Subnetworks in BlockDAG

Oleksandr Antonenko
Faculty of Mathematics, Physics and Information Technologies
Odesa I.I. Mechnikov National University
Odesa, Ukraine
antonenko@onu.edu.ua

Sergii Grybniak
Institute of Computer Systems
Odesa Polytechnic State University
Odesa, Ukraine
s.s.grybniak@op.edu.ua

Denis Guzey
Faculty of Mathematics, Physics and Information Technologies
Odesa I.I. Mechnikov National University
Odesa, Ukraine
d1onat1d@gmail.com

Oleksandr Nashyvan
Institute of Computer Systems
Odesa Polytechnic State University
Odesa, Ukraine
o.nashyvan@op.edu.ua

Ruslan Shanin
Faculty of Mathematics, Physics and Information Technologies
Odesa I.I. Mechnikov National University
Odesa, Ukraine
ruslanshanin@onu.edu.ua

***Abstract*—In the article, we study the horizontal scaling of the Waterfall or similar blockDAG networks by partitioning them into subnetworks by applying hierarchical and graph-based clustering algorithms. It leads to the reducing the network load and, in addition, to the increasing of the potential performance parameters of the underlying protocol. We consider methods of topology construction, propose clustering algorithms, and perform a simulation of a network partitioning into subnetworks.**

***Index Terms*—blockDAG, clustering, scalable blockchain, scale-out, subnetwork, Waterfall**

I. Introduction

In modern world, blockchain technologies and their generalizations have become widespread and play an important role in multiple industries. The use of blockchain to store and distribute information in decentralized distributed networks is limited by poor scalability and low transaction speed [1]. One solution is to use a directed acyclic graph (DAG) structure instead of a chain one. This leads to the emergence of blockDAG networks. The data structure in the form of a DAG allows the simultaneous creation of several blocks and has been studied in many works [2]–[7]. Other approaches to solving these problems have also been proposed: sharding [8]–[12], Layers [13], [14], etc.

In the paper, we consider the problem of scalability the blockDAG Waterfall network [15], [16]. We will focus on a concept of subnetworks which suggests that some information is not distributed throughout the network, but only among some set of nodes, which we will call a subnetwork. Since several blocks are created in blockDAG networks in one round of work and, thus, there is no need to send all transactions throughout the network, we can use subnetworks to separate the set of transactions, which does not make sense in conventional blockchain systems. Also, partitioning into subnetworks, we build methods which reduce the network latency within the subnetworks while keeping the subnetworks roughly the same size. This innovative approach allows us to reduce network load and gives us a partial solution of the problem of the simultaneous creation of blocks with some identical transactions. Note that the concept of subnetworks has been used in other way in some blockchain protocols. For example, in Ethereum 2.0 [17], Kaspa [18] and TON [12], subnetworks are used for division of nodes into different subgroups to perform additional tasks.

Thus, in the paper, we consider the division of the Waterfall network into subnetworks and appropriately dividing the transaction pool into disjoint subpools for reducing the network load.

II. Problem statement

First, let us review the necessary points related to the operation of the Waterfall network. The time of work of the Waterfall network is divided into rounds, called slots, and epochs, which are used to summarize the work of the network. At the beginning of each epoch, a list of committees for the next epoch is determined, and at the beginning of each slot, the workers who will create a block in that slot are determined.

A BlockDAG network node contains a pool of transactions that are waiting to be added to a block or finalized and also a local copy of the BlockDAG ledger. Multiple validators can work on a single BlockDAG node.

The basic idea of subnetting (partitioning into subnetworks) is that each node will belong to some single subnetwork and will only store and distribute transactions of that subnetwork. At the same time, it will receive, store and distribute all blocks of blockDAG and, thus, will have a complete copy of the entire blockDAG repository.

A. Requirements to partitioning into subnetworks

Let us begin our consideration of partitioning into subnetworks by discussing the challenges and problems that arise in the process. First, it is obvious that *all nodes containing validators must belong to some subnetwork*. Also, since

[1]https://ieeexplore.ieee.org/document/10087101

nodes from different subnetworks will have different sets of transactions and all correct transactions must be published to blockDAG, *each subnetwork must contain nodes that have validators on them*.

Since the validator does not create the block often, and the transactions must be processed as quickly as possible, *then subnetwork validators must "frequently" create blocks*. This requirement can be interpreted in a number of ways. First, it can be understood in the probabilistic sense that, on average, at least one block per slot is created in each subnetwork. Second, it can be understood in the deterministic sense that in each subnetwork it is created at least one block per slot.

Partitioning the network into subnetworks requires a method for obtaining information about the links in the network. Immediately note that, since each subnetwork must have validators, the information required for clustering must contain, in particular, information about the presence and number of validators in the network nodes, which, generally speaking, can generate new types of threats and attacks. To prevent this, we introduce the notion of "*virtual node*", and we will work with virtual nodes when partitioning the network into subnetworks.

Finally, another important requirement for a clustering method is speed. Unfortunately, most methods of clustering data are quite slow, and when handling large volumes of data, they require greater computing power. Thus, we focus on algorithms that work at speed $O(m \log m)$, where m is the number of virtual nodes.

III. Subnetting approaches

We will refer to a virtual node as a set of validators that we assume share a common transaction pool and are located on the same physical node. By default, each validator is considered as a virtual node. To combine validators into a virtual node or to change the composition of a virtual node, validators must publish the appropriate transactions to the network.

Let us consider several approaches to subnetting (partitioning into subnetworks) with all of the above in mind:

1) The block creator selection system does not change.
   a) The network is divided into subnetworks with the sole restriction that subsets cannot be too small. The advantage of this solution is the simplicity and the undemanding nature of the clustering algorithm. The disadvantage of this solution is the fact that, on smaller subnetworks, blocks will be generated much less frequently, and the frequency of block generation is not guaranteed. Thus, transactions could potentially hang for long periods of time.
   b) The network is divided into subnetworks which contain approximately equal numbers of validators. In this case, the number of blocks created in the slot should not be less than the number of subnetworks. Assuming an even distribution of block creators, on average at least one block per slot it will be created in each subnetwork.
2) The block creator selection system is tied to subnetworks, namely, block creators in each slot are selected from members of the subnetwork:
   a) The network is divided into subnetworks that contain approximately equal numbers of validators. In this case, the number of blocks, created in a slot, is a multiple of the number of subnetworks. This method works most predictably, but requires special properties from the clustering algorithm.
   b) The network is divided into subnetworks, and the number of validators in each subnetwork may differ significantly (for example, by no more than two or three times). In contrast to the previous point, in order to maintain approximately equal probability of block creation by all validators, additional mechanisms must be introduced to ensure that the probability of block creation by participants in different subnetworks is the same.

Note that in case 2) a necessary requirement is that all validators know the composition of subnetworks (the distribution of validators across subnetworks), which limits the choice of clustering methods. In this paper, we study the second approach when partitioning into subnetworks. Also, since clustering must be identical at every node, we suppose that all nodes will execute the clustering algorithm based on the finalized part of the ledger before some fixed slot.

IV. Building a network topology

A. Notations and preliminary remarks

Further we assume that the set $\mathcal{V}$ of all validators of the network is given, $|\mathcal{V}| = n$ is the number of validators, and some class $\mathcal{N}$ of subsets of $\mathcal{V}$ is given, $\mathcal{N} \subseteq 2^{\mathcal{V}}$, whose elements are virtual nodes, $|\mathcal{N}| = m$ is the number of virtual nodes. We assume that any two elements of the class $\mathcal{N}$ either coincide or do not intersect, and each of the validators belongs to some virtual node. We additionally impose a restriction that a virtual node cannot contain many validators, namely, there is such natural number L, that for any virtual node $N \in \mathcal{N}$, the inequality $|N| \leqslant L$ holds, where, by $|N|$, we denote the number of validators in the virtual node N. This condition is important, because otherwise there could be substantial problems with the uniform distribution of validators across subnetworks.

Let us formulate our problem in the described terms: it needs to partition the set of nodes $\mathcal{N}$ on k subsets (clusters, subnetworks) $Cl_1, Cl_2, \ldots, Cl_k$ so that $Cl_1 \cup Cl_2 \cup \ldots \cup Cl_k = \mathcal{N}$ and $Cl_i \cap Cl_j = \varnothing$ if $i \neq j$, which means that each virtual node is included in exactly one cluster. In what follows we will denote the set of clusters as $\mathcal{C}$, $\mathcal{C} = \{Cl_1, \ldots, Cl_k\}$.

B. Information collection methods

In what follows by era we mean some fixed period of time such that during this period of time all network participants

will create at least one block. Information about the connections between virtual nodes is collected and published within the era. Let us describe this procedure.

Let N_1 and N_2 be two arbitrary nodes, and let validator $v \in N_1$ fix q times the distance between the virtual nodes N_1 and N_2. When one of validator of node N_2 publishes a block in blockDAG, some time later this block gets to validator v. At that moment, it fixes the delay d_{q+1} according to the slot beginning and calculates a new distance between virtual nodes N_1 and N_2 by the following formula:

$$d_{v,q+1}(N_1, N_2) = \begin{cases} \frac{q}{q+1} d_{v,q}(N_1, N_2) + \frac{1}{q+1} d_{q+1} & \text{if } q \leqslant Q, \\ \frac{Q}{Q+1} d_{v,q}(N_1, N_2) + \frac{1}{Q+1} d_{q+1} & \text{if } q > Q. \end{cases}$$

Also we set $d_{v,0}(N_1, N_2) = d_0$. Hence, all validators of the node N_1 (or the physical nodes on which they reside) store information about the average delay for the blocks of each other virtual node (except for those that have not yet published blocks, or have not published blocks for a very long time, i.e. have been offline for a long time).

At the moment when it is the turn of the validator v of node N_1 to publish its block, it selects from all virtual nodes (except its own) s nodes with the least delays $d_v(N_1, N_2)$, and publishes the list of s identifiers of virtual nodes and delays to them.

C. Topology of network of virtual nodes

For any two virtual nodes N_1, $N_2 \in \mathcal{N}$ we denote by $V(N_1, N_2) = \{v \in N_1 \cup N_2 \colon \exists d_v(N_1, N_2)\}$ the set (possibly empty) of all validators, that have published distances between nodes N_1 and N_2, where we denote by $d_v(N_1, N_2)$ the published validator v distance between nodes N_1 and N_2, $d_v(N_1, N_2) = d_v(N_2, N_1)$, and write

$$D(v) = \{d \colon \exists N_1, N_2 \in \mathcal{N} \quad d_v(N_1, N_2) = d\}$$

for the set of all distances published by the validator v. Thus, if some validator $v \in N_1 \cup N_2$ publishes a distance between nodes N_1 and N_2, then it is counted as the distance from N_1 to N_2 as well as the distance from N_2 to N_1.

If $V(N_1, N_2) \neq \varnothing$, then we define $d_1(N_1, N_2)$ — the average distance between nodes N_1 and N_2 — as follows

$$d_1(N_1, N_2) = \frac{1}{|V(N_1, N_2)|} \sum_{v \in V(N_1, N_2)} d_v(N_1, N_2),$$

where the sum is taken over all published distances between nodes N_1 and N_2.

For virtual nodes N_1 and N_2 such that $V(N_1, N_2) \neq \varnothing$ we also define the distance d_{max} as follows

$$d_{max}(N_1, N_2) = \frac{1}{|N_1| + |N_2|} \sum_{v \in N_1 \cup N_2} d_v^{max}(N_1, N_2),$$

where $d_v^{max}(N_1, N_2) = d_v(N_1, N_2)$ if $v \in V(N_1, N_2)$ and $d_v^{max}(N_1, N_2) = \max\{\max D(v), d_1(N_1, N_2)\}$ if v does not belong to $V(N_1, N_2)$.

In what follows, we denote by $d(N_1, N_2)$ one of the values $d_1(N_1, N_2)$ or $d_{max}(N_1, N_2)$. We will assume that the values $d_1(N_1, N_2)$ and $d_{max}(N_1, N_2)$ are undefined if $V(N_1, N_2) = \varnothing$, that is, if no validator has published distances between nodes N_1 and N_2. Thus, we can say that the published distances $d(N_1, N_2)$ define an undirected graph $G = (\mathcal{N}, E)$, where the set of edges coincides with those pairs of nodes between which the distance is published, i.e. $E = \{(N_1, N_2) \in \mathcal{N}^2 \colon V(N_1, N_2) \neq \varnothing\}$, weights of nodes are equal to the number of validators in them $w(N) = |N|$, and weights of edges are equal to $d(N_1, N_2)$. Note that we suppose that the graph G is connected which is confirmed by numerical experiments at $s \geqslant 10$.

D. Clustering into roughly equal clusters

As mentioned earlier, we are trying to achieve an approximate equality in the number of validators in all clusters. Of course, exact equality cannot be achieved because the total number of validators $|\mathcal{V}| = n$ may not be a multiple of k, and we divide into clusters not validators, but virtual nodes, which may contain up to $|N| \leqslant L$ validators.

Therefore, we formalize the notion of approximate equality as follows: let us call the number of validators in a cluster the cluster weight and denote it by

$$w(Cl) = \sum_{N \in Cl} |N| = \sum_{N \in Cl} w(N).$$

We will say that the clusters $Cl_1, Cl_2, \ldots, Cl_k$ roughly equal in number of validators if $w(Cl_j) \leqslant w(Cl_i) + L$ for every i, $j \in \{1, ..., k\}$, $i \neq j$. It is easy to see that such partitioning into clusters always exists.

E. Balancing block generation

Let in the division of the entire network of nodes, we obtain k subnetworks (clusters) of different sizes, namely, the sizes $w(Cl_1) \leqslant \ldots \leqslant w(Cl_k)$. Let an era consist of R slots. Then the smallest subnetwork generates one block per slot, and then all the others in proportion to their size, i.e. i-th network will generate $\left\lceil \frac{w(Cl_i)}{w(Cl_1)} R \right\rceil$ blocks per era.

Let S blocks be generated by some subnetwork in an era. In the j-th slot it is generated $s_j = \left\lceil \frac{S}{R} j \right\rceil - \left\lceil \frac{S}{R} (j-1) \right\rceil$ blocks. If more than one block needs to be generated in the i-th slot, the entire transaction pool is divided into "baskets", depending on the remainder of the transaction hash divided by s_j.

V. Hierarchical clustering

When selecting clustering methods that meet the above requirements, we settled on variations of hierarchical clustering methods. Modifications of divisive and agglomerative hierarchical clustering algorithms were considered (for more details of these algorithms, see, for example, [19], [20]).

A. Agglomerative hierarchical clustering (AHC)

In this approach, we cannot guarantee that the obtained clusters will have approximately equal sizes. Nevertheless, with the specially introduced distance between clusters, we

increase the probability that the weights of the maximum and minimum clusters will not differ by more than a factor of 2. Let us denote by $M = \frac{|\mathcal{V}|}{k}$ and let

$$wc(Cl) = \begin{cases} w(Cl) & \text{if } w(Cl) \leqslant M-1, \\ M-1+\frac{w(Cl)-M}{w(Cl)} & \text{if } w(Cl) > M-1. \end{cases}$$

Also we set $P(Cl_1, Cl_2) = \frac{M}{M-wc(Cl_1)} \frac{M}{M-wc(Cl_2)}$. Then we define the distances between clusters as follows

$$\text{dist}_{max}(Cl_1, Cl_2) = P(Cl_1, Cl_2) \max_{\substack{a\in Cl_1,\ b\in Cl_2 \\ V(a,b)\neq\varnothing}} d(a,b),$$

$$\text{dist}_{min}(Cl_1, Cl_2) = P(Cl_1, Cl_2) \min_{\substack{a\in Cl_1,\ b\in Cl_2 \\ V(a,b)\neq\varnothing}} d(a,b).$$

In some sense, we can think that $d_{max}(Cl_1, Cl_2)$ is a variation of complete linkage hierarchical clustering, and $d_{min}(Cl_1, Cl_2)$ is a variation of single linkage hierarchical clustering, with an additional factor penalizing "heavy" clusters.

B. Divisive hierarchical clustering (DHC)

Classic divisive hierarchical clustering algorithms [19] work as follows:

1) Each time, some cluster is divided into two parts.
2) In the divided cluster there are the most distant points, which become the starting points of new clusters.
3) Using these points, all other points in the cluster are added to some cluster by certain rules.

We modify each of these steps according to the requirements formulated above. First, we consider the following methods for finding two vertices for the second step:

1) Find the most distant vertices in a weighted graph ($O(m^2 s \log m)$ operations).
2) Find the most distant vertices in an unweighted graph ($O(m^2 s)$ operations).
3) Find some distant vertices in a weighted graph ($O(ms \log m)$ operations).
4) Find some distant vertices in an unweighted graph ($O(ms)$ operations).

The basic idea of finding some distant vertices in a graph is as follows: take an arbitrary graph vertex, call it N_0, find the vertex N_0 farthest from vertex N_0, then find the vertex N_2 farthest from vertex N_1 in the same way.

The most distant vertex from some vertex N of the graph is found as follows: Start the traversal of the graph starting from vertex N, the most distant vertex is the last vertex in this traversal. If we use breadth-first search (BFS) as a traversal, the pair N_1, N_2 will be the furthest in terms of number of edges between them (i.e. the furthest in an unweighted graph), and if we use Dijkstra's traversal [21], then the pair N_1, N_2 will be the most distant in terms of the length of shortest path between them (as sum of weights of edges, in the weighted graph). In an arbitrary graph this double traversal algorithm will not find a pair of vertices with maximal distance. But it will still find a pair of sufficiently distant vertices, which can be taken as a base pair.

Once the two base vertices are chosen, we need to partition the graph into two clusters. The basic idea of partitioning is very simple: one by one, we enroll the nearest vertices to the base vertices into their clusters, observing two important conditions each time: *cluster size ratio is as close to the desired ratio as possible* and *both clusters are connected graphs*.

We use the following approaches to determine the vertex closest to the cluster:

1) traverse vertices by BFS (aka wave method) as if the graph were unweighted, using edge weights only to determine the order of passing neighbors of the next vertex (we process the nearest neighbors first);
2) traverse vertices by Dijkstra's algorithm in a weighted graph, passing first the nearest vertices in terms of distance from the base vertex;
3) traverse vertices by Prim algorithm [22] (as in construction of minimal spanning tree), passing first the nearest vertices in the sense of distance from any vertex of the cluster part already built at that moment.

VI. Assessing the quality of clustering

To assess the quality of clustering we will use the following indicators.

A. Silhouette

Let $N_i \in Cl_j$ and $a(N_i, Cl_j) = \frac{1}{|Cl_j|} \sum_{N\in Cl_j} d(N_i, N)$ be the average distance from node N_i to other objects of cluster (compactness) and let

$$b(N_i, Cl_j) = \min_{Cl\in\mathcal{C}\setminus Cl_j} \frac{1}{|Cl|} \sum_{N\in Cl} d(N_i, N)$$

minimum average distance from node N_i to objects from other clusters (separability). Then the estimation of clustering quality will be the following value (silhouette [23]):

$$Sil(\mathcal{C}) = \frac{1}{m} \sum_{Cl\in\mathcal{C}} \sum_{N\in Cl} \frac{b(N,Cl)-a)N,Cl)}{\max\{a(N,Cl), b(N,Cl)\}}.$$

It is easy to see that $-1 \leqslant Sil(\mathcal{C}) \leqslant 1$. The closer $Sil(\mathcal{C})$ is to 1, the better the clustering.

B. Sphericity index

We call a node N_{Cl} the center of a cluster Cl if

$$\sum_{A\in Cl} d(A, N_{Cl}) = \min\left\{ \sum_{A\in Cl} d(A,N) \colon N \in Cl \right\}.$$

Let $S(\mathcal{C}) = \frac{1}{k} \sum_{Cl\in\mathcal{C}} \frac{1}{|Cl|} \sum_{A\in Cl} d(A, N_{Cl})$ be the average distance from cluster centers to cluster elements,

$$T(\mathcal{C}) = \frac{1}{k^2-k} \sum_{A,B\in\mathcal{C}, A\neq B} d(N_A, N_B)$$

be the average distance between cluster centers and

$$I_S = \frac{T(\mathcal{C}) - S(\mathcal{C})}{T(\mathcal{C})}.$$

The parameter I_S shows the sphericity of clustering. The closer this parameter is to 1, the better the clustering.

C. Diameter index

Let $D(X) = \{d(a,b)\colon a,b \in X\}$ be the diameter of the set X and let

$$I_D = \frac{D(\mathcal{N}) - \max_{Cl\in\mathcal{C}} D(Cl)}{D(\mathcal{N})}$$

be the diameter index. This index shows how much the diameter of clusters is smaller than the diameter of the whole network, and varies from 0 to 1. The closer this parameter is to 1, the better the clustering.

D. Standard deviation index

For the set X we denote

$$D_{sq}(X) = \left(\frac{1}{|X|^2 - |X|}\sum_{A,B\in X, A\neq B} d^2(A,B)\right)^{\frac{1}{2}}.$$

Then the standard deviation index is

$$I_{sq}(\mathcal{C}) = \frac{D_{sq}(\mathcal{N}) - E(D_{sq},\mathcal{C})}{D_{sq}(\mathcal{N})},$$

where $E(D_{sq},\mathcal{C}) = \frac{1}{k}\sum_{Cl\in\mathcal{C}} D_{sq}(Cl)$.

E. Uniformity of cluster weights

Let $I_W = \left(\frac{1}{k}\sum_{Cl\in\mathcal{C}}\left(\frac{w(Cl)}{|\mathcal{V}|} - \frac{1}{k}\right)^2\right)^{\frac{1}{2}}$ be standard deviation of the relative weights of clusters from the best value.

VII. Modeling

The input data for modeling are the set of nodes of blockDAG and the number of validators on each of them. To simulate distances $d_v(N_1,N_2)$ between nodes, we use a "geographic" model, where each node is defined by a point with coordinates (x,y). The distance between nodes $d_v(N_1,N_2)$ is modeled by the formula

$$d_v(N_1,N_2) = \mathrm{delay}(N_2) + c_v d_e(N_1,N_2),$$

where $\mathrm{delay}(N_2)$ is some random block generation delay by node N_2 (depends only on node N_2), $c_v \approx 1$ is some random factor symbolizing link instability (each time generated independently) and $d_e(N_1,N_2)$ is the Euclidean distance.

In total, 30 datasets were generated, each one consisted of 300 nodes with random coordinates and a random weight from 1 to 25.

The simulation results are summarized in Table I. During the simulation process, the algorithms were divided into 2 groups: agglomerative and divisive. To specify each particular algorithm, a number of parameters were introduced to describe the internal behavior of a given algorithm. These parameters are separated by a dash or comma.

The first parameter tells the type of algorithm, the second describes the distance function (d_1 or d_{max}). For agglomerative algorithms, the third parameter describes the distance function between clusters (dist_{max} or dist_{min}). For divisive algorithms, the third parameter denotes the algorithm that is used to find the farthest nodes. The name double_traverse denotes the double traverse method, where traverse is the graph traversal method (BFS or Dijkstra), global_traverse is that traverse is run in turn from each vertex to find the maximum distance in the entire graph. The last parameter specifies the traversal method for clustering (BFS, sorted_bfs, dijkstra, prim).

VIII. Conclusion

In this paper, we consider the problem of partitioning the Waterfall network (or similar blockDAG networks) into subnetworks to optimize network interaction between nodes. The paper considers the main approaches to such partitioning, in particular, the question of the need to change the block generation rule, and the basic requirements of the node clustering algorithm. The main approach is to publish nearest neighbor metric information by each validator in blockDAG at the moment when a new block is published by this validator.

We introduce a notion of virtual node to separate the storage level of blockDAG from network level and formulate methods to collect and publish information that reflects the quality of communication of virtual nodes. Methods are formulated that allow all nodes of the blockDAG network to retrieve from the published information an undirected weighted sparse graph of links between virtual nodes. Thus, we are reduced to the problem of graph clustering of virtual nodes, with additional conditions on the low computational complexity of the algorithm, and the approximate equality of the obtained clusters by the number of validators in them. This paper presents different variations of hierarchical agglomerative and divisional algorithms, modified specifically to solve this problem. The main clustering quality metrics are also considered. In addition, modeling of these algorithms was conducted, and the results of their work on model sets were obtained.

As a result of numerical simulation, the following results were obtained: agglomerative methods of clustering have much greater non-uniformity of cluster weights (the difference is up to two times). Other than on test-source data, the clusters from one virtual node very rarely appeared, which is absolutely inadmissible. Also, the computational complexity of agglomerative methods is too high. Nevertheless, the distance function version dist_{min} performs on average better than dist_{max}.

Among the divisional algorithms, different methods of base node extraction and node clustering were considered. Slightly better results are given by methods of global search for the most distant nodes in the graph global_dijkstra and global_bfs. However, these methods require significant computational cost. The fastest methods are those based on double BFS (double_bfs) traversal. For graph partitioning, you can also use the fastest parallel width traversal of vertices (bfs or its version with sorted_bfs). The method based on these traversals works for $O(msk)$, gives basic quality and uniformity indices only slightly inferior to global_dijkstra and global_bfs methods, and works stably on different test data. Therefore, this method

TABLE I
AVERAGE CLUSTERING INDICES BY 30 DATA SETS, $k = 4$, $m = 300$, $L = 25$.

Clustering Method	Sil	I_D	I_S	I_{sq}	$I_W \cdot 100\%$
AHC, d_1, dist_{max}	0.26	0.12	0.28	0.21	5.16%
AHC, d_{max}, dist_{max}	0.25	0.10	0.24	0.21	5.16%
AHC, d_1, dist_{min}	0.40	0.24	0.49	0.36	5.87%
AHC, d_{max}, dist_{min}	0.40	0.23	0.50	0.35	5.51%
DHC, d_{max}-double_dijkstra-dijkstra	0.43	0.27	0.54	0.37	0.26%
DHC, d_1-double_dijkstra-dijkstra	0.43	0.27	0.54	0.37	0.28%
DHC, d_{max}-global_dijkstra-dijkstra	0.44	0.32	0.56	0.39	0.17%
DHC, d_1-global_dijkstra-dijkstra	0.44	0.31	0.52	0.39	0.20%
DHC, d_{max}-global_bfs-bfs	0.44	0.32	0.56	0.39	0.19%
DHC, d_1-global_bfs-bfs	0.44	0.32	0.56	0.39	0.19%
DHC, d_{max}-global_bfs-sorted_bfs	0.45	0.33	0.56	0.39	0.25%
DHC, d_1-global_bfs-sorted_bfs	0.44	0.33	0.55	0.39	0.20%
DHC, d_{max}-double_bfs-bfs	0.43	0.29	0.53	0.37	0.23%
DHC, d_1-double_bfs-bfs	0.43	0.29	0.53	0.37	0.23%
DHC, d_{max}-double_bfs-sorted_bfs	0.43	0.28	0.53	0.37	0.15%
DHC, d_1-double_bfs-sorted_bfs	0.42	0.29	0.51	0.37	0.22%
DHC, d_{max}-double_bfs-dijkstra	0.42	0.25	0.51	0.36	0.17%
DHC, d_1-double_bfs-dijkstra	0.43	0.27	0.52	0.37	0.17%
DHC, d_{max}-double_bfs-prim	0.41	0.24	0.48	0.35	1.77%
DHC, d_1-double_bfs-prim	0.41	0.26	0.51	0.35	1.66%

can be the basis of the method for partitioning the blockDAG Waterfall network into subnetworks.

In the future, the network layer needs to be augmented in connection with to the partitioning into subnetworks. Also, an important issue to study will be the robustness of the obtained clustering methods and distance metrics to intentional distortion by dishonest blockDAG participants.

REFERENCES

[1] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," in *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125.

[2] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: A Fast and Scalable Cryptocurrency Protocol," Cryptology ePrint Archive, Paper 2016/1159, 2016. [Online]. Available: https://eprint.iacr.org/2016/1159

[3] Y. Sompolinsky, S. Wyborski, and A. Zohar, "PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus," Cryptology ePrint Archive, Paper 2018/104, 2018. [Online]. Available: https://eprint.iacr.org/2018/104

[4] A. M. Khalifa, A. M. Bahaa-Eldin, and M. A. Sobh, "Blockchain and its Alternative Distributed Ledgers - A Survey," in *2019 14th International Conference on Computer Engineering and Systems (ICCES)*, 2019, pp. 118–125.

[5] K. Gai, Z. Hu, L. Zhu, R. Wang, and Z. Zhang, "Blockchain Meets DAG: A BlockDAG Consensus Mechanism," in *Algorithms and Architectures for Parallel Processing*, M. Qiu, Ed. Cham: Springer International Publishing, 2020, pp. 110–125.

[6] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "SoK: Diving into DAG-based Blockchain Systems," arXiv:2012.06128v2, pp. 1–36, 2020, https://arxiv.org/abs/2012.06128.

[7] Q. Nguyen, A. Cronje, M. Kong, E. Lysenko, and A. Guzev, "Lachesis: Scalable Asynchronous BFT on DAG Streams," arXiv:2108.01900v1, pp. 1–45, 2021, https://arxiv.org/abs/2108.01900.

[8] 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*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 17–30. [Online]. Available: https://doi.org/10.1145/2976749.2978389

[9] A. E. Gencer, R. van Renesse, and E. G. Sirer, "Short Paper: Service-Oriented Sharding for Blockchains," in *Financial Cryptography and Data Security*, A. Kiayias, Ed. Cham: Springer International Publishing, 2017, pp. 393–401.

[10] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 583–598.

[11] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 931–948. [Online]. Available: https://doi.org/10.1145/3243734.3243853

[12] N. Durov. (2019) Telegram Open Network. [Online]. Available: https://ton.org/ton.pdf

[13] C. Li and L.-J. Zhang, "A Blockchain Based New Secure Multi-Layer Network Model for Internet of Things," in *2017 IEEE International Congress on Internet of Things (ICIOT)*, 2017, pp. 33–41.

[14] H. Bai, G. Xia, and S. Fu, "A Two-Layer-Consensus Based Blockchain Architecture for IoT," in *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, 2019, pp. 1–6.

[15] S. Grybniak, Y. Leonchyk, I. Mazurok, O. Nashyvan, and R. Shanin, "Waterfall: Gozalandia. distributed protocol with fast finality and proven safety and liveness," 2022, 10 pages.

[16] S. Grybniak, D. Dmytryshyn, Y. Leonchyk, I. Mazurok, O. Nashyvan, and R. Shanin, "Waterfall: A Scalable Distributed LedgerTechnology," 2022, 6 pages.

[17] Ethereum 2.0. [Online]. Available: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md#topics-and-messages

[18] Kaspa. Subnetworks. [Online]. Available: https://kaspa.gitbook.io/kaspa/archive/archive/components/kaspad-full-node/reference/subnetworks-1

[19] L. Hubert, "Monotone invariant clustering procedures," *Psychometrika*, vol. 38, no. 1, pp. 47–62, 1973.

[20] M. Roux, "A comparative study of divisive and agglomerativehierarchical clustering algorithms," *Journal of Classification*, vol. 35, pp. 345–366, 2018.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[22] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[23] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and*

Applied Mathematics, vol. 20, pp. 53–65, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257