Locality Aware Skip Graph

Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap
KOC University, Department of Computer Engineering
Istanbul, Turkey
{yhassanzadeh13, akupcu, oozkasap}@ku.edu.tr

***Abstract*—As a DHT-based distributed data structure, Skip Graph plays a key role in the P2P cloud storage, distributed online social networks, search engines and other DHT-based applications. So far, the traditional identifier assignment algorithms do not consider the Skip Graph node's locations. Since in the Skip Graph nodes are connected together based on their identifiers, neglecting the nodes locality in the identifiers assignment results in high end-to-end latency in the overlay network which negatively affects the performance of the system. In this paper, we proposed a method to assign the Skip Graph nodes identifier's considering their location information and make the nodes locality aware. In the proposed dynamic and fully decentralized algorithm called DPAD, instead of assigning the peer's identifier uniformly at random, a locality aware identifier will be assigned to the nodes at their arrival time to the system based on their distances to some super nodes named Landmarks. Considering the locality awareness as the similarity of the distances between the nodes in the overlay and underlay network, the performance evaluation of the DPAD algorithm compare to the best known static and dynamic algorithms results in about 82% improvement in the locality awareness of the node's identifier and about 40% improvement in the search query end-to-end latency.**

***Keywords*—Huffman algorithm, Euclidean distance, Skip Graph, Locality Awareness, P2P systems, Hop-to-hop latency, End-to-end latency**

I. Introduction

Skip Graph [1] is a distributed data structure that stores data using {key, value} pairs. Data retrieval is possible by using the key associated with the specific data item. Scalability [2], fault tolerance, fast searching, correctness under concurrency and load balancing [3] make Skip Graph a suitable underlying routing infrastructure for the P2P cloud storage applications [4][5]. In such applications, there exist some clients and servers. Client owns the data item and server is one of the nodes in the system that stores that data item. Client finds the suitable server via a search in the Skip Graph. It then sends its data to to the server for the upcoming saving processes. The server will store the data item on behalf of the client. Also, the Skip Graph's ability to store and retrieve a data item is very similar to the Distributed Hash Table (DHT) [6]. This makes the Skip Graph to be considered as the DHT alternative in many of DHT's applications such as P2P storage systems [7] [8], P2P online social networks [9] and P2P search engines [10].

In the Skip Graph, each node has a unique name id and numerical id associated with itself. The nodes will be connected together based on their name ids in multiple levels. The more two nodes have common prefix bits in their name ids, they will connected to each other in the more levels of Skip Graph. In a Skip Graph, the most common search query is to find the node with the specific numerical id which is called *Search by Numerical ID*. The Skip Graph with the total number of N nodes, could perform a search (by numerical id) query by traversing $O(logN)$ hops on the average.

The performance of a Skip Graph is measured by its query processing and response time. Since the most common query of a Skip Graph is the *Search by Numerical ID*, the average end-to-end latency (the latency between the source and destination of a query) [11] of the search by numerical id would be considered as the main performance metric of a Skip Graph. The Skip Graph topology is determined based on the name ids of the nodes. Therefore the network paths in the underlying Skip Graph infrastructure heavily influenced by the name id assignment strategy. So far, the node's name id in the Skip Graph are chosen uniformly at random from an identifier space. This makes the underlying network path between two hops to be chosen uniformly at random. In such systems, there would be a huge difference between the underlying network paths in the Skip Graph infrastructure and the overlay network paths in reality. The average hop-to-hop latency (the latency between two consecutive hops) could increase and hugely affect the average end-to-end latency per search query and therefore makes the whole system inefficient in the terms of query processing and response time.

One solution to improve the latency problem in the Skip Graph is to make it locality aware such that the node's identifiers reflect their overlying network's location information. As much as two nodes are closer to each other in the overlay network, their name id would be much more similar to each other. Therefore in the Skip Graph locality awareness will be defined as assigning name ids to the nodes such that the more two nodes are close to each other, the more common prefix they would have in their name ids.

As described in the beginning of the introduction the Skip Graph could be used instead of DHT in many of DHT-based applications. Therefore, by making the Skip Graph locality aware and optimizing the end-to-end latency in its search queries, we will also optimize the query processing and response time in many of such applications. Using a locality aware Skip Graph instead of a DHT will also enable the DHT-based storage system to perform the data replications based on the location of the peers. Contrary to DHT, in the Skip Graph each node has a name id alongside with its traditional key (numerical id). Therefore, when the name ids

reflect the locality, the placement of the replicas could be handle based on their location information.

In this paper, we did the following contributions:

- We proposed the **first** dynamic fully decentralized algorithm (DPAD) to assign locality aware name ids to the nodes of Skip Graph.
- We improved the end-to-end latency of the Skip Graph search queries using our proposed DPAD name id assignment algorithm.
- We developed a simulation environment **SkipSim** for simulating the name id assignment algorithms, look up operations, fault tolerance etc. on the Skip Graph in both centralize and decentralize manner.
- We redesigned the proposed identifier assignment algorithms for DHT nodes to be compatible with the Skip Graph features, simulated them in the SkipSim and compared their performance with our DPAD algorithm.
- Our simulation results show that our DPAD algorithm 40% improves the end-to-end latency of the search queries and performs about 82% better in preserving the locality awareness of the Skip Graph nodes than the previous best known dynamic, fully decentralized counterparts.

In the rest of the paper, we first introduce the scheme of Skip Graph along with its search by numerical id and problem definition in Section II. In the Section III, we present DPAD algorithm to provide locality aware identifiers for the nodes of Skip Graph. We review the related works in the Section IV. In Section V, we evaluate our algorithm performance with respect to other similar algorithms. Finally we conclude in Section VI.

II. Skip Graph

A. Construction

Skip graph could be considered as the distributed version of the Skip List [12]. It offsets the drawbacks of the Skip List like single point of failure [13], lack of redundancy [13] and manifestation of hot spot [14].

Skip Graph [1] consist of sorted doubly-linked lists divided by the levels (Figure 1). In addition to the numerical id as in the the Skip List, each node in the Skip Graph also has a name id which is a binary string. Nodes are sorted in the level 0 based on their numerical id (like the Skip List). In i^{th} level there would be at most 2^i doubly-linked lists that are sorted in the same order of the level 0 and each node participates only in one of them, until the nodes are divided into singletons after $O(logN)$ levels on average, where N is the number of nodes.

In the i^{th} level the nodes that are located in the same sub-list would have at least i common bits prefix in their name ids. For example in Figure 1 nodes with numerical ids 12 and 39 have 2 bits common prefix in their name ids, therefore they will be in a same sorted doubly-linked sub-list in the levels 1 and 2. However, since nodes with the numerical ids 71 and 28 have only one bit common prefix, they only shared a sub-list in the level 1.

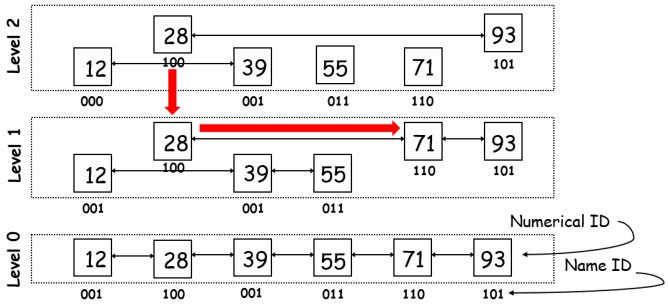

Fig. 1: Searching for numerical id 71 starting at the node with numerical id 28

B. Search by numercial ID

As a member of DHT based distributed data structure family, Skip Graph supports the regular DHT operations like *insertion* and *get*. In the Skip Graph the *get(x)* operation is defined as a search for the address of the node that has x as its numerical id [1]. This operation is defined as a *Search By Numerical ID*, and is very similar to the searching in the Skip List, except that in the Skip Graph each node could initiate a search. On the contrary, in the Skip List, all the searches should be initiated by the left top most node (start node). Skip Graph could perform this operation with logarithmic time order on the average with respect to the number of the nodes in the zero level.

Figure 1 shows an example of search for numerical id. The search initiates by the node 28 for the node that has the numerical id of 71. Since the search target 71 is greater than the numerical id of the search initiator (28), the search in all the levels with continue to the *right* side. Red arrows show the procedure of the search step by step. In the beginning of the search, at the level 2, there are two consecutive nodes with numerical ids 28 and 93 that are less than and greater than the search target (71) respectively. Therefore, the search will continue one level down (level 1) with the node 28. In level 1, node 28 will find the search target after performing a linear search.

Another example of searching for the numerical id is shown in Figure 2. A node with the numerical id of 55 initiates a search for the node with numerical id of 14. In the case that the target numerical id does not exist in the Skip Graph (like this example which a node with numerical id of 14 does not exist) algorithm returns the node with the numerical id that is the greatest node in the system that is less than the search target (12 in this case). The search starts at the topmost level (2) by the initiator of the search (55 in this example). Since the search target (14) is less than the initiator, in all the lower levels the linear search will continue to the *left* direction. In the level 2, since 55 has no left neighbor and hence it could not perform the linear search, the search will switch to the level one. In that level, the two consecutive elements (one greater than the search target and one less) will be found immediately by just checking the left neighbor of 55 (node with the numerical id 39). Therefore the search will continue at the lower level (0)

with the node that has the smaller numerical id (39). In the level 0, 39 will perform a linear search for 14 and it will reach the node with numerical id 12. At this point the search will be terminated. Since we reached to an element that is smaller than the search target, continuing the linear search will make no sense any more and node with the numerical id of 12 will be returned by the algorithm as the result of the search.

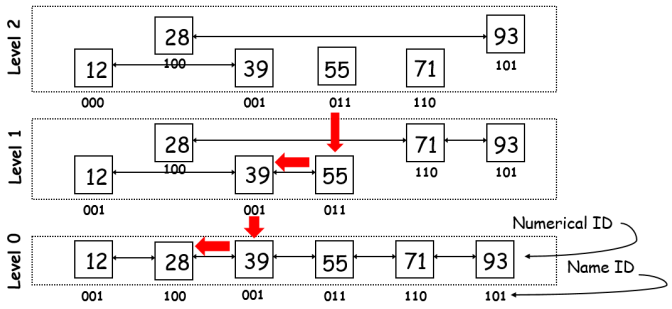

Fig. 2: Searching for numerical id 14 starting at the node with numerical id 55

III. Dynamic Prefix Average Distance (DPAD) algorithm

In our proposed *Dynamic Prefix Average Distance (DPAD)* algorithm, with N nodes in the system, we need to have at least $logN$ landmarks. Landmarks are some super peers in the system that are placed according to the expected density of the nodes in the system. To obtain a *nameID* for a single node P, this algorithm will receive two arrays as the input: *Distance* and *Average* both with the size of K which is corresponds to the number of the landmarks in the system. The i^{th} element of the *Distance* array is the distance of node P to the i^{th} landmark, whereas the i^{th} element of the *Average* array is the average distance of all the present nodes in the system to the i^{th} landmark.

In the DPAD algorithm each landmark in the system has a dynamic prefix that is determined before any node arrives to the system. In order to consider the density by employing the Huffman algorithm on the landmarks.

To generate the Huffman prefixes for the landmarks, DPAD first finds the landmark that has the minimum total distances to all other landmarks. This landmark considered as the most dense landmark in the system. DPAD then uses the distance of each landmark to that most dense landmark as its weight for the Huffman algorithm. By running the Huffman algorithm, the prefix of landmarks will be determined.

Algorithm III.1 shows the DPAD method in more details. In the beginning of the algorithm, the closest landmark to the node P will be found and the name id of the node P will have the prefix of its closest landmark (Algorithm III.1 line 1). DPAD then compares the each element of *Distance* array with the corresponding element of the *Average* array and assigns the *nameID* of that node based on that element (lines 2-6). Then it will update the *Average* matrix with the *Distance* array (line 7). Here, the update is to compute the new average distance of all to all the landmarks including the *Distance* vector of node P.

There may already exist a node in the system with the same name id that the DPAD algorithm wants to assign to the node P. This situation occurs if each member of *Distance* array of that node and node P be in the same situation with respect to the corresponding element in the *Average* array at the time that their name ids was going to be assigned. In order to handle this situation, DPAD algorithm checks to see that whether the name id it generates is already assigned to another node or not. (line 8) If the generated nameID has been already assigned, the algorithm tries to find the closest available name id to the generated nameID (line 9). DPAD will do this by finding the closest available leaf of the name id tree to the generated nameID. The name id tree is a binary tree with a null string as the root. Each right child of a node has the name id of its parent extended with a *one* bit and each left child of a node has the name id of its parent extended with a *zero* bit.

Algorithm III.1: Dynamic Prefix Average Distance (DPAD)

Input: array Distance, array Average
Output: string nameID, array Average

```
1  nameID = closestLandmarkPrefix(Distance); //Finding the
   closest landmark to the node P
2  for i=0; i < NumberOfLandmarks; i++ do
3  |  if Distance[i]>Average[i] then
4  |  |  nameID = nameID + 0;
5  |  else
6  |  |  nameID = nameID + 1;
7  Update(Distance, Average); if nameID is already
   assigned to another node then
8  |  return (nextAvailable(nameID), Average);
9  else
10 |  return (nameID, Average);
```

DPAD is a dynamic algorithm, it will generate a name id for each new arrival node based on the current situation of the system. Although by using the DPAD algorithm, the maximum size of name ids in the system may increase, in order to preserve the logarithmic behavior of Skip Graph in responding the search queries, the number of the levels in the skip graph will not be changed. Therefore, in a graph with 6 levels, two nodes may have more than 6 bits common prefixes but they will be in a same list up to the 6th level.

IV. Related Works

So far, several approaches have been proposed to assign the node identifiers in a distributed data structure. Almost all the methods have been designed for the distributed hash tables (DHTs) and there was no specific method for the case of Skip Graph. We divided the identifier assignment strategies into two classes: Dynamic algorithms and Static algorithms.

A. Dynamic Algorithms

In this class of algorithms, the identifer of a new node could be assigned at the time node arrives to the system. *LAND*

[15] chooses the identifiers of the DHT nodes uniformly at *random*. In *LDHT* [16] the DHT identifiers were considered as a conjunction of two independent binary strings: the first part is a prefix containing the *location information* of the nodes (ASN [17]), while the second part is completely chosen uniformly at *random*.

Zhou et al [18] proposed the employing of *hierarchical* address. The hierarchy consist of region, sub region, leaf region and a random part. In this method, the name ids have dynamic length. The more an area is crowded, the longer name ids its related nodes will have.

Freedman et al [19], experimentally showed that *IP prefixes* would not reflect the relation between the location of the peers. There are several example of the peers with at least 21 bits mask which are about 1000 miles away from each other while according to their IP prefixes, they would assumed to be close neighbors.

B. Static Algorithms

Static algorithms need the locality information of all the nodes in the system before generating the name ids. Therefore, in this class of algorithms, all the nodes should be present and the total node number of the system should be determined. Then the algorithm could generate the identifiers of the nodes. Hence, static algorithms could not assign the name ids upon the arrival of a new node. For each new node that joins the system, the static algorithm should be run and assign new name ids to all the nodes by considering the new relations between the location of new node and other existing nodes.

LMDS [20] does not provide the locality-based name ids in the desired format of the skip graph. On the contrary to the skip graph which the name ids are the string of binary digits containing the locality information as their common prefix, the outcome of this algorithm is an integer which could not be considered as the name id of the skip graph by itself and needs couple of extra processes.

In [21], several approaches were discussed based on *MDS* behavior to assign some kinds of identifiers to some points. Instead of choosing landmarks and positioning all the nodes with respect to the landmarks, in *PMDS* some pivots will be selected from the ordinary nodes and the points will be positioned with respect to each other. The algorithm should be executed per each new pivot and also for each new arrival node. *High dimensional scaling (HDS)* [21] selects K nodes and measures the distance of the other nodes to these K nodes. This distances would provide a K-dimensional coordination. Final coordination would be calculated using the adjacency matrix.

Allan et al [22] proposed a method based on LMDS to convert the physical coordination of the nodes to the network coordination such that the Euclidean distance between each two nodes in the network reflects the latency. Authors also used Hilbert Curve to improve the efficiency of LMDS in very dense networks, for example in a picture file which represents a graph of million pixels. To reduce the effect of the noise on LMDS in the very large networks, Lee et al [23] also suggested to divide the input into L parts, run LMDS on each part and combine the results, instead of run the LMDS on the whole input.

In Geo-Peer [24] the authors used Delaunay triangulate [25] based on the Voroni diagram [26] in order to generate a locality aware P2P system such that each peer is connected to its closest neighbor. In this method they used several message exchange steps in order to provide a Delaunay triangle between each 3 neighbors. For each new node several messaged will be exchanged in the system for the neighbor discovery, network maintenance and Delaunay triangulation. This method has some advantages and disadvantages as: a node will be connected to its closest neighbors, the approach is completely P2P and routing would be efficient. However, it would not generate locality aware identifiers. Considering the locality aware identifier assignment to each node, the maximum degree of each node (number of the links it has to its neighbors) is not static in this method and is related to the geographical location of the node. By employing this method in the Skip Graph, some nodes may have more than $O(\log n)$ links that is in contrast to the constraint of the Skip Graph and will make the search query inefficient.

Table I shows a more detailed comparison between the identifier assignment strategies in DHT-based data structures and our proposed dynamic DPAD algorithm. One criteria to compare different methods is is decentralization, which is "Full" if all the peers could perform the identifier assignment and is "Hybrid" if only some super peers (for example landmarks) could assign the node identifiers. For the case of locality awareness a method is "Full" if the nodes identifier only contains their location information and is "Hybrid" if in addition to the location information, the nodes identifier also contains a random part.

V. Simulation and Results

A. Simulation Environment: SkipSim

We designed and implemented a simulation environment named *SkipSim* for simulating the proposed algorithms on the Skip Graph and evaluating their performance in the case of preserving the locality awareness and search querie's end-to-end latency. *SkipSim* is able to generate the random topologies based on the system criteria.

In the *SkipSim*, the existence probability of the nodes in each point is proportional to their positioning with respect to the landmark points. Assuming nodes will appear near to the landmarks with higher probability, Equation 1 shows the chance of each point regarding to each landmark. In this equation, $chance_{ij}$ is the chance of i^{th} point in the screen to be selected as a node location with respect to the j^{th} landmark and d_{ij} is the distance between them. $maxDistance$ is the maximum possible distance between two nodes in the *SkipSim* (The screen diameter). In the Equation 2 $chance_i$ is the sum of all the i^{th} point's chances with respect to all the K landmarks in the system. The probability distribution of the nodes is obtained by the Equation 3. In this equation p_i is the probability that a node arrives at the point i in the screen and $\sum_{j=1}^{n} chance_j$ is the sum of all the points chance in the screen.

Method	Behavior	Decentralize	Locality Awareness
LAND[15]	Dynamic	Full	No
LDHT[16]	Dynamic	Hybrid	Hybrid
Hierarchical assignment[18]	Dynamic	Hybrid	Hybrid
LMDS family[20][21][22]	Static	Hybrid	Full
Dynamic Prefix Average Distance (DPAD)	**Dynamic**	**Full**	**Full**

TABLE I: Comparison between various methods of identifier assignment

We generated 100 random topologies (each with 64 nodes and 6 landmarks) based on the described distribution and evaluated each algorithm based on them.

$$\text{chance}_{ij} = 1 - \frac{d_{ij}}{maxDistance} \quad (1)$$

$$\text{chance}_i = \sum_{j=1}^{K} chance_{ij} \quad (2)$$

$$\text{p}_i = \frac{chance_i}{\sum_{j=1}^{n} chance_j} \quad (3)$$

B. Related algorithms implementation

We implemented the related proposed algorithms to compare their performance in the case of locality awareness and query efficiency with our proposed DPAD algorithm. The implementation details of these algorithms listed as follows:

1) LDHT: In this algorithm implementation, the name id of each node is its closest landmark prefix following with a random sub-string. In this implementation, we assigned a fix prefix to each landmark and assume each landmark defines a specific ASN by that prefix. All the nodes around a landmark will then belong to the same ASN group.

2) Hierarchical Assignment: In order to implement this algorithm, we employed the same strategy of generating Huffman prefixes for the landmarks as what we did in the DPAD algorithm. The second part of the name ids would be chosen uniformly at random from the available name ids. By this implementation the density of the landmarks will determine the regions and sub-regions. The more an area is crowded by the landmarks, Huffman algorithm would generate more sub-regions by assigning similar prefix to that landmarks.

3) LMDS: Having N nodes and K landmarks in the system a $N \times K$ *Distance* matrix will be generated such that $Distance[i][j]$ is the network distance of the i^{th} node to the j^{th} landmark. LMDS will be run on the *Distance* matrix and will output a single value for each node. Nodes will be ranked based on their LMDS value from zero to $N-1$ in the ascending or descending order. The name id of each node would be the conversion of its rank to the binary considering the system size of name ids.

4) Dynamic Prefic LMDS (DPLMDS): We designed and implemented this static algorithm by combining the Hierarchical assignment and LMDS algorithms together in order to obtain another static algorithm to compare with our proposed DPAD algorithm. In the DPLMDS algorithm, the name id of each node consist of two parts: The Huffman prefix of the closest landmark in the system to that node followed by the output of the LMDS algorithm as described above.

C. Results

1) Locality Awareness: To evaluate each name id assignment algorithm we ran the algorithm for that 100 saved random topologies. For each topology, after assigning the name ids to all the nodes, average distance of **each** node to all its neighbors in the underlying network was computed. Then the average distance of **all** the nodes in the system regarding to their underlying network's neighbors calculated. Finally, the average distance of each node to all its underlying network's neighbors in 100 random topologies was reported as the performance metric of each algorithm regarding to the locality awareness. Figure 3 shows the average distance of each node to its look up table neighbors in each algorithm. In this figure the x-axis corresponds to the algorithms whereas the y-axis shows the average distance. As it is shown in this figure, our proposed **dynamic, fully decentralized** DPAD algorithm has the same performance as the **static** DPLMDS algorithm and improves the locality awareness with the gain of about 24% in comparison to the **hybrid decentralized** Hierarchical approach and with the gain of about 82% in comparison to the random assignment (LAND) which was the only dynamic, fully decentralized algorithm before DPAD.

Figure 4 shows the average distances between all pairs of the nodes in the system based on their name ids common prefix after employing the DPAD algorithm. As this figure shows, the more two nodes are closer to each other, the more common prefix they would have in their name id.

2) End-to-end latency in the search query: In order to evaluate the effect of the name id assignment algorithms on the end-to-end latency of the search query, we ran 1000 random search by numerical id queries originated at the random nodes per each topology for 100 random topologies. By modeling the overlying network RTT with the physical distances between the nodes, we measured the total distance that a packet travels on the average to perform a search by numerical id for 1000 random queries as the performance metric of each algorithm.

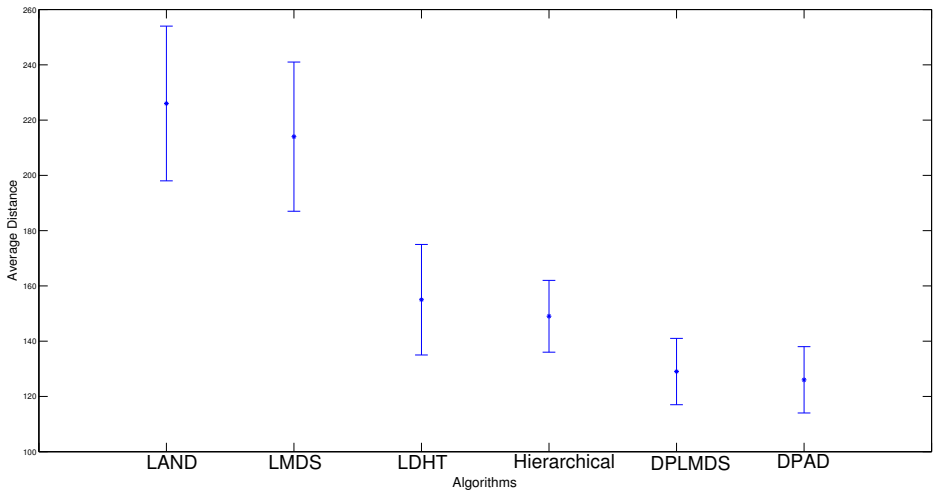

Fig. 3: Average distance of the nodes to their look-up table's neighbors in the name id assignment algorithms

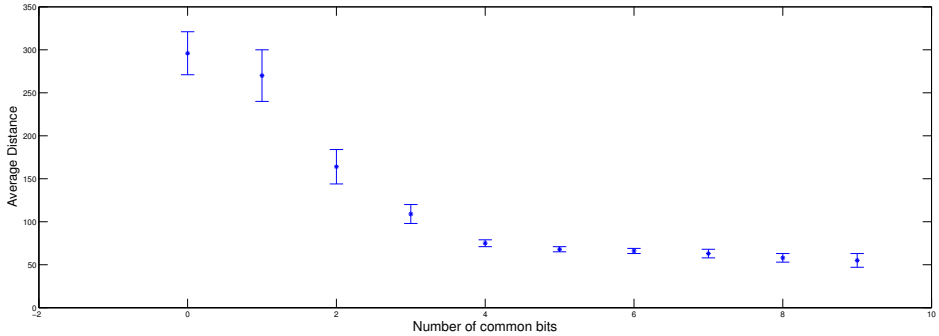

Fig. 4: Average distance between the nodes with different common prefix in the Dynamic Prefix Average Distance (DPAD) algorithm

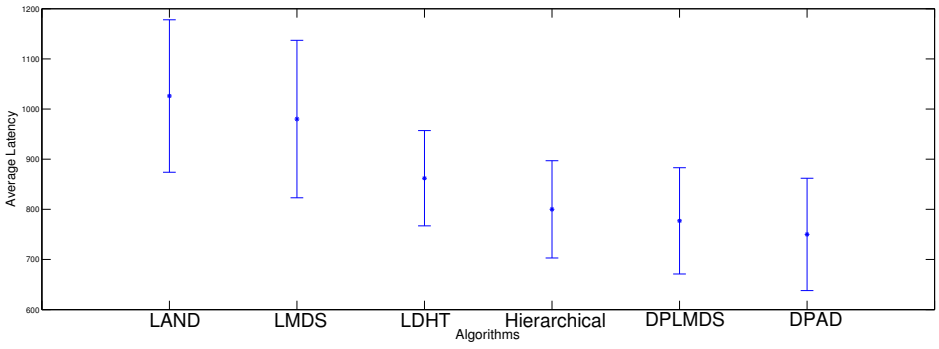

Fig. 5: Effect of name id assignment algorithms on the end-to-end latency of the search by numerical id

Figure 5 shows the performance of name id assignment algorithms on the end-to-end latency of the search by numerical id. The x-axis corresponds to the algorithms and y-axis shows the average end-to-end latency.

As Figure 5 shows, our proposed **dynamic, fully decentralized** DPAD algorithm has the similar performance to the **static** DPLMDS in the case of improving the search query end-to-end latency. In comparison to the random assignment algorithm (LAND) which was the only dynamic and fully decentralized algorithm before ours, DPAD improves the end-to-end latency with the gain of 40%. Although in comparison to the **hybrid decentralized** Hierarchical assignment of the identifiers, DPAD improves the end-to-end latency with the gain of 6%, the main difference between DPAD and Hierarchical assignment arias in the case of their decentralize behavior. In contrast to the Hierarchical assignment, DPAD assigns the nodes identifier in a fully decentralized manner. In DPAD all the existing peers in the system could assign a locality aware name id for a new node, whereas in the Hierarchical assignment only some super peers who knows the hierarchical division in the system could perform the assignment.

VI. Conclusion

In this paper with the aim of reducing the end-to-end latency in the search queries, we proposed a novel algorithm to make the Skip Graph nodes locality aware. We addressed the locality awareness problem to the method in which the Skip Graph node's name ids are assigned. The proposed algorithm called *Dynamic Prefix Average Distance (DPAD)* is a dynamic and fully decentralized algorithm which assigns the identifiers of the Skip Graph node's based on their location information at their arrival time to the Skip Graph. DPAD which is so far the only proposed locality aware identifier assignment algorithm for the Skip Graph, assigns the name ids by analysing the distance of the nodes with respect to some super nodes in the system called landmarks.

In order to analyze the performance of DPAD algorithm with other algorithms we implemented a simulator called SkipSim. In the SkipSim we analysed the performance of the DPAD algorithm with other traditional identifier assignment algorithms in the terms of nodes locality awareness and end-to-end latency of the search queries.

In the SkipSim we considered the physical distance between each two nodes as their RTT latency and measured the end-to-end latency of search by numerical id as the performance of each algorithm in the case of search queries. We also considered the average RTT latency of each node to its neighbors in the underlay network as the locality awareness metric for each algorithm. Comparing to the other proposed algorithms, our proposed DPAD algorithm improves the locality awareness of the Skip Graph nodes with the gain of about 82% and end-to-end latency of the search query with the gain of about 40% considering the dynamic and fully decentralized behaviors.

References

[1] J. Aspnes and G. Shah, "Skip graphs," *ACM TALG*, vol. 3, no. 4, 2007.

[2] A. Tanenbaum and M. Van Steen, *Distributed systems*. Pearson Prentice Hall, 2007.

[3] S. Batra and A. Singh, "A short survey of advantages and applications of skip graphs," *IJSCE*, vol. 3, no. 5, 2013.

[4] E. Udoh, *Cloud, grid and high performance computing: emerging applications*. Information Science Reference, 2011.

[5] T. Shabeera, P. Chandran, and S. Kumar, "Authenticated and persistent skip graph: a data structure for cloud based data-centric applications," in *International Conference on Advances in Computing, Communications and Informatics*, ACM, 2012.

[6] W. Galuba and S. Girdzijauskas, "Distributed hash table," in *Encyclopedia of Database Systems*, pp. 903–904, Springer, 2009.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM*, vol. 31, no. 4, pp. 149–160, 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*, vol. 31. ACM, 2001.

[9] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *2nd ACM EuroSys*, pp. 46–52, 2009.

[10] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of P2P systems*, vol. 6, pp. 68–72, 2003.

[11] B. Van Schewick, *Internet architecture and innovation*. MIT Press, 2010.

[12] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and commutative hashing," in *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, vol. 2, pp. 68–82, IEEE.

[13] M. Van Steen and A. Tanenbaum, "Distributed systems principles and paradigms," *Prentice Hall,*, vol. 1, no. 2, 2003.

[14] T. Crain, V. Gramoli, and M. Raynal, "No hot spot non-blocking skip list," in *33rd ICDCS 2013*, pp. 196–205, IEEE.

[15] I. Abraham, D. Malkhi, and O. Dobzinski, "Land: Locality aware networks for distributed hash tables," tech. rep., Tech. Rep. TR 2003-75, Leibnitz Center, The Hebrew University.

[16] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, and X. Li, "Ldht: locality-aware distributed hash tables," in *IEEE ICOIN 2008*, pp. 1–5.

[17] G. Huston, "Exploring autonomous system numbers," *The Internet Protocol Journal*, vol. 9, no. 1, pp. 2–23, 2006.

[18] S. Zhou, G. R. Ganger, and P. A. Steenkiste, "Location-based node ids: Enabling explicit locality in dhts," *Technical Report, Carnegie Mellon University*, 2003.

[19] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic locality of ip prefixes," in *5th ACM SIGCOMM*, pp. 13–13, USENIX Association, 2005.

[20] V. De Silva and J. B. Tenenbaum, "Sparse multidimensional scaling using landmark points," tech. rep., Technical report, Stanford University, 2004.

[21] U. Brandes and C. Pich, "Eigensolver methods for progressive multidimensional scaling of large data," in *Graph Drawing*, pp. 42–53, Springer, 2007.

[22] A. Allan, R. Humphrey, and G. D. Fatta, "Non-euclidean internet coordinates embedding," in *13th ICDMW, 2013*, IEEE.

[23] S. Lee and S. Choi, "Landmark mds ensemble," *Pattern Recognition*, vol. 42, no. 9, pp. 2045–2053, 2009.

[24] F. Araújo and L. Rodrigues, "Geopeer: A location-aware peer-to-peer system," in *3rd IEEE NCA 2004*.

[25] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[26] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM CSUR*, vol. 23, no. 3, pp. 345–405, 1991.