On Sharding Open Blockchains with Smart Contracts

Yuechen Tao*, Bo Li*, Jingjie Jiang†, Hok Chu Ng*, Cong Wang‡, Baochun Li§
* Hong Kong University of Science and Technology, † Future Network Theory Lab, Huawei, ‡City University of Hong Kong, § University of Toronto
{ytaoaf, bli, hcngac}@cse.ust.hk, j@introo.me, congwang@cityu.edu.hk, bli@ece.toronto.edu

***Abstract*—Current blockchain systems suffer from a number of inherent drawbacks in its scalability, latency, and processing throughput. By enabling parallel confirmations of transactions, sharding has been proposed to mitigate these drawbacks, which usually requires frequent communication among miners through a separate consensus protocol.**

In this paper, we propose, analyze, and implement a new distributed and dynamic sharding system to substantially improve the throughput of blockchain systems based on smart contracts, while requiring minimum cross-shard communication. Our key observation is that transactions sent by users who only participate in a single smart contract can be validated and confirmed independently without causing double spending. Therefore, the natural formation of a shard is to surround one smart contract to start with. The complication lies in the different sizes of shards being formed, in which a small shard with few transactions tends to generate a large number of empty blocks resulting in a waste of mining power, while a large shard adversely affects parallel confirmations. To overcome this problem, we propose an inter-shard merging algorithm with incentives to encourage small shards to merge with one another and form a larger shard, an intra-shard transaction selection mechanism to encourage miners to select different subsets of transactions for validation, as well as a parameter unification method to further improve these two algorithms to reduce the communication cost and improve system reliability.

We analyze our proposed algorithms using the game theoretic approach, and prove that they converge to a Nash Equilibrium. We also present a security analysis on our sharding design, and prove that it resists adversaries who occupy at most 33% of the computation power. We have implemented our designs on go-Ethereum 1.8.0 and evaluated their performance using both real-world blockchain transactions and large-scale simulations. Our results show that throughput has been improved by 7.2×, and the number of empty blocks has been reduced by 90%.

I. Introduction

With better security, greater transparency and improved traceability, blockchain systems attracted a significant amount of attention in both academia and industry [1]–[3]. They served as foundations for cryptocurrencies (e.g., Bitcoin [4] and DogeCoin [5]), and enabled distributed applications by using a peer-to-peer architecture rather than traditional client-server models (e.g., Ethereum [6], Namecoin [7] and IPFS [8]). However, such systems usually suffer from poor performance, particularly in terms of throughput, where it is not uncommon that users wait for excessively long periods of time for the confirmation of transactions [7]. With Bitcoin, only 7 transactions sent from users can be confirmed per second, while Visa is able to confirm 24,000 transactions per second [9].

The root cause for such a low throughput lies in the fact that every miner in blockchain systems validates the same set of transactions simultaneously. Miners can receive rewards in the form of transaction fees. When a block is confirmed, the creator of the block can receive the fees from the block's transactions. Therefore, miners all prefer to validate transactions with higher fees. If we assume that several transactions arrive at the same time, all the miners will sort these transactions in the same order according to the transaction fees and generate blocks that confirm the same set of transactions. In other words, the confirmation of transactions is serialized.

To tackle this problem, sharding is proposed to validate and confirm different sets of transactions in parallel [10]–[16] by dividing the network into multiple small groups. Transactions from different shards are validated by different miners simultaneously. However, since the validation of transactions may need transaction records in multiple shards, frequent cross-shard communication is required. For example, assume that there is a transaction between user A and B who ever conducted transactions in different shards, i.e., miners in these two shards only have parts of the transaction records. Different from the non-sharding systems, miners must communicate with each other to exchange their individual validation results to validate the transaction jointly. Different sets of new consensus protocols [12]–[18] were proposed to handle such cross-shard communication among miners from different shards, where per-shard leaders exchange per-shard validation results with each other on behalf of the miners in corresponding shards. Such mechanisms inevitably lead to heavy communication overhead.

In this paper, we propose a completely *distributed* sharding system that not only eliminates cross-shard communication, but also eliminates the need for new consensus protocols during the transaction validation process. Account-based blockchain systems, like Ethereum [6] and Hyperledge [10], support *smart contracts* recording the conditions under which certain transactions can happen. Our key observation is that if users only participate in one smart contract on the

The research was support in part by RGC GRF grants under the contracts 16206417 and 16207818, Guangzhou Development Zone Project Grant 2017GH23.

same blockchain, the transactions sent by these users can be validated and confirmed independently and in isolation without causing double spending. We propose to form one shard for those transactions whose senders are only involved in one smart contract. Miners use a random selection algorithm to decide which shard they belong to, where an honest miner can verify whether others are cheating on their identities to ensure system consistency.

Nevertheless, forming each shard corresponding to a single smart contract can be problematic. A small shard with too few transactions tends to generate a large number of empty blocks, which wastes mining power. On the other hand, if a shard is too large and occupies the majority of transactions, the number of shards and the number of parallel confirmations are reduced, which adversely affects the throughput.

To avoid empty blocks in small shards, we propose an inter-shard merging algorithm. We give miners in small shards an extra reward if they merge and form a larger shard. To achieve this, we model the miners' behavior as an evolutionary cooperative game process. In this process, during each iteration, miners will exchange their decisions on whether to merge with others, based on which they revise their selections until the Nash Equilibrium point is achieved. We obtained all the possible Nash Equilibria under different conditions through theoretical analysis. Based on this, we propose a gaming strategy as our inter-shard merging algorithm and prove that this algorithm will achieve at least one equilibrium.

To avoid marginal improvements on throughput in large shards, we propose an intra-shard transaction selection algorithm. Miners in a large shard face a significant amount of conflicts when selecting transactions, which negatively affects the throughput of the system. To optimize the system, we let miners select different sets of transactions through a congestion game [19]–[22]. We incorporate the best-reply strategy in this congestion game as our intra-shard transaction selection algorithm to achieve a pure strategy Nash Equilibrium according to the analysis in [22], [23].

However, during such a shard merging process, heavy cross-shard communication appears. Further, miners are assumed to be selfish and honest, yet system security may be affected if such an assumption does not hold, i.e., malicious nodes exist. We propose a new parameter unification scheme to handle these two problems at the same time. To solve the security problem, we force miners to follow our algorithms by enabling miners to verify whether others work in a gaming manner, and then reject blocks packed by those rule-breakers. We prove that if miners have identical input parameters of these algorithms, such a verification process can be executed through running these algorithms locally. After these algorithms are locally executed, all the miners will know how transactions are selected and how shards are merged, which also relieve miners from frequent communication. Similar to Omniledger [13], those input parameters are generated by verifiable leaders who are selected randomly with the VRF [24] algorithm.

Our original contributions in this paper are as follows. *First*, we propose a novel distributed sharding system that improves system throughput without incurring new consensus protocols for frequent cross-shard communication. *Second*, we design an inter-shard merging algorithm to dynamically merge small shards, an intra-shard transaction selection scheme to let miners select the best sets of transactions that maximize both their profits and system throughput at the same time, and a parameter unification technique to further reduce the communication cost without sacrificing security. *Finally*, through theoretical analysis, we prove that our inter-shard merging algorithm converges to a mixed strategy Nash Equilibrium, and our parameter unification scheme offers 33% attack resilience.

We have implemented our sharding system on top of go-Ethereum 1.8.0 and evaluated its performance using both real-world transactions and large-scale simulations. Our experimental results have shown that system throughput has increased by 7.2× with only nine shards. With our shard merging algorithm, the percentage of empty blocks decreases by 90% with only a 14% decrease in throughput improvement. With the intra-shard transaction selection algorithm, the system throughput is further improved by 3×.

II. Background and Motivation

A. Roles in a Blockchain System

In a blockchain system, users broadcast transactions to the network, and miners are responsible for validating those transactions according to all the transaction histories. Confirmed transactions, which are often referred to as *states*, are packed to blocks by miners. Blocks are recorded by all the miners locally in the form of linked lists, called *ledgers*. The entire network can be split into multiple small groups, called *shards* [12]–[18]. Miners in each shard maintain disjoint transaction histories and ledgers of the network. In this way, transactions in different shards are validated by different miners in parallel.

A smart contract records a transaction and the conditions under which that transaction is valid. For instance, user A can enforce a contract to transfer 2 ETH to user B if B's balance is below 1 ETH. A smart contract account will record this potential transaction and this condition. To incorporate this smart contract, a new transaction is conducted between user A and that smart contract account [6], rather than directly between users. Miners will verify whether the account balance of user B is less than 1 coin. If so, miners will confirm the transaction between user A and that smart contract, and record the balance change of user A and B in their local ledgers. With the proliferation of smart contracts (over 1.7 million smart contracts by May 2018 [25]), contract-based transactions are becoming dominant. In Ethereum, each of the top ten smart contracts has 2,998,533 transactions on average, and the most popular smart contract has 1,035,4398 transactions [26] [27].

B. Revisiting the Low Throughput of Blockchain Systems

In current-generation blockchain systems, miners validate the same set of transactions simultaneously. Such serialization of transaction confirmation is the root cause of the low throughput in blockchain systems. Miners in a blockchain system keep track of unvalidated transactions. Each time a

TABLE I
CONFIRMATION TIME WITH DIFFERENT NUMBER OF MINERS.

Number of miners	2	3	4	5	6	7
Confirmation time (sec)	218	194	113	120	103	121

miner successfully confirms a transaction, she will be rewarded a certain number of coins, namely transaction fees. To achieve maximal profits, miners always select transactions with the highest fees from those unvalidated transactions. Following this principle, all the miners are likely to select the same set of unvalidated transactions. As a result, transactions with the highest transaction fees are likely to be confirmed first before the whole network moves on to the next set of transactions. In other words, more miners do not necessarily imply higher throughput. We injected 20 transactions into go-Ethereum 1.8.0 with the settings used in Sec. VI-B1. As shown in Table I, the time it takes to confirm a transaction does not decrease as the number of miners keeps increasing beyond four.

Sharding improves the throughput of blockchain systems by validating transactions in parallel. However, randomly sharding the entire blockchain requires frequent cross-shard communication. For example, user A with an initial balance of 10 ETH transferred 8 ETH to user B, and this transaction is recorded in shard 1. If A now tries to transfer 3 ETH to user C in shard 2, miners in shard 2 would accept and confirm this invalid transaction if they fail to communicate with miners in shard 1. To validate such cross-shard transactions, miners in related shards must exchange their validation results and make joint decisions. This not only incurs a tremendous amount of communication times, but also jeopardizes system consistency if such communication fails.

Furthermore, as miners in the original Blockchain systems do not need to communicate when validating transactions, a separate consensus protocol is needed to define how such cross-shard communication should be conducted. Existing cross-shard consensus protocols require per-shard leaders to exchange validation results [13]–[16] among shards.

To summarize, an ideal sharing system should require minimum cross-shard communication, and should make different shards operate independently most of the time.

C. Data Irrelevancy in Smart Contracts

We observe that in blockchain systems with smart contracts, there are three types of senders. First, in Fig. 1(a), user A only sends transaction 1 to user B through contract 1. Second, in Fig. 1(b), transaction 2 and 3 are sent by user C. Contract 2 records the conditions when transaction 3 can happen. Similarly, transaction 2 will be confirmed as valid if the conditions recorded in contract 1 are satisfied. Third, in Fig. 1(c), user F sends transaction 4 to user G through contract 1, and transaction 5 to user H directly. Since user A only participates in one smart contract and does not conduct transactions with users directly, transaction 1 can be validated only with the transaction records of user A stored in contract 1. Conversely, other transactions in Fig. 1 cannot be validated only with the transaction records stored in contract 1. Generally speaking, *contract-based transactions sent by users that only participate in one smart contract can be validated and confirmed by a subset of miners that know the transaction history of the related contract.* Other miners can validate and confirm the remaining transactions at the same time without sacrificing system consistency. However, in the current blockchain design, such transactions are validated by all the miners in a serialized manner, which unnecessarily wastes computation power and exacerbates system throughput.

III. DISTRIBUTED SHARDING

We now present our distributed sharding design. We first introduce how to divide transactions and miners into different shards in a distributed fashion, and then introduce how users and miners work in our sharding system. Finally, we will analyze why we need our inter- and intra- shard algorithms.

A. Transactions and States Sharding

Based on our observations in Sec. II-C, transactions sent by users who only participate in the same smart contract naturally form a shard, and miners in this shard only record transactions happening in the shard. Noticing that there are some users who participate in more than one contract or have directly sent transactions to other users. Transactions sent by these users form a unique shard, called the *MaxShard*, and miners in the *MaxShard* record all the transactions in the system to validate transactions sent from these users.

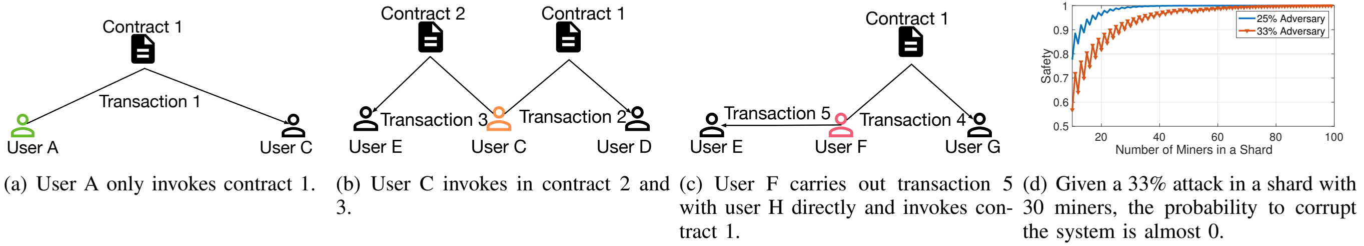
With such a formation of shards, transactions in different shards are validated independently. Miners in different shards mine in parallel without any cross-shard communication.

B. Miner Sharding

How do we securely assign miners to shards? First, permitting miners to choose shards is insecure since malicious nodes may focus on one shard to corrupt. Therefore, we need a source of randomness with which a shard has the same fraction of malicious nodes with the system. Second, the fraction of miners in a shard shall keep up with the fraction of transactions in that shard. For example, *MaxShard* may contain more transactions than other shards, thus more miners are required to validate transactions in the *MaxShard*.

Several schemes have been proposed to separate miners into shards evenly. In Omniledger [13], a verifiable leader is selected among miners with the VRF algorithm [24]. Then that leader generates the randomness. Which shard a miner belongs to is calculated with her public key and the randomness through the RandHound [28] algorithm. In this way, the separation result can be checked by everyone in the system.

To overcome the second point on distributing miners based on the fractions of transactions, we revise the proposal in Omniledger [13] as follows. Apart from generating and broadcasting the randomness, the verifiable leader is also responsible for requesting and broadcasting the fractions of transactions in different shards from miners in *MaxShard*. A miner m will know which shard she belongs to according to the following algorithm. She first sorts all the shards based on



(a) User A only invokes contract 1.

(b) User C invokes in contract 2 and 3.

(c) User F carries out transaction 5 with user H directly and invokes contract 1.

(d) Given a 33% attack in a shard with 30 miners, the probability to corrupt the system is almost 0.

Fig. 1. (a)(b)(c): A motivating example: data irrelevancy in smart contracts. Only transaction 1 sent by A through contract 1 can be validated and confirmed independently. (d): The security analysis of our miners separation mechanism.

the received fractions of transactions from the leader. $\beta_i\%$ represents the fraction of transactions in the ith shard, and i is the *ShardID*. Then she runs the RandHound algorithm [28] with which miners are separated to 100 groups evenly, and obtains a random number r ranging from 1 to 100. If $r \in [\sum_{i=1}^{s-1} \beta_i, \sum_{i=1}^{s} \beta_i]$, she is in shard s. Obviously, users can verify whether a miner is in shard s with this algorithm given that miner's public key, the randomness, as well as the fractions of transactions received from the verifiable leader.

The security analysis of our miner separation mechanism is identical to Omniledger [13]. Under the PoW consensus algorithm, Fig. 1(d) illustrates the security of a shard with a different number of miners for 25% and 33% adversaries. Obviously, a shard with more miners is harder to be corrupted.

C. Workflow in the Distributed Sharding System

When a miner receives a transaction, she first figures out whether the sender of that transaction is only involved in the current shard. If so, the miner will generate and broadcast a block whose body contains that transaction and whose header contains the current *ShardID*. There are several solutions to verify whether the sender only incorporates the current smart contract. Trivially, since miners in the *MaxShard* record all the transactions in the system, they can get the answer through checking the local states of the system and then broadcast such information to others. This will surely incur heavy query cost. A more elegant way is to let miners maintain the call graph [29] among smart contracts and users locally. In this way, miners can check the call graph instead of remotely referring to the whole history. Obviously, how to get such information of a sender is pluggable in our sharding. Our future work focuses on the design of the call graph to further improve the efficiency of our system.

When a miner X receives a block packed by another miner Y, she needs to perform two verifications to ensure the security of the system. First, X verifies whether Y really corresponds to the *ShardID* in the block header. If Y cheats on her shard, X will find that and reject the block. Then, X checks whether she is in the same shard with Y through the *ShardID* in the block header. If so, Y will record that block locally.

The system's workflow is shown in Fig. 2. User x sends transaction 1 and 2 through contract 1 and contract 2. User y sends transaction 3 through contract 1, and user z sends transaction 4 through contract 2. These transactions are broadcast

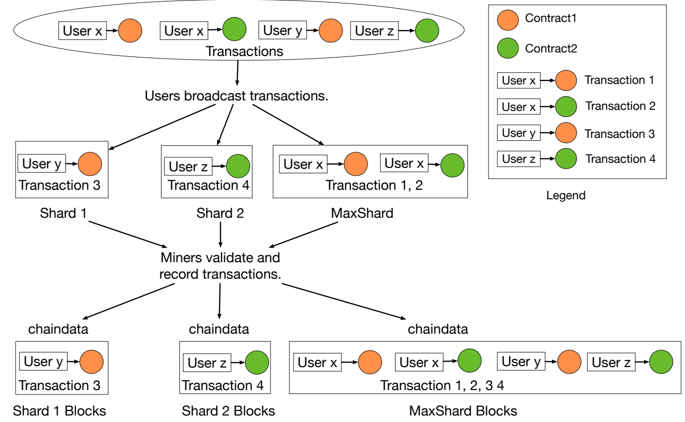


Fig. 2. Overview: our distributed sharding system. User x sends transaction 1 and 2. User y sends transaction 3. User z sends transaction 4. These transactions will be validated and recorded in different shards.

to all miners in the same blockchain network. A miner verifies whether the incoming transactions belong to her shard using the principle described in Sec. III-A. Miners in each shard validate the transactions by referring to the local ledgers. After validation, miners pack valid transactions to blocks, modify local ledgers, and broadcast the blocks. After receiving new blocks belong to their shards, miners will update the local ledgers accordingly.

D. The Necessity for Inter- and Intra- Shard Algorithms

Define the size of a shard as the number of transactions in that shard, shard sizes vary significantly based on users' activities in each smart contract. Neither too small nor too large shards are preferred in the sharding system.

If a shard is too small, empty blocks will appear. Existing research has illustrated that there is a substantially large number of empty blocks in todays blockchain systems, e.g., 20% percent Ethereum, and 19% empty blocks in Bitcoin [30], [31]. In order to enable decentralized blockchain systems, miners are awarded if they participate in the transaction validation process. When a block submitted by a miner is added to the blockchain, this miner can receive a *block reward* and the transaction fees corresponding to transactions in that block. Further, even if the block does not contain any transactions, that miner can still get the block reward. Based on such an incentive mechanism, if there are no unvalidated transactions, a miner will pack empty blocks to maximize her profit. Packing an empty block is no easier than a normal block. A great

amount of mining power is thus wasted. Therefore, we need a dynamic inter-shard algorithm to merge small shards and avoid empty blocks.

On the other hand, a large number of transactions might appear in a single shard. In the extreme case, the system only contains one large shard and one small shard, the majority of transactions are validated and confirmed in a serialized manner if all miners in the large shard choose to pack the most profitable transactions. This limits the potential for throughput improvement. Therefore, we need an intra-shard transaction selection scheme that drives miners to select different sets of transactions.

IV. Inter- and Intra- Shard Algorithms

In this section, we first propose a dynamic and distributed inter-shard merging algorithm for small shards to avoid wasting mining power. Second, we introduce an intra-shard transaction selection algorithm for miners in big shards to further improve the throughput. Third, we present our parameter unification method to ulteriorly minimize the communication cost and enhance the security of our system. Finally, we give the security analysis of our two algorithms.

A. Inter-Shard Merging Algorithm

To reduce the number of empty blocks, we encourage miners in small shards to merge with others by embracing an incentive mechanism. Under this mechanism, we model the merging behavior of miners as a cooperative game process and propose an iterative inter-shard merging algorithm.

1) Objective and Incentive for Merging: We first discuss the objective of our merging process. If the number of unvalidated transactions is larger than 0 at any time, miners can earn more money by validating transactions than packing empty blocks. Therefore, the size of the newly formed shard must satisfy the following constraint,

$$T \geq L, \tag{1}$$

where T is the size of a shard, L is the lower bound. Our objective is to make miners merge so that the size of the newly formed large shard satisfies (1).

However, miners might lose some transaction fees after merging, because there will be more miners competing on validating transactions in the new shard. Therefore, an incentive mechanism is needed to let miners merge with others.

The incentive is given in the form of coins, called the *shard reward*. The rule of distributing the *shard reward* is **if the size of the new shard satisfies (1), *all* the miners in small shards can get the same *shard reward*.** Like the *block reward*, the *shard reward* is also transferred to miners' accounts by the system. Further, our parameter unification scheme in Sec. IV-C ensures that the merging result is verifiable by all the miners.

2) Cooperative Game: Given the incentive mechanism, miners' merging behavior is a cooperative game process. For example, assume miners in shards C, D and E decide to merge at first. Their expected payoffs are the *shard reward*. After exchanging their choices between each other, those miners find that (1) can be satisfied even only two of them merge. Therefore, all of them will rely on others to form the new shard and stay in their shards. In this way, their expected payoff will change to *0*.

This is surely a gaming process, where miners may change their decision based on others' choices. However, a new shard must be stable for the sake of the consistency of the system. We then propose Algorithm 3 to form a stable shard, i.e. to achieve the equilibrium state. Through the theoretical analysis of miners' merging behavior in Sec. V, we obtain the conditions for achieving the mixed strategy equilibrium in this scenario. Algorithm 3 satisfies such equilibrium conditions, and thus serves as the strategy for miners.

However, Algorithm 3 only produces **one** stable shard. Therefore, we need to do the merging process iteratively as illustrated in Algorithm 1. First, miners in small shards cooperate with each other to form one new shard whose size is larger than L. Then miners in the remaining small shards come to the next round to form another new shard until the remaining shards cannot form a shard satisfying (1).

Algorithm 1: Iterative Merging Algorithm

Input: The number of shards, the size of the small shards, and the *shard reward*

1 **for** *Remaining small shards who can form a new shard* **do**

2 Miners in small shards use Algorithm 3 to merge a new shard with the *shard reward*.

The complexity of Algorithm 1 is as follows. Denote the number of small shards as S. There are at most $\frac{S}{2}$ new shards after merging. In other words, Algorithm 3 will be performed at most $\frac{S}{2}$ times. According to Sec. V-B, the complexity of Algorithm 3 is $\mathcal{O}(M * \log \frac{1}{E}))$, where M is the number of subslots, and E is the desired error value. Therefore, the time complexity of Algorithm 1 is $\mathcal{O}(S \log \frac{1}{E} M)$.

B. Intra-Shard Transaction Selection Algorithm

For shards that contain too many transactions, we further design a distributed intra-shard transaction selection method to improve the throughput of large shards by letting miners select different sets of transactions. First, we model the miners' behavior in selecting transactions as a *congestion game*. Then, we give a transaction-selection algorithm which will achieve equilibrium in this gaming process.

Miners validate transactions because of the transaction fees. Given such an incentive, we can model the miners' transaction selection process as a congestion game process [19]–[22]. In a congestion game, there are limited resources, and the expected profits a player can get differ based on the resources one selects and the number of players choosing the same resources.

For example, only miner X selects a transaction with the highest transaction fee at first. In this way, her expected profit is the transaction fee of that transaction. After miners exchange their choices, others find that only miner X selects the most expensive transaction, so others will select that transaction to improve the expected profit. Then miner X is most likely to get

zero transaction fee, because almost all the miners compete for this transaction at the same time. In other words, the profit of a miner changes either she or others select different transactions.

Therefore, we let miners select different sets of transactions through this gaming process with Algorithm 2. According to [22] [21], our transaction selection game will reach the pure strategy Nash Equilibrium state since all the miners follow the best-reply strategy. The complexity of the best-reply strategy has been proved to be $\mathcal{O}(uT^2)$ in past researches [32]–[35], where u is the number of miners and T is the number of transactions.

Algorithm 2: Transaction-Selection Algorithm

Input: The transactions set, the miners set, and the initial transaction set selected by each miner

1 **while** *Some miner can get a higher expected profit* U_i *by selecting transaction* σ_i **do**

2 Pick a miner i who can improve her expected profit by selecting transaction σ_i, that is $U_i(\sigma_i) \geq U_i(j)$.

3 Select transaction σ_i for miner i.

The miners' expected profit U_i used in Algorithm 2, which is modeled as the utility function in the congestion game process, is calculated as follows. Assume there are u miners and T transactions in a shard. The strategy vector of all the miners is $\{\sigma_1, \sigma_2, ..., \sigma_u\}$, the number of miners who choose the transaction j is n_j, and $j \in \{1, 2, ..., T\}$. The transaction fee of the jth transaction is f_j. The expected payoff of miner i selecting transaction j is,

$$U_{i,j} = \tfrac{f_j}{n_j+1}. \quad (2)$$

C. Parameter Unification

First, we claim that we still need to reduce the communication frequency among miners and improve the system security. We then introduce the parameter unification mechanism to solve these two problems at the same time.

In Algorithm 1 and 2, miners need to exchange their choices for several iterations to achieve the final decision on shards merging and transaction selection. Considering the number of transactions and miners is huge in the real implementations, it will be costive for miners to communicate with each other.

Further, the system may be attacked by malicious nodes. This is because miners cannot verify whether others do merging or transaction selection jobs with our algorithms. In this way, the adversary can incorporate multiple malicious nodes and let them corrupt specific shards or transactions together. On the contrary, if the verification of merging or transaction selection is enabled, the adversary can hardly corrupt any shards or transactions. For example, suppose miner A knows that a transaction should not be assigned to miner C. If malicious miner C packs a block containing that transaction, miner A can do verification and reject that block.

To enable miners to verify whether others work using our algorithms, we incorporate the following parameter unification method. We just let all the miners have identical input parameters of these two algorithms, which are others' random initial choice, the miners set, and the shards set or the transactions set. We now argue that in this scenario, such verification can be enabled. In this scenario, it is obvious that all the miners will get the same output representing the merging results by executing Algorithm 1, which also holds for Algorithm 2. If honest ones compare others' merging or transaction selection behavior with that output, they can find whether others are cheating on which shard to merge or which transaction to validate. The blocks generated by those liars who do not follow Algorithm 1 or Algorithm 2 will be rejected by honest ones. Further, since miners can execute Algorithm 1 and Algorithm 2 locally, communication does not exist among miners. In conclusion, miners are enforced to follow our algorithms without communication overhead using our parameter unification method.

To unify the input parameters in Algorithm 1 and Algorithm 2, we use the same mechanism discussed in Sec. III-B on generating randomness by verified leaders in Omniledger [13]. In our design, the verifiable leader generates the others' random initial choices, and broadcasts it with the current miners set, the shards set or the transactions set to the whole network.

D. Security Analysis

We will analyze the security of our two algorithms with the parameter unification scheme separately in this section. We assume an infinite pool of malicious nodes, and we use the binomial distribution to model the number of malicious nodes in a single shard. These are practical due to the unpredictability of the leader selection process.

During the inter-shard merging process, the adversary can only corrupt the newly formed shard when she is the leader, and the new shard has more than $\frac{1}{2}$ fraction malicious nodes at the same time. So she must be the leader for several continuous times until there are enough malicious nodes in the new shard. The probability that the newly formed shard is corrupted is,

$$\textstyle\sum_{k=0}^{l} f^k \cdot (1 - P_s), \quad (3)$$

where the adversary has f fraction computation power, P_s the probability that a single shard is not corrupted analyzed in Sec. III-B, l is the consecutive rounds that the adversary controls. Based on (3), when $l \to \infty$, given a 25%-adversary, the failure probability of our inter-shard merging algorithm is $8 \cdot 10^{-6}$.

During the intra-shard transaction selection process, we assume that the transaction fees obey the binomial distribution. Given this assumption, the probability of selecting a transaction with t coins of transaction fee is,

$$P_t = \begin{pmatrix} t \\ N \end{pmatrix} \cdot (\tfrac{1}{2})^N, \quad (4)$$

where N is the total transaction fees. Similar to the analysis on the single shard security, the probability of corrupting a single transaction i is,

$$P_i = P\left[c > \left\lfloor \frac{n}{2} \right\rfloor\right] = \textstyle\sum_{k=0.5n}^{n} \begin{pmatrix} n \\ k \end{pmatrix} f^k (1-f)^{n-k}, \quad (5)$$

where *n* is the number of miners on that transaction. To corrupt a transaction, the adversary must control a leader, and generate a randomness with which enough malicious nodes can validate that transaction together. Similarly, the adversary must control the leader repeatedly and continuously until she generates that randomness. Therefore, the probability that the system is corrupted with our intra-shard algorithm is,

$$\sum_{k=0}^{l} f^k \cdot \sum_{t=1}^{T}(P_i \cdot P_t), \tag{6}$$

where *T* is the total number of transaction fees. When $l \to \infty$, with a 25%-adversary and 200 transaction fees in total, the corruption probability is $7 \cdot 10^{-7}$.

V. A Game Theoretic Perspective

In this section, we analyze miners' merging behavior from a cooperative game theory perspective. First, we theoretically formulate the merging game in which miners decide to merge with others. Second, we analyze the sufficient and necessary conditions when the merging game achieves the mixed strategy Nash Equilibrium. Finally, we propose a distributed algorithm for the miners to make merging decisions which will converge to an equilibrium.

TABLE II
Notations

y_m	The size of the new shard merged by m shards.
x_i	Replicator dynamics of player i.
x_{-i}	Replicator dynamics vector of players except i
G	The *shard reward*.
C_i	Cost of player i for merging with others.
L	The minimum size of the newly formed shard.
$U_{i,j}$	The expected payoff of i selecting transaction j.
$U_{Y,i}$	The payoff of player i if she merges with others.
$U_{N,i}$	The payoff of i if she doesn't merge with others.
$\bar{U}_{Y,i}(x_i)$	The average payoff of player i if she merges.
$\bar{U}_{N,i}(x_i)$	The average payoff of i if she doesn't merge.
$\bar{U}_i(x_i)$	Average payoff of i using mixed strategy.

A. Formulation of the Merging Game

For simplicity of notations, we let player i represent miners in shard i. Assume that there are m players merging among N players. The number of transactions in the newly formed shard is denoted as y_m. Denote that the number of transactions in these shards as $c_1, c_2, ..., c_m$, respectively. The total number of transactions in the new shard is:

$$y_m = \sum_{i=1}^{m} c_i. \tag{7}$$

Recall the incentive mechanism, if y_m, which is the size of the newly merged shard, is larger than L, all the players will get the shard reward G. Further, merging with others can lose some profits, noted as C. Therefore, if player i merges with others, the utility function is:

$$U_{Y,i} = Pr(y_m > L) * G - C_i, \forall i \in [0, N], \tag{8}$$

where C_i is the merging cost of player i, and $Pr(y_m > L)$ is the probability if the number of transactions for the newly formed shard is larger than the lower bound.

There is no cost for the staying players. The utility function for them is

$$U_{N,i} = \begin{cases} Pr(y_m > L) * G, \forall m \in [1, N] \\ 0, \text{ if } m = 0. \end{cases} \tag{9}$$

We use the definition of the evolutionarily stable strategy (ESS) in [36] [37] to represent the player' final stable states. A strategy a^* is an ESS if and only if, $\forall a \neq a^*$, a^* satisfies:

- equilibrium: $U_i(a, a^*) \leq U_i(a^*, a^*)$,
- stability: if $U_i(a, a^*) = U_i(a^*, a^*), U_i(a, a) < U_i(a^*, a)$,

where $U_i(a_1, a_2)$ is the utility of player i when she uses strategy a_1 and another player uses strategy a_2.

Since all the players are selfish, they will cheat if cheating can improve their payoffs, since all the players are uncertain of other players' actions and utilities. In such a case, to improve their utilities, the players will try different strategies in every play and learn from the strategic interactions using the methodology of understanding-by-building. During the process, the percentage of players using a certain pure strategy may change. Such a population evolution can be modeled by replicator dynamics. Specifically, let x_a stand for the probability of a player using pure strategy $a \in \mathcal{A}$, where $\mathcal{A} = 0, 1$ representing whether to merge with others. By replicator dynamics [38], the evolution dynamics of x_a are given by the following differential equation:

$$\dot{x_a} = \eta[\bar{U}(a, x_{-a}) - \bar{U}]x_a, \tag{10}$$

where $\bar{U}(a, x_{-a})$ is the expected payoff of a player using pure strategy a, x_{-a} means others use strategies except a, $\bar{U}$ is the expected payoff of all players, and η is a positive scale factor.

The sufficient and necessary conditions that the mixed strategy Nash Equilibrium is achieved are analyzed in the technical report [39].

B. Algorithm to the Nash Equilibrium

In this part, we present the distributed inter-shard merging algorithm by iteratively updating the possibilities of whether to join others to converge to the final equilibrium.

Since $\dot{x}_i = 0$ is the sufficient and necessary condition for the system's equilibrium, we just need to converge to the solution of the replicator dynamics to get the final equilibrium. Hence, we can use the classical gradient descent algorithm [37] to reach the solution iteratively.

The discretized dynamic replicator is

$$x_{i,a_i}(t+1) = x_{i,a_i}(t) + \eta[\bar{U}_i(a_i, x_{-i}(t)) - \bar{U}_i(x_i(t))]x_{i,a_i}(t), \tag{11}$$

where t is the slot index and $x_{i,a_i}(t)$ is the probability of player i using strategy $a_i \in \mathcal{A}$ at slot t. To make the calculation more accurate, we divide each slot to S subslots and each player can choose whether to merge with others in each subslot.

Based on (11), we give the algorithm in Algorithm 3. Miners in different small shards only need to exchange the statistic data at the end of each slot t.

The complexity of Algorithm 3 is as follows. Since this is a iterative algorithm, denote E as the final error when the convergence is achieved, and e_t as the error in the tth

Algorithm 3: One-Time Shard Merging Algorithm

Input: the step size η, the slot index $t = 0$

1 **while** $x_{i,a}$ *does not change* **do**
2   **for** $q = 1 : M$ **do**
3   i tosses a coin with probability $u_{i,a}$ using strategy a.
4   i computes her utility with (14).
5   i approximates $\bar{U}_i(a, x_{-i}(t))$ with (12).
6   i approximates $\bar{U}_i(x_i(t))$ with (13).
7   i updates the probability of strategy a_i using (11).
8   i sends the statistic data and its selection to others.
    $t = t + 1$.

iteration, and r_t as the convergence rate at tth iteration. Based on the relationship between the convergence rate and the error, we have $\frac{e_{t+1}}{e_t} = r_t$. According to [40], [41], r_t is bounded by $\mathcal{O}(\frac{1}{t})$. Therefore, $\frac{e_{t+1}}{e_t} \leq \frac{1}{2}$, which can be re-written as $e \leq (\frac{1}{2})^t$. In other words, the number of iterations T satisfies $T \leq \log \frac{1}{E}$. In each iteration, there is a linear operation with complexity $\mathcal{O}(M)$. Therefore, the total complexity is $\mathcal{O}(M \log \frac{1}{E})$.

$\bar{U}_i(a_i, x_{-i}(t))$ and $\bar{U}_i(x_i(t))$ used in Algorithm 3 are calculated as follows.

$$\bar{U}_i(Y, x_{-i}(t)) = \frac{\sum_{s=1}^{S} U_i(t,s)\mathrm{a}_i(t,s)}{\sum_{s=1}^{S} \mathrm{a}_i(t,s)}, \tag{12}$$

$$\bar{U}_i(x_i(t)) = \frac{1}{S} \sum_{q=1}^{S} U_i(t,s), \tag{13}$$

where $a_i(t,s)$ *represent the strategy of player i at subslot s in slot t.* $U_{i,a}(t,s)$ is,

$$U_{i,a}(t,s) = \begin{cases} G_i - C_i & , i \text{ merges, and (1) is satisfied,} \\ -C_i & , i \text{ merges, and (1) is not satisfied,} \\ G_i & , i \text{ stays, and (1) is satisfied,} \\ 0 & , i \text{ stays, and (1) is not satisfied.} \end{cases} \tag{14}$$

VI. Evaluation

A. Methodology

Testbed Environment: We implemented our sharding system with inter- and intra- shard algorithms on a private chain with go-Ethereum v1.8.0. The test network is composed of nine c5.large servers on AWS, corresponding to nine miners respectively. The consensus protocol is Proof-of-Work (PoW). The gas limit per block is 0x300000, where at most 10 transactions are contained, and the timestamp starts at 0x00. Since our sharding mechanism does not focus on intra-shard consensus process, we just set the number of miners in each shard as 1. Miners stop validating transactions until all the injected transactions are confirmed. We do not use real transactions in the Ethereum. Instead, we register multiple smart contracts, and each of them records an unconditional transaction that transfers money to a specified destination. Transactions in our experiments will invoke these smart contracts. The distribution of transactions and the workloads of miners will be illustrated in each experiment in detail.

Schemes to compare:

- Ethereum: the original non-sharding design.
- ChainSpace: a sharding platform where smart contracts are supported. ChainSpace differs with our sharding design in that ChainSpace separates miners and transactions into shards randomly, incurring new cross-shard consensus protocols and heavy cross-shard communications.

Benchmark: We use Ethereum as the benchmark when evaluating the performance improvement of our sharding system. The main performance metric is *throughput improvement*, which is computed as W_E/W_S, where W_E and W_S are the waiting time until all the transactions are validated in Ethereum and a sharding scheme respectively.

We conducted the following set of experiments to evaluate the benefit of sharding, the effect of our inter-shard merging algorithm and the impact of our intra-shard transaction selection algorithm.

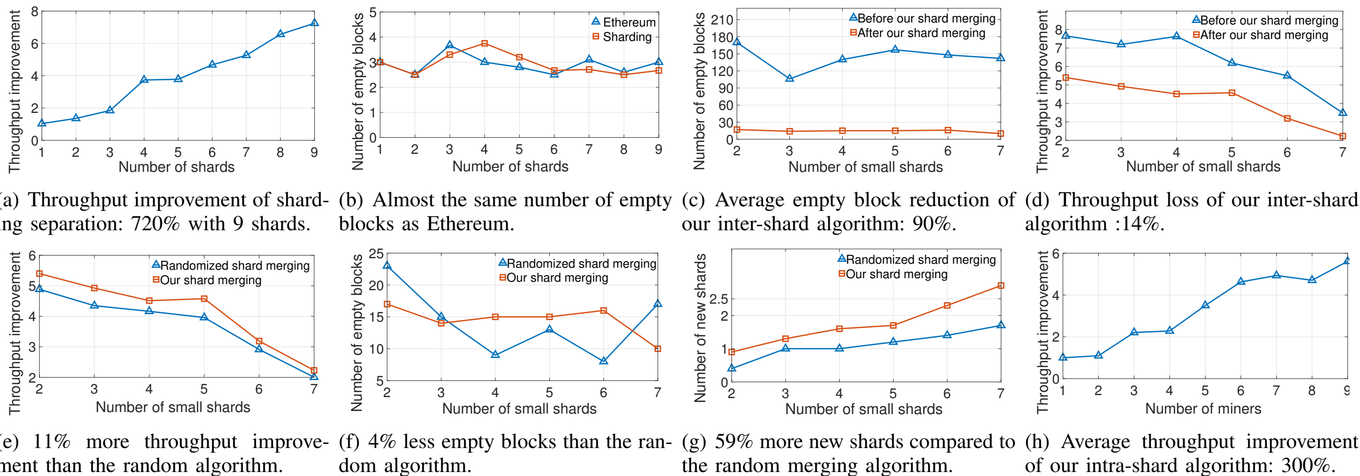
- We compared the throughput between our sharding separation mechanism and Ethereum (Sec. VI-B1).
- We compared the throughput improvement and the cross-shard communication times between our sharding separation mechanism and ChainSpace (Sec. VI-B2).
- We examined the reduction of empty blocks and the throughput improvement of our inter-shard merging algorithm (Sec. VI-C). We further compare the reduction of empty blocks and the throughput improvement between our inter-shard merging algorithm and a randomized merging algorithm, where miners decide whether to merge with others with a probability of 0.5 (Sec. VI-C).
- We evaluated the throughput improvement of our intra-shard transaction selection algorithm (Sec. VI-D).

We further conducted large-scale **simulations** to compare the performance between our algorithms and the optimal solution.

B. The Benefit of Contract-based Sharding

1) Compared with Ethereum: We explore the performance of our sharding separation mechanism with different numbers of shards. We inject 200 transactions into the system with 1 to 9 shards. By incorporating different smart contracts, those transactions will be distributed to multiple shards automatically. The numbers of transactions in these shards obey a uniform distribution. In other words, the number of transactions in each shard is $\frac{200}{s+1}$ under a system with s smart contracts, where s is larger than 1 and $s+1$ indicates that there is a MaxShard. When there are 9 shards, each shard will have 22 transactions on average. The mining process is configured with 0x40000 difficulty, under which a miner can pack one block in one minute on average. The result is shown in Fig. 3(a) and Fig. 3(b).

Fig. 3(a) shows that throughput increases near linearly with respect to the number of shards. When there are 9 shards, the throughput improvement reaches 720%. From Fig. 3(b), there is no vital difference in the number of empty blocks between Ethereum and our sharding design. This is because there does not exist small shards in this circumstance. Since the mining power of shards is almost the same and transactions



(a) Throughput improvement of sharding separation: 720% with 9 shards.

(b) Almost the same number of empty blocks as Ethereum.

(c) Average empty block reduction of our inter-shard algorithm: 90%.

(d) Throughput loss of our inter-shard algorithm :14%.

(e) 11% more throughput improvement than the random algorithm.

(f) 4% less empty blocks than the random algorithm.

(g) 59% more new shards compared to the random merging algorithm.

(h) Average throughput improvement of our intra-shard algorithm: 300%.

Fig. 3. (a)(b): Performance of **our sharding design** without **small shards**. (c)(d): Performance of **our inter-shard merging algorithm**. (e)(f): Compared with random merging, **our inter-shard merging algorithm** is more effective. (g)(h): Throughput improvement of **our intra-shard transaction selection algorithm**.

are separated evenly, miners in different shards will complete the mining process almost at the same time. Therefore, the numbers of unvalidated transactions in these shards are always larger than 0 during the confirmation process of all the inject transactions, indicating (1) is satisfied at any time. In this way, miners can get more profits by validating transactions than only packing empty blocks.

2) Compared with ChainSpace: We first compare the throughput improvement between our sharding design and ChainSpace. To ensure the result is not affected by the difference in intra-shard consensus protocols, we unify the transaction confirmation speed in a non-sharding manner. By revising the mining difficulty of PoW to 0xd79, a miner confirms 76 transactions per second both with our sharding design and ChainSpace in a single shard. We inject 24000 transactions into the system with 1 to 9 shards. In ChainSpace [15], we need to set the number of shards manually, and transactions will be distributed evenly and randomly. Therefore, the numbers of transactions in these shards also obey a uniform distribution.

The throughput improvement of our sharding design and ChainSpace is shown in Fig. 4(a). Both of these two schemes parallel the system effectively. The throughput improvement increases near linearly with the number of shards.

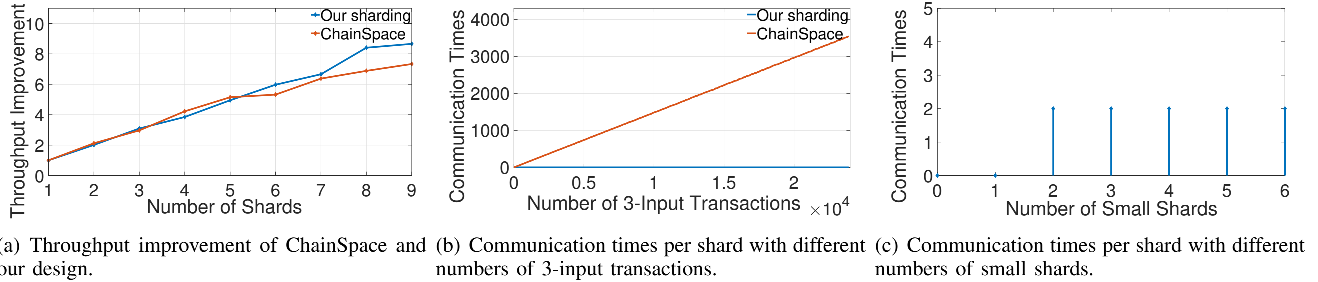
Then we compare our sharding design and ChainSpace on the communication cost with relation to the number of transactions. There are 9 shards, and a single miner can confirm 76 transactions in a non-sharding manner. We inject different numbers of transactions into the whole system. All the injected transactions have 3 inputs. The validation of transactions with 3 inputs needs the account information from 3 users. We repeat this injecting process for 20 times and record the average per-shard communication times to make the results more valid.

The reason for only injecting 3-inputs transactions is as follows. The communication complexity for validating one cross-shard transaction in ChainSpace has been shown as $\mathcal{O}(N^2)$, where N is the number of miners who participate in the communication process. According to the variable-controlling approach, we let $N \leq 3$ to better explore the relationship between the communication cost and the number of transactions. In ChainSpace, a 3-input transaction will be randomly separated into a shard. In this way, the validation of a 3-input transaction needs the information from at most 3 shards, indicating that the communication among miners in up to 3 shards is needed.

The result in Fig. 4(b) shows the difference on communication times per shard between ChainSpace and our sharding design. The blue line corresponds to our design, whose communication times stays at 0. In our design, all of those 3-input transactions will be validated in the MaxShard without any cross-shard communication. On the contrary, the communication times per shard in ChainSpace increases linearly with the number of injected 3-inputs transactions. In the real implementation, the number transactions may be much larger than 24000 and the number of inputs may be not only 3, indicating huge communication cost.

Finally, we examine the communication cost of our sharding design with relation to the number of small shards. There are 7 shards in total with different numbers of small shards, and a single miner can confirm 76 transactions in a non-sharding manner. We inject 24000 transactions in total into the system. We only inject 1000 transactions into a small shard. And we inject more than 3600 transactions into a shard with normal size to ensure that there are 24000 transactions in total. Miners in small shards will start merging at the time slot 0x00.

The result shows that the communication times per shard remains to be 2 under different numbers of smart contracts. Based on our parameter unification scheme, miners in different shards only need to submit the number of transactions to the verifiable leader which is determined in the miner separation process. Then the leader generates and broadcasts the random-



(a) Throughput improvement of ChainSpace and our design.
(b) Communication times per shard with different numbers of 3-input transactions.
(c) Communication times per shard with different numbers of small shards.

Fig. 4. (a): The throughput improvement of our sharding design is not worse than ChainSpace. (b): Our sharding design has zero communication cost when validating transactions, while the communication cost in ChainSpace correlates with the number of transactions linearly. (c) Our sharding design only incurs O(1) communication cost during the merging process.

ness. This process only incurs two communications.

C. Inter-Shard Merging Algorithm

1) Compared with our sharding separation mechanism: With our inter-shard merging algorithm, we record the reduction of empty blocks and the throughput improvement with different numbers of small shards. A miner can pack one block in one minute on average with a c5.large server on Amazon EC2 in this experiment. There are 2 to 7 small shards among the 9 shards. We only inject 1 to 9 transactions into a small shard, and more than 22 transactions into a regular shard while the total number of transactions remains at 200. We record the number of empty blocks in 212 seconds, which is the total confirmation time when each shard has 22 transactions in the experiment of Sec. VI-B1. The results are shown in Fig. 3(d) and Fig. 3(c).

Fig. 3(c) shows that the average number of per-shard empty blocks is 152, while there are only 15 empty blocks on average after our inter-shard merging algorithm. This means our inter-shard algorithm reduces $\frac{152-15}{152} = 90\%$ empty blocks.

From Fig. 3(d), when the number of small shards increases, the throughput improvement decreases. This is because miners in small shards will not validate anything after the first block is packed. Therefore, a certain amount of mining power is wasted on empty blocks. Further, after shard merging, the throughput improvement is $\frac{5.20-4.48}{5.20} = 14\%$ less than the one before merging. The reason is in a merged shard, miners from multiple small shards will mine the same set of transactions in a serialized manner, indicating a decreasing of transaction validation speed.

2) Compared with a randomized merging algorithm: We compare the performance of our inter-shard merging algorithm with a random merging algorithm, where miners in small shards randomly choose whether to merge with others with a probability of 0.5. The experiment settings are the same to the one illustrated in Sec. VI-C1. At some random point, all the miners are at an equilibrium state as defined in Sec. V-A to form a stable shard, and the algorithm also stops here. We compare the throughput improvement, the number of empty blocks, and the number of new shards as shown in Fig. 3(e), Fig. 3(f) and Fig. 3(g).

From Fig. 3(e), our shard merging algorithm improves the throughput by 448% on average, and the randomized algorithm improves it by 403% on average. From Fig. 3(g), our algorithm has $\frac{1.78-1.12}{1.12} = 59\%$ more new shards, further indicating the higher throughput improvement. From Fig.3(f), our merging algorithm has 14.6 per-shard empty blocks on average, and the random algorithm has 15.3 per-shard empty blocks on average, i.e., our algorithm reduces $\frac{15.3-14.6}{15.3} = 4\%$ more empty blocks than the randomized algorithm on average.

In conclusion, our inter-shard merging algorithm reduces empty blocks greatly with little throughput improvement loss.

D. Intra-Shard Transactions Selection Algorithm

We evaluate the throughput improvement of our intra-shard transaction selection algorithm with different numbers of miners. We inject 200 transactions into a single shard with at most 9 miners. A miner can pack one block in one minute on average, and keep mining until all the transactions are confirmed. The result is shown in Fig. 3(h). According to the result, our intra-shard transaction selection algorithm has an average throughput improvement of 300% with 9 miners.

E. Large-Scale Simulations

We simulate our two inter- and intra- shard algorithms **on large scales** with Python 3.0. Through the simulations, we further verify the effectiveness of our algorithms and the correctness of our experimental results.

1) Inter-Shard Merging Algorithm Simulation: We simulate the number of the newly formed shards of our inter-shard merging algorithm as shown in Fig. 5(a). We randomly generate different numbers of transactions in multiple small shards, and record the numbers of new shards according to our inter-shard algorithm. From Fig. 3(e) and Fig. 3(g), we can find that the more new shards after merging, the more throughput improvement is achieved. **Therefore, we can use the number of new shards to represent the throughput improvement.** We thus compare the number of new shards in our inter-shard merging algorithm with the optimal value. The system throughput is maximized when the size of all the new shards is L which is defined in (1), i.e., the number of small shards is $\frac{\#transactions}{L}$.

Fig. 5(a) shows that our algorithm can achieve 80% of the optimal performance on average. The reason for such high

performance is as follows. In our algorithm, a player repeats the determination process based on the updated merging probability several times at each iteration. According to randomized algorithm theory [23], repeating increases the success probability, indicating the higher probability for getting the optimal solution.

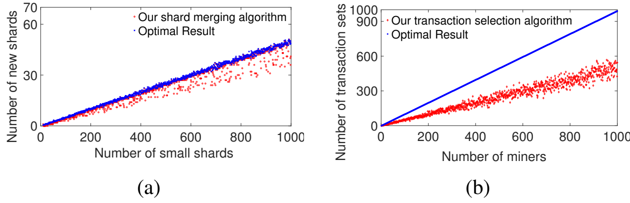


Fig. 5. (a): Our merging algorithm only loses 20% throughput improvement on average than the optimal result. (b): Our transaction selection algorithm loses about 50% throughput improvement on average.

2) Intra-Shard Transaction Selection Algorithm Simulation: We also simulate the number of transaction subsets under our transaction selection algorithm as in Fig. 5(b). We randomly generate the transaction fees of multiple transactions and record the numbers of transaction sets according to our intra-shard algorithm. According to the previous analysis in Sec. II-C, if miners validate different sets of transactions, the system throughput will be improved. Therefore, **the number of transaction sets can represent the throughput improvement of the system.** The optimal situation happens when all the miners validate different sets of transactions. In this way, the number of transaction sets is the same as the number of miners. Fig. 5(b) shows that our algorithm loses 50% in throughput on average compared to the optimal solution.

The near 50% performance loss results from the existence of the situation that there is a transaction set with much higher transaction fees than others, where the equilibrium is that everyone chooses that transaction set with the highest transaction fee. Under such circumstances, the throughput improvement is 1, i.e., this is a serialized system, which will pull down the average performance of our intra-shard transaction selection algorithm.

In conclusion, we got the following evaluation results:

- Our sharding mechanism achieves 720% throughput improvement when there are only 9 shards (Sec. VI-B1).
- Under the same consensus achievement efficiency in a single shard, our sharding mechanism achieves almost the same throughput improvement to ChainSpace.
- Our sharding incurs constant communication times. Conversely, the communication times in ChainSpace increases linearly with different numbers of transactions.
- Our merging algorithm reduces 90% empty blocks with 14% decrease in throughput improvement (Sec. VI-C).
- Our merging algorithm has 11% higher throughput improvement than the randomized merging algorithm, 59% more new shards and 4% less empty blocks on average (Sec. VI-C). In large-scale simulations, our shard merging algorithm is near-optimal, with 20% throughput loss on average.
- Our intra-shard transaction selection algorithm has 300% throughput improvement on average (Sec. VI-D).

VII. Related Work

Existing sharding schemes that divide the states among different shards [13]–[16] all require a new consensus protocol. Leaders in input and output shards of a cross-shard transaction are responsible for conducting cross-shard communication to reach a consensus on whether this transaction is valid. Input shards have transaction records related to the inputs of a cross-shard transaction, and output shards will record this transaction after it is confirmed. RSCoin [16] adopts a Two-Phase Commit protocol. A transaction is first committed to leaders of input shards. If accepted by the majority of input leaders, it will be committed to leaders of output shards for final validation. In Chainspace [15], miners in input shards first reach an intra-shard consensus, leaders in input shards directly communicate with leaders in output shards to reach a cross-shard consensus. In RapidChain [14], leaders in output shards will generate one intra-shard transaction for each input and send it to the corresponding input shard. Leaders of input shards then send their validations back to output shards for final approval. In Omniledger [13], several validators selected from miners are responsible for validating transactions, and leaders will collect these validation results and return the joint decision to users.

Further, all the protocols mentioned above incur frequent cross-shard communication. To validate one cross-shard transaction, there will be at least 2 rounds of cross-shard communication. One round communication between two shards can require up to $\mathcal{O}(n^2)$ bits of network transfers [13], where n is the number of nodes participating in the communication. When there are a large number of cross-shard transactions, the communication overhead will be extremely high. In our sharding system, a shard is formed by transactions whose inputs and outputs are within the same shard. Therefore, no cross-shard communication is necessary to validate transactions. Our sharding system only needs minimal cross-shard communication to update the statistical information of each shard.

On the other hand, sharding proposals where states are not divided result in heavy storage costs since all the miners need to store all the transaction histories [42]. Per-shard validating peers in sharding systems like Zilliqa [11], Corda [43], and Elastico [44] store the *entire* states of the system to validate cross-shard transactions. Since validating peers have complete information of the system, they can use the traditional consensus protocols like Practical Byzantine Fault Tolerance (PBFT) [45] to correctly validate cross-shard transactions. However, sharding systems need to reconfigure shards and reselect validating peers periodically to prevent the *Sybil attack* [46]. Therefore, all the miners must store the entire states of the system in the long run. The storage cost is still high for each miner. In contrast, our sharding scheme divides the isolated states into independent shards and miners in these shards do not need to store the complete information of the system. Therefore, the storage cost is significantly reduced.

VIII. Conclusion

In this paper, we proposed a distributed sharding system that improves the throughput of blockchain systems with smart contracts. Our solution requires minimum cross-shard communication without new consensus protocols. By adopting an inter-shard merging algorithm and intra-shard transaction selection algorithm, miners independently join a shard and mine different sets of transactions that maximize the system throughput. To reduce the communication cost and improve security, we designed a new parameter unification mechanism and provide the corresponding security analyses. Based on our analysis, our inter- and intra- shard algorithms can converge to a Nash Equilibrium. We have implemented our sharding system on go-Ethereum v.1.8.0 and evaluated its performance by comparing it with Ethereum and ChainSpace with injected transactions and large-scale simulations. Our experimental results confirmed that our sharding algorithms can significantly improve system throughput. Our future work will concentrate on reducing the query cost on whether a user incorporates multiple smart contracts and the storage overhead of miners in the MaxShard.

References

[1] S. Maiyya, V. Zakhary, D. Agrawal, and A. E. Abbadi, "Database and Distributed Computing Fundamentals for Scalable, Fault-tolerant, and Consistent Maintenance of Blockchains," in *ACM VLDB*, vol. 11, no. 12, 2018, pp. 2098–2101.

[2] S. Maiyya, V. Zakhary, M. J. Amiri, D. Agrawal, and A. El Abbadi, "Database and Distributed Computing Foundations of Blockchains," in *ACM SIGMOD*, 2019, pp. 2036–2041.

[3] S. Han, Z. Xu, Y. Zeng, and L. Chen, "Fluid: A Blockchain Based Framework for Crowdsourcing," in *ACM SIGMOD*, 2019, pp. 1921–1924.

[4] S. Nakamoto, "Bitcoin: A Peer-to-peer Electronic Cash System," 2008.

[5] T. D. Team, "Announcing New Dogecoin Foundation," http://foundation.dogecoin.com/.

[6] V. Buterin *et al.*, "Ethereum white paper, 2014," https://github.com/ethereum/wiki/wiki/White-Paper.

[7] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A Global Naming and Storage System Secured by Blockchains." in *USENIX ATC*, 2016, pp. 181–194.

[8] J. Benet, "IPFS-content Addressed, versioned, p2p file system," in *arXiv preprint arXiv:1407.3561*, 2014.

[9] Blocksplain, "Blockchain speeds & the scalability debate," https://blocksplain.com/2018/02/28/transaction-speeds/.

[10] C. Cachin, "Architecture of the Hyperledger Blockchain Fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, vol. 310, 2016.

[11] T. Z. Team, "The ZILLIQA Technical Whitepaper," https://docs.zilliqa.com/whitepaper.pdf.

[12] T. E. Team, "On Sharding Blockchains," https://github.com/ethereum/wiki/wiki/Sharding-FAQs.

[13] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A Secure, Scale-out, Decentralized Ledger via Sharding," in *IEEE SP*, 2018, pp. 583–598.

[14] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *ACM CCS*, 2018, pp. 931–948.

[15] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A Sharded Smart Contracts Platform," in *arXiv preprint arXiv:1708.03778*, 2017.

[16] Q. Yao, "Central Bank Encrypto-currencyAnalysis of RSCoin System," in *Caijing Weekly*, vol. 13, 2017, pp. 20–22.

[17] J. Wang and H. Wang, "Monoxide: Scale out Blockchains with Asynchronous Consensus Zones," in *USENIX NSDI*, 2019, pp. 95–112.

[18] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *ACM SIGMOD*, 2019.

[19] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[20] F. Michal and N. Noam, "Potential and Congestion Games," https://hujieconcs.wordpress.com/lecture-notes/.

[21] I. Milchtaich, "Congestion Games with Player-specific Payoff Functions," in *Games and Economic Behavior*, vol. 13, no. 1. Academic Press, 1996, pp. 111–124.

[22] T. Heikkinen, "A Potential Game Approach to Distributed Power Control and Scheduling," in *Computer Networks*, vol. 50, no. 13. Elsevier, 2006, pp. 2295–2311.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2009.

[24] S. Micali, M. Rabin, and S. Vadhan, "Verifiable Random Functions," in *Foundations of Computer Science, 1999. 40th Annual Symposium on Foundations of Computer Science*. IEEE, 1999, pp. 120–130.

[25] A. Vikati, "Ranking Ethereum Smart Contracts," https://medium.com/@vikati/ranking-ethereum-smart-contracts-a27e6f622ac6.

[26] SFOX, "What 29,985,328 Transactions Say About the State of Smart Contracts on Ethereum," https://blog.sfox.com/what-29-985-328-transactions-say-about-the-state-of-smart-contracts-on-ethereum-2ebdba4bea1c.

[27] Google, "Google Public Ethereum Dataset," https://bigquery.cloud.google.com/dataset/bigquery-public-data.

[28] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable Bias-resistant Distributed Randomness," in *IEEE SP*, 2017, pp. 444–460.

[29] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, no. 3, pp. 216–226, 1979.

[30] K. Mitchell, "Get Rich by Mining Empty Blocks on Ethereum," https://coinedtimes.com/get-rich-by-mining-empty-blocks-on-ethereum/.

[31] S. Zheng, "An Analysis of Empty Blocks on Ethereum," https://coinedtimes.com/get-rich-by-mining-empty-blocks-on-ethereum/.

[32] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode, "Nashification and the Coordination Ratio for a Selfish Routing Game," in *Springer International Colloquium on Automata, Languages, and Programming*, 2003, pp. 514–526.

[33] H. Ackermann, H. Röglin, and B. Vöcking, "On the Impact of Combinatorial Structure on Congestion Games," in *Journal of the ACM*, vol. 55, no. 6, 2008, p. 25.

[34] S. Chien and A. Sinclair, "Convergence to Approximate Nash Equilibria in Congestion Games," in *Elsevier Games and Economic Behavior*, vol. 71, no. 2, 2011, pp. 315–327.

[35] M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis, "Structure and Complexity of Extreme Nash Equilibria," in *Elsevier Theoretical Computer Science*, vol. 343, no. 1-2, 2005, pp. 133–157.

[36] J. M. Smith, *Evolution and the Theory of Games*. Cambridge university press, 1982.

[37] B. Wang, K. R. Liu, and T. C. Clancy, "Evolutionary Cooperative Spectrum Sensing Game: How to Collaborate?" in *IEEE Transactions on Communications*, vol. 58, no. 3, 2010.

[38] R. Cressman, *Evolutionary Dynamics and Extensive Form Games*. MIT Press, 2003, vol. 5.

[39] "On Sharding Open Blockchain with Smart Contracts," http://bit.ly/2lJIzbI.

[40] R. Ganti, "Convergence Rate of Gradient Descent Algorithm," https://rkganti.wordpress.com/2015/08/21/convergence-rate-of-gradient-descent-algorithm/.

[41] N. Z. Shor, "Convergence Rate of the Gradient Descent Method with Dilatation of the Space," in *Springer Cybernetics*, vol. 6, no. 2, 1970, pp. 102–108.

[42] Z. Xu, S. Han, and L. Chen, "CUB, A Consensus Unit-Based Storage Scheme for Blockchain System," in *IEEE ICDE*, 2018, pp. 173–184.

[43] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An Introduction," in *R3 CEV*, 2016.

[44] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol for Ppen Blockchains," in *ACM CCS*, 2016, pp. 17–30.

[45] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery," in *ACM TOCS*, vol. 20, no. 4, 2002, pp. 398–461.

[46] J. R. Douceur, "The Sybil Attack," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 251–260.