2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

OptChain: Optimal Transactions Placement for Scalable Blockchain Sharding

Lan N. Nguyen, Truc D. T. Nguyen
CISE Department
University of Florida
Gainesville, FL, 32611
Email: {lan.nguyen,truc.nguyen}@ufl.edu

Thang N. Dinh
CS Department
Virginia Commonwealth University
Richmond, VA, 23284
Email: tndinh@vcu.edu

My T. Thai
CISE Department
University of Florida
Gainesville, FL, 32611
Email: mythai@cise.ufl.edu

***Abstract*—A major challenge in blockchain sharding protocols is that more than 95% transactions are cross-shard. Not only those cross-shard transactions degrade the system throughput but also double the confirmation time, and exhaust an already scarce network bandwidth. Are cross-shard transactions imminent for sharding schemes? In this paper, we propose a new sharding paradigm, called OptChain, in which cross-shard transactions are minimized, resulting in almost twice faster confirmation time and throughput. By treating transactions as a stream of nodes in an online graph, OptChain utilizes a lightweight and on-the-fly transaction placement method to group both related and soon-related transactions into the same shards. At the same time, OptChain maintains a temporal balance among shards to guarantee the high parallelism. Our comprehensive and large-scale simulation using Oversim P2P library confirms a significant boost in performance with up to 10 folds reduction in cross-shard transactions, more than twice reduction in confirmation time, and 50% increase in throughput. When combined with Omniledger sharding protocol, OptChain delivers a 6000 transactions per second throughput with 10.5s confirmation time.**

I. Introduction

Blockchain has emerged as a disruptive and transformational technology, with great potential and benefits, offering a promising new decentralized economy without the risk of single point of failures, monopoly, or censorship [1]. Examples of these systems are ranging from the cryptocurrency such as Bitcoin [2] and Ethereum [3], to other infrastructures and application domains such as the Internet-of-Things [4], [5] and Digitial Health [6], [7]. Unfortunately, the performance level of existing decentralized systems based on the Blockchain technology is too low to realize that vision, e.g., 7 transactions per second (tps) and up to 60 minutes confirmation time for Bitcoin and 10 tps/12 minutes for Ethereum.

To this end, *blockchain sharding*, which splits the Blockchain into multiple disjoint parts, called shards, each maintained by a subgroup of nodes, has been proposed as a prominent solution for Blockchain scaling. Since each node only needs to communicate with a few (hundreds) nodes from the same shard to maintain a small chunk of blockchain, sharding reduces substantially the storage, computation, and communication costs. This is different from legacy blockchain systems, e.g., Bitcoin and Ethereum, in which all nodes need to communicate to maintain the same copy of blockchain, thus, the performance is limited by the nodes average processing capabilities. The latest sharding approaches such as Omniledger [8] and Rapidchain [9] can handle thousands of transactions per second with confirmation time about a few dozens of seconds.

All existing sharding approaches, however, face the same challenge of handling *cross-shard* transactions, which involve data from more than one shards. To prevent the double-spending problem [10], all shards that involve a cross-shard transaction (tx) need to execute multiple-phase protocols to confirm the tx. This can multiple fold increase both the latency and the effort to confirm the tx, comparing to the case when the tx need to be processed by only one shard. In turns, extra efforts in confirming txs may lead to higher tx fees. To make it even worse, more than 95% of the transactions are cross-shard [8], [9]. Previous sharding approaches often place txs into shards randomly to balance the load among the shards. It is natural to ask "is there a smart transactions placement strategy that reduces the cross-shard txs, making sharding even faster?"

In this paper, we propose OptChain, a sharding-agnostic framework that boosts the performance of existing (and future) sharding approaches via optimizing the placement of txs into shards. OptChain learns the pattern from the past txs to decide the shard-location for incoming txs based on 1) whether such placement reduces the cross-shard txs and 2) load balance among the shards. Specifically, OptChain works on top an unexplored graph construction for transaction networks in UTXO model [2], termed *Transactions as Nodes* (TaN), and introduces a new score, termed *Temporal Fitness* to assess the suitability of placing an incoming transaction to each shard.

The *TaN network* is constructed by abstracting each transaction as a node and there is a directed edge (u, v) if tx u uses tx v as an input. This TaN is an online directed acyclic graph (DAG), in which nodes arriving one by one. This construction is different from existing abstraction of transaction networks in which transactions are abstracted as edges among the nodes made of addresses [11].

The *Temporal Fitness* score is composed by two component scores *Transaction-to-Shard* (T2S) and *Latency-to-Shard* (L2S). The T2S scores between a transaction u and a shard S_i, measures the probability that a random walk from a node/tx u ends up at some node in S_i (see section IV.B), i.e., hence, how likely the transaction should be placed into the shard without causing further crosshard txs. The L2S score estimates the processing delay when placing the transaction to a shard. Importantly, the protocol to estimate the two scores is lightweight and is executed at the users side.

Practicality. Our solution $OptChain$ can be implemented

The first two authors contribute equally to this paper.

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00059

with simple modification in user-side software, e.g., wallet. That is $OptChain$ does not interfere with the core consensus protocols and, hence, can be integrated into almost all sharding approaches. Specifically, as computing the T2S score only requires the information on the input txs, it can be done efficiently at the user side by modifying the existing Simple Payment Verification protocol [2], i.e., users do not need to download the complete transaction history. Based on the latencies observed from the shard, the wallet software at the user side can use OptChain to make the decision on which shard he/she wants the tx to be processed.

To validate our approach, we measure the performance of an enhanced version of OmniLedger [8] with our OptChain approach on existing Bitcoin transactions. The results indicate that OptChain can effectively reduce the cross-shard txs up to 10 folds, cut the txs confirmation time by 93%, and at the same time, increase the throughput by 50%. While we only test OptChain with Omniledger, we predict a similar level of improvement in performance when combining OptChain with other sharding protocols such as Rapidchain.

Our contribution. We summarize our contribution as follows.

- We introduce a new way of sharding txs, reducing cross-shard txs via an optimal placement of txs into shards. This simple idea effectively reduces the cross-shard txs and boosts the performance comparing to the random placement in existing sharding approaches.
- We investigate a new abstraction of transactions network (TaN) in which transactions are abstracted as nodes rather than edges among addresses in previous studies. This new abstraction results in an online DAG that can provide new ways to analyze the transaction stream in UTXO-based ledgers. We provide various charateristics of this TaN on Bitcoin transactions consisting of 298,325,121 nodes and 696,860,716 edges.
- We introduce a novel algorithm, called OptChain, that analyze the stream of txs in TaN network to make the optimal placement of txs into shards. OptChain is simple, lightweight, and can be easily implemented into existing wallet software.
- Our comprehensive experiments with an enhanced Omniledger protocol on real Bitcoin transactions affirm the significant benefit of our approach in cutting down the confirmation time and boosting throughput.

Organization. The rest of this paper is structured as follows. We summarize the related work in Section II. Section III discusses the basis of handling cross-shard transactions in blockchain sharding and provides some observations as well as primary goals of the transactions placement strategies. In Section IV, we investigate the TaN network and present our OptChain algorithm. The experimental design and results are presented in Section V. Finally, Section VI gives the concluding remarks.

II. Related work

Blockchain sharding. Several blockchain sharding protocols [8], [9], [12]–[15] have been proposed to address the scalability issue in legacy blockchain. In typical blockchain sharding, the entire state of the blockchain is splitted into partitions called *shards* that contain their own independent piece of state and transaction history. The key idea is to parallelize the available computation power, dividing it into several smaller shards where each of them processes a disjoint set of transactions.

Currently, most of existing sharding protocols are built on top of the UTXOs model. The most notable exception is Ethereum 2.0 [15] which is the next development phase of the Ethereum blockchain [3], employing the account model. Unlike the UTXO model, each transaction in the account model has only one input and one output.

The three core components of an existing sharding protocol are 1) how to (randomly) assign nodes into shards to form shard committees; 2) an intra-shard consensus protocol to execute by shard committees (often a BFT protocol [8], [9] but also can be a Nakamoto-like protocol [16]); and 3) an atomic protocol for cross-shard transactions. In this work, we focus on the third component in which we aim to optimize the placement of transactions, hence mitigate the negative impact of cross-shard transactions on the performance of existing sharding approaches [8], [9].

Transaction networks. In our method, we abstract the relation between transactions under a graph representation. In previous work, Kondor et al. [11] addressed the transactions network as a graph where each node represents a user address, each directed edge between two nodes is created if there is at least one transaction between the corresponding addresses. Our work is fundamentally different in which we abstract transactions as nodes while an edge between two nodes represents the behavior that one transaction spends an output of the other. By such representation, we model the problem of transaction sharding to be an online graph partitioning problem with temporal balancing, where nodes in a graph is divided into disjoint subsets and the objective is to identify which subset should contain a new arriving node.

Online and Balanced DAG Partitioning. Online graph partitioning has been addressed in the literature [17], [18]. Stanton et al. [17] and Abbas et al. [18] have proposed multiple natural, simple heuristics and compared their performance to Metis [19], a fast and offline method. The objective of their algorithms is to partition network vertices into almost equal disjoint sets while minimizing number of crossing edges (edge whose two endpoints are in two different set). However, these works are fundamentally different to our work, in which we would like to minimize number of cross-shard transactions rather than crossing edges. Furthermore, even their works eventually guarantee the balances between sets, we concern more on temporal balancing, in which we would like the number of vertices on each sets should be almost equal for any given of time.

III. Cross-shard transactions and Transactions Placement Strategies

In this section, we present an overview of state-of-the-art transaction sharding procedures. Although blockchain sharding itself could help boost the transactions confirmation process, nonetheless, the high amount of cross-shard transactions is one of the obstacles that hinder the system from getting better performance. Hence, we illustrate its negative impact which motivates us to devise the OptChain to tolerate this issue.

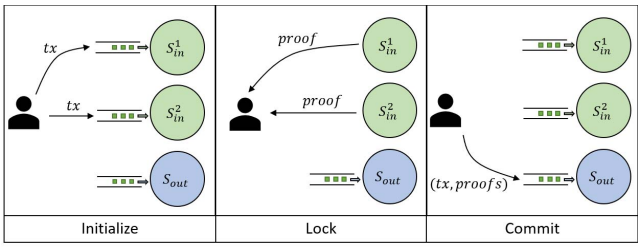


Fig. 1: Handling cross-shard transactions in Omniledger [8]

A. Atomic Commit Protocols for Cross-shard Transactions

As both cross-shard protocols in [8] and [9] are built on top of ledgers using UTXO model, we begin with the presentation of this model.

Unspent Transaction outputs (UTXO) model. The UTXO model was introduced in original Nakamoto protocol for Bitcoin [2]. In this model, transactions may have multiple inputs and outputs where an output is assigned with credits and locked to a user's address. This newly created transaction output is referred to as a UTXO. The UTXOs may then be used as inputs of another transaction, and after this transaction is committed to a block, those UTXOs will be marked as *spent* and cannot be used again. For simplicity, we will consider a transaction tx with two inputs tx_{in}^1, tx_{in}^2.

Cross-shard Transactions. Let S_{in}^1, S_{in}^2, and S_{out} denote the shards that contain tx_{in}^1, tx_{in}^2 and tx, respectively. If all the shards S_{in}^1, S_{in}^2, and S_{out} are the same, we have a same-shard transaction, otherwise the transaction is cross-shard (cross-TX).

If transactions are placed into shards randomly, it was shown that the probability for a typical transaction having two inputs and one output to be a cross-shard transaction is about 94% [8], assuming 4 shards, and 99.98%, assuming 16 shards.

Committing Cross-TXs in OmniLedger [8]. OmniLedger proposed a novel atomic protocol to commit cross-TXs consistently among all shards. The protocol locks all input transactions at the input shards before committing the output transaction(s) to output shard(s).

1) **Initialize**. A user creates a cross-TX tx whose inputs spend UTXOs, e.g., tx_{in}^1, tx_{in}^2, from some input shards, e.g., S_{in}^1, S_{in}^2. The client gossips the cross-TX and it eventually reaches all input shards.
2) **Lock**. All input shards validate the transactions within his shard. If the transactions are valid, they are marked spent on the shard's ledger, and a *proof-of-acceptance* is gossiped; otherwise a *proof-of-rejection* is gossiped.
3) **Commit**. If all input shards gossip the proof-of-acceptance, the client can gossip an *unlock-to-commit transactions* that eventually reach all output shards. In turn, each output shard validates the transaction and includes it to his ledger. However, if even one input shard issued a proof-of-rejection, then the transaction cannot be committed and has to abort. The client then can gossip an *unlock-to-abort* message to reclaim the fund.

Committing Cross-TXs in RapidChain [9]. Rapidchain uses a "yanking" mechanism, in which the input transactions, e.g., tx_{in}^1, tx_{in}^2, will be first moved to from the input shards, e.g., S_{in}^1, S_{in}^2, to the output shard, e.g., S_{out} via an inter-committee protocol. After all the input transactions are successfully "yanked" to the output shard, then the final transactions can be added to the ledger of the output shard.

B. Performance Penalty for cross-TXs.

Comparing to same-shard transactions, cross-TX incurs much higher confirmation time, communication and computation costs.

Longer confirmation time. In Omniledger [8], a cross-TX will easily double the confirmation time of those in the same-shard transactions. For the same-shard transaction, the user only needs to submit the transaction to the shard and wait for confirmation. The confirmation time will only be the round time trip between the users and the shard pluses the time for the shard committee to agree on the transactions. A cross-TX will incur extra time to confirm the input transactions as the input shards as well as another round time trip between the users and the shard committees. The same is applied for Rapidchain cross-TX protocol as each cross-TX incurs extra round-time trip among the committees as well as the waiting time for input transactions to agree on 'yanking' transactions between shards.

Extra communication and computation cost. For a typical cross-TX with 2 inputs and one output, the communication cost will triple that of a same-shard transaction as all the three shard committees and the user need to communicate to confirm the transaction. The same holds for the computation cost.

Thus, assuming uniform time and cost to handle transactions in shards, each cross-TX will *double confirmation time* and *triple the bandwith consumption and computation cost*. Therefore, if we can reduce the fraction of cross-TXs to 20%, we will cut the confirmation time by more than 40% and more than double the throughput.

C. OptChain: A Transactions Placement Strategy

Random Placement. In existing sharding approaches [8], [9], transactions are placed randomly into shards. Often, the hashed value of a transaction is used to determine which shards the transaction will be placed into. This will balance the amount of transactions per shard, however, cause almost all transactions to be cross-TXs. For a typical transaction having two inputs and one output to be a cross-shard transaction is about 94%, assuming 4 shards, and 99.98%, assuming 16 shards [8]. To be specific, the systems do not consider the relationship between transactions on sharding process, which makes a majority of transactions eventually become cross-TXs.

Smart Transaction Placement. Ideally, the best method is to groups well-connected transactions into a same shard. By that way, we can minimize the number of verification steps (step 2) of cross-TXto get proof-of-acceptance. Moreover, the current state of shards should also be considered. We would like to avoid situations where some shards are extremely busy (i.e huge number of transactions in queue to wait for verification) while some are idle. Intuitively, with a random selection, we can balance the number of transactions in each shards. However, some transactions could take more time to processed than the others. Therefore, in simulation, we observe that there are some certain moments the random selection will eventually cause extremely imbalance on queue sizes between shards.

OptChain. Motivated by observations on the limits of OmniLedger and RapidChain, in this paper, we investigate an optimization problem in which users need to determine the best shard to submit their transactions in order to minimize cross-TXs while guaranteeing the temporal balance, thus, shorten the confirmation time and boost the overall system throughput. Specifically, the ultimate goal of our smart transactions placement OptChain are:

1) **Fast Confirmation Time**: As major txs are same-shard txs and load are distributed evenly, txs get confirmed in much shorter time.
2) **High Throughput**: Same-shard txs require less time and communication to confirm, thus, the system throughput gets significantly boosted.

The above goals are obtained via optimizing two indirect goals

1) **Cross-TX Minimization**: Reduce the number of cross-shard transactions by grouping related transactions into a same shard.
2) **Temporal Balancing**: To distribute load evenly among shards to increase parallelism and reduce queuing time.

In practice, we aim to deploy OptChain as a user-side software. By monitoring its own transactions as well as the information on the loads and confirmation time at the shards, a client software can make decision on which is the best shard to submit transactions. It is important that users do not have to store the whole blockchain to optimize the placement.

In the next section, we will propose a lightweight, yet, efficient solution, answering the question: *How to identify an appropriate shard for a new transaction such that in the long future, our system can achieve four main goals as described?*

IV. OptChain Algorithm

In this section, we propose an algorithm used by OptChain to place transactions into shards. First, we introduce the system model, in which we represent transactions under a directed network. Under this model, we utilize a well-known PageRank analysis to propose T2S-score, which is to measure how likely a transaction should be placed into the shard. Then, we propose a mathematical model to estimate confirmation latency for placing a transaction into shards, which we called L2S score. Finally, we describe how OptChain places transactions into shards based on the combination of T2S and L2S scores.

A. Transaction-as-Node Network and Partitioning

To observe the relation between transactions, we model the set of transactions under a graph representation, which is defined as follows:

Definition 1 (TaN Network): A TaN network of a set of transactions is presented as a directed graph $G = (V, E)$ where V is the set of transactions and E is a set of directed edges in which there exists $(u, v) \in E$ if the transaction u uses the UTXO(s) of transaction v.

To have a clear picture on how a TaN network looks like, we construct a TaN network from a set of transctions in the first 508,241 blocks of the Bitcoin blockchain which was gathered by MIT [20]. The most recent transaction of the dataset was issued on February 2018. From this dataset, we construct a TaN network of 298,325,121 nodes and 696,860,716 edges. Nodes that do not have any outgoing edges represent coinbase transactions (rewards for mining Bitcoin blocks). On the other hand, nodes that do not have any incoming edges represent transactions whose UTXOs have not been spent. TaN network is a directed acyclic graph since a transaction only uses UTXO(s) of past transactions. Therefore, TaN network can be sorted in a topological order, which exactly reflects the order of appearance of transactions.

Fig. 2a shows the plot of the degree distribution of the constructed TaN network in log-log scale. The degree distribution of TaN network exhibits *power-law* distribution with average in- or out- degree of ≈ 2.3. There are total 508,241 coinbase transactions (transactions without inputs), and 369,2947 transactions whose UTXO(s) have not been spent. Also, there are 37,108 transactions without any outputs or inputs. As can be seen from the degree distribution in Fig. 2b, major nodes (93.1%) in TaN network have the in-degree lower than 3. As regards the out-degree, 97.6% of nodes have the out-degree lower than 10, and 86.3% lower than 3.

We investigate the average degree of nodes of TaN network overtime and plot in Fig. 2c. We observe that for a major of time, the average degree of TaN network is stable and consistent. There are only 2 periods the average degree significantly changes, which is the first 1 millions transaction and around 80,000,000th one. This behavior does not happen regularly because: (1) the first period is that the system needs to generate a lot of coinbase transactions for funding and (2) there was a flooding attack at the second period [21], which causes mining pools to create a lot of transactions with high degree to clean up "trash" transactions.

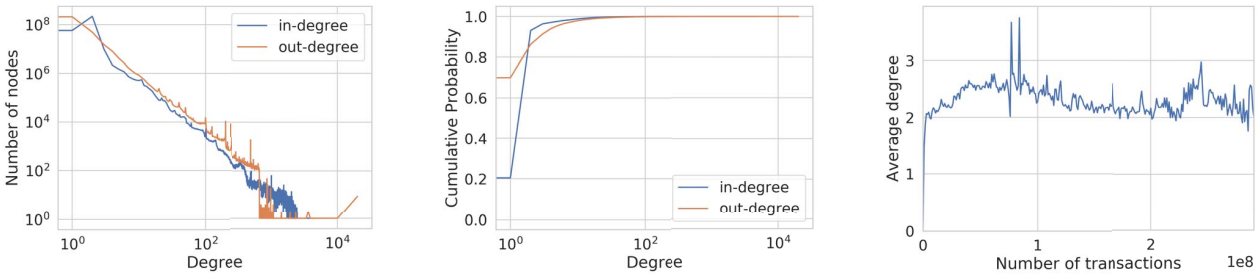
By distributing transactions into shards, the task of transactions sharding eventually become partitioning the TaN network into k disjoint subsets of nodes $\mathcal{S} = \{S_1, ..., S_k\}$, where k is the number of shards and S_i denotes a set of transactions under management of shard i. We have $\cup_{i=1}^{k} S_i = V$ and $S_i \cap S_j = \emptyset$ for all $i, j = 1 \rightarrow k$. For simplicity, we also call S_i as the shard i.

Given a transaction represented by node $u \in V$, denote $S(u) \in \mathcal{S}$ as a shard containing u. Let $\mathcal{S}_{in}(u)$ as a set of input shards of u. Then u is a cross-TX iff $\mathcal{S}_{in}(u) \neq \{S(u)\}$. Therefore, we consider the task of distribute a new transaction into shards as an online partitioning to the TaN network. To be specific, given a TaN network $G = (V, E)$, a set of k disjoint subsets of nodes $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ and a new arriving transaction (node) u, our task is to identify $S(u) \in \mathcal{S}$.

B. Transaction-to-Shard Score

In this part, we will propose a metric to measure the fitness score between the new arrival transaction and shards, which we called Transaction-to-Shard (T2S) score. This score is motivated by a well-known graph measurement, called PageRank. To be specific, PageRank assigns numerical weighting to each node of a network with the purpose of measuring node relative importance within the network. A PageRank vector is a weighted sum of the probability distribution obtained by taking a sequence of random walk steps starting from a specified initial distribution. PageRank has been proven to be an efficient method on graph local partitioning, in which nodes in a same set of partition tend to have a similar weight.

Although regular PageRank computation could cost considerable runtime, we utilize the trait that TaN is acyclic and the order of nodes' appearance is also TaN's topological order to devise a fast T2S-score computation of each nodes. To be



(a) Degree distribution (b) Cumulative distribution (c) Changes in average degree

Fig. 2: TaN network statistics

specific, we represent the T2S-scores of a transaction u to shards under a k-dimensional vector $p(u)$, in which the entry i measures the fitness between u and shard i. Denote $r[i]$ as the value of entry i of vector r. Also, let $N_{in}(u)$ ($N_{out}(u)$) as a set of input (output) transactions of u. $p(u)$ is computed by the following equation:

$$p(u) = \alpha s(u) + (1-\alpha) \sum_{v \in N_{in}(u)} \frac{p(v)}{|N_{out}(v)|}$$

where α is a constant in $(0, 1]$ and $s(u)$ is a starting vector for a node u. $s(u)$ is initiated as follows:

- If u is a new arriving node, $s(u) = \{0\}^k$
- Otherwise, $s(u)$ receives a value $\frac{1}{|S(u)|}$ at the entry corresponding to shard $S(u)$ and 0 elsewhere.

Clearly $p(u)$ can easily be calculated in the manner similar to Bread-First Search, in which we start from nodes who have no input (coinbase transaction). Therefore, computing $p(u)$ for all $u \in V$ costs $O(k(|V| + |E|))$ runtime complexity, which is very expensive as the TaN network grows. However, we observe that: after placing a new transaction u into shard $S(u)$, the change on $s(v)$ for all $v \in V$ does not impact the fitness score of all nodes other than normalization scale. To be specific, assume at time $t+1$, u arrives and is placed into shard i ($S(u) = S_i$). Denote $p^{(t)}(v)$ is the T2S-score vector of node v at time t and $S_i^{(t)}$ is the shard i at time t. We have:

- If $v \neq u$, $p^{(t+1)}(v)[i] = \frac{|S_i^{(t)}|}{|S_i^{(t)}|+1} p^{(t)}(v)[i]$ and $p^{(t+1)}(v)[j] = p^{(t)}(v)[j]$ for all $j \neq i$.
- If $v = u$, $p^{(t+1)}(v)[i] = \alpha \frac{1}{|S_i^{(t)}|+1} + \frac{S_i^{(t)}}{S_i^{(t)}+1} p^{(t)}(v)[i]$ and $p^{(t+1)}(v)[j] = p^{(t)}(v)[j]$ for all $j \neq i$.

Therefore, rather than computing $p(v)$ for all $v \in V$ from scratch to get T2S score of a new transaction u, we propose the following methods for the faster calculation. First, we introduce two other vectors $p'(v), s'(v)$ associated to each node $v \in V$ in addition to $s(v)$ and $p(v)$. If a transaction v is placed in shard j then $s'(v)[j] = 1$ and $s'(v)[i] = 0$ for all $i \neq j$, $s'(v)$ is fixed right after v is placed. Given a new transaction u, $p(u)$ can be computed as follows.

- If u is a coinbase transaction, there is no calculation needed.
- Otherwise, we set $p'(u) = (1-\alpha) \sum_{v \in N_{in}(u)} \frac{'p\ (v)}{|N_{out}(v)|}$. The T2S score of node u is $p(u) = \{\frac{p'(u)[i]}{\sum_v s'(v)[i]}\}_i^k$.

Then, after placing u into shard i ($S(u) = S_i$), we update $p'(u) = p'(u) + \alpha s'(u)$. Overall, the computation of $p(u)$ now only costs $O(|N_{in}(u)|k)$. As TaN network has been shown to be scale-free, the average computation, thus, costs only $O(k)$.

Discussion. Why do we develop the new T2S score instead of using existing graph partitioning methods to minimize the number of cross-TX? The best way yet unrealistic to minimize the number of cross-TXs is that we know the structure of the TaN network beforehand, i.e. all transactions have been arrived, and apply graph partitioning algorithms. But even those, does such algorithm improve the performance of sharding blockchain? The answer could be not likely. In experiment, we applied a well-known graph partitioning tool, called **Metis** k-way [19], to get the partitioning on transactions. Metis aims at partitioning the graph into disjoint sets of nodes with almost equal size while minimizing the number of edges whose two endpoints belong to different sets. However, when simulating, if we put transaction exactly like in the Metis solution, the system's throughput and confirmation latency are highly impacted as the Metis solution tends to put large amount of consecutive transactions into one shard. More details of such experiments will be shown in Section V.

A more realistic method, but simple, is **Greedy**. To compare with Metis, given n txs and k shards, we set the maximum number of transactions that a shard can have as $(1+\epsilon)n/k$. Then we consider transactions sequentially. Considering a transaction u arrives at a moment t, denote $S_i^{(t)}$ as a set of transactions in shard i at time t. The greedy algorithm computes the cost of placing u on shard j as $f(u, j) = |\mathcal{S}_{in}(u) \setminus S_j^{(t)}|$. Then the algorithm places u into a shard j which has the maximum $f(u, j)$ and its size has not exceeded $(1+\epsilon)n/k$. Intuitively, the greedy solution will help reduce the number of cross-TXs. However, this solution does not take the global view on TaN network structure while only considering connection of one-hop away from u. Thus, in the long future, the performance of greedy algorithm becomes undesireable.

To prove the efficiency of T2S-score, we compare the percentage of cross-TXs between Omniledger, Greedy solution and a solution which is similar to Greedy except that after computing $p(u)$, we put u into a shard with the highest T2S-score, i.e. $\text{argmax}_i p(u)[i]$. We set $\alpha = 0.5$ and we call such

method **T2S-based**. With both Greedy and T2S solution, we set $\epsilon = 0.1$.

We use the same data set from MIT. First, we ran the algorithms from scratch where all shards are empty at the beginning. The result is presented in Table I. It is easy to see that our solution significantly reduces the number of cross-TXs and overcomes Greedy solution with a huge margin. Even Metis is unrealistic, we put its results in the table as a baseline for comparison.

TABLE I: Percentage of cross-TXs when running from scratch

k	Metis	Greedy	Omniledger	T2S-based
4	1.66 %	24.62 %	80.82 %	9.28 %
8	3.09 %	27.02 %	90.33 %	12.52 %
16	4.70 %	28.14 %	94.87 %	15.73 %
32	6.91 %	28.69 %	97.09 %	18.94 %
64	9.91 %	28.97 %	98.18 %	21.65 %

Next, we consider at a certain moment, the system already places a certain amount of transactions into shards. We then apply the algorithms with a set of new arrival transactions and compare the number of cross-TXs in such set. To be specific, first, we use Metis to partition the TaN network of 30 millions Bitcoin transactions into k shards. Then we apply the algorithm to place transactions of a sequence of next 1 millions transactions into k shards. The results is presented in Table II. Again, our solution showed its efficiency comparing with Greedy and OmniLedger in term of minimizing the number of cross-TXs.

C. Latency-to-Shard Score and Temporal Fitness

We have shown that a simple solution using T2S-score could significantly reduce the number of cross-TXs. However, this is not a ultimate goal of our algorithm design. What if there is a huge amount of transactions with the same "fittest" shard w.r.t T2S-score arriving sequentially? Trivially, putting all those transactions into the same shard is not a good solution. Therefore, in this part, we propose a mathematical model to estimate confirmation latency of a transaction under transaction sharding. We call the estimated latency *Latency-to-Shard* (L2S) score. Hence, the best transaction placement is the one that maximizes the T2S-score while minimizing the L2S-score.

Let consider a moment a new transaction u arrives and system shards are $S_1, ...S_k$. Assume if u is placed in shard j, u will need proof-of-acceptance from a set of shards $\mathcal{S}_j$. We model the communication time between a user who creates u and the shard S_i under exponential distribution $l_c^{(i)}(t) = \lambda_c^{(i)} e^{-\lambda_v^{(i)} t}$, where $\frac{1}{\lambda_c^{(i)}}$ is expected communication time, which could be collected through frequently sampling between the user and shard S_i. Also, for each shard S_i, we model the verification time of S_i under exponential distribution $l_v^{(i)}(t) = \lambda_v^{(i)} e^{-\lambda_v^{(i)} t}$, where $\frac{1}{\lambda_v^{(i)}}$ is expected verification time, which could be estimated from observation of recent consensus time of shard i and its current queue size. With high precision, it is likely that $\lambda_v^{(1)} \neq ... \neq \lambda_v^{(k)} \neq \lambda_c^{(1)} \neq ... \neq \lambda_c^{(k)}$

Therefore, a probability distributed function of time to get proof-of-acceptance from shard S_i is modeled as follows.

TABLE II: Number of cross-TXs when running from a certain stage of the system

k	Greedy	Omniledger	T2S-based
4	335,269	837,356	112,657
8	407,747	922,073	172,978
16	441,267	960,935	226,171
32	449,032	979,323	282,108
64	454,321	988,144	366,854

Algorithm 1 Transaction Sharding in OptChain

Input: $G(V, E)$, $S_1, ...S_k$, $p' : V \rightarrow \mathbb{R}^+$, α and a new transaction u
Output: a new state of $S_1, ...S_k$

```
# Compute T2S-scores of u
p'(u) = (1 − α) Σ_{v∈N_in(u)} p'(v) / |N_out(v)|
p(u) = { p'(u)[i] / |S_i| }_{i=1}^{k}
# Compute L2S-scores of u
for j = 1 → k do
    E(j) = ∫_0^∞ t ∫_0^t f_v^(j)(x) f_v^(j)(t − x) Δx Δt
end for
# Put u into shard with the highest Temporal Fitness score
s_u = argmax_i ( p(u)[i] − 0.01 · E(i) )
S_{s_u} ← S_{s_u} ∪ {u}
# Update after placing u into S_{s_u}
p'(u)[s_u] = p'(u)[s_u] + α
Return S_1, ...S_k
```

$$f^{(i)}(t) = \int_0^t l_c^{(i)}(x) l_v^{(i)}(t-x)\Delta x = \frac{\lambda_c^{(i)}\lambda_v^{(i)}}{\lambda_v^{(i)} - \lambda_c^{(i)}}\left(e^{-\lambda_c^{(i)} t} - e^{-\lambda_v^{(i)} t}\right)$$

while the cumulative distributed function is:

$$F^{(i)}(t < T) = \int_0^T \frac{\lambda_c^{(i)}\lambda_v^{(i)}}{\lambda_v^{(i)} - \lambda_c^{(i)}}\left(e^{-\lambda_c^{(i)} t} - e^{-\lambda_v^{(i)} t}\right)\Delta t = \frac{\lambda_v^{(i)}}{\lambda_v^{(i)} - \lambda_c^{(i)}}(1 - e^{-\lambda_c^{(i)} T}) - \frac{\lambda_c^{(i)}}{\lambda_v^{(i)} - \lambda_c^{(i)}}(1 - e^{-\lambda_v^{(i)} T})$$

As the user can send request for verification simutaneously to shards in $\mathcal{S}_j$, the probability that verification process is done by time T is computed as

$$F(t < T) = \prod_{S_i \in \mathcal{S}_j} F^{(i)}(t < T)$$

Thus, the probability distributed function of time for the user to get all proof-of-acceptance if place u into shard j is modeled as follows

$$f_v^{(j)}(t) = \frac{\Delta F(t)}{\Delta t} = \sum_{S_i \in \mathcal{S}_j} \frac{\lambda_c^{(i)}\lambda_v^{(i)}}{\lambda_v^{(i)} - \lambda_c^{(i)}}\left(e^{-\lambda_c^{(i)} t} - e^{-\lambda_v^{(i)} t}\right) \prod_{S_r \in \mathcal{S}_j \setminus S_i} F^{(r)}(t)$$

Similarly, we can find the probability distribution for u to get confirmation from shard j, which is:

$$f_c^{(j)}(t) = \frac{\lambda_c^{(j)}\lambda_v^{(j)}}{\lambda_v^{(j)} - \lambda_c^{(j)}}\left(e^{-\lambda_c^{(j)}t} - e^{-\lambda_v^{(j)}t}\right)$$

Therefore, the L2S-score of u if putting into shard j is computed by

$$\mathcal{E}(j) = \int_0^\infty t \int_0^t f_v^{(j)}(x) f_v^{(j)}(t-x)\Delta x \Delta t$$

In overall, for a given transaction, we would like to balance between the T2S and L2S-score. Specifically, given a new arrived transaction u and a shard j, we define the *Temporal Fitness* score between u and j as $p(u)[j] - 0.01 \cdot \mathcal{E}(j)$. OptChain then places u into the shard whose has the highest Temporal Fitness score. The procedure of transaction sharding in OptChain is presented in detailed by Alg. 1.

V. Experiments

The aim of this experiment is to evaluate the impact of OptChain on Blockhain system's latency and throughput. To demonstrate the strengths of our proposed solution, the performance of OptChain is rigorously compared to OmniLedger, Metis, and the Greedy heuristic discussed in section IV.

A. Experiments Design and Configuration

The experiments presented in OmniLedger [8] used the first, arguably simple, 10,000 Bitcoin blocks which contain only 10,093 transactions in total. Additionally, of those transactions, 99.1% are coinbase, which cannot be cross-TXs. At that point in time, Bitcoin was still in an initial phase in which there was only about 1 transaction per block and most of them were coinbase transactions. Therefore, their experimental results do not represent the impact of cross-TXs.

Our experiments are conducted on the first 10 million Bitcoin transactions, taken from the dataset in [20]. From this dataset, we build a TaN network that contains 10,000,000 nodes and 19,958,051 edges. As opposed to OmniLedger, our dataset is much bigger with more chance of creating cross-TXs. Specifically, with 16 shards, the OmniLedger's random placement produces 9,349,979 cross-TXs comprising 93.5% of the whole set.

We develop a sharding-based blockchain simulation and run the transactions placement algorithms to calculate the latency and throughput. Particularly, we use the OverSim framework [22] to simulate a Bitcoin-like blockchain system on OMNeT++ 4.6 [23], which is a discrete event-based network simulator. The bandwidth of all connections between nodes is set to 20 Mbps and a latency of 100 ms is imposed on all communication links. We set the block size of 1MB because this is currently the block size limit in Bitcoin. The average size of a transaction is about 500 bytes, so we put about 2000 transactions in one block. A shard is assigned with about 400 validators and one leader that are randomly placed at different coordinates. In our simulation, the distance between nodes affects the communication latency. Each shard implements a queue (or mempool) to store incoming transactions that have not been processed yet.

Our simulation involves a set of clients that continuously issue transactions from the dataset to the system at a predefined rate. We re-implement the mechanism for handling cross-TXs as decribed in Section III. Before sending a transaction u for verification, a client will run a transactions placement algorithm to determine the shard $S(u)$ to place that transaction.

TABLE III: Experiment configuration

Number of transactions	10,000,000
Block size	1 MB
Transactions per block	2,000
Network bandwidth	20 Mbps
Number of shards	4, 6, 8, 10, 12, 14, 16
Transactions rate (tps)	2000, 3000, 4000, 5000, 6000
Algorithms	OptChain, Metis k-way, OmniLedger, Greedy

Getting rid of Omniledger's bottleneck. In Omniledger, users gossip each transaction to all nodes in the network, thus, all nodes must receive the whole blockchain (and also the unconfirmed transactions, if any). Thus, even when the number of nodes and shards go to infinity, the throughput of nodes in Omniledger is still limited by typical network bandwidth of the nodes. To address this issue, users will direct each transaction directly to the input shards and output shards of the transaction.

Transaction placement strategies. We compare OptChain with three transaction placement methods.

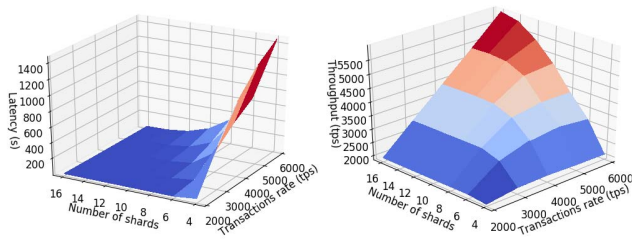
- OmniLedger: The default random placement strategy in Omniledger.
- Greedy: Greedy placement strategy, discussed in Section IV-B.
- Metis k-way: Running (offline) Metis partitioning algorithm to partition the network into k shards.

As Metis k-way is an offline algorithm, it is not a realistic transactions sharding scheme. However, if we can put transactions as in Metis solution, we can minimize the number of cross-TXs. Our purpose is, therefore, to see whether only minimizing the number of cross-TXs can improve the system performance. Therefore, we first input the whole TaN network to get its Metis solution and then use the resulting partitions to determine $S(u)$ for each transaction u.

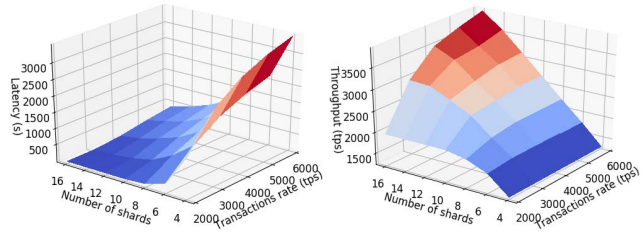
We carry out the experiment with various configurations by combining different number of shards and transaction rates. Specifically, we vary the number of shards from 4 to 16 to conform with the experiments presented in OmniLedger [8]. With regard to the transactions rate, which represents the rate at which transactions are sent to the system, we refer to the VISA-level throughput of 4000 transactions per second (tps) to set the transactions rate from 2000 tps to 6000 tps. Table III summarizes all the parameters and configuration of the experiments.

B. Experimental results

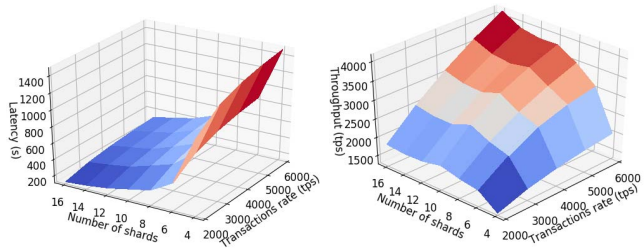
In Fig. 3, we summarize the system's average latency and throughput when running the algorithms with different combination of transactions rates and number of shards. The thoughput is calculated by taking the number of transaction divided by the total time for all transactions get committed. The latency of a transaction is measured by the time from when the transaction is sent until it is committed to the blockchain. In this part, we would like to have more insight on how different configuration impacts the performance of OptChain as well as other transaction sharding methods in term of throughput and latency.



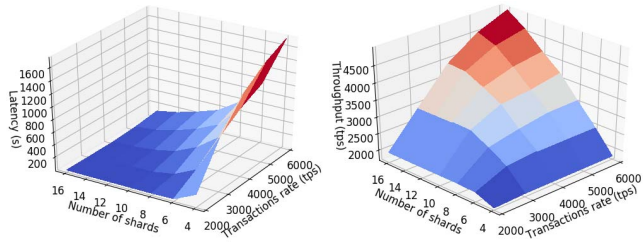
(a) OptChain



(b) OmniLedger



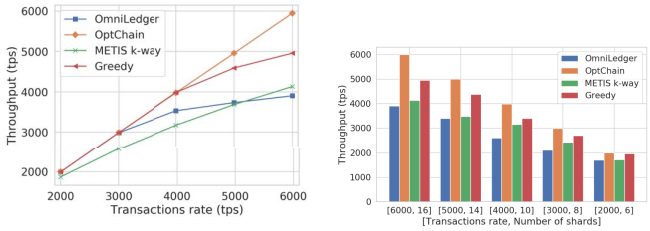
(c) METIS k-way



(d) Greedy

Fig. 3: Impact of different transactions rates and number of shards on the latency and throughput

1) Maximum Throughput: From Fig. 3, it is easy to see that all the methods achieve their highest throughput at transaction rate of 6000 and 16 shards. However, except OptChain, the other three algorithms produce throughput much lower than the input rate, which shows that these three are incapable of handling such transaction rate in this setting. We observe that even OptChain does not always run well in all configurations, i.e. throughput is lower than transaction rate. But for each value of transaction rates, OptChain always has a certain configuration on the number of shards guaranteeing no backlogging in the system. To be specific, with a transaction rate of 2000, OptChain is totally healthy with at least 6 shards. This number in transaction rates of 3000, 4000, 5000, 6000 is 8, 10, 14, 16 respectively. Meanwhile, the Omniledger system needs at least 16 shards to be able to process up to 3000



(a) Number of shards = 16 (b) Varying transactions rate and #shards

Fig. 4: System throughput

transactions per second. With 16 shards, Greedy is only able to process with transactions rate up to 5000. Therefore, given a same transaction rate, OptChain needs a lower number of shards than other methods to avoid backlogging in the system.

For a more comprehensive comparison, Fig. 4b presents the maximum system throughput at different pairs of value of transactions rate and number of shards. As can be seen, no other transaction sharding methods can reach up to the same level as OptChain. For example, the maximum throughput OptChain can achieve at 16 shards is 34.4%, 30.5%, and 16.6% higher than that of OmniLedger, Metis, and Greedy, respectively.

Metis has been proven to be the best method, yet unrealistic, to distribute transactions into shards in order to minimize the number of cross-TXs. High number of cross-TXs is claimed by Omniledger [8] to be the main factor limiting the system performance. However, when we place transactions as in Metis's solution, the throughput never inline with transaction rate. For example, Fig. 4a shows the throughput of the algorithms as we fix the number of shards to 16 and vary the transactions rates. As previously stated, OptChain, Omniledger and Greedy are comfortable to a certain rate within such range. But Metis's throughput never reaches the transaction rate. Thus, we conclude that *high number of cross-TXs is not a sole factor hindering Blockchain sharding performance.*

What cause Metis's throughput being worse than other three solutions? We take more insights on the timeline that transactions are finally committed. We set the transaction rate to be 6000 and 16 shards. Then we count the number of transactions getting committed in each 50-seconds period and plot the results as in Fig. 5. It is easy to see that OptChain, OmniLedger, and Greedy produce almost consistent number of committed transactions over each period of 50 seconds. Meanwhile Metis is not efficient during the first 500 seconds. Also, Metis tends to agitate more than other three solutions, which could imply congestion on shards in some certain moments, i.e. some shards get more transactions than the other. The huge drop in the end of each line in Fig. 5 is because the simulator has reached the end of the dataset in which no more transactions are sent to the system.

To observe the congestion, we plot out the change on queue sizes of shards under four sharding methods as in Fig. 6. The congestion on Metis method is from the fact that Metis tends to put a large amount of consecutive transactions into one same shard. Thus, the pattern, in which some shards are overwhelmed while the others have no transactions, happens frequently in Metis. Greedy also met a situation that some

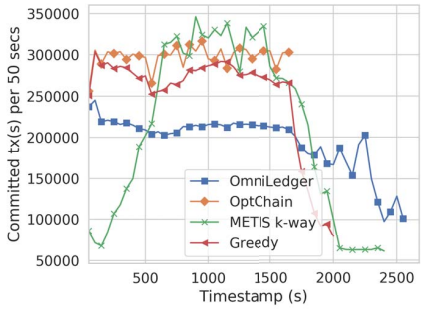


Fig. 5: Number of committed transactions across time

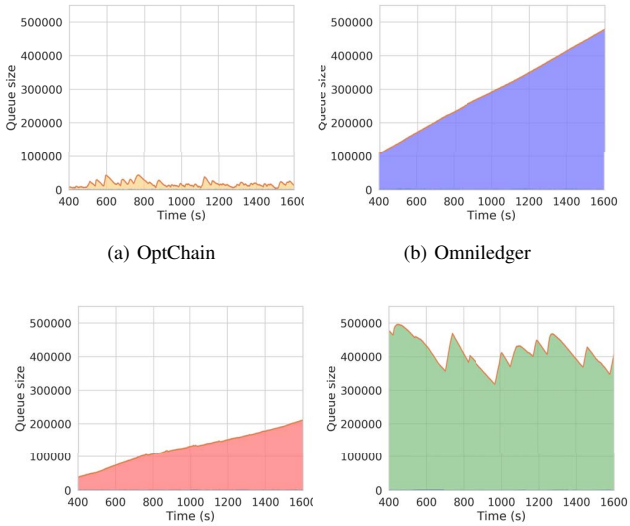
shards have no transaction in several moments but in overall, transactions are splitted more equally than Metis among remaining shards. Such event does not happen in OptChain and OmniLedger. However, as OmniLedger is not capable of running with a transactions rate of 6000 tps, shards queue size will increase linearly overtime. OptChain performs the best in term of load balancing among shards. We can see that both the maximum and minimum queue size in OptChain are consistent and stable. Also, in worst case, a queue size in OptChain only reaches up to $\approx$ 44,000 transactions while these numbers in Metis, Greedy and OmniLedger are 507000, 230000 and 499000 respectively.

To have a clearer observation of this behavior, we calculate the ratio between the maximum and minimum queue size of each algorithm at each simulation timestamp and compare them. Fig. 7 illustrates this comparison where we can clearly see how inefficient the Metis and Greedy are in terms of temporal balance. Being unable to handle the load balancing among shards not only affects the throughput negatively but also notably increases the system's average latency as we shall see in the following experiments.

2) Transaction Latency: Next, we compare the average (maximum) transaction latency, i.e., transaction *confirmation time*, among the different sharding approaches. As demonstrated in Fig. 3, all four sharding methods share the same behavior: at a certain transaction rate, the average transaction latency decreases significantly when the number of shards increases. Therefore, all four methods performs their best in term of latency with the configuration of 16 shards as the number of transactions in each shard is low enough.

To closely evaluate the latency, we varied the transaction rate while keeping the number of shards at 16 as shown in Fig. 8a. Clearly there exists a considerable gap between OptChain's average latency and other methods' results. OptChain always achieves the best performance comparing to the others. In fact, at the transactions rate of 4000 tps, the system only takes 8.7 seconds to process a transaction in average. At low transactions rate (i.e., 2000 to 3000 tps), we can see that all algorithms except for Metis have good latency in general. This is because the system can balance between throughput and transaction rate at these ranges, thus no backlogging happens.

As we increase the rate to 6000 tps, our algorithm still maintains a good latency, while we can see a significant increase in other algorithms. In particular, at 6000 tps, we reduce up to 93% the latency comparing to the OmniLedger.



(a) OptChain (b) Omniledger

(c) Greedy (d) Metis k-way

Fig. 6: Maximum and minimum queue size of shards over time

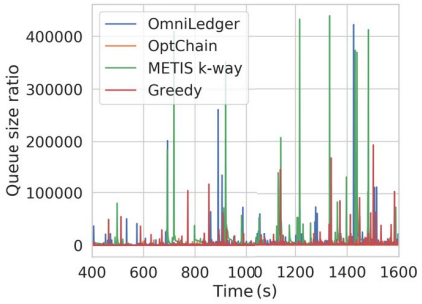
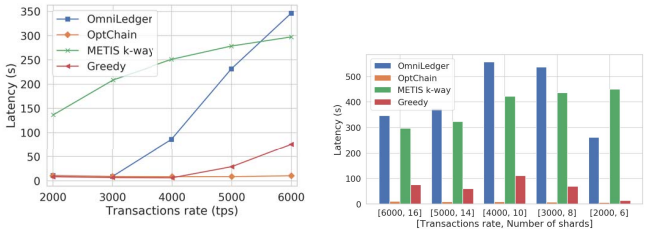


Fig. 7: Queue size ratio

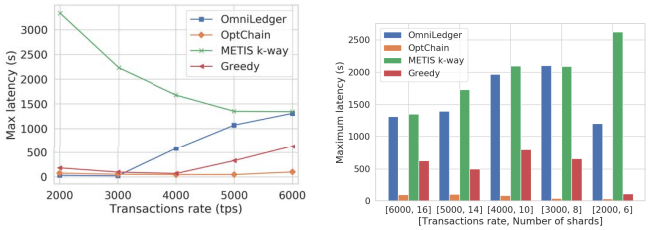
This comes from the fact that OmniLedger can not tolerate 6000 transactions per second in such settings. Thus, the queuing delay (time staying in queue) of a transaction will increase overtime and come to infinity. This behavior can also be observed in Greedy. As for Metis, even though it has the least amount of cross-shard transactions, it still gets really high average latency. This issue can be explained as Metis failed to achieve the temporal balance in queue size between shards as we mentioned above, causing there were only some active shards at a time and exacerbating the final average latency.

Next, we measure the system's latency with different combinations of transactions rate and number of shards as in Fig. 8b. Specifically, we set the configuration in the same way as Fig. 4b. Again as can be seen in Fig. 8b, OptChain obtains the best performance in term of balancing throughput and transaction rate. At these configurations, the highest average latency of OptChain is only 10.5 seconds at the transactions rate of 6000 tps and 16 shards, while OmniLedger reaches 346.2 seconds at this configuration. In addition, at the transactions rate of 4000 and 10 shards, OptChain reduces up to 98% the latency as compared to OmniLedger.



(a) Number of shards = 16 (b) Vary transactions rate and #shards

Fig. 8: Average transaction latency



(a) Number of shards = 16 (b) Vary transactions rate and #shards

Fig. 9: Maximum transaction latency

Fig. 9a illustrates the maximum latency to process transactions with 16 shards at different transaction rates. At 6000 tps, OptChain takes at most 100.9 seconds for a transaction while OmniLedger, Metis, and Greedy respectively takes 1309.5, 1345.9, and 628.9 seconds. This result, together with Fig. 8, again confirms that our algorithm significantly reduces the transaction processing latency. Additionally, in Fig. 9b, we also plot the result of this metric at different combinations of transactions rate and number of shards as in the previous Fig. 8b. Among these configurations, the highest latency we ever reach is 102.7 seconds at the transactions rate of 5000 tps and 14 shards. At this point, OmniLedger, Metis, and Greedy respectively takes at most 1397.0, 1730.0, and 497.0 seconds to process one transaction.

For a more detailed view on the impact of algorithms on system's latency, in Fig. 10, we present the cumulative distribution of the latency when we set the shards number to 16 and transactions rate 6000 tps. As can be seen, up to 70% of the transactions are processed within 10 seconds with OptChain. For other algorithms, within this time frame, only 41.2%, 7.9%, and 2.4% of the transactions were completed with Greedy, OmniLedger, and Metis, respectively.

C. Summary of results

The experiments conducted in this section notably show that the proposed OptChain significantly outperforms other transaction sharding methods. To be specific, OptChain could reduce up to 93% the latency and increase 50% the throughput in comparison to OmniLedger. Furthermore, OptChain is more scalable than OmniLedger where it can scale up to the rate of 6000 transactions per second and 16 shards. Meanwhile, for a certain transaction rate, OptChain requires less shards than OmniLedgers to guarantee the system performance without backlogging. In overall, Optchain's good performance comes

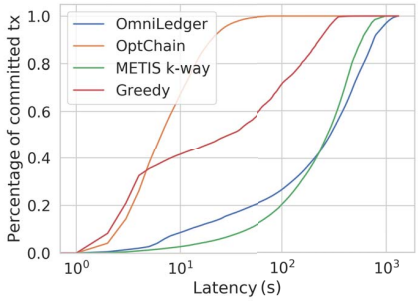


Fig. 10: Latency distribution

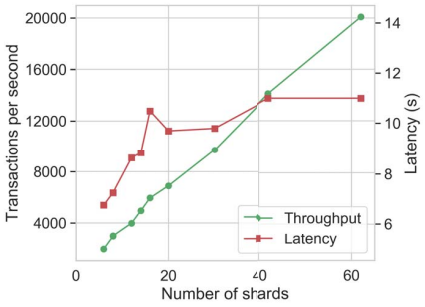


Fig. 11: OptChain scalability

from the fact that OptChain is able to concurrently handle two main factors that hinder the system performance: (1) reducing cross-TXs and (2) temporally balancing workload between shards.

The highest transaction rate OptChain can scale up to (throughput is equal to transaction rate) with multiple number of shards is plotted out as in Fig. 11. The best throughput of OptChain is almost linear with the number of shards and it can reach above 20,000 tps with 62 shards. More importantly, when the throughput is comfortable with transaction rate, OptChain guarantees that the confirmation delay is never more than 11 seconds.

VI. Conclusion

Handling cross-shard transactions has been a major challenge in research on blockchain sharding as they are the main factor that limits the scalability of the system. In this paper, we have proposed OptChain, a scalable protocol to optimally place transactions into shards, which guarantee reducing number of cross-shard transaction as well as temporally balancing workload between shards. In experiments, OptChain reduces the latency by 93% and increases the throughput by 50% in comparison with the state-of-the-art sharding system, OmniLedger. Finally, our empirical evaluation demonstrates that OptChain scales smoothly with transactions rate up to 6,000 transactions per seconds with 16 shards and potentially handle much higher rate with more shards.

Acknowledgements

This work was supported in part by NSF EFRI-1441231, NSF CNS-1814614, and DTRA HDTRA1-14-1-0055.

REFERENCES

[1] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] Ethereum, "ethereum/sharding." [Online]. Available: https://github.com/ethereum/sharding/blob/develop/docs/doc.md

[4] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[5] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, "Internet of things, blockchain and shared economy applications," *Procedia computer science*, vol. 98, pp. 461–466, 2016.

[6] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control," *Journal of medical systems*, vol. 40, no. 10, p. 218, 2016.

[7] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[8] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger." *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.

[9] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: A fast blockchain protocol via full sharding."

[10] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, p. 2, 2015.

[11] D. Kondor, M. Pósfai, I. Csabai, and G. Vattay, "Do the rich get richer? an empirical analysis of the bitcoin transaction network," *PloS one*, vol. 9, no. 2, p. e86197, 2014.

[12] D. George and S. Meiklejohn, "Centrally banked cryptocurrencies," in *Network and Distributed System Security Symposium 2016*, 2015, pp. 1–14.

[13] 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*. ACM, 2016, pp. 17–30.

[14] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

[15] "Sharding introduction rd compendium," https://github.com/ethereum/wiki/wiki/Sharding-introduction-RD-compendium, 2018, accessed: 2019-01-12.

[16] "Monoxide: Scale out blockchain with asynchronized consensus zones," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, 2019.

[17] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1222–1230.

[18] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: an experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

[19] G. Karypis and V. Kumar, "Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[20] "Bitcoin network dataset," https://senseable2015-6.mit.edu/bitcoin/, 2018, accessed: 2019-01-12.

[21] J. Pearson, "Wikileaks is now a target in the massive spam attack on bitcoin," Jul 2015. [Online]. Available: https://motherboard.vice.com/en_us/article/ezvw7z/wikileaks-is-now-a-target-in-the-massive-spam-attack-on-bitcoin

[22] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007, pp. 79–84.

[23] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and , 2008, p. 60.