2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

SwiftFabric: Optimizing Fabric Private Data Transaction Flow TPS

Mingxuan Li[12], Dongdong Huo[124], Chao Liu [12], Han Wang [12], Yazhe Wang [1],
Yu Wang [1], Ping Zou[3], Yandong Li[3], Zhen Xu [1]
[1]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
[3]*Beijing Aerospace Smart Manufacturing Technology Development Co., Ltd, Beijing China*
[4]*Corresponding author*
[1]{limingxuan2, huodongdong, wanghan1996, wangyazhe, wangyu xuzhen}@iie.ac.cn
[1]liuchao19@mails.ucas.edu.cn
[3]{zouping, liyandong}@casicloud.cn

***Abstract*—As permissioned chain technology is getting more and more important in blockchain technology, Hyperledger Fabric–one of its representative–has received extensive attention. However, previous works just focused on plaintext transmission and made TPS optimization that cannot be applied in ciphertext transmission situation, which is usually required on permissioned chain under actual circumstances. This paper will be the first to focus on private data transaction flow to optimize Fabric TPS, without changing the normal call to the Fabric SDK (Software Development Kit). We modify the process of writing data to the transient database after the completion of the simulated transaction, optimize the data structure in transient database, and simplify the process of verifying hashes. At the same time, professional SmartContract experiment cases and experiment plans are designed to simulate large-scale commercial applications, which are used to measure the TPS before and after optimization in the Fabric private data transaction mode. The results show that the optimized Fabric model can greatly promote the performance of the private data transaction flow.**

***Keywords*-Blockchain; TPS; Private; Data;**

I. Introduction

The emergence of Bitcoin [1] brought blockchain technology into people's attention. Blockchain has public ones, such as Bitcoin [1] and Ethereum [2] – the most popular public chains, and permissioned ones. As there are no centralized server nodes in public chain network, the public chain can be open to everyone around the world, so that anyone can query transactions, send transactions, etc. in it and each node can freely join or exit the blockchain network, and participate in the reading and writing of data on the chain. The underlying technology of the public chain is mainly used in financial fields such as releasing virtual currency. However, its poor performance, openness and transparency make it difficult to be applied in B2B business between the existing centralized business community alliances. As for permissioned blockchain, the Hyperledger Fabric is a permissioned blockchain architecture initiated by the Linux Foundation [3], which provides a set of enterprise-level distributed ledgers and fills the gap in the application of the public chain between the enterprise alliances.

The Hyperledger Fabric architecture has undergone mainly two versions of iterations. The original 0.6 version [4] can only be used for commercial verification and cannot be applied to real-world scenarios. The main reasons are that all methods are concentrated on the Peer node, merely supporting a single channel, the structure is simple, and the scalability, security, and isolation are inherently insufficient.

After 1.0 [5] and subsequent versions, Fabric architecture was redesigned, and the consensus service was stripped out into Orderer nodes, which provides functions such as transaction sorting, packing blocks, and submitting ledger. After version 1.1 [6], the Hyperledger Fabric proposed the private data as a new feature to compensate for the shortcomings of private data protection. This feature was officially applied in version 1.2 [7] and is continuously updated and enhanced in subsequent releases. The latest version of the Fabric is updated to version 1.4 [8] and will be the first LTS (Long-Term Support) version recommended for production.

The performance of the alliance chain can be nearly 100 times higher than that of the public chain. However, due to the inherent performance problems of the blockchain, in quite a few cases the native Fabric system still cannot carry the mainstream commercial activities on the Internet. Therefore, the performance optimization problem of the Hyperledger Fabric is still an important problem worthy of discussing.

Previous evaluation and optimization of Fabric technology and solutions just focus on the traditional non-private data transaction flow, and the case of evaluation use is relatively simple, generally a single official transfer SmartContract(sample02) [9], which is called chaincode by a joint name in Fabric. The invoke chaincode parameters are generally (a, b, x): a transfers x $ to b. Those solutions usually use Cache [10] in marshal and unmarshal, parallel processing in the verification phase, splitting Peer function and other methods to improve the performance of blockchain. The evaluation version is the version before the 1.4LTS [8]. However, the traditional data transaction process has several disadvantages that make it unable to meet the organizer's need to protect private data. First, to meet the need for a group of organizations in the channel to keep some confidential data from other organizations on the channel, they can choose to create new channels. But each time they create a channel, there will be additional management overhead and the new channel will take up a lot of physical resource issues. Second, all channel participants are unable to keep some of the data private. To solve these problems, Fabric proposed the transaction flow mode of private data after the 1.1 experimental version [6], which enables endorsement, submission, or query of private data in a subset of the organization in the channel without having to recreate the data channel. At the same time, Orderer nodes can only get the hash of the transaction data, and cannot get the real information of the transaction data. This mode well protects users' private information.

During the transaction execution of chaincode, users obtain and save private data to the transient database, and Peer nodes transmit private data stored in the transient database to other relevant nodes through the Gossip protocol. After the dissemination is completed, the Orderer node receives the hash value returned by the Peer node and sorts it together with other non-private data. And after the relevant nodes compare the hash values, all relevant Peer nodes finally update the block information to the private database of the ledger. In this process, the user's transaction information is transparent to the Orderer node. So this process is more secure than traditional data transaction processes. The iterative versions of the Fabric continue to optimize and enhance the private data flow. The private data flow of Fabric has become an extremely important way and has received more and more attention.

We focus on the performance evaluation and optimization of the Hyperledger Fabric private data flow model. This paper is the first to experiment the 1.4 LTS version [8] of the private data process mode through professional experiment cases and experiment scenarios. To improve the TPS performance of Fabric in private data mode, we performed the following four things:
1. Construct an experiment case that simulates a commercial chaincode; design an experiment middleware; perform a private data transaction flow experiment to obtain the TPS of the Fabric benchmark.
2. Modify the process of completing the simulation transaction into the transient database, and do dual guarantee optimization.
3. Modify the information that the data propagate to other Peer nodes, and transmit the hash value after the endorsement to other nodes.
4. Modify the process of comparing hashes, instead of recalculating the private data hash value.

This paper divides into 8 sections. The second section introduces the Fabric1.4 [8] architecture and private data transaction flow; the third section introduces the experimental design; the fourth section introduces the experimental results; the fifth section introduces related work, and the rest three parts introduces our future work, contribution, and acknowledgment.

II. Fabric Architecture and Private Data Flow

A. Main Components of Fabric

Hyperledger Fabric's architecture includes Client node, CA node, Orderer node and Peer node, which includes Endorser node, Committer node, Leader node. Fabric Architecture is show in Fig. 1

- Client node: deploys the user's application or CLI command terminal. The registered user obtains the legal certificate and private key, and then executes the user's command. First, the transaction proposal is sent to the endorser node for request endorsement. When enough endorsement results are collected, the simulation result executed by endorsement nodes is encapsulated into ordinary transaction information and sent to Orderer node through Broadcast service interface for sorting, and then broadcasted to all Peer nodes after generating block in the channel.
- CA node: as a certificate authority in the Fabric architecture, CA node provides user management and certificate services such as user registration and certificate issuance based on the RESTful [11] interface. This type of node manages membership information in the Fabric network, the life cycle and rights control functions of the digital certificate based on the digital certificate and the standard PKI(Public Key Infrastructure) [12]. After logging in through the client, the user registers as a legitimate user and uses the ECDSA algorithm [13] to generate a public and a private key. The MSP is set up to verify and manage the user identity, and the identity certificate in the Fabric conforms to the X.509 [12] standard specification. All legal members need to obtain the identity certificate signed by the authentication before accessing the network. It is not necessary to access the CA node all the time while the Fabric system is running, so the Fabric-CA node is separated into a component.
- Orderer node: Hyperledger Fabric provides two sorting services, solo and Kafka [14]. The solo mode means that the Orderer is a single node, suitable for the test environment, and the multi-node kafka mode is used in the actual product environment. Our experiment and optimization model are based on the kafka consensus. The Orderer node accepts the transaction message request through the Broadcast service interface, submits the message to the consensus component for sorting, and then adds it to the local cache transaction message list, cuts the packaged new block according to the rule of the block, and submits the local ledger of the Orderer node. At the same time, the block request message is processed through the Deliver() service interface, and the requested block data is obtained from the local ledger and then sent to the leader node of the organization and broadcasted to other nodes in the channel organization.
- Peer node: includes endorser node, committee node, leader node, and so on. Endorser node: responsible for simulating the implementation of the signature proposal,

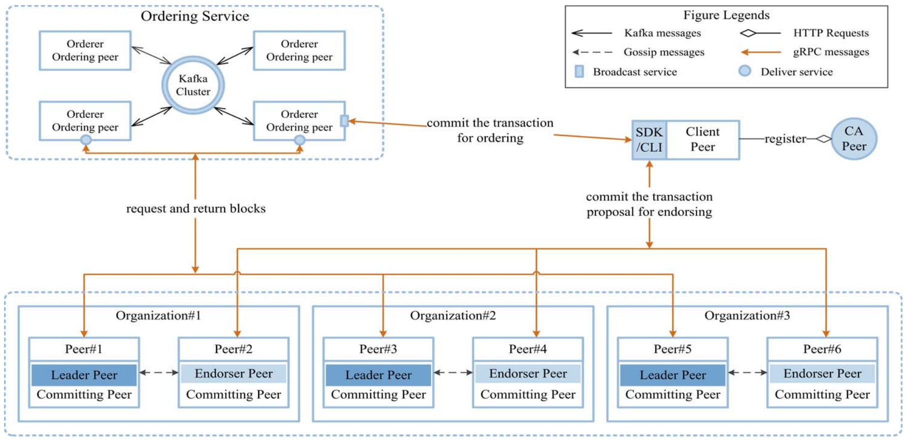
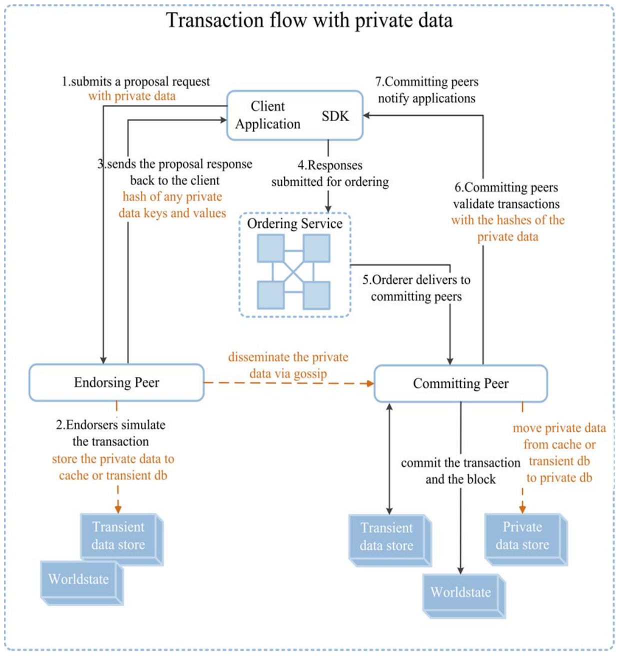


Fig. 1. Fabric Architecture

and endorsing the simulation execution result read-write set and transaction proposal, indicating the result of the simulation execution of the approved transaction proposal. Committee node: All Peer nodes on the same channel default to the committee node on the channel organization, maintain the account ledger data of the channel on the node, and are responsible for verifying the transaction, calling the VSCC system chaincode to check whether the endorsement information satisfies the instance. The specified endorsement policy is implemented, and then the MVCC checks the validity of the transaction information and submits the local ledger. The leader node represents the organization, establishes a gRPC communication connection with the Orderer node through the Deliver service interface, and requests the ledger block data of the specified channel. At the same time, the leader node distributes the received block data to other nodes in the organization through the Gossip protocol [15] to broadcast transactions. In addition, the Peer node in the channel organization synchronizes the missing data (block data and private data) based on the anti-entropy algorithm [16], and timely updates all the node books in the organization to ensure data consistency.



Transaction flow with private data
1.submits a proposal request with private data
Client Application
SDK
7.Committing peers notify applications
3.sends the proposal response back to the client hash of any private data keys and values
4.Responses submitted for ordering
6.Committing peers validate transactions with the hashes of the private data
Ordering Service
5.Orderer delivers to committing peers
Endorsing Peer
disseminate the private data via gossip
Committing Peer
2.Endorsers simulate the transaction store the private data to cache or transient db
commit the transaction and the block
move private data from cache or transient db to private db
Transient data store
Worldstate
Transient data store
Private data store
Worldstate

Fig. 2. Private Data Transaction Flow

B. Private Data Transaction Flow

The private data flow [17] is a new feature proposed by Fabric in version 1.1 [6]. It is officially applied in version 1.2 and is continuously supported and optimized in subsequent versions. When private data collections are used in chaincodes, different from traditional ones, the flow of private data can protect the confidentiality of transaction proposals, endorsements, and submission of private data. The private data transaction flow is shown in Fig. 2

- 1. For reading or writing private data, the client application submits a function that calls the chaincode to request an endorsement node belonging to the authorized organization in the collection. Private data in the chaincode or data used to generate private data is submitted in the transient database of the transaction proposal.
- 2. The endorser node simulates the execution of the transaction, stores the private data in the transient data store (the local temporary storage area in the Peer node), and sends the private data to the authorized Peer through the gossip protocol according to the data set policy in the configuration file.
- 3. The endorser node returns the result response of the transaction to the client application in public data. The result of the processing contains the key of the private data and the hash value of the content. Private data itself is not returned to the application.
- 4. The client application submits the received transaction

information (including the hash of the private data) to the Orderer node. The Orderer node packs the blocks according to the normal flow. Blocks containing private data hashes are distributed to individual Peer nodes. All Peer nodes on the channel can consistently and validate transactions that contain private data hashes without knowing the actual private data content.

- 5. During block verification, the authorized Peer determines whether it has permission to read private data based on the collection policy. If they have permission, they will first verify the local private data to determine whether they have received private data during the chain endorsement phase; if not, they will pull from other peers. They then validate private data and private data hashes in the block and submit the transaction and block. After verification and submission, a copy of the private data is moved to the private database and the private data write set store. Finally, the local private data in the Peer's transient data store will be deleted.

III. Experimental Design

During the process of deploying the Fabric, we monitor the hardware information and find that the Peer node occupies more hardware resources, which is the main bottleneck affecting the performance of the Fabric. In this section we will detail our experiment cases and methods to make a good foundation for evaluating the versatility of optimization. Besides, the optimization processing method for each module of the Peer node will be described in detail.

A. Experiment Tools

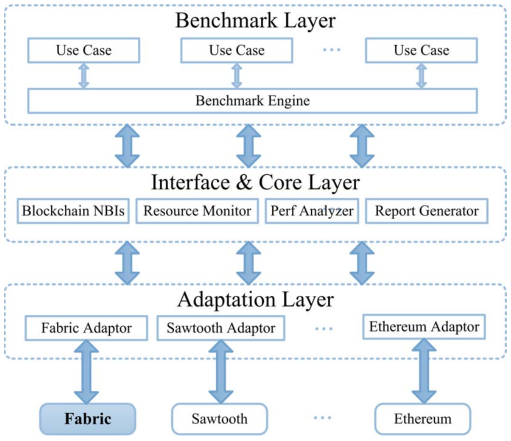


Fig. 3. Caliper Architecture

Caliper [18], as shown in Fig. 3, is a experiment tool of blockchain benchmark performance that has been accepted as the latest Hyperledger version. It can be used to experiment different blockchain implementations. Users can do performance experiments on their own blockchain network with a defined experiment set, obtain a series of experiment results and generate reports with many performance indicators, such

TABLE I
EXAMPLE RESULT

Test	Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
1	initMarble	100	0	25.3tps	4.11s	1.04s	2.53s	13.4tps
2	initMarble	100	0	50.5tps	5.79s	0.95s	3.46s	13.8tps
3	readMarble	100	0	25.2tps	0.98s	0.41s	0.72s	22.0tps
4	readMarble	100	0	50.5tps	2.43s	0.59s	1.53s	23.6tps
5	transferMarble	100	0	25.3tps	3.64s	1.11s	2.47s	14.6tps
6	delete	100	0	25.3tps	3.35s	0.78s	2.19s	14.9tps

as transaction success rate, transaction throughput, transaction latency, resources, consumption and so on. In this paper we will use Caliper for performance evaluation.

B. Experiment Case

Using the officially provided (marbleprivate) [19], we set up the permutation and combination of the eight experiment functions and experiments with caliper. Before experimenting, the middleware of the experiment needs to be written, which can experiment the basic impact on TPS in detail. The chaincode represents a standard enterprise alliance application, mainly used to simulate a more complex asset transfer solution. The contract has eight application requirements, four of which are privacy-related functions:
Initmarble: creates a product, which includes the product ID, color, size, owner, and price; Determines whether the marble is already in the ledger, and if not, write the information of the product into the ledger.
Delete: deletes the corresponding product according to the product ID;
ReadPrivateDetails: queries the product information in the private data collection in the ledger according to the product ID;
TransferMarbel: change the owner of the product through the transaction;

We write specific middleware that simulates normal trading scenarios between companies, including initializing new product information, product warehousing, product inquiries, and product outbound operations. As shown in TABLE I, according to the actual situation, because the product out of the warehouse must have the product in the ledger, the initial product should be placed in the first place, the product out of the operation placed at the end, and the other operations in the middle can be arranged and combined. The test plan can fully cover the basic functions of select, delete, update, insert. It is perfectly possible to simulate an application between enterprises. The following TABLE I shows test solution: **Name** is operation name. **Succ** is number of successes. **Fail** is number of fail. **Send Rate** is rate of sending a transaction. **Max Latency** is maximum latency of feedback. **Min Latency** is minimum latency of feedback. **Avg Latency** is average latency of feedback. **Throughput** is rate of transactions are processed, which is the true TPS.

C. Dual Protection through Cache Optimization

Optimization of dual protection on Peer and Orderer. After the simulation is executed, the gossip protocol is called to distribute the private data set to the Peer node specified in advance, and the local private data is stored in the local transient private database where we optimize the code. Before the private data is stored in the transient private database, the cache is made in memory and the private data is stored in the local database simultaneously. When other peers receive data, they write to the local transient database on the one hand, and also do the cache on the Peer node. When the committer accounting node needs to submit private data, it will judge. If there is no cache information, Fabric system goes to the transient to read the information. If not, the cache information is directly submitted to the private database. If there is a problem such as downtime, the data in the transient is written to the private database. Fabric system also deletes the associated private data in the transient private database and cleans up the expired data.

D. Parallel Calculation of Hash Value

The Process that the endorsement node sends the simulation result to the client is separated into two parts. The private data needs to be calculated hash after the endorsement, and the endorsement node waits for the hash value of the private data to be sent to the client together with the public data where there will be waiting for each other. Instead, We quarantine the sending process into tow parallel processes and no longer send private data and public data sets together to the client. This method can greatly reduce the waiting time when the private data transaction type and the public data transaction type are simultaneously transmitted.

E. Optimize the Hash Calculation Process

Monitor calculated hashes and reduce data migration. In the consensus phase, the hash value of the private data is transmitted to the client through the peer node together with the public data. When the client collects transactions sent by all the endorsement nodes, the information is sent to the Orderer node. The private data is sent directly to the relevant Peer node through the gossip algorithm. Through experiments, we found that when the peer database of the Peer node receives the private data sent by the endorsement node, there is a process of waiting for the peer node to pull the transient database information and to calculate the hash. In order to reduce the waiting time, we set up a monitor to surveille the transient database. When the private database receives the private data, it immediately starts hashing according to the namespace. The calculated hash value and the original private data are combined into a new structure in the transient database, and the namespace is used as a new index.

F. Optimize Hash Verification

Modify the Process of Comparing Hashes. Fabric native system privat data is propagated to other nodes through the gossip protocol. Before applying the method of private transaction flow, it is necessary to configure trusted subgroups in advance. Therefore, the organizations of the transaction of the private data flow trust each other. However, there is still possibility that data can be tampered with. So the fabric's native system pulls the hash value in the block of the orderer node and then at the commit node pulls the private data in the Cache or transient data base and calculates the hash of it. Next, we compared hash values, if they match, the hash value will be written to the blockchain, and the private data to the private database from Cache or the transient database. Once the above processes are done, the private data in the Cache or transient database will be deleted. When comparing hashes, we parallelize and batch the process. At the same time, we omit the process of extracting data in the commit node, and in combination with the process in IV-E, only the hash value of the namespace and the corresponding private data is pulled and compared with the hash in the orderer block. If they are consistent, the transient information is directly transferred to the private database. This greatly reduces the waiting time for calculations and improves system performance. In the process of comparing hashes, if data loss occurs due to the propagation process of the gossip algorithm, data and hash values need to be pulled again. This situation will only occur under extreme conditions.

G. Security Analysis

Security performance analysis: In this section, we reorganize the impact of our optimization process on the overall security of the SwiftFabric. In the optimization process of IV-B, the data is cached in memory. However, because of the dual protection in the transient database, this optimization method does not result in the absence of data sets as a result of to physical damage such as power outages and downtime. There is no impact on the integrity of the data. The optimization method for IV-C is to perform parallel processing using the characteristics of the GO language itself, and can fully utilize the resources of the CPU for data processing. There is no impact on the security of the fabric itself. IV-D and IV-E use parallel computing methods and set up monitoring points to reduce the data migration process. At the same time, the hash value is recalculated at the peer receiving end, and the hash value is compared to prevent the data from being tampered with. The security of the private data flow is thus guaranteed.

IV. Experimental Results

In this section, we will introduce the SwiftFabric performance experiment result. We use a 1.4 LTS Fabric [8] architecture, and seven local servers connected by a 1 Gbit/s switch. Each server is equipped with two Intel Xeon CPU 1220 v5 processors @3.00 GHz, for a total of Four cores and four threads and 16 GB of RAM. Therefore, the network problem is not in our consideration. At the same time, kafka/zookeeper is used as the Orderer node, and the docker [20] container is managed by Kubernetes [21] technology. We establish a large docker cluster to better simulate an enterprise-level alliance permission blockchain system. We tested the private data of the unmodified version of Fabric 1.4 [8] with the most IV-A experimental scheme.

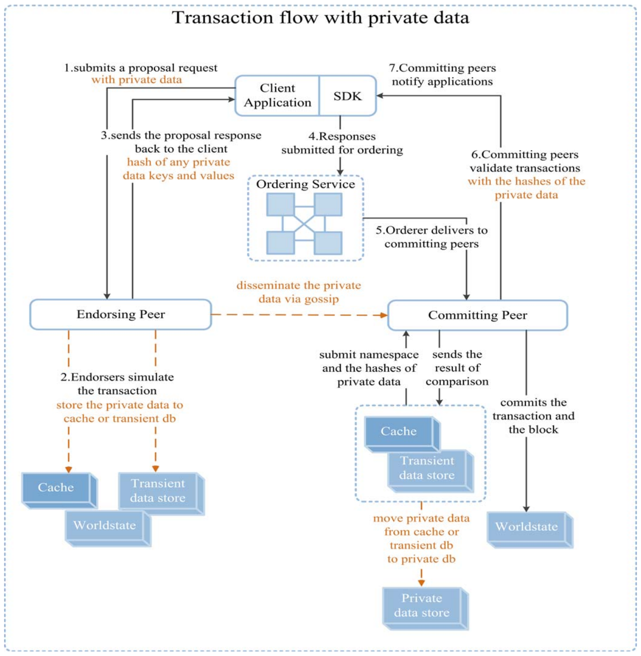


Fig. 4. SwiftFabric Architecture

TABLE II
PUBLIC DATA FLOW

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
1	initMarble	10000	0	4904.6tps	25.35s	4.12s	21.03s	511.2tps
2	readMarble	10000	0	5000.0tps	5.94s	0.42s	2.68s	2764.4tps
3	transferMarble	10000	0	4881.3tps	21.39s	5.89s	22.33s	464.6tps
4	delete	10000	0	4961.6tps	20.52s	3.78s	17.69s	489.0tps

By comparing the non-private data flow test table TABLE II and the test data TABLE III of the private data flow, we found that the transaction process of non-private data streams does not need to calculate the hash value because it does not need to write the simulated transaction result to Cache or the transient database. Instead, after the simulated transaction is completed, the transaction result is sent to the Orderer Node, and the hash value does not need to be recalculated when submitted to the ledger. The tested TPS performance is averagely 1.05 times higher than the private data. At the same time, it is obvious that the private data flow in the case of sending TPS is 10000, initMarble, and transferMable, and the actual Throughput of delete is average 462.2TPS, and the Troughput of readMarble is 2730.6TPS. Obviously, the reading ledger takes a short time to write to the ledger. By observing the two graphs, we found that both the transaction flow of non-private data and private data are not very efficient, and the efficiency of writing ledger is much lower than the reading ledger.

A. Cache Optimization

In this section, we will simulate the transaction result cache to MemDB for processing, before the private data set is written to the Cache or transient database. The original Fabric has two databases, LevelDB [22] and CouchDB [23]. In order to increase the speed, we still use the LevelDB [22] database for

TABLE III
PRIVATE DATA FLOW

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
1	initMarble	10000	0	4900.8tps	37.14s	9.01s	21.81s	496.7tps
2	readMarble	10000	0	4999.3tps	6.23s	0.69s	3.18s	2730.6tps
3	transferMarble	10000	0	4826.1tps	47.86s	12.99s	30.06s	430.1tps
4	delete	10000	0	4907.3tps	32.13s	8.78s	20.69s	460.0tps

TABLE IV
CACHE OPTIMIZATION

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
1	initMarble	10000	0	4910.6tps	29.89s	8.21s	18.18	933.9tps
2	readMarble	10000	0	5000.0tps	5.81s	0.57s	2.66s	2750.1tps
3	transferMarble	10000	0	4826.3tps	38.31s	10.96s	23.96	800.8tps
4	delete	10000	0	4906.4tps	32.13s	7.78s	16.61s	776.4tps

the transient data. This database is a native database of the Fabric written by GO language, which is more efficient than CouchDB [23]. The comparison of the optimized cache before and after the optimization of the private data flow results as shown in TABLE III and TABLE IV, the read ledger before the optimization readMarble is 2730.6TPS, and the optimized readMarble is 2750.1TPS. The comparison shows that the gap between the two is not obvious. This is because the Fabric is implemented by reading the WorldState database when reading the ledger information. The minor difference is due to the delay in writing data into the ledger. The TPS for writing data operations is nearly 2 times better than the performance of the original Fabric. It is obvious that the value of max latency and min latency in the optimized Fabric transaction in the private data transaction is greatly reduced compared with the value before the optimization. This shows that the direction of optimization we have chosen is extremely correct. At the same time, because we are doing the cache, the data is written to the transient database through the parallel method. When the system reads the data in the cache, if there is a power outage or downtime, it will enter the transient database to read the data, or submit the data. Therefore, we use the experiment to power off the computer during the transaction process. We found that the private data operation before the power outage can be read out in the ledger. This experiment shows that we can not only improve the efficiency of reading and writing private data, but also protect the integrity of data.

B. Optimize Results of Endorsement Nodes

Since the endorsement node calculates the hash value separately and sends the public transaction process, in order to verify the optimization result of our endorsement node, we design a new test case: we mixedly send the private and the public data transaction flow as TABLE III and TABLE V show, also the results of the original fabric architecture and the optimized fabric. We can clearly see that due to the parallel sending of hash values and public data, the TPS of private data and public data are improved. In particular, public data is increased to nearly 2.5 times because it no longer waits for calculating hash data for private data. And the tps of private data has also increased to 1.5 times of the original.

TABLE V
OPTIMIZE RESULTS OF ENDORSEMENT NODES

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
	public							
1	initMarble	10000	0	4998.9tps	20.00s	3.07s	18.16s	1205.2tps
2	readMarble	10000	0	5000.2tps	4.95s	0.38s	2.44s	4512.1tps
3	transferMarble	10000	0	4820.1tps	18.35s	4.98s	20.15s	1187.3tps
4	delete	10000	0	4910.5tps	19.20s	3.31s	16.32s	1208.5tps
	private							
1	initMarble	10000	0	4905.7tps	36.28s	8.61s	20.356s	762.8tps
2	readMarble	10000	0	4999.3tps	6.05s	0.73s	3.49s	3059.3tps
3	transferMarble	10000	0	4719.4tps	46.14s	11.87s	28.21s	654.3tps
4	delete	10000	0	4910.3tps	30.11s	7.93s	20.61s	716.1tps

TABLE VI
HASHES VERIFICATION

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
	public							
1	initMarble	10000	0	4904.3tps	20.23s	3.61s	17.28s	1998.1tps
2	readMarble	10000	0	4999.6.0tps	3.97s	0.39s	2.08s	3129.3tps
3	transferMarble	10000	0	4881.3tps	19.36s	4.71s	21.06s	1802.1tps
4	delete	10000	0	4967.5tps	18.19s	3.01s	15.06s	1846.3tps
	private							
1	initMarble	10000	0	4900.3tps	26.25s	6.5s	18.36s	1941.0tps
2	readMarble	10000	0	4999.6.0tps	4.80s	0.60s	2.58s	3019.6tps
3	transferMarble	10000	0	4826.1tps	35.16s	8.71s	24.06s	1795.3tps
4	delete	10000	0	4907.3tps	22.19s	5.00s	15.63s	1806.9tps

C. Hashes Verification

In this section, we will focus on the comparison verification process of the hash before the private data is submitted to private database. When we optimize the combination of IV-D and IV-E, the process of calculating the hash no longer waits for the propagation process of the oderer. In the committee node, only the namespace and the corresponding hash value in Cache or the transient database are pulled, and no other information. If there is a hash of the transaction block in the compared data, the priavte data in Cache or the transient database will be directly submitted to the private database and the hash value will be written to the ledger. If the data flow is lost due to network error, the hash value of the block needs to be recalculated to ensure the integrity of the block data. After the optimization is completed, the performance of the fabric and the performance of the original fabric structure are as shown in the following Table VI and Table III: The latency of max and min is found to be significantly reduced. And the actual written TPS is nearly 4 times better than the performance of the original fabric.

D. Comprehensive Optimization Results

We integrate our optimization solution and integrate the optimization method of Cache and hash-optimized methods. Since the privacy verification phase is completed, the private data is directly written into the privacy database through the transient database. The hash value of the transaction data is directly written to the ledger after completion. Therefore, the combination of the two can collectively optimize the process of private data transaction flows. As shown in the TABLE III and TABLE VII. It is clear that our optimization program can greatly improve the process of the entire private data on the basis of the original. In **Succ, Fail, Send Rate, Max Latency, Min Latency, Avg latency, Troughtput** several performance indicators have been greatly improved. Especially Troughtput of write data. It nearly increases to 10 times of the original one.

TABLE VII
COMPREHENSIVE OPTIMIZATION RESULTS

Test	Name	Succ	Fail	SendRate	MaxLatency	MinLatency	AvgLatency	Throughput
	public							
1	initMarble	10000	0	4904.9tps	19.20s	2.82s	17.80s	4620.8tps
2	readMarble	10000	0	5000.0tps	4.84s	0.32s	1.51s	4866.6tps
3	transferMarble	10000	0	4881.5tps	19.20s	4.44s	20.15s	4432.3tps
4	delete	10000	0	4961.2tps	18.01s	2.34s	15.26s	4545.5tps
	private							
1	initMarble	10000	0	4900.5tps	30.23s	7.30s	18.52s	4564.1tps
2	readMarble	10000	0	4999.9tps	6.46s	0.55s	3.36s	4635.9tps
3	transferMarble	10000	0	4826.2tps	40.23s	10.47s	28.53s	4299.8tps
4	delete	10000	0	4907.4tps	28.65s	5.40s	217.58s	4438.7tps

V. RELATED WORK

Research on Hyperledger Fabric has been a constant concern of blockchain researchers. And many commercial applications have been used. We have researched a large number of Fabric related literature. There are not many papers on improving the performance of Hyperledger Fabric. We found that basically all papers proposed optimization solution based on conventional data flow methods. Thakkar P's paper [24] is the first one to improve the performance of Fabric. Their work has contributed to our paper. The paper mainly divides its work into two parts. The first half provides six instructions to configure parameters to achieve optimal performance. In the preliminary pre-experimental stage, our paper is in the pre-experimental stage, and the basic configuration is optimal according to the parameter optimization index. The second half of the paper defines three major performance bottlenecks in the Fabric transaction process. i. Repeat the verification of the X.509 certificate in the endorsement phase. ii, block verification. iii, the process of writing to the Couch database. The author made three changes to this, 1. the endorsement policy validation cache in the cryptographic component (3x) 2. the concurrent endorsement policy validation (7x) 3. Enhance the effect of batch read and write optimization of existing couchDB in the state verification and submission phase. These modifications can increase the overall TPS by a factor of 16. We borrowed some of their methods in experimental design. But since their modified version is Fabric 1.0 [5], and a lot of content has been applied by Fabric 1.1 [6] and its subsequent versions. Our optimized version 1.4 [8] has been fused in many ways. Gorenflo, Christian et al [25] has mainly proposed to increase the TPS from 3000 to 20000. 1. separate the metadata from the data, only send the transaction ID to the consensus layer. 2. parallel and cache, parallel and cache are implemented in the verification endorsement policy and semantic verification phase. 3. the use of memory to access data on critical paths, world state stored in memory, providing a lightweight hash table. The table can quickly obtain the data needed for the verification phase, and put the storage of the immutable block into the write-optimized storage cluster. 4. resource separation, Peer's endorsement role and committee role are split on different physical machines. This solution cannot solve the problem of how to deal with data persistence after the machine is down. Future work will be stored through distributed data such as spark. But our work has been able to use the leverdb database to help us do data persistence, even if there is a machine downtime, the data can be completely retained. And the paper is also aimed at traditional data transactions flow.

VI. Future Work

In the work that follows, we will improve the performance of the secure private data flow of the Hyperledger Fabric from the consensus level. Fabric now only supports CFT (Crash Fault Tolerance), and the future version of the Fabric supports the P-BFT(Practical-Byzantine Fault Tolerance) algorithm [26], then we can further improve the performance of the Fabric's private flow by studying the private data flow in the consensus layer for the optimization method under P-BFT [26]. We have packaged the SwiftFabric source code and will open it to GitHub in the future.

VII. Conclusion

In this paper, our contribution mainly rests on the optimization of private data process of version 1.4 [8] of the Hyperledger Fabric, increasing the TPS to nearly 10 times in our experiment environment. The main work is in the optimization of private data flow, through detailed experiment cases, unified experiment methods, making the experiment results more convincing. By changing the way temporary data is stored, we can achieve faster data caching, and more importantly ensure data persistence. At the same time, based one the basic characteristics of the authentication hash of the private data flow, and given that subgroup members have high credibility when users applying private data transaction flow, we ensure that the efficiency of data dissemination is improved on the one hand, and that unnecessary hash calculation time is reduced on the other hand, which can efficiently improve the performance of the Hyperledger Fabric.

VIII. ACKNOWLEDGMENT

This work was financed by Beijing Municipal Science and Technology Project (No. Z191100007119007), and supported by The National Key Research and Development Program of China (No. 2018YFB1004001)

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Consulted*, 2008.
[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[3] R. M. Stallman and G. E. Manual, "Free software foundation," *El proyecto GNU–Fundación para el software libre*, 1986.
[4] Fabric-0.6. https://github.com/hyperledger/fabric/tree/v0.6.0-preview.
[5] Fabric-1.0. https://github.com/hyperledger/fabric/tree/release-1.0.
[6] Fabric-1.1. https://github.com/hyperledger/fabric/tree/release-1.1.
[7] Fabric-1.2. https://github.com/hyperledger/fabric/tree/release-1.2.
[8] Fabric-1.4. https://github.com/hyperledger/fabric/tree/release-1.4.
[9] ChainCode02. https://github.com/hyperledger/fabric/blob/release-1.4/examples/chaincode/go/example02/chaincode.go.
[10] A. J. Smith, "Cache memories," *ACM Computing Surveys (CSUR)*, vol. 14, no. 3, pp. 473–530, 1982.
[11] L. Richardson and S. Ruby, *RESTful web services.* " O'Reilly Media, Inc.", 2008.
[12] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet x. 509 public key infrastructure certificate and crl profile," 1999.
[13] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.
[14] N. Garg, *Apache Kafka.* Packt Publishing Ltd, 2013.
[15] A.-M. Kermarrec and M. Van Steen, "Gossiping in distributed systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 2–7, 2007.
[16] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 1, pp. 8–32, 1988.
[17] Fabric-Private-Data. https://hyperledger-fabric.readthedocs.io/en/latest/private-data-arch.html?from=groupmessage&isappinstalled=0#private-data-reconciliation.
[18] Caliper. https://hyperledger.github.io/caliper/.
[19] PrivateMarble. https://github.com/hyperledger/fabric-samples/tree/release-1.4/chaincode/marbles02_private.
[20] C. Anderson, "Docker [software engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102–c3, 2015.
[21] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
[22] A. Dent, *Getting started with LevelDB.* Packt Publishing Ltd, 2013.
[23] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: the definitive guide: time to relax.* " O'Reilly Media, Inc.", 2010.
[24] P. Thakkar, S. Nathan, and B. Vishwanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," 2018.
[25] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second," 2019.
[26] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, pp. 173–186, 1999.