A Decentralized Sharding Service Network Framework with Scalability

Shubin Cai, Ningsheng Yang, and Zhong Ming(✉)

Nation Engineering Laboratory for Big Data System Computing Technology, CCSE, Shenzhen University, Shenzhen City 518060, Guangdong Province, China
{shubin,mingz}@szu.edu.cn, ningsheng.yang@qq.com

Abstract. Blockchain is a decentralized distributed service network framework which ensures the consistency among service nodes in byzantine faulty network. However, the very restricted performance and enormous energy consumption makes blockchain faltering. In this paper, we proposed a sharding blockchain framework with linear scalability. Unlike other frameworks with sharding, our model needs no centralized organization to assemble messages from subcommittees. We also redesigned the block-generating algorithm to accelerate generating block. Our framework can reach nearly linear scalability with scale of service network while tolerating no more than 1/4 adaptive byzantine adversaries. Furthermore, we designed simulation experiments with up to 1000 virtual nodes to proof our theoretical scaling properties. In our experiments, our model performs better than Bitcoin-NG when the size of network more than around 400 nodes. But delay in Bitcoin-NG grows far more than ours. Also, our framework wins out over ELASTICO while our simulation platform contains less than 1000 nodes.

Keywords: Blockchain · Sharding · Ring paxos

1 Introduction

Blockchain, introduced firstly in Bitcoin [1], is a decentralized distributed service network framework which ensures the consistency among service nodes in byzantine faulty network. The significate difference between centralized distributed service network and blockchain embodies that blockchain has no autocratic supervisor and no one can deny the executed operations. A great number of virtual currency application, certification platforms and Internet of Things (IoTs) platforms have been deploying the blockchain infrastructure [35–39]. They hope to reduce the cost of maintaining centralized system via blockchain.

1.1 Workflow in Blockchain

The workflow of blockchain can be described as follows. Clients broadcast requests to all service nodes and then service nodes validate these requests

H. Jin et al. (Eds.): ICWS 2018, LNCS 10966, pp. 151–165, 2018.
https://doi.org/10.1007/978-3-319-94289-6_10

according to its local database independently. After that, every service node chooses several executable requests and tries to capsule them into a block following the given rules. Block contains two parts called header and body respectively. Header mainly records a hash value of previous block, the digest of body, timestamp and a special random string called nonce. Sequence among blocks is decided by hash value instead of global timer. The body in the block contains several requests chosen by service node.

If a server received a legal block during generating-block, it would cease packaging and append the received block to its local chain. After that, server will restart packing following the new block. However, A dilemma, called fork, may emerge occasionally when one server received several legal blocks in a very short time. Fork makes disagreement among servers and damages the consistency of system. The consensus mechanism in blockchain with one-way linked list rules that the longest chain (contains the most number of blocks) is legal and the short ones will be discarded. Traditionally, servers only acknowledge the first-received block. When more than half servers in blockchain network agrees on all blocks, the system is correct.

1.2 Problems and Challenges

By now, the most attractive research on blockchain includes security and scalability. The security problem in blockchain mainly focus on cryptography and attacks in network including DDoS, sybil attack and double-spending [8,9]. The system state in decentralized blockchain is decided by voting from all servers. But propagation delay and malicious nodes in the network may cause disagreement. Many researchers have been contributing their painstaking effort to find solutions. A significate discovery was found by Eyal Ittay and Emin Gün Sirer whose paper [5] pointed out that the original Bitcoin consensus mechanism ignores the propagation delay. The propagation delay may misdirect some honest nodes accept wrong blocks from malicious nodes. Eyal Ittay and Emin Gün Sirer stipulated that servers should randomly select a legal block instead of adopting the first one. Also, this new rule indicates that majority (more than half) is not enough, the percentage of malicious nodes in the blockchain network must no more than a quarter.

Meanwhile, the scalability of blockchain also draws scholars to explore. There are three main factors constrain the throughput. Firstly, the size of block and the form of chain greatly restrict the throughput in decentralized environment. The most intuitive optimization into our mind is expanding the capacity of block. The bigger block contains more requests. However, the expansion protocols in Bitcoin [16,17] arouse fierce controversy in community. Protesters insisted that fat block would occupy more time in transmission and increase the risk in data corruption. Thus some scholars turned to alter the shape of chain. Yonatan Sompolinsky and Aviv Zohar put forward the tree [6] and DAG [7] to replace the chain successively. In each round, the blockchain in form of tree or DAG can build more than one block. However, the sequence of blocks with the same precursor can hardly be decided owing to erratic transmission delay. The view on

these blocks may not same and therefore, undermines the consistency in system. The another shortcoming is that some requests may be packed into multiple blocks. Therefore, the pattern of reforming chain thirsts for a global sorting algorithm.

Secondly, there's no centralized node in blockchain which requires clients sending requests to every server. In high-traffic scenarios, the large-scale broadcasting will rise a broadcast storm and make system unable to respond. Decrease the scale of sending message may relieve the pressure of the system. An effective method to reduce the sending message is sharding who divides requests into several sets and maps these requests to the corresponding committees. Although requests in different sets are disjoint, these requests may still be contradictory in execution. Hence the order of requests needs to be unified.

The last pivotal element is communication protocol in byzantine generals network. Since byzantine generals problem proposed by Lamport et al. in [10], many researchers have been devoting their efforts to enhancing the scalability in byzantine consensus protocols [19–26]. Discussion in byzantine generals problem can be divided into two different situations. The ideal one is that every node in network is honest but crash-prone. The optimal algorithm can reach $O(n)$ communication complexity via electing a special subcommittee to administer the whole net [11]. Another situation seems very undesirable and challenging because of spiteful nodes in the net. For many years, many scholars had proposed some protocols with exponential communication complexity [12]. A significate enhancement was made by Srikanth et al. [13], whose method reaches polynomial communication complexity. In addition, some researchers pointed out that state-machine replication (SMR) is fundamental approach to building fault-tolerant distributed system [27]. They introduced atomic broadcast [28–33] as an important communication primitive. Despite the large number of atomic broadcast algorithms had been proposed in byzantine network. However, few of them works in leaderless conditions [34]. There has great potential in applying atomic broadcast to decentralized service network.

In short, the methods to enhance the performance concentrate on two directions.

Leader Election. Leader can determine the global order of requests which can reduce the overhead. For example, Bitcoin-NG [14]. In fact, the common consensus mechanism, such as Proof of Work (PoW) [1] and Proof of Stack (PoS) [4], used in the traditional blockchain are also forms of the leader election. There are two problems threaten this framework. One is how to select leader fairly and another is how to ensure system consistency when multiple leaders were selected. There are more or less centralization problems because of the existence of autocrat. For example, PoW has the problem of computing centralization.

Sharding. Sharding amortizes heavier communication task into two steps—inner committee and among committees. After sharding, different groups in the

system can independently select the disjoint requests from the clients and package them into blocks. The biggest advantage of sharding is that different groups can generate blocks independently which improves the throughput of the system. However, sharding also has two problems. First of all, we cannot guarantee that every committee is reliable because the percentage of malicious nodes may exceed 1/2 in some groups. Another threat is how to achieve the agreement on sequence of blocks among servers. The representative architecture contains ELASTICO [2], Ethereum [15], etc.

1.3 Contribution

In this paper, we propose a scalable distributed framework based on sharding and atomic broadcast protocol. We innovatively introduce and reform ring paxos [3] to generate blocks instead of PoW. Furthermore, we shrink the scalability of inputs that clients need only broadcast its requests to members in a committee. Our framework can reach nearly linear scalability with the number of service nodes while tolerating no more that 1/4 adaptive byzantine adversaries.

In order to verify that our framework performs better than ELASTICO and Bitcoin-NG, we built a simulation experiment with up to 1000 nodes. Because we don't use PoW in generating blocks, our frame performs much better than ELASTICO. Also, our model works better than Bitcoin-NG while the scale of network exceeds around 400 nodes.

2 System Framework

Before formally introducing our model, we firstly explain the terms and some basic assumptions in Sects. 2.1 and 2.2. Then we will elaborate the workflow and details in our model (Sect. 2.3 to Sect. 2.6).

2.1 Terminology Interpretation

Definition 1 (Group). *We divide servers into several groups with fixed number of members (the size of committee is C). If the proportion of malicious nodes in a group exceeds 1/2, we label this group with trustless. On the contrary, the group will be marked as reliable.*

Definition 2 (Working State and Division State). *When some committees reach a predetermined threshold and all committees have worked for long enough time, system will stop working and enter the division state briefly. Meantime, the original group will be split into at most two subgroups. After redistribution, each group will return to working state.*

2.2 Assumption

Clients and Requests. Provided that all clients have their own global unique identifiers (GUID), public and private key pairs. Every request will be encrypted and cannot be tampered.

Servers. Assuming that all service nodes also have GUID and key pairs. Each node is either honest or malicious. Furthermore, All honest nodes do not go offline forwardly and when the honest one off line, he will eventually go online again as soon as possible. The percentage of malicious nodes in our system never exceeds 1/4.

Network and Communication. Assuming that the network is asynchronous which means that message may be disordered, retransmitted, and lost during transmission. But the content of message will not be tampered. Each node can communicate directly with other nodes and clients.

Committees. Every committee is either reliable or trustless. The proportion of malicious nodes in reliable committee is less than 1/2. The First group in our system called $group_0$ which is reliable. Each committee initialised with C members. The number of members in normal group ranges from $\lceil \frac{C+1}{2} \rceil$ to $2C$. The committee contains no more than C members needs at least $\lceil \frac{C+1}{2} \rceil$ ballots to pass proposals while the bigger group needs more than half ballots.

2.3 System Synopsis

Dividing servers into pieces of committees is our kernel conception. As we mentioned before, the decentralized framework with sharding needs a global sort algorithm to array blocks. Therefore, we introduce a tree structure to decide the sequence of groups. Each node in this tree corresponds to a segment of chain and the group responsible for maintaining it respectively. Through this tree, we can sort committees by tree traversal algorithm.

Workflow in working state mainly contains three steps. First of all, each group independently makes up blocks following atomic broadcast protocol. Second, leaders in all groups exchange blocks between each other and synchronize blocks within their group. At last, all service nodes independently proofread blocks and write the executable them into their local database. The form of blockchain can be depicted as Fig. 1.

When some groups reached the predetermined threshold, they will make an appointment on when shall splitting and inform this timing to others. In our system, the obese group (size of $2C$) will be split into at most two pieces with size of C. After synchronizing blocks, other groups will know the alteration and update their view. After the division, the original group ceased and the segment of chain it maintained will be sealed.

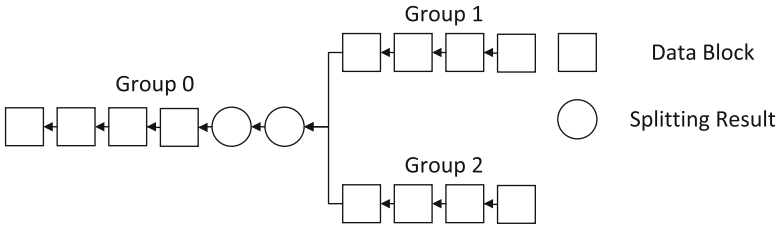


Fig. 1. The form in our blockchain framework

2.4 Generating Blocks

Different from the traditional method in generating blocks, we exploit atomic broadcast to build blocks inner group. Atomic broadcast can eliminate divergence and guarantee the consistency within cluster, so we introduce atomic broadcast as the communication protocol of block generated inside of the group. In this paper, we use a variant algorithm of ring paxos. Algorithm 1 presents variant ring paxos.

Algorithm 1. Variant Ring Paxos

```
1: Task1 (Leader)
2: // Upon received enough requests R
3: b = generateBlock (R)
4: broadcast (b, others)
5: let ring be the overlay ring within the committee
6: Task2 (AllNodes)
7: // Upon receiving b
8: if first (ring) then
9:     initiate (poll)
10:    append (poll)
11:    send (successor, poll)
12: end if
13: Task3 (AllNodes)
14: // Upon receiving send (successor, poll)
15: if not last (ring) then
16:    append (poll)
17:    send (successor, poll)
18: else
19:    broadcast (poll, others)
20: end if
```

In Algorithm 1, we introduce additional two variables. *Ring* is a sequence deciding the order of members within group. *Poll* contains every node's opinion on whether or not to execute requests. Notice that *poll* is append-only and encrypted by all members. Furthermore, leader is not only the first node in the *ring* but also the last one.

First of all, leader packages some requests into a block and broadcasts it to others. After that, leader will rise a *poll* and send it along with the *ring*. While receiving a *poll*, node will attach its view and encrypt it. When receiving the poll, leader will broadcast the it to other members. After that, each node will check the *poll* and execute the requests who earns more than half votes.

With further observation we can find that variant ring paxos requires a leader and a *ring* in each round. We must offer an equitable means to elect a leader and build a *ring*. When the committee was born, a special block, called synchronizing signal, was found. This special block is unpredictable. Therefore, we can use seed to generate the first leader in new group. Building *ring* is actually much easier. We can harvest several different rings by revolving the *ring* with different header (e.g., Fig. 2). However, leader may collapse during his work. There are several anomalies during generating block.

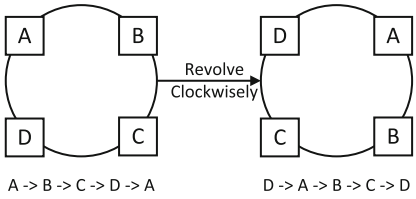


Fig. 2. Revolving original ring generates new ring

1. Leader failed in packing & broadcasting requests. Leader may collapse or refuse to do his duty deliberately and his committee had to create an empty block. When the number of nodes is less than $C/2$, this group cannot perform function as well and perish. The rest nodes will re-join the network and enter into other group.
2. Leader failed in passing poll. When leader failed in sending poll to his successor, the successor will rise as new leader to finish the job when the quorum lives.
3. Leader failed in broadcasting poll. The second last node in the ring will perform as the leader to complete current round.
4. Other nodes failed in sending poll. The failed node will be skipped. If the group still contained more than $C/2$ members, then this group still worked.

2.5 Intra-committees Synchronization

After executing Algorithm 1, leader will send his block and poll to other leaders. When leader receiving blocks with their polls, he will broadcast to his mates. As we mentioned before, we introduce a tree structure mapping groups to the nodes in the tree. So we can use tree traversal algorithm to make an agreement on the sequence among groups.

After synchronizing blocks, a vital task is to elect leaders in next epoch. It is obvious that purely use the hash value of block built in their group is unfair

because the current leader can elaborately choose requests and find a special combination to achieve reappointment. But the blocks from other committees are not all predictable. Therefore, we re-organize the blocks along with the order of groups and build a Merkle Hash Tree. The hash value inside the root, called *seed*, will decide who is the next leader.

Also, leader may offline in this step. We must develop a policy to handle the accidents.

5. Leader failed in synchronization among groups. If leader failed in synchronization, then other groups would regard the result from failed leader as an empty block and the block failed to be synchronized will be discarded. However, the group with failed leader cannot take step with system due to failure in synchronization and they need to elect a temporary leader to complete synchronization.
6. Leader failed in synchronization in his own group. Every node in this group doesn't know who is next leader and this committee can only generate an empty block in next epoch. Then, this tricky problem converts into situation 5.

Apparently, malicious leader can deliberately make up divergence between committees. So we need two round broadcast to erase disagreement. In the first round, leader will send and receive blocks and polls with each other. After enough interval, every leader will broadcast his absence and try to fulfill the hollow. In ideal environment, our model needs only two times broadcast in this synchronization.

2.6 Splitting Committee

Splitting committee also requires fairness. Hence we reuse the *seed* mentioned in Sect. 2.5 to split groups. The jobs in splitting committee are checking the quality of nodes, slicing obese groups and building ring in new groups.

Different from generating blocks, splitting committee is the most wasteful operation in our framework because we require all servers executing PoW to proof their authenticity. Because evil forces can rise sybil attack with fabricating thousands of virtual service nodes. When a node finding an answer which is up to the mark then he broadcasts his answer to the others in original group. When receiving PoW results from other nodes, server will independently sort them.

Committees who reach the peak will be sliced into at most two pieces with size of C. So leader in this round will generate at most two synchronization signal. Nodes who fail to establish a group will be discarded and re-join the network. This mechanism also ensures the quality of servers in service network. Algorithm 2 presents splitting committee. However, it is not in a hurry to impart this modification to clients. Supposing that clients don't know the alteration and send their requests to the elapsed group. The members in original team, scattered into two new teams, will reply clients with modification. Because newborn groups inherit their parental rules and extend them. The requests mapping to the old group must fall into a group derived from its parent.

Algorithm 2. Splitting Committee

```
    Task1 (AllNodes)
 2: answer = PoW (seed)
    broadcast (answer, others)
 4: Task2 (AllNodes)
    // Upon receiving answers from othersnodes
 6: sort (results)
    Task3 (Leader)
 8: groupNumber = 0
    // Upon receiving C answers marked as A
10: while groupNumber < 2 do
        b = generateBlock (A)
12:     Variant Ring Paxos (b)
        ++ groupNumber
14: end while
    Task4 (AllNodes)
16: // Upon receiving a synchronizing signal A
    ring = sort (A)
18: updateGroup ()
```

After receiving synchronizing signal, the next task is establishing a ring in each newborn committee. Because the answer from every node is unpredictable. Therefore, the sorted sequence of answers will be regarded as the sequence of nodes in their new ring.

3 Analysis of System

In this section, we present the analysis for how our framework prevents potential threats and works securely. Also, we will explain the value of some key parameters in our paper.

Without loss of generality, we assume that the network contains N servers who have equivalent resource allocation. The rate of malicious nodes in our system is $f(0 \leq f < 1/2)$ and the number of malicious nodes is $N_m(N_m = N \cdot f)$. The normal capacity of group is C and the number of groups is K. To make it easier, we suppose $N = C \cdot K$. Other assumptions have already been introduced in Sect. 2.2.

3.1 Security Analysis

System security defined as the rate of trustless groups in our system will never reach $2f$ while the proportion of malicious nodes in the system less than f.

Proof Sketch. We use N_m^1 and N_m^2 to describe the number of malicious number in trustless committees and in reliable committees respectively. It's obvious that $N_m = N_m^1 + N_m^2 (0 \leq N_m^1, N_m^2 \leq N_m)$. Malicious power will try to maximize the divergence among nodes. The only method to contaminate honest node is

occupying more than half sets in the committee and misdirect the minority of honest ones to follow the evil. The target of the evil can be marked in Eq. (1).

$$max\{\lfloor \frac{N_m^1}{\lceil \frac{C+1}{2} \rceil} \rfloor \cdot C + N_m^2\} \tag{1}$$

We can find that Eq. (1) reach the peak when $N_m^2 = 0$ and the maximum value is $\lfloor \frac{f \cdot C}{\lceil \frac{C+1}{2} \rceil} \rfloor \cdot C$. The limitation of the first factor in maximum value, in Eq. (2), shows that the rate of trustless groups in our system will never reach $2f$ while the proportion of malicious nodes in the system less than f. Therefore, the security of system can be guaranteed with no more than half trustless groups which requires $f < 1/4$.

$$\lim_{C \to \infty} \frac{f \cdot C}{\lceil \frac{C+1}{2} \rceil} = 2f \tag{2}$$

3.2 Performance Analysis

In this section, we will analyze three kernel index in decentralized service network framework including throughput, delay and number of message. Also, we make a compare with Bitcoin, Bitcoin-NG and ELASTICO. In our analysis, we weigh these index within a single interval, the expectancy time of PoW, when system operates steadily. Meanwhile, we unify the format of block in different framework and the size of committee. Furthermore, there's no collapse and no malicious behavior in system.

Delay. Intuitively, delay in our paper means the time from packing block to reach agreement in the system. Given a time t and a ratio $x(0 < x \leq 1)$, the x points the smallest time difference y that at least $x \cdot N$ of the nodes at time t report the same state prefix up to $t - y$. Formally, the (ϵ, δ) consensus delay of system is the ϵ-percentile δ-point consensus delay. We require that at least 75% of nodes agree on the state of system during at least 90% of the time. The delay in Bitcoin can be regarded as the sum of time expenditure of PoW and the upper quartile in propagation delay. However, the delay in Bitcoin-NG can be described as only the upper quartile in propagation delay because there are multiple blocks generated in leader's term of office. The situation in ELASTICO is more complex. In each period, ELASTICO requires all nodes to finish PoW proofing themselves and then broadcast their proof to establish committees. After that, ELASTICO exploits PBFT [18] to generate block within each group and send the block to the first group, called final committee. Then the final committee runs the same PBFT protocol to agree on the final result and broadcast to the whole network. The scene in ours is much easier comparing to ELASTICO. Within the committee, variant ring paxos needs two times broadcast and C times unicast. Then leaders will send messages to each other and check the missing blocks in two round. After that, leader will broadcast its reply to all members in his group which requires another broadcast. We use PoW to present the expectancy time in

Proof of Work, B to presents the upper quartile in expectancy time of broadcast in whole network, S presents the expectancy time in unicast. To simplify the compare, we assume that the upper quartile in expectancy time of broadcast within cluster is exactly $\frac{1}{k} \cdot B$ when cluster contains $\frac{1}{k}$ members. Table 1 shows the delay in these four framework. If we regarded the time spending on unicast and broadcast as same value T in any scale network. Then the consensus delay in ELASTICO is $PoW + 7T$ and the delay in ours is $(C + \frac{3}{k} + \frac{2}{C}) \cdot T$. Supposing the PoW is 600 s and a block with 1 MB needs around 1 s to disseminate, the arrangement of C can be bounded by $0 < C < 600$ when we make consensus delay in ours shorter than it in ELASTICO. However, the recommend size of committee in ELASTICO is also around 600 while the experiment in ELASTICO fixed the capacity of group to 100 which is upper bound of performance in PBFT.

In Intra-committees Synchronization, we require two times broadcast for leaders to exchange their blocks. The bottleneck in our system is here. As the scale of the system expands, the time delay in synchronization rises along with the number of committees groups.

Table 1. Delay in four decentralized framework

Framework	Delay
Bitcoin	$PoW + B$
Bitcoin-NG	B
ELASTICO	$PoW + (1 + \frac{6}{k}) \cdot B$
Ours	$C \cdot S + (\frac{3}{k} + \frac{2}{C}) \cdot B$

Throughput. Throughput shows the faculty for packaging and responding requests from clients. We analyze the throughput in just one period and use the throughput of Bitcoin as benchmark. In every period, 10 min in expectancy, Bitcoin will generate just one block. The output in Bitcoin-NG can be pre-set according to consensus delay. Therefore, the throughput in framework can be described as Eq. (3) and Table 2 shows the Throughput under ideal conditions, on average.

$$Throughput = \frac{Number of Blocks}{Expectancy Delay} \tag{3}$$

Number of Message. This index indicates the scalability in decentralized system and the consumption of bandwidth. Bitcoin and Bitcoin-NG are both reach the cost at $O(n)$. During committee formation in ELASTICO, they need all nodes to identify themselves and find their new groups in next epoch which results in $O(nc)$ messages. Generating block within committee and re-organizing blocks in centralized group both need $O(c^2)$ owing to PBFT. Hence the scalability of ELASTICO can reach $O(nc^2)$. Our framework introduced variant ring

Table 2. Expectancy throughput

Framework	Throughput
Bitcoin	$1/(PoW + B)$
Bitcoin-NG	$1/B$
ELASTICO	$k/(PoW + (1 + \frac{6}{k}) \cdot B)$
Ours	$k/(C \cdot S + (\frac{3}{k} + \frac{2}{C}) \cdot B)$

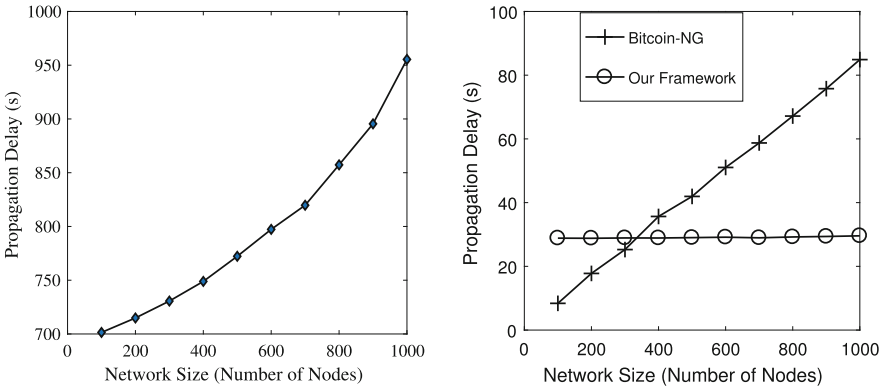
paxos as generating algorithm to shrink the number of message to $O(n^2)$ at the cost of high delay. Generating blocks within committee requires $3C$ messages and synchronizaiton between committees needs $\frac{n^2}{C}$ messages. After that, leaders will broadcast blocks within their clusters at cost of N messages. Therefore, the message complexity is $O(n^2)$ in our framework.

4 Experiment

We simulated a decentralized distributed platform with 1000 virtual nodes inside. For sake of saving time, we introduced event driven mechanism instead of timer. Every node in our platform has the same resource allocation and they can directly communicate with all other nodes which means the consumption of bandwidth may very huge. The benefit of equivalent allocation is that we can use a random number generator to decide the leader in each round without great energy consumption. The size of block is 1 MB and the bandwidth of network card is 100 Mpbs and we forced nodes to use up their bandwidth as far as possible in our experiments. For sake of reducing the interference of accidental factors, we recorded continuous 10 rounds in our experiments.

We pre-set the time expenditure of PoW as 600 s. The size of committee in our experiments was fixed at 100 nodes. The result in our simulation experiment is obvious that ELASTICO has the worst performance in propagation delay. There are two fatal shortcomings in ELASTICO. One is that ELASTICO requires PoW to calculate identities in each round which consumes around 10 min and the another one is the limitation in PBFT. PBFT mainly has three steps to reach agreement within a cluster. First step is that leader broadcasts block within committee. The second step requires everyone broadcast their view to other members in the cluster and this giant broadcast will happen again in the third step. The whole cost of message is $2C^2 + C$. Furthermore, the growth of groups will aggravate the burden of centralized committee because blocks from all subcommittees will be send to every member in centralized cluster and the huge number of message will consume the bandwidth of nodes in their center rapidly (Fig. 3).

The same threat in ELASTICO also exists in our sharding framework. However, we replace the PBFT by variant ring paxos to reduce the number of message at the cost of more delay and we also exploit two times broadcast to reach agreement among leaders instead of the centralized institution.



(a) The results of ELASTICO

(b) The results of Bitcoin-NG and our framework

Fig. 3. The average propagation delay in our experiments

When the scale of network exceeds 400 nodes, Bitcoin-NG performs worse than our model. Although Bitcoin-NG needs only N messages in each synchronization. However, the bandwidth of leader will becomes the bottleneck in huge-scale system.

5 Future Work

It seems that our model is highly efficient according to the analysis and experiments above. However, there indeed exists some shortcomings in our system. First of all, service nodes in our model need to storage all clients' and other servers' UID and public key which requires a great piece of memory. The memory may come to the new bottleneck in our servers. Secondly, our framework has two different states. When our system enters into the splitting state, our system cannot handle the requests because all servers need to prove that they're not virtual nodes via PoW. That means our framework may remain silence aperiodically.

In the future study, we will make further efforts to refine details in our framework and optimize it. Also we will pay attention to the development of atomic broadcast and its applications.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grants NSFC 6167050817 and 61672358.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. bitcoin.org (2009)
2. Luu, L., Narayanan, V., Zheng, C., et al.: A secure sharding protocol for open blockchains. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 17–30. ACM (2016)

3. Marandi, P.J., Primi, M., Schiper, N., et al.: Ring Paxos: a high-throughput atomic broadcast protocol. In: IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 527–536. IEEE (2010)
4. https://en.wikipedia.org/wiki/Proof-of-stake
5. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
6. Sompolinsky, Y., Zohar, A.: Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881 (2013). http://eprint.iacr.org/
7. Lewenberg, Y., Sompolinsky, Y., Zohar, A.: Inclusive block chain protocols. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 528–547. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_33
8. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_24
9. Newsome, J., Shi, E., Song, D., Perrig, A.: The sybil attack in sensor networks: analysis and defenses. In: Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks, IPSN 2004, pp. 259–268. ACM, New York (2004)
10. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
11. Gilbert, S., Kowalski, D.R.: Distributed agreement with optimal communication complexity. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 965–977. Society for Industrial and Applied Mathematics (2010)
12. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
13. Toueg, S., Perry, K.J., Srikanth, T.K.: Fast distributed agreement (preliminary version). In: Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, pp. 87–101. ACM (1985)
14. Eyal, I., Gencer, A.E., Sirer, E.G., et al.: Bitcoin-NG: a scalable blockchain protocol, pp. 45–59 (2015)
15. Ethereum Foundation. Ethereum's white paper (2014). https://github.com/ethereum/wiki/wiki/White-Paper
16. Andresen, G.: Bitcoin improvement proposal 101 (2015). https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki
17. Garzik, J.: Bitcoin improvement proposal 102 (2015). https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki
18. Castro, M.: Practical Byzantine fault tolerance and proactive recovery. In: Symposium on Operating Systems Design and Implementation, pp. 173–186. USENIX Association (1999)
19. Guerraoui, R., Huc, F., Kermarrec, A.-M.: Highly dynamic distributed computing with Byzantine failures. In: Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing, PODC 2013 (2013)
20. Bracha, G.: An O(log n) expected rounds randomized Byzantine generals protocol. J. ACM **34**, 910–920 (1987)
21. Lamport, L.: Fast paxos. Distrib. Comput. **19**(2), 79–103 (2006)
22. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, pp. 153–168. USENIX Association (2009)

23. King, V., Saia, J.: From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 464–478. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04355-0_47
24. King, V., Lonargan, S., Saia, J., Trehan, A.: Load balanced scalable Byzantine agreement through quorum building, with full information. In: Aguilera, M.K., Yu, H., Vaidya, N.H., Srinivasan, V., Choudhury, R.R. (eds.) ICDCN 2011. LNCS, vol. 6522, pp. 203–214. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17679-1_18
25. Braud-Santoni, N., Guerraoui, R., Huc, F.: Fast Byzantine agreement. In: Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing, pp. 57–64. ACM (2013)
26. King, V., Saia, J.: Breaking the $O(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. J. ACM **58**, 18:1–18:24 (2011)
27. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
28. Cristian, F., Mishra, S.: The Pinwheel asynchronous atomic broadcast protocols. In: International Symposium on Autonomous Decentralized Systems (ISADS), Phoenix, Arizona, USA (1995)
29. Defago, X., Schiper, A., Urban, P.: Total order broadcast and multicast algorithms: taxonomy and survey. ACM Comput. Surv. **36**(4), 372–421 (2004)
30. Ekwall, R., Schiper, A., Urban, P.: Token-based atomic broadcast using unreliable failure detectors. In: Proceedings of the International Symposium on Reliable Distributed Systems (SRDS), pp. 52–65 (2004)
31. Fritzke, U., Ingels, P., Mostefaoui, A., Raynal, M.: Faulttolerant total order multicast to asynchronous groups. In: Proceedings of International Symposium on Reliable Distributed Systems (SRDS), pp. 578–585 (1998)
32. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Distributed Systems, Chap. 5, 2nd edn. Addison-Wesley (1993)
33. Pedone, F., Schiper, A.: Optimistic atomic broadcast. In: Kutten, S. (ed.) DISC 1998. LNCS, vol. 1499, pp. 318–332. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0056492
34. Poke, M., Hoefler, T., Glass, C.W.: AllConcur: leaderless concurrent atomic broadcast. In: International Symposium on High-Performance Parallel and Distributed Computing, pp. 205–218. ACM (2017)
35. Hueber, O.: The blockchain and the sidechain innovations for the electronic commerce beyond the Bitcoin's framework. Int. J. Transit. Innov. Syst. **6**, 88–102 (2018)
36. Oliveira, L.C.D., Simoyama, F.D.O., Grigg, I., et al.: Triple entry ledgers with blockchain for auditing. Int. J. Audit. Technol. **3**(3), 163 (2017)
37. Feld, S., Schönfeld, M., Werner, M.: Traversing Bitcoin's P2P network: insights into the structure of a decentralized currency. Int. J. Comput. Sci. Eng. **13**, 122–131 (2014)
38. Ouaguid, A., Abghour, N., Ouzzif, M.: A novel security framework for managing Android permissions using blockchain technology. Int. J. Cloud Appl. Comput. (IJCAC) **8**(1), 55–79 (2018)
39. Asharaf, S., Adarsh, S.: Advanced topics in blockchains. In: Decentralized Computing Using Blockchain Technologies and Smart Contracts: Emerging Research and Opportunities (2017)