LB-Chain: Load-Balanced and Low-Latency Blockchain Sharding via Account Migration

Mingzhe Li, *Student Member, IEEE,* Wei Wang, *Member, IEEE,* and Jin Zhang, *Member, IEEE*

Abstract—Blockchain sharding has been increasingly used to improve blockchain systems' performance, in which a blockchain is split into multiple smaller, disjoint shards. In practice, however, sharding can only achieve limited throughput and latency improvement, especially for the *user-perceived transaction confirmation delay.* The performance degradation is believed to be caused by the cross-shard transactions. However, we show, through comprehensive system deployment and measurement studies, that the main culprit is the *imbalanced transaction load* on different blockchain shards. To address this problem, we propose a novel sharding system, called LB-Chain, which *dynamically* balances the transaction load on different shards by periodically *migrating active accounts* from heavily-loaded shards to less-loaded ones. We have implemented a prototype of LB-Chain, and evaluated its performance through large-scale blockchain deployment using real-world transaction traces. Extensive experiments confirm that LB-Chain significantly boosts sharding performance, reducing the transaction confirmation delays by up to 90% while increasing the transaction throughput by more than 10%. The delay difference between different accounts is also reduced dramatically, leading to improved fairness in the system.

Index Terms—Blockchain, blockchain sharding, load balance, account migration

1 Introduction

Blockchain has been instrumental for enabling decentralized digital currencies [24], [38], and has drawn tremendous attention from academia and industry. However, as the number of transactions surges in the existing blockchain systems, throughput scalability becomes a major challenge in system deployment. Blockchain sharding has been proposed as an effective solution for scaling the throughput of blockchain systems [23]. It splits a blockchain into multiple disjoint parts, called shards. Each shard is maintained by a subgroup of nodes, and different shards execute disjoint transactions in parallel.

While sharding improves blockchain throughput, however, *there is still a significant throughput gap between the existing sharding protocols and the potential throughput speedup.* Ideally, the system throughput should increase proportional to the number of shards. Nevertheless, it is observed that existing sharding systems result in over 30% throughput loss [25] compared to the ideal case, degrading the throughput scalability. More importantly, most existing blockchain sharding protocols overlook another important performance issue: *how to improve user-perceived transaction confirmation delay* (TCD) [41]. The user-perceived transaction confirmation delay means the delay between the time that a transaction is sent by a user until it is committed into the blockchain. This is an important metric because users (aka accounts, clients) are concerned about how quickly the transactions they send can be committed into the blockchain. However, existing blockchain sharding solutions still suffer from high latency. It is observed that the user-perceived TCD in existing solutions reaches more than hundreds of seconds [25], which is disruptive to the user experience.

Cross-shard transactions and *imbalanced transaction load* might be the crux for the above performance degradation [35], [25], [37], [27], [14]. A cross-shard transaction represents a transaction sent from one shard to another, which typically incurs additional communication overhead. Transaction load imbalance refers to the unbalanced number of transactions processed in different shards, resulting in many shards having more transactions than they can process. Both of them may harm the performance of the sharding system. However, the ***first*** research gap is that little work has analyzed in real systems that *how much performance degradation can be impaired by cross-shard transactions and transaction load imbalance, and who dominates the impact on performance.* Some studies claim that the cross-shard transaction is the main culprit in performance loss [25], [28]. Some other works argue that the imbalanced transaction load dominates the performance loss of blockchain sharding [27], [37]. However, their arguments are not confirmed in a real system. The ***second*** research gap is that among those works focusing on transaction load balance [16], [27], [37], they mainly focus on proposing account allocation algorithms, however, *neglecting to design a secure and efficient account and transaction* ***migration*** *protocol in a real blockchain sharding system.* In

- *M. Li is with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China, and with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong (email: mlibn@cse.ust.hk).*
- *W. Wang is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong (email: weiwa@cse.ust.hk).*
- *J. Zhang is with the Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (email: zhangj4@sustech.edu.cn).*
- *J. Zhang and W. Wang are the corresponding authors.*

practice, it is not enough to have only an account allocation algorithm, it is more essential to design a migration protocol in real systems so that accounts and transactions can actually be migrated between shards based on the allocation results to achieve load balance in the system.

To fill the first research gap, we present a systematic study in a real blockchain sharding system (QuarkChain [3]) to show that, it is the *imbalanced transaction load* on different shards that dominates the high delay and limited throughput in general cases. In previous blockchain sharding systems [35], [10], each account is randomly bound to a specific shard. However, the number of transactions generated by each account varies dramatically, which results in a severe load imbalance on different blockchain shards. Through measurements based on real Ethereum transactions, we find that the number of transactions executed by a heavily-loaded shard is more than 5× than a less-loaded shard (Sec. 2.4). A heavily loaded shard causes longer user-perceived transaction confirmation delays because the nodes cannot process transactions as fast as the user send them. A shard with low load, on the other hand, suffers from a decrease in throughput because fewer transactions can be processed. Specifically, it is found that the transaction load imbalance causes up to thousands of seconds of user-perceived confirmation delay and 35% throughput loss (Sec. 3).

Driven by the above findings, we then propose LB-Chain, a novel blockchain sharding framework that achieves Load Balance on different shards. It fills the second research gap by proposing an intelligent account and transaction migration protocol to achieve efficient and secure migrations between shards. The main idea of LB-Chain is that, it periodically predicts the upcoming number of transactions for accounts and uses the prediction results to determine which accounts should be allocated to which shards (i.e., *account allocation algorithm*) for improved load balance. Based on the allocation results, the sharding nodes (miners) in LB-Chain utilize our core design: a secure and efficient *account migration protocol*, to migrate accounts from heavily loaded shards to lightly loaded shards, thus achieving transaction load balance on each shard in the system.

The design of LB-Chain faces two main challenges. ***First***, how to propose an efficient and secure migration scheme in blockchain sharding? To balance the load on different shards, account migration is required. Because merely moving a transaction to other shards will cause the execution failure, as other shards have no information about the accounts that are associated with the transaction. More importantly, malicious nodes may attack the system during account migration, and simple account migration mechanisms also causes performance loss (explained in next paragraph). Therefore, a secure and efficient account migration protocol is necessary to protect the security during migration without excessive performance loss. This is an important point that has been overlooked in previous works. ***Second***, how should an account allocation algorithm determine which and how many accounts should be migrated? Specifically, the load balance results also rely on the account allocation decision (which account to be migrated to which shard). Moreover, it is infeasible to allocate all accounts in practice, as there are numerous accounts in a large-scale system. Therefore, a practical account allocation scheme is required to balance the loads among shards with only a small number of accounts being allocated.

Secure and Efficient Migration for Account and Transaction. To address the first challenge, we propose a secure and efficient account and transaction migration scheme.

Simply migrating the account states causes severe *security issue*. Unlike traditional databases [34], [11], [30], security is particularly important in blockchain systems. In the process of migrating account states, malicious nodes may launch various attacks, such as *generating invalid messages, sending repeated migration messages (replay attacks), etc.* Therefore, we make several designs to secure the account migration. For example, to prevent invalid messages, any account migration message needs to be verified and pass the consensus. To prevent replay attacks, we set a unique serial number (named migration nonce) for the migration message of each account, and the continuity of the nonce is required to be verified for security.

A straightforward migration scheme *degrading system efficiency* if it cannot properly handle the transaction migration related to an account. To achieve transaction migration for improved efficiency, we propose schemes that 1) migrate the queuing transactions along with the account migration and 2) postpone the validations for newly arrived transactions to prevent them from being aborted early (explained in detail in Sec. 5.2). These schemes reduce the transaction validation failure probability, increasing system performance. We also design to raise the execution priority for the account state migration. As the account migration can be processed quickly, the transactions associated with the account can be thus handled quickly, improving system efficiency.

Practical Account Allocation. To address the second challenge, we propose an account allocation algorithm to improve the load balance on different shards by moving as few accounts as possible. The algorithm periodically exploits the predicted upcoming transactions and the existed queuing transactions to calculate the loads for a few *hot accounts*. Based on the calculated loads, the algorithm allocates hot accounts for better load balance, and finally determines the account allocations. Therefore, the proposed account allocation algorithm improves the load balance on shards with only a small number of accounts being allocated. This helps reduce both the complexity of the account allocation algorithm and the number of accounts that need to be migrated, thus improving system efficiency.

We summarize the main contributions of this paper as follows:

- **Measurement Studies:** We conduct systematic measurement studies using a real blockchain sharding system to justify that the imbalanced transaction load is the main culprit to the performance loss of blockchain sharding in general cases.
- **Account and Transaction Migration:** We propose and implement an efficient and secure migration scheme for accounts and transactions in LB-Chain. This scheme is secure under the blockchain sharding scenario, and it maintains high efficiency under real implementations.

- **Account Allocation:** In LB-Chain, we propose and implement a practical account allocation algorithm that can improve the transaction load balance by moving only a few hot accounts.
- **System Implementation:** We develop a prototype for LB-Chain and conduct extensive experiments. Experimental results based on real Ethereum transaction trace show that, compared with existing blockchain sharding schemes, LB-Chain effectively balances the load among shards, reducing user-perceived transaction confirmation delay by up to 90%. Moreover, LB-Chain also achieves near-optimal throughput compared with an ideal load balance scheme.

2 Background, Motivation and Related Work

2.1 Blockchain and Blockchain Sharding

Blockchain, as a promising decentralized technology, has a great potential in numerous scenarios and systems [33], ranging from the cryptocurrency (e.g., Bitcoin [24] and Ethereum [38]), to other infrastructures and applications (e.g., Internet-of-Things [26], [17], Digitial Health [20], [8]). Unfortunately, existing blockchain systems suffer low transaction throughput and high latency issues, which hinder blockchain adoption in many systems that require high transaction throughput and real-time transaction processing.

Several blockchain sharding protocols [23], [41], [18], [35], [42], [14], [15] have been proposed to address the throughput scalability issue in legacy blockchain systems (e.g., Bitcoin and Ethereum). Unlike the legacy blockchain where all nodes need to communicate to maintain the same copy of the blockchain, sharding splits the nodes into multiple groups (shards). Each shard maintains its independent piece of state and transaction history, and executes different transactions in parallel. Among these works, however, they focus on designing and implementing various sharding systems to improve the scalability for legacy blockchain. *Most of them do not to analyze the reasons affecting the sharding performance.*

2.2 UTXO/Account Model and Existing Transaction Placement Strategy

A natural question for blockchain sharding is how to place transactions on different shards. There are two models: UTXO model and account model. In UTXO (Unspent Transaction Output) model [9], [24], transactions are placed in different shards independently according to the transaction ID [18], [23]. While in the account model [38], the transaction is placed to different shard according to its sending account [35], [10]. Therefore, the transactions sent by the same sending account are placed in the same shard. The account model is usually thought to be more universal than UTXO as it can easily support smart contracts [7]. Additionally, the account model can be extended and used in more complex scenarios and applications other than cryptocurrency [12], [40]. Therefore, in this paper, our system is built on the *account model*.

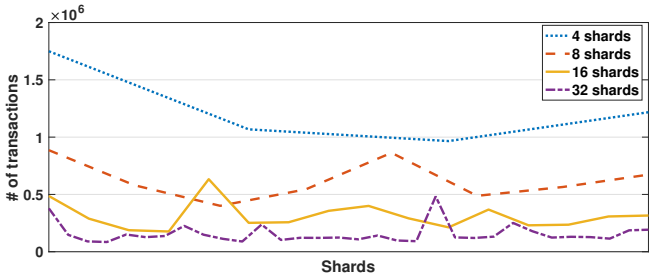


Fig. 1. Transaction distribution across shards. Each line represents the transaction load on each shard for the corresponding number of shards

2.3 Performance Metrics of Blockchain Sharding Systems

One objective of blockchain sharding is to improve the throughput of blockchain. The system throughput is measured by *transaction per second (TPS)*, meaning the number of transactions that can be executed per second. Ideally, the throughput should scale out linearly with the number of shards increasing. However, it is observed that the TPS could degrade over 30% compared with the ideal transaction rate in sharding systems such as OmniLedger [25], which is quite severe. Our measurements also reveal similar observations where up to 35% TPS loss occurs in the existing sharding system (Fig. 2c).

From the user's perspective, what matters more than system throughput is how long it takes for the transactions they send to be packed into blocks (i.e., *user-perceived transaction confirmation delay (TCD)*). Reducing user-perceived TCD is also another design target of blockchain sharding. However, existing blockchain sharding systems cause huge user-perceived TCD (hundreds of seconds in OmniLedger [25], and up to thousands of seconds in our experiments). Some sharding systems claim they achieve short transaction confirmation delays [41], [14]. However, the latency in their works is defined as the time from when a transaction is packed into a block to when that block is committed. This latency ignores the queuing time between when the transaction is submitted by the user until the transaction is packed into the block. Therefore such delay is less meaningful to the users.

Finally, not only do users suffer from high TCD, but there are also significant differences between the transaction confirmation delays of different users. We name this difference of the average transaction waiting time (TCD) among users (accounts) as *fairness*. Poor fairness inevitably detracts from the user experience, therefore maintaining good fairness is essential in practical systems. However, we observe in our measurement study that existing sharding system has poor fairness performance.

Consequently, it is essential to design a scalable sharding system that achieves increased system throughput, reduced transaction confirmation delay, and improved fairness among accounts.

2.4 Transaction Load Imbalance and Cross-shard Transactions

To scale out blockchain sharding systems, it is essential to investigate which factors affect sharding performance.

Transaction load imbalance and cross-shard transaction are believed to be two factors that injure existing blockchain sharding systems' performance [25], [35], [37], [27], [28].

Transaction load imbalance represents the difference in the number of transactions processed by different shards, resulting in some shards being busy while others idle. In the account model, load imbalance becomes even worse than that in the UTXO model, since once an account is allocated to a certain shard, all its transactions are allocated to that shard. Therefore, hot accounts (account that involves a great number of transactions) easily overwhelm the load of the shard they are in, resulting in uneven load among shards.

We conduct a measurement study to evaluate the severity of load imbalance in existing sharding systems. Specifically, we use real-world Ethereum transactions and distribute them according to the existing random transaction placement scheme (refer to Sec. 2.2 in detail). Fig. 1 shows that the existing transaction placement strategy causes severe load imbalance across shards. For example, in 32 shards, a heavy-loaded shard has more than $5\times$ number of transactions than a light-loaded shard.

Cross-shard transaction is generated when one transaction is transmitted from one shard to another. In this paper, cross-shard transactions include those that span two shards (e.g., normal transfer transactions, and simple smart contract transactions with a single step). Compared with an intra-shard transaction, a cross-shard transaction typically involves additional time and network overhead. The reason is that existing blockchain sharding systems usually design a series of multi-round protocols for cross-shard transactions to prevent double-spending [10], [41], hindering the efficiency of transaction processing.

2.5 Related Works

Some works explore the impact of cross-shard transactions on blockchain sharding performance [22], [25], [28]. As a typical example, authors in [25] claim that cross-shard transaction causes huge impact on the sharding performance, but the conclusion is mainly derived from theoretical analysis and simulations. In next section, our experiments in real systems will show that it is the imbalanced load that causes a great degradation on performance, especially on the user-perceived confirmation delay.

A few related works have studied load balancing in blockchain sharding [37], [27], [16], [19]. For instance, [37] proposes a load balancing mechanism based on transaction load prediction and account relocation algorithm. In [19], the authors propose a load balancing framework in sharded blockchains in which objects (e.g., accounts) are frequently reassigned into shards. The authors in [16] propose a load balancing scheme using the graph partitioning algorithm. However, those works focus mainly on the algorithm design for account allocation. *Their works do not involve the design and implementation of a practical account **migration** mechanism in real sharding systems.* One of the main contributions of LB-Chain is to propose a secure and efficient account migration mechanism, which is not studied in the previous works. Moreover, we conduct measurement studies in real systems to justify the performance degradation caused by load imbalance, as discussed in the next section.

Load balancing is an important issue in traditional distributed databases [34], [11], [30]. However, distributed databases and blockchain sharding are inherently different [29]. There are Byzantine nodes in blockchain who can behave arbitrarily wrong. While in distributed databases, nodes are typically assumed honest or can only crash. Therefore, blockchain systems require higher security guarantee compared to databases. Due to different security assumptions, it is more challenging to design practical migration mechanisms in blockchain sharding to achieve load balancing. For example, how to perform secure migration in blockchain sharding where there is no trusted coordinator? LB-Chain proposes a secure and efficient migration mechanism to help the blockchain sharding system balance its load.

3 Measurement Study

We conduct measurement studies in a real sharding system to analyze the negative impact of transaction load imbalance and cross-shard transactions on user-perceived transaction confirmation delay and system throughput, respectively. The results shows that, in our experiments, it is the imbalanced transaction load that results in high latency and limited throughput.

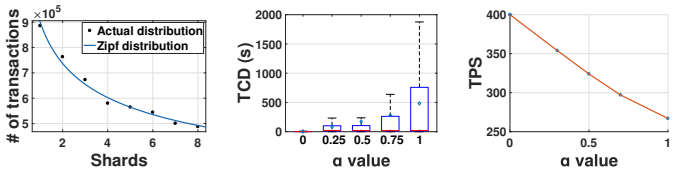
3.1 Basic Experiment Settings

Our measurement study is based on a well-known public blockchain sharding project named QuarkChain [3]. QuarkChain's implementation is based on Ethereum. In the experiments, we deployed 32 r5.xlarge EC2 instances in different regions, each with a quad-core processor and 32G memory. 8 shards are implemented, and 800,000 transactions are generated. We manually adjust the cross-shard transaction ratio and the number of transactions (loads) on different shards. For the blockchain parameters, we set a 500 transaction limit for each block, a target of 10-second block creation interval (resulting a 50 TPS transaction processing capacity for each shard), and a 30 Mbps end-to-end bandwidth. The average transaction generation rate is set as 50 TPS per shard by default. These parameter settings are practical and reasonable in real systems, which is similar to previous work [35] and Ethereum. We think that the results of our experiments are somewhat *generalizable*, since our experimental environment is practical and the system we based on has a similar underlying architecture to many other blockchain sharding systems.

3.2 Impact of Transaction Load Imbalance

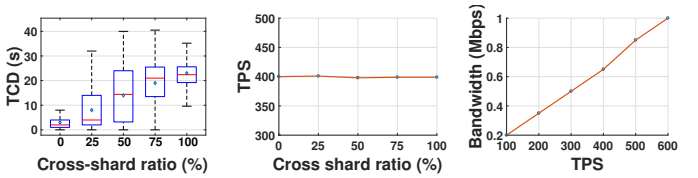
We first evaluate the performance degradation caused by transaction load imbalance on throughput and delay. The results illustrate that transaction load imbalance causes significant degradation on user-perceived TCD and TPS.

To eliminate the influence caused by cross-shard transactions, we controlled the ratio of cross-shard transactions as 0. We changed the skewness of transaction distribution on shards (i.e., change the number of transactions in different shards) to conduct the evaluation. We analyzed the transactions in Ethereum and found that each shard's real transaction distribution can be fitted into Zipf distribution. Fig. 2a



(a) Fitting Zipf distribution (b) α value vs. TCD (c) α value vs. TPS

Fig. 2. Results for transaction load imbalance.



(a) Cross-shard ratio vs. TCD (b) Cross-shard ratio vs. TPS (c) Bandwidth cost for cross-shard txs

Fig. 3. Results for cross-shard transactions.

shows the number of transactions for shard No. 1 to shard No. 8, which nicely fits the Zipf distribution. Therefore, in our experiments, the transaction distribution is assumed to follow Zipf distribution. We change the exponent parameter α of Zipf distribution to control the skewness of transaction distribution in the experiments, where a larger α means a more imbalanced load distribution.

Fig. 2b and Fig. 2c show that transaction load imbalance prolongs user-perceived TCD to more than 1,000 seconds and decreases TPS by up to 35%. Specifically, the TPS dramatically drops and TCD increases when the load becomes imbalanced. Here we only show the results up to $\alpha = 1$ since the α value in real transaction distribution is usually less than 1. It is expected that the impact on TCD and TPS will increase as the load becomes more imbalanced.

3.3 Impact of Cross-shard Transactions

We now measure the performance degradation caused by cross-shard transactions, and show that cross-shard transaction is *not* a main factor that affects user-perceived transaction confirmation delay and system throughput.

To eliminate the influence caused by transaction load imbalance, we kept the number of transactions on each shard to be the same (i.e., 50 TPS per shard). We then changed the ratio of cross-shard transactions to observe the effect of cross-shard transactions on TCD and TPS.

The impact of cross-shard transactions on user-perceived TCD is demonstrated in Fig. 3a. As expected, *the cross-shard transactions do influence TCD*, but the influence is small. Specifically, the TCD caused by cross-shard transactions is one order of magnitude smaller than load imbalance. For example, on average only 22s TCD is caused even the cross-shard ratio is 100%. The TCD increases when the cross-shard transaction ratio increases. The reason is that existing sharding systems usually use a series of multi-stage protocols to process cross-shard transactions, thus increasing the confirmation delay. It is worth noting that the transaction confirmation delay does not increase infinitely, as the upper limit of the cross-shard ratio is 100%. As a result, the transaction confirmation delays caused by cross-shard transactions are small in general.

As shown in Fig. 3b, *TPS remains almost constant* when we change the cross-shard transaction ratio. The reason is that the network overhead caused by the cross-shard transactions is small compared with the bandwidth limitation. Specifically, Fig. 3c illustrates the net bandwidth cost by cross-shard transactions under different TPS, in which the cross-shard ratio is set as 100%. We found that even 600 cross-shard transactions are executed per second, the bandwidth cost by cross-shard transactions is only 1 Mbps, which is far less than the practical bandwidth limitation.

Justification of Our Results. Some previous arguments (e.g., [25]) suggest that cross-shard transaction mainly causes the performance degradation. We speculate that our results differ from theirs due to the following points. First, their evaluation is based on the UTXO model, in which each transaction has a larger size than in the account model, which occupies more bandwidth. Second, they conduct evaluations under extremely heavy load (hundreds of TPS per shard), whereas we use a general and more practical load setting (as mentioned in Sec. 3.1). Finally, their deductions and evaluation are based on theoretical analysis and simulations, without the support of a real implementation.

3.4 Summary of Measurement Study Results

To sum up, in general, transaction load imbalance causes most of the negative effect on sharding performance. Specifically, load imbalance causes extremely long user-perceived TCD, and the impact on TCD is an order of magnitude bigger than that of cross-shard transaction (hundreds of seconds vs. dozens of seconds). Additionally, load imbalance causes remarkable TPS reduction (up to 35%), while cross-shard transaction causes no influence on TPS in the general case. It is worth noting that, although our measurement is based on QuarkChain, the results above and analysis can be *generally applied* to existing account model-based blockchain sharding systems, as most of the underlying protocols are similar (based on Ethereum).

4 System Overview

In light of the observations that load imbalance is the dominant factor that degrades sharding performance, we propose LB-Chain, in which smart account allocation and migration schemes are designed to achieve load-balanced and scalable sharding.

4.1 System Model and Overview

LB-Chain is a blockchain sharding system for improved transaction load balance. Like existing blockchain sharding systems [14], [18], [23], LB-Chain consists of multiple P2P nodes (miners). All the nodes are split into multiple shards. Each shard maintains its independent ledger (blockchain), account information, and transaction history. Transactions are sent via different accounts (aka clients, users) into the blockchain sharding system. Similar to many previous works [35], [41], [3], a transaction is sent to one or multiple

nodes in the network. The nodes then follow the gossip protocol and route the transaction to the corresponding shard. Like many other systems [18], [23], [35], [14], nodes are connected by a partially synchronous peer-to-peer network, in which messages sent by a node can reach any other nodes with optimistic, exponentially-increasing time-outs. Finally, as mentioned, our system is built on the account model.

The system architecture is illustrated in Fig. 4. There are mainly two parts in LB-Chain: the allocation service who performs account allocation and the sharding network who conducts account migration. To *balance the load* on different shards, the allocation service (explained in Sec. 4.2) periodically performs account allocation. The allocation scheme predicts the number of transactions for accounts and uses the predicted results to decide which shard an account should be allocated to. To actually *achieve improved load balance in blockchain sharding network*, the sharding nodes then, according to the allocation results, perform account migrations to migrate accounts from the previous shard to the newly allocated shard and migrate their transactions correspondingly.

4.2 Account Allocation

In account allocation, the *transaction prediction* is performed first to predict the number of future transactions. Based on the predicted results, the *account allocation algorithm* is then performed to decide the accounts' migrated locations for improved load balance. The account allocation is performed by a third-party entity named allocation service, which is assumed to be trustworthy in our paper. Many other works also assume similar third-party entities for various functionalities such as ordering services [31], [5], [4], [32] or smart contract service providers [39], hence we think such an entity is practical.

Transaction Prediction. Transaction prediction [21], [36] is an essential yet challenging part of the system. To improve the load balance among shards, it is necessary to accurately predict how many transactions will be generated by the accounts in the future. This allows the subsequent account allocation algorithm to calculate a more load-balanced account allocation result. Moreover, the number of transactions sent by different accounts changes dynamically over time. Therefore, the prediction is performed periodically based on the epoch.

To perform transaction prediction, the allocation service periodically retrieves transaction history from the sharding network. Using machine learning, the allocation service then predicts the upcoming number of transactions generated by different accounts and shards.

Account Allocation Algorithm. A well-performed account allocation scheme should achieve load balance with high efficiency. However, *performing prediction and allocation for all the accounts is infeasible*, as a large-scaled system contains numerous accounts. Fortunately, we find that a few accounts (hot accounts) generate most of the transactions (e.g., 100 hot accounts generate more than 50% of the transactions in Fig. 5b) in practice. Seen in this light, we configure the allocation service to only *allocate for the hot accounts*, while leaving the rest of the accounts unmoved.

The account allocation algorithm is executed periodically. It determines which shard the accounts and their transactions will be migrated to. Specifically, at the beginning of each epoch, based on the prediction results, it moves few hot accounts from the heavily-loaded shard to lightly-loaded shard to improve load balance, and stops when there is no improvement.

4.3 Account and Transaction Migration

To actually achieve improved load balance on each shard, the account and transaction migration is then performed after account allocation. The migration is performed by nodes (miners) in the sharding network. The nodes in LB-Chain periodically obtain account allocation results from the allocation service and perform migrations. If an account is reallocated to a new shard, it should be migrated along with its transactions.

To improve transaction load balance, the transactions are required to be moved from one shard to another. However, this is not straightforward. Solely changing the location of transactions causes transaction execution failure, as other shards do not have the states that are associated with the transactions. Thereby, an account migration scheme is required to enable the account current state (e.g., balance, nonce) be moved across shards.

To enable account state migration between shards, we design the *account migration transaction*. It is responsible for containing and sending the account state across the shard and is generated by the nodes in corresponding shard. However, there are various *security issues* faced during account migration. To address it, we make specific designs for the account migration transaction, and modify the intra-shard consensus accordingly. Moreover, we requre the migration process to be verified by other nodes via intra-shard consensus to secure the account migration.

As accounts being migrated between shards, the transactions associated with them also need to be migrated accordingly to ensure the efficiency of the system. To achieve transaction migration for *improved efficiency*, we propose to migrate the queuing transactions along with the account and postponing validations for newly arrived transactions to prevent new transactions from being aborted. We also propose to raise the execution priority for the account migration transaction, so that the account migration transactions and other related transactions can be processed quickly. According to the above designs, the account and transaction migration can be processed efficiently.

5 Account and Transaction Migration

We introduce how we design the account and transaction migration scheme in LB-Chain in this section and leave behind the explanation of account allocation in Sec. 6.

At the beginning of every epoch, according to the account allocation algorithm result, if an account is allocated to a new shard that is different from the shard it is allocated in the last epoch, the account should be migrated from the previous shard (i.e., *source shard*) to the new shard (i.e., *destination shard*). Besides, the upcoming new arrival transactions of the migrated accounts should be executed

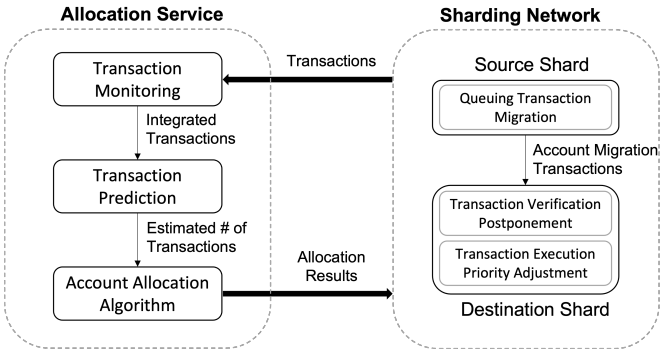


Fig. 4. System architecture.

by corresponding destination shards. The objective of the account migration is to safely migrate accounts to proper shards according to the allocation results, with low throughput loss, low latency, and high fairness.

Basic Knowledge. To design a practical account migration scheme, we must know how existing blockchain sharding systems work. There are several shards in the system. Each shard maintains several accounts, the nodes (i. e., minors) belong to the shard maintains the state of its accounts and process the transactions generated from the accounts. The account state contains basic information about an account, such as balance, current transaction nonce [38] (which indicates the sequence number of the transaction in the account). Each transaction of an account has some basic fields, such as the sender account, the receiver account, transaction nonce, transfer value, the signature, etc. When a transaction is packed into a block, the miners should perform verification for the transaction to check whether the sender's balance is enough, whether the nonce is successive, whether the signature is correct, etc. The nonce should be successive to guarantee security (e.g., preventing replay attacks).

5.1 Secure Account Migration

When an account is reallocated to a new shard, in order to execute its upcoming transactions, the nodes in the destination shard should create this account and maintain its state. However, it is not easy to notify the new shard and share the account state information among independently operating shards. Therefore, we propose a new type of transaction named *account migration transaction*, which is generated by the nodes. This special cross-shard transaction is used to notify the destination shard about the state of the migrating account.

Ensuring Security. A naive account migration scheme is vulnerable to various attacks. Malicious nodes may send wrong migration transaction (i.e., *transaction manipulation*), send the same transaction multiple times (i.e., *replay attack*), or refuse to send the transaction (i.e., *silence attack*) to intercept the account migration process. Therefore, to ensure that the account migration process is resistant to typical malicious behaviors mentioned above, we propose the following mechanisms.

To prevent *transaction manipulation*, each account migration transaction should be verified by sharding nodes. Nodes in a shard reach consensus on transactions, and the account migration transactions are then sent to corresponding destination shards. Besides the basic verification, each account migration transaction contains several unique values of fields that needs to be verified: i) the sender account and the receiver account of the transaction should be the same (the account to be moved), ii) the account migration transaction should be signed by the node proposing the block in the source shard, iii) the transferred value of the account migration transaction should equal the balance of the account to be moved. iv) the source and destination shards in the account migration transaction should be the same as the account allocation result (each node caches the account allocation results for recent epochs for validation).

A malicious node could save the account migration transaction and resend it later to launch *replay attack.* To prevent that, we add an extra field in the account migration transaction to maintain its sequence number in migration transactions of the account (called the *migration nonce*). The migration nonce should be maintained in the state of each account. When reaching consensus, each node should verify whether the migration transaction of a certain account have consecutive migration nonce. Only if the nonce is consecutive can the transaction be passed the verification.

Another malicious behavior is that a malicious node in the source shard *does not send* the account migration transactions to destination shards. In this case, the client or the destination shard can inform the source shard (similar to [35]). Specifically, the client who does not receive a transaction confirmation response after a timeout can inform the source shard. The destination shard can also send a notification to the source shard if it finds that the account migration transaction has not been received after a timeout (by checking the account allocation results). The notification thus allows account migration transactions to be resent by other nodes in the source shard.

Migration Procedure. When an account migration occurs, each node in the source shard locally generates the account migration transaction and signs it using its own signature. The block producer (e.g., the miner who has the chance to produce a block) packages the account migration transactions generated by itself along with normal transactions into the block, and broadcasts the block in the shard. All nodes in the shard should verify all the transactions, and reach a consensus. After the account migration transactions successfully pass the consensus, the account migration transactions are sent to corresponding destination shards (along with normal transactions). The source shard deletes the migrated account while the destination shard constructs the account and updates the account state. If a client wants to query its account or transaction states, it can send a request to the blockchain network, and the nodes will route the request to the corresponding shard according to the account allocation results, and that shard will return the query result to the client. By the way, to ensure that the balance of the migrated account does not change during migration, the account migration transaction requires no transaction fee. To encourage nodes to package the account migration transactions, the one who packages the account migration transaction into the block will be rewarded by the system (similar to the

mining reward).

5.2 Efficient Transaction Migration

The above mechanisms enable secure account migration. However, to further improve the efficiency of our system, we need to deliberately handle the queuing transactions in the source shards, the newly arrived transactions in the destination shards, and the account migration transactions.

Migration of Queuing Transactions in Source Shard. This design aims at handling queuing transactions. Specifically, according to the basic mechanism, the account in the source shard will be removed, and the state will be cleared after the account is migrated. However, if the migrated account has queuing transactions that are waiting to be packaged in the source shard, verifications for those transactions will fail. More seriously, the failure of verification for those transactions will cause the nonce to be discontinuous. As a result, all the verifications of subsequent transactions sent by the migrated account will fail, which results in a large throughput loss. To address this problem, we propose the mechanism that requires the source shard nodes to send the migrated accounts' queuing transactions to the destination shard after the account is migrated.

Postponement of Transaction Verification in Destination Shard. This design aims at handling newly arrived transactions. Specifically, the account migration takes time, during the account migration process, the verification for the newly arrived transactions generated by the migrated account will fail, as the account state in the destination shard is not updated (the migration transaction is waiting to be packaged). Failing to verify these new transactions leads to the nonce incoherence, again causing significant throughput loss. Therefore, we design to postpone their verification. As a result, the migrated accounts' newly arrived transactions can be waited in the destination shard's queue and be executed after the account is moved.

Adjustment of Transaction Execution Priority in Destination Shard. This design aims at handling account migration transactions. Particularly, another significant problem of the basic migration mechanism is the long waiting time for the account migration transactions. Be noted that the account migration transactions are the bottleneck of the operation, as all the queuing and upcoming transactions of the migrating account rely on the successful operation and package of the migration transaction. Therefore, the long waiting time for an account migration transaction will inevitably prolong the migration process, reducing throughput and increasing delay. The account migration transaction thus should be given the highest priority. Therefore, we raise the execution priority of the account migration transaction among all transactions. As a result, it can immediately be executed once received.

With the above mechanisms, we overcome the challenges of achieving account and transaction migration in real blockchain sharding systems. Moreover, the proposed migration scheme ensures security, and achieve an increase in throughput and a reduction in transaction confirmation delay.

6 Account Allocation

We now introduce our account allocation design, which aims to make the transaction load in different shards balanced while keeping the number of account migrations within a low level to reduce the migration overhead. It consists of two parts: transaction prediction and account allocation algorithm.

6.1 Transaction Prediction

To allocate accounts, the allocation service needs to periodically predict the number of transactions that will be produced by different shards and accounts in every epoch. Various prediction methods have been used for different purposes in blockchain systems [21], [36]. In this paper, the allocation service collects historical transaction statistics (e.g., the number of transactions of each account and digital currency prices), and uses a 2-layer Long Short-Term Memory (LSTM) model [13] to predict the number of transactions for each account in the following epochs. Every layer of the LSTM model consists of 100 neurons, with a dropout equal to 0.001. The loss function of LSTM is set to be MSE.

Feasibility. Learning is computationally intensive and time-consuming. To achieve a practical account allocation scheme, we increase the prediction interval to estimate every epoch's number of transactions for the next N epochs at one prediction. An example of learning results of 200 epochs transaction prediction is shown in Fig. 5a, in which the prediction is accurate enough for the following account allocation (detailed discussion in Sec. 7).

Besides, performing prediction for every account is infeasible in practice, as millions of accounts are in a large-scale system (Sec. 7). Additionally, most of the accounts only send a few transactions, thus there is not enough data to support the learning for accurate predictions. We find that a small portion (<0.02%) of hot accounts send most (>50%) of the transactions (see Fig. 5b) in practice. Seen in this light, we only focus on the hot accounts for every shard and predict the transaction generated by them in every epoch. For the other light accounts, we integrate them as an *aggregated account* for each shard, and predict a total number of transactions for the aggregated account. Similarly, the allocation algorithm only focuses on hot accounts to allocate and migrate. We will see that we can achieve similar performance by doing this while dramatically reducing the computational complexity.

6.2 Account Allocation Algorithm

When obtaining the transaction prediction results, the allocation service periodically determines the locations (shards) for accounts and for their generated transactions in each small epoch. We first formulate the account allocation problem and show its NP-hardness. Then, we propose a heuristic algorithm to solve the account allocation problem.

Problem Formulation. We state the account allocation problem as follows:

$$\min \quad \frac{\sum_{i\in\mathcal{S}}[\sum_{j\in\mathcal{A}}[(n_j^t + q_j^{t-1})\cdot x_{i,j}^t] - \bar{l}^t]^2}{|\mathcal{S}|} \tag{1}$$

$$\text{s.t.:} \quad n_j^t,\, q_j^{t-1} \in \{0,1,2,\ldots\},\ \forall j \in \mathcal{A},$$
$$x_{i,j}^t \in \{0,1\},\ \forall i \in \mathcal{S},\ \forall j \in \mathcal{A}.$$

The goal of the objective function is to minimize the variance of the number of transactions between shards (i.e., improve load balance) in epoch t. Specifically, for a given account $j \in \mathcal{A}$, where $\mathcal{A}$ is the set of accounts, n_j^t represents the amount of the predicted upcoming transactions of account j during the upcoming epoch t. q_j^{t-1} means the amount of the queuing transactions of account j remained in last epoch $t-1$. For each shard $i \in \mathcal{S}$, where $\mathcal{S}$ is the set of shards, $x_{i,j}^t$ means whether the account j is located in shard i in epoch t. $x_{i,j}^t = 1$ means the account j is located in shard i during epoch t, and $x_{i,j}^t = 0$ otherwise. We define $l_j^t = (n_j^t + q_j^{t-1})$ as the load for account j in epoch t. $\bar{l}^t$ means the average number of transactions that will be executed by each shard, calculated by:

$$\bar{l}^t = \frac{\sum_{j \in \mathcal{A}} l_j^t}{|\mathcal{S}|}. \tag{2}$$

The account allocation problem can be reduced to the k-partitioning problem, which is NP-hard [2]. Therefore, we propose a heuristic account allocation algorithm with better efficiency and acceptable performance.

Algorithm 1 Account Allocation Algorithm for Epoch t

1: INPUT: $\mathcal{S}$, $\mathcal{A}_{hot}$, n_j^t, q_j^{t-1}, l_j^t, $x_{i,j}^{t-1}$, m_i^t, $\bar{l}^t$ for all i, j
2: $x_{i,j}^t \leftarrow x_{i,j}^{t-1}$, $V_t = \tilde{V}_t = \frac{\sum_{i \in \mathcal{S}} [\sum_{j \in \mathcal{A}_{hot}} (l_j^t \cdot x_{i,j}^t) + m_i^t - \bar{l}^t]^2}{|\mathcal{S}|}$
3: Sort each shard $i \in \mathcal{S}$ by its load $(\sum_{j \in \mathcal{A}_{hot}} (l_j^t \cdot x_{i,j}^t) + m_i^t)$ in descending order, save to a sorted shard list S_{heavy}, find the most heavy-loaded shard i_{heavy} and the most light-loaded shard i_{light}
4: Sort the accounts in i_{heavy} by loads l_j^t in descending order, save to a sorted account list A_{heavy}
5: **while** $\sum_{j \in \mathcal{A}_{hot}} (l_j^t \cdot x_{i_{heavy},j}^t) + m_{i_{heavy}}^t > \bar{l}^t$ **do**
6: **for** j in A_{heavy} **do**
7: Move j from i_{heavy} to i_{light}, update $\tilde{V}_t$
8: **if** $\tilde{V}_t < V_t$ **then**
9: $x_{i_{light},j}^t \leftarrow 1$, $x_{i_{heavy},j}^t \leftarrow 0$, $V_t \leftarrow \tilde{V}_t$
10: Update the load on each shard, update S_{heavy}, i_{heavy}, i_{light} and A_{heavy}
11: **go to** line 5
12: **end if**
13: **end for**
14: Remove i_{heavy} from S_{heavy}, update i_{heavy}, i_{light} and A_{heavy}
15: **end while**
16: OUTPUT: $x_{i,j}^t$ for all i, j

Algorithm Design. Solving the account allocation problem is time-consuming and extremely inefficient in real systems. Therefore, when performing the account allocation algorithm, the allocation service only decides the migration locations for hot accounts. The rest of the accounts (aggregated account) are kept fixed on each shard.

The intuition of the proposed account allocation algorithm is to move as few accounts as possible to balance the load. In each epoch, the algorithm iteratively moves hot account from the heavy-loaded shard to the light-loaded shard to improve load balance. The load balance level in epoch t is defined as:

$$V_t = \frac{\sum_{j \in \mathcal{S}} [\sum_{j \in \mathcal{A}_{hot}} (l_j^t \cdot x_{i,j}^t) + m_i^t - \bar{l}^t]^2}{|\mathcal{S}|}, \tag{3}$$

which represents the variance of the transaction amounts between shards. The definition is similar as Eq. (1). However, $\mathcal{A}_{hot}$ here is the set of *hot* accounts. m_i^t represents the predicted number of transactions for shard i generated by its aggregated account (as mentioned in Sec. 6.1) during upcoming epoch t. In addition, $\bar{l}^t$ here is calculated as:

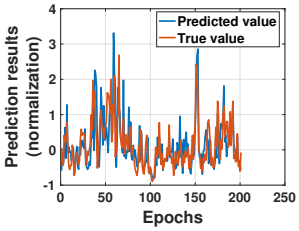
$$\bar{l}^t = \frac{\sum_{j \in \mathcal{S}} m_i^t + \sum_{j \in \mathcal{A}_{hot}} l_j^t}{|\mathcal{S}|}. \tag{4}$$

The account allocation algorithm is shown in Algorithm 1. According to the allocation results in last epoch $t-1$ and the prediction results in epoch t, the algorithm initializes the loads and the load variance V_t (line 2). In each subsequent iteration, the hottest account is selected from the most heavy-loaded shard and its transactions are moved to the most light-loaded shard (line 3-7). Noting that based on Sec. 5.2, the queuing transactions q_j^{t-1} are also migrated. Meanwhile, the new variance $\tilde{V}_t$ is calculated (line 7). If the transaction load balance is improved (line 8), the result of this migration will be retained, and next iteration will be entered after updating the parameters (line 9-11). Otherwise, the account is not migrated and the algorithm tries to move less hot account. If all the hot accounts in i_{heavy} are failed to improve V_t, the algorithm then tries to move hot accounts in less heavy shard (line 14). The algorithm stops when there is no load balance improvement for all the shards with loads larger than average. It is worth noting that the algorithm will converge and finish because that V_t decreases monotonically, and only when V_t decreases should an account be migrated.

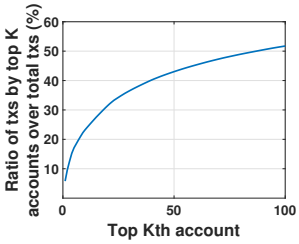
7 Implementation and Evaluation

We implemented a prototype of LB-Chain based on QuarkChain (an Ethereum-based sharding project). The system is written in GO language with 3000+ lines of code, while the prediction algorithm is written in Python. Our system is deployed on Amazon EC2 with r5.xlarge instances for sharding nodes, r5.8xlarge instances for the allocation service, and r5.24xlarge instances for clients. The allocation service connected to several nodes in each shard via RPC in order to communicate. We performed our experiment on up to 32 shards using up to 256 sharding nodes distributed in different regions. The nodes in the sharding network communicate via the gossip protocol.

We evaluate the performance of our system by replaying 15 million historical Etheruem transactions (including normal transfer transactions, and simple smart contract transactions with a single step) sent by more than 1.5 million accounts. For the trasanctions related to smart contract, we simulate the invocation relationships of these smart contracts and implement them through transactions. Unless otherwise specified, we randomly selected three sets of transactions (5 million continuous transactions in each

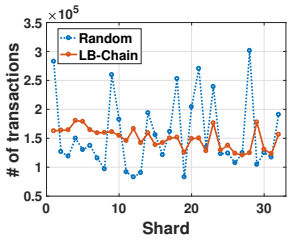


(a) Prediction results for an account

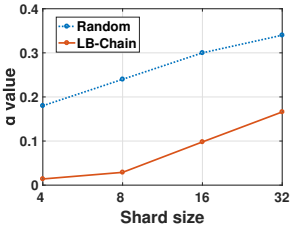


(b) tx ratio of top 100 accounts

Fig. 5. Prediction feasibility analysis.



(a) Load distribution among shards



(b) α value comparison

Fig. 6. Load balance comparison.

set) and showed the average results of them. The account migration epoch t is set as 10 minutes. Additionally, we use a practical setting where the end-to-end bandwidth was limited to 30Mbps, and we set a 500 transaction limit for each block and a target of 10-second block creation interval as default (resulting a 50 TPS transaction processing capacity for each shard). The total transaction sending rate is set as (50×number of shards) TPS.

Baselines. We benchmark LB-Chain against two baselines.

Random Allocation: *Random Allocation* represents the existing random transaction placement scheme (Sec. 2.2) without migration, where the accounts are placed randomly to a particular shard, and the transactions are placed according to the sender account's address.

Ideal Allocation: In *Ideal Allocation*, all transactions are assumed to arrive at the very beginning. It solves the k-partition problem for all the transactions based on all accounts to decide the account allocation results. This algorithm is a theoretical upper bound that our account allocation scheme can achieve. This algorithm is infeasible in a real implementation, as it is extremely time-consuming.

7.1 Prediction Results

Before the demonstrations of LB-Chain performance, we first justify the feasibility of our prediction approach. Fig. 5a shows an example of the prediction results for one top account. In the predictions, we use the features in the last 50 epochs to predict the number of transactions in the upcoming one epoch. In our implementations, the predicted values are close to the actual values, with an average of 11% errors. Another observation is that single learning can estimate the results for multiple epochs in the future (e.g., 200 epochs).

Fig. 5b illustrates the number of transactions generated by top accounts in a randomly sampled set of transactions (over 8,000,000 transactions and 800,000 accounts). Results show that less than 100 top accounts (out of 800,000) send more than half of the transactions, a general case in Ethereum transaction dataset. Therefore, the prediction method is reasonable and feasible, in which the allocation service only needs to predict a limited number of top accounts.

7.2 Load Balance

We now evaluate whether LB-Chain can balance the number of transactions on different shards. Fig. 6a shows the load distribution results for 32 shards. It is observed that in our LB-Chain system, the number of transactions across different shards was more evenly distributed than *Random*. For instance, the heavy-loaded shard has 4× number of transactions higher than a less-loaded shard in *Random*, whereas LB-Chain improves it to 0.5×. We noticed that our scheme still cannot achieve an ideally completely balanced allocation. This is because the allocations and migrations can only be conducted on the granularity of account under the account model. There are several extremely hot accounts who send more transactions than a single shard can handle. Thus there are peaks appeared on the corresponding shard where the extremely hot accounts are located. This inherent problem cannot be solved by any other algorithms. We also use the exponent parameter α in Zipf distribution to evaluate the load balance level. As shown in Fig. 6b our migration mechanism improved the imbalanced transaction distribution and reduce α from 0.24 to 0.029 in 8 shard setting, (the smaller the α, the more balanced the load).

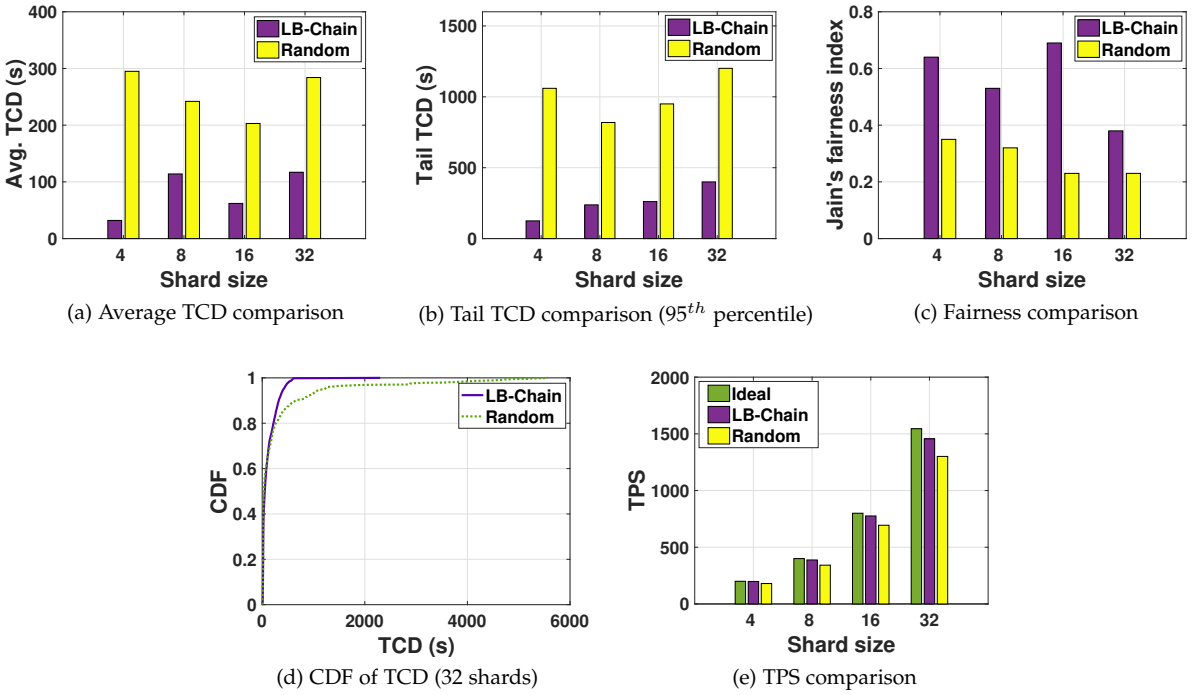
TABLE 1
Performance Improvement

Shard size	4	8	16	32
Avg. TCD improvement ratio (%)	89.2	52.9	69.5	58.8
Tail TCD improvement ratio (%)	88.2	70.9	72.5	66.7
Fairness improvement ratio (%)	82.9	65.6	200	65.2
TPS improvement ratio (%)	10	13.4	11.8	11.9
TPS improvement upper bound (%)	11.1	16.9	15.2	17.7

7.3 Confirmation Latency and Fairness

The transaction load imbalance affects the transaction confirmation delay dramatically. In this experiment, we analyzed the average delay over all accounts in Fig. 7a, and the 95 percentile account delay in Fig. 7b. Together with the Table. 1, we see that the transaction confirmation delay was reduced in all cases after the migration, with a maximum reduction of nearly 90%.

We also analyzed the effect of the migration mechanism in LB-Chain on fairness. In our experiment, we use Jain's Fairness Index [1] to measure the fairness of transaction confirmation delay among different accounts. A larger index value (close to 1) represents a fairer case. Fig. 7c shows that Jain's Fairness Index is improved by more than 60% in all cases by using LB-Chain. Besides, as shown in Fig. 7d, 99% accounts wait less than 500 seconds in LB-Chain while in *Random Allocation*, 10% accounts wait for more than 1,000



(a) Average TCD comparison (b) Tail TCD comparison (95^{th} percentile) (c) Fairness comparison

(d) CDF of TCD (32 shards) (e) TPS comparison

Fig. 7. Main performance evaluation results.

seconds. The results further demonstrate the improvement of our migration mechanism on the confirmation delay and fairness.

7.4 System Throughput

The transaction load imbalance also impacts the system throughput, hence we evaluated the system TPS. As shown in Fig. 7e, our mechanism improves the system throughput over 10%, which is very close to that achieved by the ideal allocation. Noting that the ideal allocation has very high computational complexity, which is impossible to achieve in real-time in a blockchain system. To make a more detailed analysis, the gap between LB-Chain and the ideal solution is mainly due to the following reasons. First, LB-Chain does not allocate for all accounts but only allocates the hot accounts. Second, the ideal scheme assumes no prediction error. Third, transactions in LB-Chain arrive online, whereas in *Ideal*, transactions are assumed to arrive simultaneously, so it does not waste time to wait for the online transactions' arrival.

7.5 Performance in Different Loads

We also evaluate the performance improvement of LB-Chain under different load stresses. Intuitively, when all shards are overloaded (or underloaded), load balance will never bring any improvement. Therefore, it is worth investigating that in which load range LB-Chain will bring performance gain. In the experiment, we set $1\times$ load as the default setting described at the beginning of this section. We change the load by adjusting the transaction sending rate. The result in Table 2 shows that LB-Chain can improve the throughput on a wide range of load variety, although the improvement space becomes smaller when the load diverged from the optimal load. Furthermore, as the load imbalance problem becomes severer when the number of shard increases, LB-Chain improves performance on a wider range of load variety with the shard size expands.

TABLE 2
TPS Improvement Ratio (%) in Different Loads

Shard size	4	8	16	32
$\times$1.6 load		3.5	5.6	6.9
$\times$1.4 load		4.8	9.2	8.5
$\times$1.2 load	3.4	7.8	9.7	9.2
$\times$1 load (default)	10	13.4	11.8	11.9
$\times$0.8 load	2.1	6.7	9.8	9.5
$\times$0.6 load		1.4	4.7	4.5

8 Analysis and Discussion

8.1 Security

We first analyze the security of LB-Chain. We mainly analyze security during the account migration phase. For the rest parts of our system, since the system is based on the existing blockchain sharding system QuarkChain, we can achieve the same security guarantees as they do.

Preliminary Knowledge of QuarkChain. QuarkChain uses PoW (Proof of Work) consensus protocol and utilizes a beacon shard and multiple common shards to jointly ensure system security. Common shards are responsible for processing transactions (as described in the paper), while the beacon shard assists in the verification of cross-shard transactions (similar to Ethereum 2.0 [6]). Typically, 50% of the network-wide computing power is allocated to the beacon shard, and the remaining 50% is allocated to the common

shards, resulting in the system being able to tolerate attacks with less than 25% of malicious computing power. Readers could refer to [3] for more details.

LB-Chain guarantees security during the account migration phase. We assume that the number of malicious nodes does not exceed the maximum number that the consensus mechanism can tolerate, which is reasonable. Under such assumption, LB-Chain guarantees that: 1) The transactions will be correctly routed to the corresponding shard by the honest nodes. 2) The account migration transactions can be safely processed during account migration. First, after receiving the account allocation results broadcast by the allocation service, each node in the network will update its local routing information (e.g., distributed hash table, DHT). Therefore, when an account sends new transactions into the network, the transactions will be correctly routed to new corresponding shards. Second, the security of the account migration transaction is protected by the consensus mechanism. Only the valid account migration transactions can pass the consensus (Sec. 5.1). However, the account migration transaction is essentially a cross-shard transaction. Therefore, we need to further analyze the security of cross-shard transactions to ensure the security of the account migration transaction.

Security of Cross-shard Transactions. Since LB-Chain is developed based on QuarkChain, our cross-shard transaction processing scheme is similar to QuarkChain. Specifically, a cross-shard transaction (e.g., transfer fund from one account to another, the account migration transaction belongs to this) is split into two parts: fund withdraw and fund deposit. The withdraw of the transaction sender is executed in the source shard first. The security of this part is ensured by the consensus scheme. Then, the beacon shard verifies the withdraw part of the transaction (to finalize the withdraw part). After the verification is passed, the deposit of the transaction receiver is sent to the destination shard, and the destination shard executes the second half of the transaction through the consensus mechanism. Since the deposit part will be broadcast in the destination shards, there will be an honest node to operate the deposit sooner or later. Therefore, cross-shard transactions' atomicity is guaranteed (named eventual atomicity [35], [3]), hence ensuring the security of cross-shard transactions.

8.2 Generality

Our system has good generality. First, our protocol can work under different transaction distributions. In this paper, we use real Ethereum transaction data to evaluate the performance of LB-Chain, and the evaluation results show great performance gain. Ethereum is one of the most famous blockchain systems. Many previous works [35], [10] also perform evaluations using its transaction data. Therefore, the evaluation results measured based on the Ethereum transaction data are credible. More importantly, our protocol can be easily generalized to other systems whose transaction distribution is similar to Ethereum transaction data (i.e., a small number of hot accounts send a large number of transactions). As for those situations where account behavior and transaction distribution are different from Ethereum transactions, our system can also be modified to adapt. Specifically, in some cases where a small number of accounts do not send a large number of transactions, our system can aggregate multiple accounts together. Then our system treats the aggregated accounts as a whole and performs prediction, allocation, and migration for them. Through aggregating accounts, we can simulate the situation of a small number of accounts sending a large number of transactions. Therefore, we can achieve a higher prediction accuracy and a better load balancing result.

Second, our protocol can be generalized to different systems with various consensus mechanisms. Similar to [3], [35], our system is based on PoW consensus. Although our system is based on the PoW consensus mechanism, it can also be generalized to the BFT-type consensus systems with minor changes. This is because our account migration protocol is orthogonal to the consensus scheme design.

Third, our protocol can be generalized to consortium blockchains. Our system is designed based on the public blockchain. However, it can be easily extended to consortium blockchains and private blockchains. The security and decentralization considerations in the consortium blockchains are not as important as the public blockchain. Whereas, the performance requirements are relatively high in consortium blockchains. This is actually suitable for our design, as our main goal is to improve the performance of the blockchain system.

Finally, our mechanism could be extended to areas other than cryptocurrency in the future. Our mechanism design is based on the account model, in which each transaction is associated with a specific account. Most of the operations in our mechanism are also account-related (e.g., predicting the number of transactions sent by accounts, migrating accounts). This makes it difficult for our mechanism to be extended to the UTXO model. However, unlike UTXO, which is limited to the application of cryptocurrency, the account model has a broader application space (e.g., IoT, digital healthcare, and edge computing). Although our system is more suitable for payment scenarios because we consider only simple transactions in this work, our future work will address exactly the load balancing problem in the case of having complex smart contract transactions. Therefore, our system will have broader application scenarios in the future.

8.3 Feasibility

In this part, we discuss the feasibility of LB-Chain. The feasibility analysis will be divided into two parts, the blockchain sharding network part, and the allocation service part.

First, since our protocol is based on the existing mature blockchain sharding system (QuarkChain), *our system is highly feasible in the blockchain sharding network part.* Due to space limitation, this paper only described the proposed core scheme, account migration. For example, for the problems of node changes in the network and how nodes are allocated to shards, we can use the Cuckoo rule and distributed randomness generation scheme to solve them in the shard reconfiguration (e.g., once a day) stage [41]. Moreover, for security reasons, our system will not perform account migrations during shard reconfiguration phase. In this work, our protocol is built on QuarkChain, so we adopt QuarkChain design for the rest of the system

except for the account migration part (e.g., bootstrapping, node joining and leaving, shard reconfiguration, and cross-shard transaction processing). Because our mechanism has strong generality, it can also be used in many other sharding systems except QuarkChain.

Second, *our protocol has high feasibility in the allocation service part.* In our design, we introduced the role of allocation service (Sec. 4.2). As mentioned before, the allocation service is a third-party entity. Many existing works have also introduced the role of third-party entities in their design of blockchain systems (e.g., ordering service, smart contract service provider, interoperability service provider). Similar to them, the allocation service can be designed to be either centralized or decentralized [5], according to the specific situation. It can also be designed to be trusted [5] or even untrusted [32]. Note that due to the diversity of the allocation service described above, we do not encourage the allocation service to be responsible for too many tasks, such as requiring the allocation service to secure the system. This is mainly because, too much control by a third party can easily lead to a centralized system, which inherently deviates from the decentralized nature of the blockchain. More importantly, too much centralization can also tend to make the system less secure. However, no matter what the design is, our transaction prediction and account allocation are all centralized algorithms, which will bring additional computational overhead to the allocation service nodes running the algorithms. However, in our design, we have fully considered this issue and reduced the computational overhead as much as possible. As discussed in Sec. 6.1, in our design, only a few accounts need to be predicted each time. Besides, each prediction interval is quite long (e.g., one day), and the allocation service can predict the number of transactions in multiple epochs in one prediction. All of the above mechanisms can reduce computational overhead and provide higher feasibility for our system.

9 Conclusions

Existing blockchain sharding suffers significant performance degradation. However, little research has rigorously studied the performance in blockchain sharding systems. In this paper, we first justified that transaction load imbalance is the dominant factor in degrading system performance through measurement studies. Based on the observation, we proposed LB-Chain. This framework scales out sharding systems through account and transaction migrations among shards, which guarantees transaction load balance between shards. We have implemented LB-Chain based on QuarkChain. Extensive experiment results based on EC2 deployment and real Ethereum transactions show that LB-Chain dramatically outperforms the widely-adopted random transaction placement strategy with more balanced load, less delay, higher throughput, and better fairness among accounts. Notably, LB-Chain reduces up to 90% confirmation latency, increases the throughput by more than 10%, and improves the fairness by more than 60%.

References

[1] Jain's fairness index. https://en.wikipedia.org/wiki/Fairness_measure.

[2] K-partiton problem. https://en.wikipedia.org/wiki/Partition_problem.

[3] Quarkchain. https://quarkchain.io.

[4] M. J. Amiri, D. Agrawal, and A. El Abbadi. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1347. IEEE, 2019.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[6] V. Buterin. Ethereum 2.0 spec–casper and sharding, 2018. *Available [online].[Accessed: 30-10-2018]*.

[7] V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[8] G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable cities and society*, 39:283–297, 2018.

[9] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895*, 2015.

[10] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*, pages 123–140, 2019.

[11] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. El Abbadi. Squall: Fine-grained live reconfiguration for partitioned main memory databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 299–313, 2015.

[12] K. Francisco and D. Swanson. The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. *Logistics*, 2(1):2, 2018.

[13] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

[14] Z. Hong, S. Guo, P. Li, and W. Chen. Pyramid: A layered sharding blockchain system. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[15] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan. Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding. *IEEE Internet of Things Journal*, 8(6):4291–4304, 2020.

[16] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE INFOCOM*, 2022.

[17] M. A. Khan and K. Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[18] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[19] M. Król, O. Ascigil, S. Rene, A. Sonnino, M. Al-Bassam, and E. Rivière. Shard scheduler: object placement and migration in sharded account-based blockchains. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 43–56, 2021.

[20] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado. Blockchain distributed ledger technologies for biomedical and health care applications. *Journal of the American Medical Informatics Association*, 24(6):1211–1220, 2017.

[21] K. Li, Y. Tang, J. Chen, Z. Yuan, C. Xu, and J. Xu. Cost-effective data feeds to blockchains via workload-adaptive data replication. In *Proceedings of the 21st international middleware conference*, pages 371–385, 2020.

[22] Y. Liu, J. Liu, J. Yin, G. Li, H. Yu, and Q. Wu. Cross-shard transaction processing in sharding blockchains. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 324–339. Springer, 2020.

[23] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, 2016.

[24] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[25] L. N. Nguyen, T. D. Nguyen, T. N. Dinh, and M. T. Thai. Optchain: optimal transactions placement for scalable blockchain sharding.

In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 525–535. IEEE, 2019.

[26] O. Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5(2):1184–1195, 2018.

[27] N. Okanami, R. Nakamura, and T. Nishide. Load balancing for sharded blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 512–524. Springer, 2020.

[28] L. Ren and P. A. Ward. Understanding the transaction placement problem in blockchain sharding protocols. In *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0695–0701. IEEE, 2021.

[29] P. Ruan, T. T. A. Dinh, D. Loghin, M. Zhang, G. Chen, Q. Lin, and B. C. Ooi. Blockchains vs. distributed databases: Dichotomy and fusion. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1504–1517, 2021.

[30] M. Serafini, R. Taft, A. J. Elmore, A. Pavlo, A. Aboulnaga, and M. Stonebraker. Clay: fine-grained adaptive partitioning for general database schemas. *Proceedings of the VLDB Endowment*, 10(4):445–456, 2016.

[31] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*, pages 105–122, 2019.

[32] J. Sousa, A. Bessani, and M. Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 51–58. IEEE, 2018.

[33] M. Swan. *Blockchain: Blueprint for a New Economy*. O'Reilly, 2015.

[34] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proceedings of the VLDB Endowment*, 8(3):245–256, 2014.

[35] J. Wang and H. Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 95–112, 2019.

[36] Y. Wang, Q. Zhang, K. Li, Y. Tang, J. Chen, X. Luo, and T. Chen. ibatch: saving ethereum fees via secure and cost-effective batching of smart-contract invocations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 566–577, 2021.

[37] S. Woo, J. Song, S. Kim, Y. Kim, and S. Park. Garet: improving throughput using gas consumption-aware relocation in ethereum sharding environments. *Cluster Computing*, pages 1–13, 2020.

[38] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

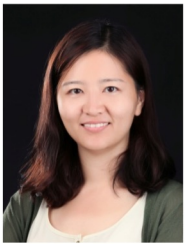
[39] K. Wüst, S. Matetic, S. Egli, K. Kostiainen, and S. Capkun. Ace: Asynchronous and concurrent execution of complex smart contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–600, 2020.

[40] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han. When mobile blockchain meets edge computing. *IEEE Communications Magazine*, 56(8):33–39, 2018.

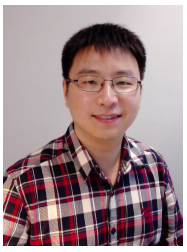
[41] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[42] M. Zhang, J. Li, Z. Chen, H. Chen, and X. Deng. Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 358–367. IEEE, 2020.

Mingzhe Li is currently a Ph.D. candidate with Department of Computer Science and Engineering, Hong Kong University of Science and Technology, and Southern University of Science and Technology. He received his B.E. degree in communication engineering from Southern University of Science and Technology in 2016. His research interests are mainly in blockchain sharding, consensus protocol, blockchain application, network economics, and crowdsourcing.



Jin Zhang is currently an assistant professor with Department of Computer Science and Engineering, Southern University of Science and Technology. She received her B.E. and M.E. degrees in electronic engineering from Tsinghua University in 2004 and 2006, respectively, and received her Ph.D. degree in computer science from Hong Kong University of Science and Technology in 2009. Her research interests are mainly in mobile healthcare and wearable computing, wireless communication and networks, network economics, cognitive radio networks and dynamic spectrum management.



Wei Wang is currently an assistant professor with Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto in 2015. Prior to that, he obtained the B.Eng. and M.Sc. degrees from Shanghai Jiao Tong University. He is also affiliated with HKUST Big Data Institute. His research interests cover the broad area of networking and distributed systems, with a special focus on big data and machine learning systems, cloud computing, and computer networks.