

[6] R. Sairam, S. S. Bhunia, V. Thangavelu, and M. Gurusamy, "NETRA: Enhancing IoT security using NFV-based edge traffic analysis," *IEEE Sensors J.*, vol. 19, no. 12, pp. 4660–4671, Jun. 2019.

[7] M. Wazid, A. K. Das, V. Odelu, N. Kumar, M. Conti, and M. Jo, "Design of secure user authenticated key management protocol for generic IoT networks," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 269–282, Feb. 2018.

[8] K.-K. R. Choo, Z. Yan, and W. Meng, "Blockchain in industrial IoT applications: Security and privacy advances, challenges and opportunities," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4119–4121, Jun. 2020.

[9] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[10] H. Liu, P. Zhang, G. Pu, T. Yang, S. Maharjan, and Y. Zhang, "Blockchain empowered cooperative authentication with data traceability in vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4221–4232, Apr. 2020.

[11] Y. Yu, Y. Li, J. Tian, and J. Liu, "Blockchain-based solutions to security and privacy issues in the Internet of Things," *IEEE Wireless Commun.*, vol. 25, no. 6, pp. 12–18, Dec. 2018.

[12] J. Gao *et al.*, "A blockchain-sdn-enabled Internet of vehicles environment for fog computing and 5G networks," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4278–4291, May 2020.

[13] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8076–8094, Oct. 2019.

[14] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[15] J. Peng, K. Cai, and X. Jin, "High concurrency massive data collection algorithm for IoMT applications," *Comput. Commun.*, vol. 157, pp. 402–409, May 2020.

[16] L. Xu, W. Chen, Z. Li, J. Xu, A. Liu, and L. Zhao, "Locking mechanism for concurrency conflicts on hyperledger fabric," in *Proc. Int. Conf. Web Inf. Syst. Eng.*, 2019, pp. 32–47.

[17] M. J. Amiri, D. Agrawal, and A. El Abbadi, "ParBlockchain: Leveraging transaction parallelism in permissioned blockchain systems," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 1337–1347.

[18] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich, "Blurring the lines between blockchains and database systems: The case of hyperledger fabric," in *Proc. Int. Conf. Manag. Data*, 2019, pp. 105–122.

[19] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The effect of IoT new features on security and privacy: New threats, existing solutions, and challenges yet to be solved," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1606–1616, Apr. 2019.

[20] J. Huang, L. Kong, G. Chen, M.-Y. Wu, X. Liu, and P. Zeng, "Towards secure industrial IoT: Blockchain system with credit-based consensus mechanism," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3680–3689, Jun. 2019.

[21] M. Debe, K. Salah, M. H. U. Rehman, and D. Svetinovic, "IoT public fog nodes reputation system: A decentralized solution using ethereum blockchain," *IEEE Access*, vol. 7, pp. 178082–178093, 2019.

[22] K. Lei, M. Du, J. Huang, and T. Jin, "Groupchain: Towards a scalable public blockchain in fog computing of IoT services computing," *IEEE Trans. Services Comput.*, vol. 13, no. 2, pp. 252–262, Mar./Apr. 2020.

[23] H. Liu, D. Han, and D. Li, "Fabric-IoT: A blockchain-based access control system in IoT," *IEEE Access*, vol. 8, pp. 18207–18218, 2020.

[24] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K.-C. Li, "A secure FaBric blockchain-based data transmission technique for industrial Internet-of-Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3582–3592, Jun. 2019.

[25] P. Nasirifard, R. Mayer, and H.-A. Jacobsen, "FabricCRDT: A conflict-free replicated datatypes approach to permissioned blockchains," in *Proc. 20th Int. Middleware Conf.*, 2019, pp. 110–122.

[26] W. Al Amiri, M. Baza, K. Banawan, M. Mahmoud, W. Alasmary, and K. Akkaya, "Towards secure smart parking system using blockchain technology," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2020, pp. 1–2.

[27] S. Zhang, E. Zhou, B. Pi, J. Sun, K. Yamashita, and Y. Nomura, "A solution for the risk of non-deterministic transactions in hyperledger fabric," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2019, pp. 253–261.

[28] C. Gorenflo, L. Golab, and S. Keshav, "XOX Fabric: A hybrid approach to transaction execution," 2019. [Online]. Available: arXiv:1906.11229.

[29] P. Ameri, N. Schlitter, J. Meyer, and A. Streit, "NoWog: A workload generator for database performance benchmarking," in *Proc. IEEE 14th Int. Conf. Dependable Auton. Secure Comput. 14th Int. Conf. Pervasive Intell. Comput. 2nd Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, 2016, pp. 666–673.

[30] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A framework for analyzing private blockchains," in *Proc. ACM Int. Conf. Manag. Data*, 2017, pp. 1085–1100.

[31] (2019). *Hyperledger Caliper*. [Online]. Available: https://www.hyperledger.org/projects/caliper

[32] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *Proc. IEEE 26th Int. Symp. Model. Anal. Simulat. Comput. Telecommun. Syst. (MASCOTS)*, 2018, pp. 264–276.