GeoSharding—A Machine Learning-Based Sharding Protocol

Hardik Ruparel, Shreyashree Chiplunkar, Shalin Shah, Madhav Goradia, and Mahesh Shirole

Abstract Sharding is one of the most prominent concepts which involves the division of the network into shards for concurrent processing of transactions. Different sharding protocols are being implemented in blockchains to enhance its scalability. The existing blockchain systems create shards using proof-of-work consensus protocol. This research aims at developing a machine learning-based sharding process that uses the nodes' geographical locations—latitudes and longitudes. IP addresses of the nodes are mapped to geographical coordinates, and these coordinates are then divided into shards using a suitable clustering algorithm. The nodes in the shards are geographically closer, thereby reducing the propagation delay in the network during intra-shard communication. GeoSharding has been tested to be significantly faster as compared to PoW-based sharding. This optimizes the network sharding process, thus escalating the scalability to a new level.

Keywords Blockchain · Sharding · Clustering · Proof of work · Machine learning

H. Ruparel (✉) · S. Chiplunkar · S. Shah · M. Goradia · M. Shirole
CE & IT Department, Veermata Jijabai Technological Institute, Mumbai, India
e-mail: hruparel_b15@it.vjti.ac.in

S. Chiplunkar
e-mail: suchiplunkar_b15@it.vjti.ac.in

S. Shah
e-mail: spshah_b15@it.vjti.ac.in

M. Goradia
e-mail: gmbipin_b15@it.vjti.ac.in

M. Shirole
e-mail: mrshirole@it.vjti.ac.in

D. Patel et al. (eds.), *IC-BCT 2019*, Blockchain Technologies,
https://doi.org/10.1007/978-981-15-4542-9_10

1 Introduction

Blockchain is an immutable and a shared ledger that enables the process of recording values known as transactions and tracking assets in a business network. Virtually anything of value can be recorded, tracked and traded on a blockchain network, reducing risk and cutting costs for all involved [1]. Being decentralized in nature, a blockchain is highly secure as no single user can alter or remove an entry in the blockchain since it would require changing all the blocks, which is realistically impossible.

However, one of the most pivotal problems in the current blockchain architecture is scalability. Today's representative blockchain such as bitcoin [2] takes 10 min or longer to confirm transactions and achieves a throughput of 7 Tx/s. In comparison, the average throughput of the current centralized systems is around 2000 and 56,000 Tx/s during holidays. For a wide-scale adoption of blockchain, this issue needs to be addressed as soon as possible [3].

Effective throughput in the overlay network is defined as the percentage of nodes that receive the propagated blocks within an average block interval period. 10% of the nodes in the network would be unable to keep up if the transaction rate exceeds the 90% effective throughput, potentially resulting in a denial of service to users and reducing the network's effective mining power [3].

$$X\% \text{ effective throughput} = \frac{\text{block size}}{X\% \text{ propagation delay}} \tag{1}$$

The formula in Eq. 1 is crucial for understanding why reparameterization like increasing the block size and decreasing the latency provides limited benefits. The bitcoin community has put forth various proposals to modify the parameters like block size and block interval that could make the system scalable. Two guidelines should be followed to ensure that at least 90% of the nodes in the network have sufficient throughput:

- **Throughput limit**. Given today's 10 min average block interval, the block size should not exceed 4 MB. Maximum throughput of at most 27 Tx/s is obtained at 4 MB block size.
- **Latency limit**. If full utilization of the network's bandwidth is to be achieved, the block interval should not be smaller than 12 s [3].

Hence, a completely new and scalable architecture is needed to be developed to increase the transaction rate of blockchains. As per our understanding, one can achieve scalability in five planes like network, consensus, storage, view and side plane. In our approach, we aim at achieving scalability in the network plane using geographical sharding. This paper is organized as follows: Sect. 2 discusses the basic concepts used in the paper. Section 3 discusses the proposed system. The results of the experiment and comparisons with the existing systems are presented in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Basic Concepts

2.1 Sharding

Sharding is the division of the network into independent groups of nodes called shards for concurrent processing of transactions. For sharding blockchain systems, the nodes will only maintain a portion of the data and not the entire information [4]. However, each node does not load the information on the entire blockchain, thus helping in scalability. In blockchain, sharding can be done at different levels like network level, transaction level and computational level.

- **Network sharding**. Network sharding is a technique that allows the network to be segregated into smaller groups of nodes called *shards*.
- **Transaction sharding.** Whenever a transaction reaches the network, it is assigned to a specific shard. The assignment of a transaction to a particular shard is based on a few bits of the sending address of the transaction. This prevents the double-spending problem as transactions sent by a user are assigned to a particular shard only.
- **Computation sharding**. The sharding of computational resources in the blockchain network via an overlay above the consensus process is called *computational sharding*. Computations can be performed efficiently with some shards acting as mappers and the rest as reducers for a map-reduce task.

2.2 Clustering

In machine learning, clustering is the process of combining objects based on their attributes and aggregating them according to their similarities. There are many clustering techniques like *distance-based, density-based, interval or statistical-based*. In our research, we use distance as the similarity measure for creating clusters. In the case of blockchain, network sharding is done for nodes, where nodes in a cluster are closer to each other than the nodes in another cluster.

Various distance measures like *Euclidean, Squared Euclidean, Manhattan, Correlation, Haversine,*, etc., can be used for measuring the distance between data points. However, we require a formula that measures the distance between locations on the Earth's surface because Earth is a globe. So, we use Haversine distance, which is suitable for spherical shapes.

Haversine Distance. The Haversine formula helps in calculating the shortest distance between any two points on a spherical surface using their latitudes and longitudes measured along the surface. This formula is given in Eq. 2, and it is important in navigation [5].

Table 1 Comparison between k-means and DBSCAN

Parameters	**K-means**	**DBSCAN**
Number of clusters	*k*	Depends on the density of the data points
Size of clusters	Similar-sized	Differs across clusters
Shape of clusters	Prefers spherical clusters	Handles skewed or randomly shaped clusters

$$d = 2r \arcsin\sqrt{\sin^2\left(\frac{\phi 2 - \phi 1}{2}\right) + \cos(\phi 1)\cos(\phi 2)\sin^2\left(\frac{\lambda 2 - \lambda 1}{2}\right)} \quad (2)$$

where d—the distance between the two points (along a great circle of the sphere), r—the radius of the sphere, $\phi 1$, $\phi 2$—the latitude of points 1 and 2, respectively, and $\lambda 1$, $\lambda 2$—the longitude of points 1 and 2, respectively.

2.3 Comparison of Clustering Algorithms

An appropriate clustering algorithm needs to be chosen for efficient formation of clusters. Since clustering algorithm is used for network sharding, the parameters such as distance function and number of clusters are chosen accordingly. Different clustering algorithms that can be used are *k-means, mean-shift, DBSCAN, expectation–maximization (EM), agglomerative, hierarchical,* etc. The two clustering algorithms suitable to our motive of clustering GPS points are:

- **K-means**. K-means clustering [6] classifies a given dataset into k number of clusters, where k is specified in advance. The clusters are then represented as points, and all the data points are associated with the nearest clusters. These clusters are then recomputed and adjusted until a desired result is reached.
- **DBSCAN**. DBSCAN [6] is another approach that makes an assumption that a cluster is a connected region with relatively dense points. DBSCAN algorithm requires two parameters: the maximum distance (*eps*) and the minimum number of points (*minPts*) in the given region which are required to form a cluster (Table 1).

Since the benefits offered by DBSCAN clustering algorithm are not relevant to our results and also the number of clusters cannot be specified beforehand, we choose k-means as the algorithm for clustering GPS points.

2.4 Similarity Measure in Leader Election

Various measures can be used for calculating the similarity between two strings, out of which Hamming and Levenshtein distance are most suitable in this context:

- **Hamming distance**. The number of places at which the corresponding characters in two equal length strings are different
- **Levenshtein distance**. The minimum number of single-character edits (insertions, deletions or substitutions) required to change one string into the other.

Hamming distance calculates similarity much faster than Levenshtein distance, but it has some limitations. This can be explained using an example: Consider the strings "12345", "51234" and "67890". The Hamming distance between the first and the second string is 5 and that between the first and the third string is also 5. On the other hand, the Levenshtein distance between the first and the second string is 2 and that between the first and the third string is 5. It can be seen that the first two strings are similar, but the Hamming distance fails to determine this. Also, comparing the similarity values obtained from both the distance measures, we have found that Hamming gives a very narrow range of values, whereas Levenshtein gives a far wider range of values. Considering Hamming, similarity measure will no longer be a distinguishing factor in leader election. Hence, we choose Levenshtein distance.

3 Proposed System

Our proposed system aims at achieving scalability and improving efficiency in the network plane of the blockchain architecture. There are existing systems that implement sharding for achieving scalability. However, these systems are not very efficient due to various factors. Some of the factors are: (a) time taken to create a shard and thus higher downtime of the network in the event of shard recreation, (b) propagation delay in intra-shard communication and (c) higher computational resources required in the existing systems. Thus, we aim to improve the efficiency by introducing the concept of geographical sharding for the network plane.

3.1 Parameters Considered for Analysis

The following parameters are used for the analysis of our proposed system:

- **Propagation delay**. In computer networks, propagation delay is the amount of time it takes for the head of the signal to travel from the sender to the receiver. It can be computed as the ratio between the link length and the propagation speed over the specific medium [7].

- **Computational resources**. A computational resource is a resource used by some computational models in the solution of computational problems [8]. PoW is a task, which requires huge computational resources.
- **Shard creation time**. Shard creation time is defined as the time taken for dividing the network into shards. Our research aims at reducing shard creation time.
- **Delay in joining of new nodes**. New nodes cannot join the network at any time. They have to wait for the shard reformation epoch and perform PoW. If they could not compute PoW within a specific time, they have to wait for the next epoch. Our research aims at including all such nodes during the next epoch itself and not delay it further.
- **Network downtime**. Downtime refers to a period of time during which a computer system, a server or a network is shut off or unavailable for use. There are various reasons for downtime, and in blockchain, one of these reasons is the shard reformation phase.

3.2 Geographical Mapping

The first step in implementing GeoSharding is mapping the Internet protocol (IP) addresses of the nodes to locations on the Earth's surface. Various APIs are available that allow us to do IP lookup and return the details in XML format. The details of all the IP addresses are stored by regional Internet registries. In the response file, we obtain the latitudes and longitudes which will be used in the Haversine formula to calculate the distance between two nodes.

3.3 Leader Election

To increase the throughput of the blockchain system, a leader is elected from each shard at every epoch [9]. Every node within a shard during a leader election epoch calculates the target value, which is the hash of its shard name and the current block number. Every node then calculates the hash of its address which includes the IP address and its port number. The similarity between these two strings of hashes forms a factor in deciding the leader of the shard [10]. The similarity between these two strings of hashes is calculated using Levenshtein distance. The more similar the hash of the node's address and the hash of the shard name and block number, the more are the chances of that node becoming the leader.

Once the distance is calculated, it is broadcasted to every node within the shard [11]. The distances received are also verified by the receiving nodes. All nodes have an array list of ages depicting the time since every node has become a leader. The node which has become a leader recently has a lower age value. Every node then calculates the cumulative scores based on the ages and the distances received from other nodes. To give a fair chance to the nodes which are not elected as a leader for a long time,

the age can be given a higher weight than the distance, like 60% weightage to the age factor and 40% weightage to the distance factor. This cumulative score is called as the leader competence score. The node which has the maximum score becomes the leader for that epoch. The address of the elected leader is then broadcasted to every other node within the shard. The age of the leader node is decreased, and the ages of other nodes within the shard are increased. After network sharding, every node has the shard information in the form of a structure. The structure of the shard is:

Struct *Shard* **contains**
    shardName string //name of the shard in string format
    shardlist[ ] Address //addresses of all the nodes within the shard
    agelist[ ] float64 //ages of all the nodes within the shard
end

Algorithm 1 Leader Election Algorithm

1: **function** ELECT(shard, currentBlockNumber, myAddress)
   Input: *shard* - a structure that every node receives during network sharding. *currentBlockNumber*- an integer stating the current block number in the blockchain. *myAddress* - the address of the node executing the program
   Output: Leader of the shard
2: $A \leftarrow$ Hash(*shard.shardName, currentBlockNumber*)
3: $B \leftarrow$ Hash(*myAddress*)
4: *distance* $\leftarrow$ Levenshtein distance (A, B)
5: Broadcast *distance* to the nodes in *shard.shardlist*
6: Verify the distances received from other nodes
7: Compute the competence score of each node to become the leader using *distance* and age of the node from the *shard.agelist*
8: Broadcast the leader
9: **if** *leader* = *myAddress* **then**
       Broadcast "I am the leader"
10: Decrement the age of the leader node and increment the ages of all other nodes

4 Results

The result analysis is completely based on the assumption that proof of work for establishing mining identities takes approximately 10 min. The proposed sharding technique is applied to the GeoLite2 database which contains over 3.3 million entries. The objective function which is used to optimize the clusters is composed of the Haversine distance function. The Ethereum and bitcoin network have around 25,000 and 7000 reachable nodes, respectively. So, to analyze the results of sharding, we consider a larger network size, and the database is sampled randomly to choose 33,645 records for clustering. The results written in the following section are obtained by the implementation of k-means [6, 12] in GoLang [13], and the clustering algorithm

Fig. 1 Geographical sharding produced using k-means clustering algorithm for $k = 3$

has been executed 10 times to reduce the variations in the result. This entire process of training and forming seven clusters took an average of 0.84434 s.

Figure 1 shows a snapshot of the proposed method for creating shards. These shards are created using the k-means clustering algorithm, where the value of k=3. The nodes in each shard are geographically closer than the nodes in other shards.

4.1 Dataset Description

GeoLite2 City database provided by the MaxMind Developers [14] was used for the implementation of the concept. This dataset contains a total of 3.366332 million records and 10 columns viz:
network (IP in string), *geoname_id* (integer), *registered_country_geoname_id* (integer), *represented_country_geoname_id* (integer), *is_anonymous_proxy* (bo-olean), *is_satellite_provider* (boolean), *postal_code* (string), *latitude* (decimal), *longitude* (decimal), *accuracy_radius* (integer). The attributes used in formation of clusters are *network*, *latitude* and *longitude*.

Preprocessing and Sampling. The database is first preprocessed to handle missing values. The dataset contains some records with missing latitude and longitude values. Since the number of such records is only 19, these records are ignored. 33,645 records were chosen randomly without replacement from the remaining records and then supplied as an input to the clustering algorithm.

Table 2 Time taken for running clustering algorithm on 33,645 records

#Clusters	Total time (s)	Average time per clustering (s)
3	2.5391	0.2539
5	4.0836	0.40836
7	8.4434	0.84434
10	13.9184	1.39184
15	26.8803	2.68803

Fig. 2 Output snippet for geographical sharding

```
C:\Users\Hp\Desktop\Sharding>go run clustering.go -k 15
Number of records after sampling: 33645
Total records in data set: 3366332
Time elapsed: 3.0184119s

C:\Users\Hp\Desktop\Sharding>go run clustering.go -k 10
Number of records after sampling: 33645
Total records in data set: 3366332
Time elapsed: 1.3414136s

C:\Users\Hp\Desktop\Sharding>go run clustering.go -k 7
Number of records after sampling: 33645
Total records in data set: 3366332
Time elapsed: 958.4384ms

C:\Users\Hp\Desktop\Sharding>go run clustering.go -k 5
Number of records after sampling: 33645
Total records in data set: 3366332
Time elapsed: 500.2538ms

C:\Users\Hp\Desktop\Sharding>go run clustering.go -k 3
Number of records after sampling: 33645
Total records in data set: 3366332
Time elapsed: 332.1319ms
```

4.2 Speed Analysis

The speed of the proposed system is measured by the time taken to form the clusters. Figure 2 shows the time taken for clustering when the number of clusters specified is 15, 10, 7, 5 and 3. This program was run 10 times, and the average time taken for clustering using specified number of clusters is shown below in Table 2. The table shows that the average time taken for creating 15 shards in a network of 33,645 nodes is 2.68803 s, which is considerably faster than the general PoW-based method since the method requires every node to solve PoW and listen to PoW results of other slower nodes for at least 600 s.

4.3 Scalability of the Proposed System

A simple and most efficient way to check the scalability is by increasing the sample size of the dataset, which is fed to the clustering algorithm. The results in Table 3 show the time taken to form 10 shards for different sizes of the dataset.

Table 3 Time taken for running clustering algorithm on different sizes of dataset

#Nodes (Sample size)	Time taken for shard creation (s)
3367	0.0746991
33,645	1.6607684
3,36,487	9.3331801
33,64,864	226.1304

Considering the current Ethereum network size, even if we increase its size 13 times to 0.336 million nodes, the time it takes for creating 10 shards is as little as 9.3331801 s, which is at most 1.5% of the time taken for PoW-based techniques [15]. For the network size of 3.3 million nodes, it takes 226.1304 s to form 10 shards, which still saves at least 70% of the time utilized for shards formation in PoW-based techniques. Thus, even if a network of such a huge size is reached, the time taken by GeoSharding to form shards will be at most 30% of the time taken for PoW-based techniques, which is a significant advantage in making the system faster and scalable.

4.4 Efficiency and Security

The quality of clusters determines the efficiency of the clustering algorithm. Figure 1 shows that the clustering algorithm has created discrete clusters and divided the nodes into shards efficiently. This map is developed with the help of Google My Maps [16] which enables you to visualize the points on the world map. Each point on the map is an IP address of a node at that location. A csv file containing longitude, latitude and a cluster id of the points is given as an input.

All the nodes run the clustering algorithm, and the shard that each node belongs to is decided in consensus by all the nodes. Since it is considered that at least 2/3rd of the nodes in network are honest, a node will always be assigned to the correct shard, thereby making the system secure. This process of sharding is repeated every 20 hours; hence, this sharding approach will significantly decrease the downtime of the network. Every new node that joins the network has to wait for the shard reformation epoch.

4.5 Leader Election

The leader election algorithm needs to be run once every leader epoch. This algorithm is proposed in Sect. 3.3. Below is an output snippet of leader election run for three epochs. A leader is chosen such that every node is given a fair chance. Figure 3 shows the addresses of the nodes in the shard and the similarity distances computed

```
C:\Users\Hp\Desktop>go run IdSet.go
New Leader Epoch
Electing Leader
Similarity score is: map[127.0.0.1 8086:10 127.0.0.1 8087:11 127.0.0.1 8080:10 127.0.0.1 8081:9 127.0.0.1 8082:7 127.0.0.1
 8083:8 127.0.0.1 8084:10 127.0.0.1 8085:11]
Age: map[127.0.0.1 8085:1 127.0.0.1 8086:1 127.0.0.1 8087:1 127.0.0.1 8080:1 127.0.0.1 8081:1 127.0.0.1 8082:1 127.0.0.1 8
083:1 127.0.0.1 8084:1]
Leader Competence Score: map[127.0.0.1 8080:4.6 127.0.0.1 8081:4.2 127.0.0.1 8082:3.4 127.0.0.1 8083:3.8 127.0.0.1 8084:4.
6 127.0.0.1 8085:5 127.0.0.1 8086:4.6 127.0.0.1 8087:5]
Leader is  127.0.0.1 8085

New Leader Epoch
Electing Leader
Similarity score is: map[127.0.0.1 8086:10 127.0.0.1 8087:9 127.0.0.1 8080:9 127.0.0.1 8081:8 127.0.0.1 8082:7 127.0.0.1 8
083:10 127.0.0.1 8084:11 127.0.0.1 8085:10]
Age: map[127.0.0.1 8081:1.1 127.0.0.1 8082:1.1 127.0.0.1 8083:1.1 127.0.0.1 8084:1.1 127.0.0.1 8085:0.5 127.0.0.1 8086:1.1
 127.0.0.1 8087:1.1 127.0.0.1 8080:1.1]
Leader Competence Score: map[127.0.0.1 8086:4.66 127.0.0.1 8087:4.26 127.0.0.1 8080:4.26 127.0.0.1 8081:3.86 127.0.0.1 808
2:3.46 127.0.0.1 8083:4.66 127.0.0.1 8084:5.06 127.0.0.1 8085:4.3]
Leader is  127.0.0.1 8084

New Leader Epoch
Electing Leader
Similarity score is: map[127.0.0.1 8085:10 127.0.0.1 8086:10 127.0.0.1 8087:8 127.0.0.1 8080:8 127.0.0.1 8081:9 127.0.0.1
8082:7 127.0.0.1 8083:9 127.0.0.1 8084:10]
Age: map[127.0.0.1 8086:1.2 127.0.0.1 8087:1.2 127.0.0.1 8080:1.2 127.0.0.1 8081:1.2 127.0.0.1 8082:1.2 127.0.0.1 8083:1.2
 127.0.0.1 8084:0.5 127.0.0.1 8085:0.6]
Leader Competence Score: map[127.0.0.1 8080:3.92 127.0.0.1 8081:4.32 127.0.0.1 8082:3.52 127.0.0.1 8083:4.32 127.0.0.1 808
4:4.3 127.0.0.1 8085:4.36 127.0.0.1 8086:4.72 127.0.0.1 8087:3.92]
Leader is  127.0.0.1 8086
```

Fig. 3 Output snippet of leader election for three leader epochs

by them. This distance is the Levenshtein distance which is explained in Sect. 3.3. Also, the corresponding current ages of the nodes are shown. Leader competence score is calculated for every node using the ages and the similarity distances as shown in Eq. 3:

$$Leader\_competence\_score = 0.4 \times similarity\_distance + 0.6 \times age \quad (3)$$

The factors 0.4 and 0.6 assure that every node is given a fair chance to become a leader. The node with the highest competence score is elected as the leader. Further, the age of the elected leader is decremented to a value of 0.5, and the ages of other nodes are incremented by 0.1 so that other nodes have a greater chance to become the leader in the next epoch.

4.6 Comparison with the Existing Solutions

Elastico [15] uniformly partitions the mining network (securely) into smaller committees (shards), each of which processes a disjoint set of transactions. Communication in Elastico takes place between the nodes in the shards and the directory committee. Elastico was the first one to propose the idea of network and transaction sharding. However, the problem of scalability cannot be solved alone by these sharding techniques as the intra-shard communication is still cumbersome since the nodes in one shard might be located at different locations, thus leading to propagation delay. Our proposed system solves this problem by using clustering

Table 4 Comparison of performance parameters with related systems [15, 17, 18]

Parameters	Proposed system	Elastico [15]	Zilliqa[17, 18]
Propagation delay	Low	High	High
Consumption of computational resources	Low	High	High
Shard creation time	Very low	High	High
Network downtime	Very low	Moderate	Moderate
Joining of new nodes	At shard reformation epoch	At shard reformation epoch if PoW is solved or else at next epoch	At Shard Reformation Epoch if PoW is solved or else at next epoch

algorithms to geographically cluster nodes into shards and process transactions in parallel, thus reducing the propagation delay.

Zilliqa [17, 18] uses proof of work to form shards. The nodes are sorted according to the nonce values obtained from the PoW results, shards are created and the results are communicated. This is a very time-consuming process causing a greater downtime of the network during shard formation epoch. Our system uses k-means for sharding which results in faster creation of shards.

Table 4 shows the comparison of our proposed system with the existing systems. The proposed system is analyzed to provide the following benefits as compared to the existing scalable architectures:

Propagation Delay. In the existing blockchain systems with sharding, all nodes perform proof of work and are divided into shards based on the PoW result. This results in shards that consist of randomly located nodes.

In GeoSharding, the nodes in a particular shard are closer to each other than the nodes in other shards, as shown in Fig. 1. Hence, the propagation delay during intra-shard communication is less in GeoSharding as compared to the existing sharded systems.

Computational Resources. Proof of work is a task which requires huge computational resources. However, GeoSharding has very moderate system requirements, and the clustering program can be run by any node. This would result in an increase in the mining power and hence the efficiency of the system.

Shard Creation Time. Proof of work is a time-consuming process. However, GeoSharding creates shards in a very short time since it uses a clustering algorithm like k-means, and hence, the shard setup phase is optimized. Shard reformation occurs every 20 h.

Joining of New Nodes. All nodes willing to join the network are assigned to specific shards based on the clustering algorithm, whereas in the existing systems, if a new node fails to compute PoW in a specific time, it is denied to join the network in that epoch.

Reduction in Downtime of the Network. The downtime of the proposed system is significantly less in comparison with other systems since the shard creation phase is optimized to obtain better results.

5 Conclusion

GeoSharding is a scalable and efficient algorithm for clustering nodes based on the latitude and the longitude values. Proof-of-work-based sharding technique first requires a considerable amount of time to solve PoW for establishing mining identities. It requires additional time to broadcast the network sharding result of all the nodes and reaches a consensus on these shard lists. This paper proposed sharding the network on the basis of k-means algorithm which is fast, scalable and accurate in creating shards. GeoSharding accelerates this process of sharding the network, considerably faster than PoW-based sharding. Also, the number of messages communicated with each peer node is 2/3rd of the messages communicated in PoW-based sharding. For every shard reformation epoch, the time taken to form shards in the worst case will still be better than the time taken for PoW-based network sharding, thereby improving the overall throughput of the architecture.

References

1. Gupta M.: Blockchain for Dummies. 2nd IBM Limited edn. John Wiley & Sons, Inc, Hoboken, NJ (2018)
2. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System
3. Croman, K., Decker, C., Eyal, I., Gencer, A., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E., Song, D., Wattenhofer, R.: On Scaling Decentralized Blockchains (A Position Paper)
4. Sharding in Blockchain, https://medium.com/edchain/what-is-sharding-in-blockchain-8afd9ed4cff0
5. Haversine Formula, https://en.wikipedia.org/wiki/Haversine_formula
6. Ganeshan D.: GPS Clustering and Analytics, http://web.cs.wpi.edu/~emmanuel/courses/cs528/F17/slides/papers/deepak_ganesan_GPS_clustering.pdf
7. Propagation Delay, https://en.wikipedia.org/wiki/Propagation_delay
8. Computational Resources, https://en.wikipedia.org/wiki/Computational_resource
9. Eyal, I., Gencer, A., Sirer, E., Renesse, R.: Bitcoin-NG: A Scalable Blockchain Protocol. In: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation, Santa Clara, CA, USA (2016)
10. Obeidat, A., Gubarev, V.: Leader Election in Peer-to-Peer Systems
11. Perlin/noise-Library, https://github.com/perlin-network/noise
12. K-means implementation in Go, https://github.com/MathieuMailhos/gomeans
13. The Go Programming Language, https://tour.golang.org
14. GeoIP2 City Dataset,https://dev.maxmind.com/geoip/geoip2/geoip2-city-country-csv-databases/
15. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A Secure Sharding Protocol For Open Blockchains

16. Google MyMaps, https://www.google.com/mymaps
17. The Zilliqa Team: The Zilliqa Project: A Secure, Scalable Blockchain Platform. Version 1.0 (May 2018)
18. The Zilliqa Team (Aug. 2017) The ZILLIQA Technical Whitepaper. Version 1