Future Generation Computer Systems 84 (2018) 32–46

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Decentralized and locality aware replication method for DHT-based P2P storage systems

Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, Öznur Özkasap *

Department of Computer Engineering, Koç University, İstanbul, Turkey

HIGHLIGHTS

- We propose a dynamic and fully decentralized locality aware replication algorithm.
- We propose a landmark-based locality aware name ID assignment algorithm.
- We optimized Skip Graph simulator SkipSim for large-scale performance analysis.
- Our approach GLARAS improves the access delay of public and private replications.
- Our approach LANS improves the locality awareness of name IDs.

ARTICLE INFO

Article history:
Received 19 November 2017
Received in revised form 2 February 2018
Accepted 3 February 2018
Available online 17 February 2018

Keywords:
Skip Graph
Distributed hash table
DHT
P2P cloud storage
Locality aware network for DHTs
Replication

ABSTRACT

Skip Graph, a type of DHT, plays an important role in P2P cloud storage applications, where nodes publicly or privately store, share, and access data. Nowadays P2P storage systems are widely using replication to support data availability, reliability, and maintainability. With replication, the main consideration is determining peers to replicate the data. Traditional replication algorithms are partially randomized and employ rigid assumptions about nodes' distribution. This results in high access delay between nodes and their closest replicas, which degrades the system performance. We propose *GLARAS* , a dynamic and fully decentralized locality aware replication method for Skip Graph. In contrast to the traditional algorithms, which replicate based on strict assumptions about the distribution of nodes, *GLARAS* aims to approximate the underlying distribution by interacting with a very small subset of nodes and minimize the average access delay of replication accordingly. To ensure *GLARAS* performs at its best, we also propose a dynamic fully decentralized landmark-based locality aware name ID assignment namely *LANS* . This ensures that the nodes' distances in the overlay and the underlying network are consistent with each other. Our extensive experiments and analysis results demonstrate that compared to the best existing decentralized locality aware replication, *GLARAS* improves the average access delay of public and private replications by about **13%** and **17%**, respectively. Likewise, in comparison to the best existing decentralized locality aware name ID assignment, *LANS* improves the locality awareness of name IDs and the end-to-end latency of search queries in Skip Graph with the gains of about **19%** and **8%**, respectively. The average replication's access delay of a Skip Graph-based P2P storage system that employs *GLARAS* and *LANS* has an improvement gain of about **2.7** over the best state-of-the-art algorithms. Since Skip Graph is a DHT, any other DHT-based P2P storage service would benefit from our solution.

1. Introduction

A peer-to-peer (P2P) storage system consists of peers (i.e., nodes) where each peer corresponds to a device in the real world (e.g., smartphones, personal computers, servers). There are two roles in a P2P cloud storage: **data owner** and **data requester**. A data owner holds a set of data objects and intends to share them with a certain subset of nodes: its corresponding data requesters. In a P2P storage system, each node can be both a data owner and data requester of other data owners, simultaneously.

In a DHT-based P2P storage system [1–3], there exists a structured overlay topology where each node knows a small subset of nodes (i.e., neighbors) in the system, and keeps their addresses as (*ID*, *address*) pairs in a table named the lookup table of that node. The predefined topology, accompanied by the lookup tables of nodes, enables them to efficiently search for each other as well as each other's data objects in a fully decentralized manner.

Skip Graph [4] is a DHT-based routing infrastructure where each node has two identifiers: a name ID and a numerical ID. Name ID is a binary string and numerical ID is a non-negative

* Corresponding author.
E-mail addresses: yhassanzadeh13@ku.edu.tr (Y. Hassanzadeh-Nazarabadi), akupcu@ku.edu.tr (A. Küpçü), oozkasap@ku.edu.tr (Ö. Özkasap).

https://doi.org/10.1016/j.future.2018.02.007
0167-739X/

integer. Connectivity patterns of Skip Graph nodes are determined based on the common prefix length of their name IDs. Address of a node is efficiently retrievable in a fully decentralized manner by searching for either of its identifiers. The ability to perform efficient, fully decentralized and concurrent search operations in a scalable manner accompanied with the flexibility in the connectivity patterns of nodes based on the name IDs emerge Skip Graph as the routing infrastructure in many P2P applications including P2P storage systems [5–8]. Likewise, Skip Graph can be used as an alternative of other DHTs like Chord [9] in various P2P applications.

To reduce the query load on the data owner, provide data availability and reliability, ease of accessibility, fault tolerance, and maintainability, the data owner makes copies of its data objects on some other nodes of the system, which are called the data owner's corresponding **replicas** [10–15]. In **public replication**, every node is considered as a potential data requester. In **private replication**, only a subset of nodes are the data requesters of a data owner. While we do not discuss access control, it can be done via encryption and key management for private replication scenarios. After replication, each data requester queries the **closest replica** to itself, i.e., the replica with the minimum access delay. We consider the access delay of replication as the round-trip time (RTT) between the data requester and the replica. The performance of a P2P storage system in terms of query processing and response time is highly correlated with the average access delay of replication where for a specific data owner, the average is taken over all its data requester nodes.

The traditional decentralized replication algorithms applicable on a Skip Graph based P2P storage system are randomized replication [16–22], replicating on a subset of data owner's neighbors [17,23–25], and replicating on the search path between the data owner and some of its data requesters [26,27]. These replication algorithms aim at improving the average access delay of replication by making explicit rigid assumptions about the nodes' distribution in the overlay network. Randomized replication assumes a uniform distribution of data requesters in the overlay network, replication on neighbors and replication on path assume a high density of data requesters around the data owner or on search paths to the data owner, respectively. The very major disadvantage of all these traditional algorithms is that instead of trying to adapt with the distribution of data requesters, they mainly go with their explicit rigid assumptions. This brings a low average access delay for the parts of the system that follow their assumptions while enforces a considerably large average access delay on the rest of the system. Consequently, the overall average access delay of replication increases significantly, which degrades the system performance. For example, randomized replication improves the average access delay of replication in the parts of the system with the uniform distribution of data requesters, while it almost fails in the other parts.

Addressing the aforementioned problems, we propose a dynamic and fully decentralized locality aware replication algorithm that is called **Growing Locality Aware Replication Algorithm for Skip Graph (GLARAS)** to improve the average access delay of replication in P2P storage systems. *GLARAS* enables locality awareness by dividing the data requesters into subgroups and placing a unique replica for each subgroup such that the minimum average access delay between data requesters inside each subgroup and their assigned replica is achieved. *GLARAS* is dynamic in the sense that it can decide on the placement of replicas based on the current system state. Likewise, it is fully decentralized and can be run by any data owner locally. As the input, *GLARAS* needs only some system-wide public information. In the case of private replication, *GLARAS* also needs the name IDs of data requesters. This means, the data owner should know the nodes that will have private access to its replicated data objects, which is a plausible assumption. *GLARAS* aims at minimizing the average access delay in a churn-free Skip Graph, where nodes are assumed to be online all the time and have unlimited bandwidth and capacity. Availability, load balancing, and capacity restrictions of nodes are orthogonal issues that we plan to address in our future work. Furthermore, maintaining the consistency of replicas is beyond the scope of this paper, and can be addressed by, for example, [28]. *GLARAS* is applicable on top of any prefix-based structured P2P cloud storage that defines data owner and data requester roles. By prefix-based, we mean that the neighboring relations are based on the common prefix length of nodes' identifiers. For example, *GLARAS* is applicable on P2P cloud storages [29–31], which work on top of Kademlia [32] that is a prefix-based DHT.

GLARAS works on top of a landmark-based locality aware name ID assignment strategy. As name IDs indicate the neighboring information of Skip Graph's nodes, we define locality awareness of name IDs as follows: the longer common prefix of nodes' name IDs in the overlay network corresponds to the lower pairwise latency between them in the underlying network. Therefore, a Skip Graph that is built on locality aware name IDs would benefit from the improved end-to-end latency of search operations, which is defined as the total pairwise latency of consecutive nodes on the search path [33]. To improve the locality awareness of name IDs, we propose a dynamic and fully decentralized landmark-based locality aware name ID assignment algorithm that is called **Locality Aware Name ID assignment algorithm for Skip Graph (*LANS*)**. Landmarks are not nodes of Skip Graph; they are just placed to make the latency of nodes toward them measurable during the join procedure and they do not have any major computation or memory overhead. *LANS* improves the query processing and response time of the Skip Graph, and therefore any DHT-based P2P application can also benefit from such a locality aware Skip Graph.

Our contributions are as follows:

- We propose ***GLARAS*, a dynamic and fully decentralized locality aware replication algorithm** for Skip Graph.
- As an independent contribution, we propose *LANS*, **a dynamic and fully decentralized locality aware name ID assignment** for Skip Graph.
- We optimized the Skip Graph simulator, SkipSim [34], to support 4 times larger scale simulations (up to 4096 nodes), implemented the best existing DHT-based identifier assignment and decentralized replication algorithms on SkipSim, and compared them with *GLARAS* and *LANS*, respectively.
- The simulation results show that compared to our previously proposed LARAS algorithm [35], which acts as the best existing decentralized locality aware replication algorithm for Skip Graph, *GLARAS* is much faster, and improves the average access delay of public and private replications with gains of about **13%** and **17%**, respectively.
- Based on the simulation results, compared to our previously proposed DPAD algorithm [33], which acts as the best existing decentralized locality aware name ID assignment for Skip Graph, *LANS* improves the locality awareness of name IDs, and end-to-end latency of search queries in Skip Graph with gains of about **19%** and **8%**, respectively.
- The average access delay of replication for a Skip Graph that employs *GLARAS* and *LANS* is about **2.7** times better over LARAS and DPAD.

In the rest of this paper, we discuss the related works in Section 2. Our system model and the Skip Graph structure are presented in Section 3. We present our proposed *LANS* algorithm in Section 4. Our proposed *GLARAS* is presented in Section 5. Our simulation setup followed by the performance results are presented in Sections 6 and 7, respectively. We conclude in Section 8.

2. Related works

2.1. Locality-based identifier assignments

2.1.1. Dynamic identifier assignments

In the dynamic identifier assignment strategies, the identifier of a new node is assigned at the time it joins the system by any node of the system, or by some specific nodes act as coordinators. **LAND** [36,37] is a DHT-based identifier solution where each node of the system can assign a random identifier to a new node. In **LDHT** [38] the identifiers are binary strings of fixed length, and have prefix and body parts similar to *LANS* where the prefix part is the Autonomous System Number [39] of the node and the body part is generated randomly by taking the hash of the node's IP address. **Hierarchical** identifier assignment [40] considers a predefined division of the geographical space into regions, based on the expected number of nodes.

Toda et al. [41] propose a locality aware name ID assignment approach that assigns a randomized name ID to the new node that joins the system. The initial randomized name IDs may later be changed to provide better description of locality awareness. In their proposed approach by joining a new node $O(\log n)$ name IDs may be changed in expectation with high probability. Since name IDs define the connectivity, the nodes with updated name IDs are required to update their neighboring information, which enforces a noticeable communication overhead in large system scales.

2.1.2. Static identifier assignments

To assign nodes' identifiers, a static identifier assignment needs the locality information of all the nodes in the system. If another node joins the system, the static identifier assignment algorithm should be re-executed with the updated locality state of the system, which may change the identifier of some nodes. This makes the static identifier assignments inefficient for P2P applications. **Landmark Multi-Dimensional Scaling (LMDS)** [42–44] scales down the nodes' latency-based coordinate to a single integer value, which is not in the format of the name IDs, and needs extra refinements to be in the shape of name IDs. Likewise, to provide locality awareness **Geo-Peer** [45] generates a Delaunay triangle [46] among each three neighbors based on Voroni diagram [47]. Although Geo-Peer does not provide the locality awareness as we define for Skip Graph, it follows the same goal as *LANS*. As the result of applying Geo-Peer, each node is connected to its closest neighbors in a fully decentralized manner. In Geo-Peer neighbors of a node are determined based on the geographical location of the node, which results in sub-sets of nodes with more than $O(\log n)$ neighbors. This violates the architecture and efficiency of Skip Graph.

Table 1 represents a comparison between various DHT-based identifier assignment strategies. We call an identifier assignment *dynamic*, if it is able to assign the identifier of nodes at their arrival time. A *static* identifier assignment needs all the nodes to be available in the system to assign their name IDs. From the decentralization point of view, an identifier assignment is *full* if any node in the system can assign an identifier to another node. An identifier assignment is *hybrid* if only some special nodes in the system (e.g., coordinators) can execute the assignment algorithm. From the locality awareness point of view, we consider an identifier assignment as *full*, if it purely considers the locality information of nodes. We call an identifier assignment *hybrid* if it applies some kind of randomness in the assignment of identifiers. An identifier assignment is not locality aware if it does not consider the locality information of nodes.

2.2. Locality-based replication

2.2.1. Decentralized replication

In a decentralized locality-based replication algorithm, the data owner places the replicas in a fully decentralized manner, without the need to communicate with any special node as the coordinator. There exist several decentralized locality-based replication algorithms each considering a certain aspect of the system:

Randomized replication algorithms place replicas uniformly random [16], or by employing hash functions [17–22,49]. Randomized replication is suitable for P2P systems with uniformly random distribution of nodes in the underlying system where randomly choosing replicas results in a low expected value of average access delay.

In **replication on path** [27], the data owner places its replicas on the search paths from some of the data requesters to itself. LAR [26] replicates on the path between the data requester and the data owner in an adaptive manner based on the traffic load that each intermediate node on the search path tolerates. In general, replication on path suits the cases where the data requesters are distributed in disjoint dense regions. Replicating on the path that passes through all those dense regions results in a low average access delay of replication. However, in the case of public replication, or when the data requesters are scattered across the system, dedicating all the replication degree on a few paths results in high average access delay.

In **replication on neighbors** [17], a data owner places its replicas on a subset of its neighbors. Using AURA [24], each node frequently exchanges its state information such as available bandwidth and load with its overlay neighbors. A data owner then selects a subset of its neighbors with desirable properties as the replicas. A greedy algorithm to find the nodes with higher replication potential based on certain metrics is proposed in [25]. Whenever a data owner wants to replicate, it sends the objective function to its neighbors. Each neighbor evaluates the objective function based on its own properties and compares the evaluated value with its neighbors. If one node holds the best local value of the objective function among its overlay neighbors, it announces itself as a replication candidate to the data owner. Otherwise, replication function evaluation continues on the neighbors recursively. Rollerchain [23] divides the DHT nodes into groups based on their availability information and distributes the data objects among the groups. An assigned data object to a group is replicated on every node inside that group. In general, replication on neighbors helps most of the queries for the data owner's objects to be answered by its neighbors. Likewise, in non-locality aware P2P systems where neighbors of a node are uniformly distributed across the system, replication on neighbors results on placing replicas randomly across the system. However, in locality-aware P2P systems where nodes' neighbors are mostly placed close to the node, replication on neighbors results in the concentration of the majority of replicas, which uplifts the average access delay of nodes in the far away replica-free regions.

In addition to the P2P solutions, several replica placements approaches with similar locality concerns are addressed in distributed data grids [50]. A Quality-of-Service (QoS) aware replication algorithm for data grid is presented in [51], where a distributed dynamic programming approach to maximize the QoS for each data requester is proposed. The QoS metric in the presented solution is a function of the number of hops between the data requester and the closest replica. Although applying such an objective in our system reduces the traffic load and hence improves the QoS, it nevertheless does not necessarily reduce the average access delay of replication, since the number of hops between two nodes in our system does not necessarily represent their end-to-end latency. Periodic Optimizer [52] aims to improve the response

Table 1
Comparison of various methods of identifier assignment.

Method	Behavior	Decentralization	Locality awareness
LAND [36,37]	Dynamic	Full	No
LDHT [38]	Dynamic	Hybrid	Hybrid
Hierarchical assignment [40]	Dynamic	Hybrid	Hybrid
LMDS family [42,44,48]	Static	Hybrid	Full
LANS	**Dynamic**	**Full**	**Full**

time in distributed data grid by balancing the load of replicas that improves the average available bandwidth. As mentioned earlier, load balancing of replication is beyond the scope of this paper.

2.2.2. Centralized replication

The centralized replication algorithms are executed by some coordinator nodes, which gather the state information of all or some of the nodes in the system and replicate accordingly [11,53]. Some centralized replication algorithms are multi-objective, and aim to optimize the quality of service, average occupied bandwidth, access delay, and replication cost by a greedy strategy [54] or modeling the replication as a multi-objective optimization problem [25,55,56]. Centralized replication algorithms are subject to the single point of failure, and they enforce a noticeable communication overhead, which makes them inappropriate for P2P applications.

Table 2 presents a comparison of various locality-based replications. We call a replication algorithm *fully* decentralized if the data owner can place its replicas without the help of any special node (e.g., coordinator). From the locality awareness point of view a replication algorithm is *full*, if it merely considers the locality information of nodes without randomness. A replication algorithm is *hybrid* if it considers some sort of randomness in the replica's placement. A replication algorithm is *not locality aware* if it ignores the node's locality information in replica's placement. In terms of behavior, a replication algorithm is *dynamic* if the data owner can place its replicas at any arbitrary time with its local view of the system. A replication algorithm has *static* behavior, if the data owner needs to wait and obtain the global view of nodes to place its replicas.

3. Preliminaries

3.1. System model

System capacity: In our view of a Skip Graph-based P2P storage system, the system capacity is a constant that is denoted by n and defined as the expected maximum number of nodes in the system. In this paper, without loss of generality, we assume n is a power of 2.

Skip graph nodes: Each node in the Skip Graph overlay represents a single peer in the real world. In our system model, the Skip Graph is churn-free [57] i.e., there is no arrival and departure of nodes, and all the nodes are considered to be available all the time. We assume that the Skip Graph nodes can communicate directly with each other in the underlying network once they know each other's addresses. To efficiently find each other's addresses, nodes use the Skip Graph overlay to perform a search for numerical ID [4] or a search for name ID [35].

Landmarks: In our system model, we presume the existence of some super nodes that are called landmarks. Landmarks are not considered as nodes of the Skip Graph. The purpose of landmarks' placement is to make the latency of nodes toward landmarks measurable during the join procedure of nodes. Landmarks are not supposed to carry major computation or memory overhead. In this paper, we denote the set of all landmarks by L, and the size of L that corresponds to the number of landmarks in the system by $|L|$. We assume $|L| = \lceil \log n \rceil$, however in general *LANS* and *GLARAS* are functional with any $|L|$ of size $O(\log n)$. We tag landmarks in an arbitrary fixed order from 1 to $|L|$, and place them according to the expected density of nodes in the underlying system. We consider a reverse correlation between the nodes' density and their pairwise latency in the underlying network, which is based on RTT. The high density of nodes in an area corresponds to the high probability of landmark existence there. The higher density of a group of nodes implies their lower average pairwise latency in the underlying network. The nodes' density in the underlying network does not necessarily correlate with their geographical locations.

Replication: We consider replication as the process of determining the place of the replicas. After the placement of replicas is done, all data objects of the data owner are copied to each of the replicas. We consider the data requesters as the authorized set of nodes having access to these data objects. Access control is a separate problem, which can be addressed for example by encrypting the data objects of the data owner and distributing the keys among the data requesters [58]. In our view of a P2P storage system, every peer including the data requesters and even the data owner itself is a potential replica candidate.

3.2. Skip graph

3.2.1. Structure

Skip Graph [4] is a decentralized routing infrastructure that is considered as the distributed version of Skip List [59–61]. A Skip Graph with n nodes has $O(\log n)$ levels, which are numbered starting from zero. Fig. 1 shows a Skip Graph with 10 nodes and 4 levels where levels numbered from 0 to 3. Each Skip Graph node has a unique numerical ID, a unique name ID, and exactly one element inside each level. In Fig. 1, the nodes' elements are represented by squares with numerical IDs enclosed and name IDs located at the bottom. Numerical ID is a non-negative integer and name ID is a binary string of length $O(\log n)$ bits. In our Skip Graph based P2P storage system, we consider the numerical IDs as the hashed value of the corresponding peer's IP address.

In level zero, all nodes are sorted based on their numerical IDs in a doubly linked list. In upper levels, nodes are split into multiple lists based on their name IDs. In general, in level i, there are 2^i doubly linked lists, and nodes that are located in the same list have at least **i-bit common prefix in their name IDs**. For instance, in level 2 of Fig. 1, there are 4 doubly linked lists, and nodes that are located in each list have at least 2-bit common prefix in their name IDs. For example, nodes with numerical IDs of 11, 41, 43, and 71, which are placed in the same linked list, have at least 2-bit common prefix in their name IDs. On the other hand, the only node with the name ID prefix of 11 is the node with numerical ID of 93, which is placed solely on a separate list of size one on level 2.

In a Skip Graph with the capacity of n nodes and $O(\log n)$ levels, each node has at most two neighbors in each level and $O(\log n)$ neighbors in total. In Fig. 1, left and right neighbors of the node with numerical ID 43 in level zero are nodes with numerical IDs of 41 and 67, respectively. Likewise, in both levels one and two, the left and right neighbors of the node with numerical ID of 43 are nodes with numerical IDs of 41 and 71, respectively. Finally, in level three, the node does not have any right neighbor and has a left neighbor with numerical ID of 11.

Table 2
Comparison of various methods of replication strategies.

Strategy	Decentralization	Locality awareness	Behavior
Randomized [16–22]	Full	No	Dynamic
On Path [26,27]	Full	No	Dynamic
On Neighbors [17,23–25]	Full	Hybrid	Dynamic
Objective-based [54–56]	No	Full	Static
GLARAS	**Full**	**Full**	**Dynamic**

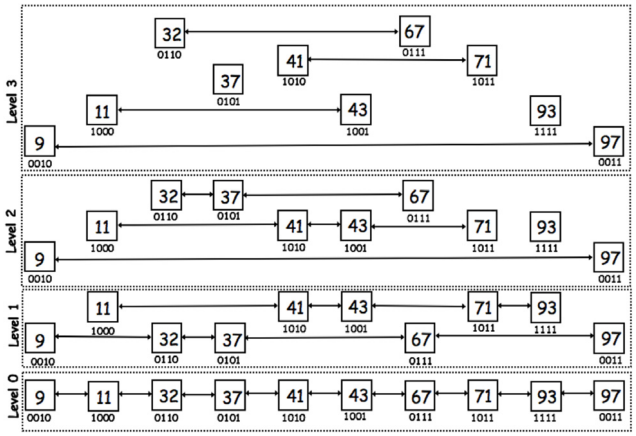

Fig. 1. An example Skip Graph with 10 nodes and 4 levels. Each node has an element in each level that is represented by a square. The integer value inside each element is the corresponding node's numerical ID. The binary string below each element is its corresponding node's name ID.

A Skip Graph can be employed as a DHT, by mapping each data object to a Skip Graph node using a hash function [4]. A lookup operation for a data item is done by a search for numerical ID with the hashed value of that data object as the target numerical ID of the search.

3.2.2. Search for numerical ID

Every node in the Skip Graph can initiate a search for a numerical ID [4]. The node that initiates the search is called the **search initiator**, and the numerical ID that the search initiator searches for is called the **search target**. The search for numerical ID is done in a fully decentralized manner. As the result of the search, the address of the node with the greatest numerical ID less than or equal to the search target is returned to the search initiator. For example, in Fig. 1, as the result of a search for numerical ID of 11, the address of the node with numerical ID of 11 is returned to search initiator. However, a search for numerical ID of 33, which does not exist in the Skip Graph, results in the address of the node with numerical ID of 32 to be returned back to the initiator. 32 is the greatest available numerical ID in the Skip Graph that is less than the search target. Having n nodes in the Skip Graph, a search for numerical ID is done by traversing $O(\log n)$ nodes in expectation with high probability. Detailed descriptions and examples of the search for numerical ID are available in [33].

3.2.3. Search for name ID

In our previous work, we proposed a search for name ID algorithm [35] done in a fully decentralized manner by traversing $O(\log n)$ nodes in expectation with high probability. As the result of a search for name ID, the address of the node that owns the most similar name ID to the search target is returned to the search initiator. By similarity, we mean the name ID that has the longest common prefix with the search target. For instance, a search for name ID of 1010 initiated by any node results in the address of the node with name ID of 1010 (and numerical ID of 41) to be returned back to the search initiator. However, as the result of a search for name ID of 0100, which does not exist in the Skip Graph, one of nodes' addresses with name IDs of 0110, 0101, or 0111 that have the longest common prefix of 2 bits with the search target is returned back to search initiator. The exact search result that is returned depends on the search path starting from the initiator. Our proposed *GLARAS* replication approach employs search for name ID to map a replication candidate from its workspace to the original P2P storage system. In a locality aware Skip Graph, all possible search results for a name ID are the most suitable nodes to the locality information that is described by the target name ID. Hence from the locality point of view, all of the possible search results are posed similarly in the underlying network to the search target name ID, there is no coarse-grained discrimination among them. Detailed descriptions and examples of the search for name ID are available in [35].

4. Locality Aware Name ID assignment algorithm for Skip Graph (*LANS*)

4.1. Algorithm overview

In our proposed Locality Aware Name ID assignment algorithm for Skip Graph (*LANS*), every node and landmark is assigned a latency-based coordinate as a tuple of size $|L|$ where L corresponds to the set of all landmarks in the system. The ith element of a node's (or a landmark's) coordinate corresponds to its round trip time toward the ith landmark of the system. The latency-based coordinate system is hence determined by the landmarks and is not necessarily of a specific shape e.g., rectangular. Every node or landmark is able to efficiently compute its own latency-based coordinate. Before any node arrives at the system, *LANS* assigns a binary string of variable length to each landmark based on its latency-based coordinate, which is called **dynamic prefix**. Using *LANS*, a node's name ID is composed of two parts: **prefix and body**. The concatenation of prefix and body parts constitutes the name ID of the node. The prefix part of a node's name ID is the dynamic prefix of the closest landmark to it. By the closest landmark, we mean the one with the minimum latency toward the node. The body part of a node's name ID is computed based on its latency-based coordinate.

Once a *new node* joins the system, it contacts each of the landmarks and computes its own latency-based coordinate. Next, the *new node* invokes *LANS* and computes its name ID by using its latency-based coordinate. Every *new node* should know at least one of the Skip Graph's nodes, which is called the introducer of *new node*. *new node* contacts its introducer for checking the availability of its generated name ID. The introducer returns back the most similar name ID (the one with the longest common prefix with the generated name ID) that is not owned by any node in the Skip Graph and hence is free to use by *new node*. Without loss of generality, landmarks can keep the address of a few nodes of Skip Graph. The *new node* can request the list from one of the landmarks, pick one node from the list and contact it as its introducer.

4.2. Algorithm description

4.2.1. Initialization

Based on the placement of landmarks, *LANS* considers a latency-based coordinate for the node that executes it, as well as each of the landmarks. The latency-based coordinate of each node or landmark is denoted by a tuple of size $|L|$ where L is the set of landmarks. Eq. (1) illustrates the latency-based coordinate of node j where $coor_{j,i}$ is the pairwise latency between node j and landmark i in the system. The latency-based coordinate of landmarks is also computed in the same way, by measuring their latency toward each of the other landmarks.

$$coor_j = \left[coor_{j,1} \quad coor_{j,2} \quad \dots \quad coor_{j,|L|}\right]. \tag{1}$$

The initialization phase of *LANS* is done only once at the birth time of the system, and before any node arrives. Based on the landmarks' placement, the system is divided into **regions**. Each region is identified by a landmark and the set of nodes, which have that landmark as their closest landmark. As a result of the initialization phase of *LANS*, each landmark is assigned a dynamic prefix based on the expected number of nodes in its region. The assignment of dynamic prefixes provides a hierarchical distinction among the regions. Since the initialization phase is a one-time operation and the landmarks are assumed to be churn free, the dynamic prefixes of landmarks never change. Dynamic prefixes of landmarks hold the prefix property i.e., no dynamic prefix is a prefix of another one. To generate the dynamic prefixes, we apply a 2-mean clustering approach [62] recursively on the landmarks set L based on the latency-based coordinate of landmarks. The first recursion of the 2-mean clustering bisects the landmarks set into two clusters; one with the dynamic prefix of 0, and the other one with 1. The next recursion on each of these clusters results in at most 4 clusters with the dynamic prefixes of 00, 01, 10, and 11. This recursive clustering continues until each cluster encloses exactly one landmark. At that point, each landmark inherits the dynamic prefix of its cluster. Based on the nature of 2-mean clustering, the longer the prefix of a landmark is, the more nearby landmarks it has in the latency-based coordinate system.

4.2.2. Name ID generation

When node j arrives at the system, it is supposed to know the address of its introducer denoted by α_j. Node j then invokes *LANS* shown by Algorithm 4.1 . As the inputs, *LANS* receives the address of node j, its introducer α_j, and the set of all landmarks denoted by L. The output of *LANS* is the name ID of node j, which is denoted by $nameID_j$. On behalf of node j, *LANS* contacts each of the landmarks by executing the *contactLandmark* function. On receiving address of landmark i, *contactLandmark* returns the latency between node j and landmark i that is denoted by $coor_{j,i}$, the latency-based coordinate of landmark i that is denoted by $coor_i$, and the dynamic prefix of landmark i that is denoted by $prefix_i$ (Algorithm 4.1 Lines 1–2). Once all the landmarks are contacted, as shown by Eq. (1), *LANS* constructs node j's latency-based coordinate denoted by $coor_j$ (Algorithm 4.1 Line 3).

Having the latency-based coordinate available, *LANS* finds the closest landmark to node j as the one with the minimum corresponding latency. The closest landmark to node j is denoted by *closestLand* (Algorithm 4.1, Lines 4). To generate the body part of node j's name ID, *LANS* computes $closestLandDir_i$, which is the normalized direction vector between *closestLand* and each landmark i in the system where $i \neq closestLand$. Likewise, *LANS* computes the $nodeDir_i$, which is the normalized direction vector between node j and each landmark i in the system where $i \neq closestLand$. The direction vector is computed based on the latency-based coordinate of nodes and landmarks. For each landmark i, after computing $closestLandDir_i$ and $nodeDir_i$, *LANS* gauges the distance between these two vectors, which is represented by $diff_i$. The landmark with the minimum $diff_i$ value is selected by *LANS* as the best matched landmark, and is denoted by *bestMatchedLand* (Algorithm 4.1, Lines 5–9).

Algorithm 4.1: LANS

Input: Node j, introducer α_j, set of landmarks L
Output: String $nameID_j$

```
// contacting landmarks to receive the latency toward them, their coordinate, and prefix
1  for each landmark i ∈ L do
2      (coor_{j,i}, coor_i, prefix_i) = contactLandmark(i);
// constructing the latency-based coordinate of node j as an array of the obtained coor_j
3  construct coor_j;
// finding the closest landmark to node j
4  closestLand = argmin_i{coor_{j,i}};
// finding the best-matched landmark to node j
5  for each landmark i ∈ L except closestLand do
6      closestLandDir_i = (coor_closestLand − coor_i) / ||coor_closestLand − coor_i||;
7      nodeDir_i = (coor_j − coor_i) / ||coor_j − coor_i||;
8      diff_i = ||closestLandDir_i − nodeDir_i||;
9  bestMatchedLand = argmin_i{diff_i};
// generating the name ID for node j
10 body = prefix_bestMatchedLand;
11 body = padBody(body, coor_{j,closestLand});
12 nameID_j = prefix_closestLand + body;
// checking for the availability of the generated name ID by contacting node j's introducer
13 nameID_j = α_j.checkAvailability(nameID_j);
```

Among all the landmarks in the system, the best-matched landmark to node j is the one that provides the best description of node j's latency-based positioning inside the region it is inscribed. *LANS* initializes the body part of node j's name ID with the dynamic prefix of the best-matched landmark to node j (Algorithm 4.1, Line 10). To complete the rest of node j's name ID, *LANS* invokes *padBody* routine, which extends the body part to $\lceil \log n \rceil$ bits where n is the system capacity. *padBody* fulfills the extension by adding the $\lceil \log n \rceil - |body|$ leftmost bits of $coor_{j,closestLand}$ binary representation to the body part where $coor_{j,closestLand}$ corresponds to the latency between node j and its closest landmark (Algorithm 4.1, Line 11). After the body part is constructed, *LANS* generates name ID $nameID_j$ by concatenating the $prefix_{closestLand}$ and *body* parts (Algorithm 4.1, Lines 12).

To prevent the name IDs duplication, *LANS* checks the availability of the generated name ID for node j by contacting α_j. On receiving the check for availability request, α_j returns the most similar name ID that does not belong to any node in the Skip Graph (Algorithm 4.1, Line 13). *checkAvailability* conducts a search for $nameID_j$ [35], which returns back the address of the node that has the longest common prefix with $nameID_j$ in Skip Graph. If result's name ID is not identical to $nameID_j$, it conveys that $nameID_j$ is not held by any other node, and *checkAvailability* returns it. Otherwise, *checkAvailability* decomposes $nameID_j$ into prefix and body parts. *checkAvailability* then concurrently checks the availability of predecessors and successors of $nameID_j$, and returns the first free to use predecessor or successor. Predecessor and successor of a name ID have the same prefix part as the name ID, with the integer value of body part one less and one more than the name ID body part's integer value, respectively. The availability of predecessor and successors is checked by conducting a search for

their name IDs. It is important to note that since *LANS* considers the body part of name IDs of size $\lceil \log n \rceil$ bits, there are $2^{\lceil \log n \rceil} = n$ possible predecessors and successors for a certain name ID on the aggregate. Since n is considered as the expected maximum number of nodes in the system, for a generated $nameID_j$, there always exists one free to use predecessor or successor. Based on our simulation results, for *LANS* to generate a name ID the average number of searches that are conducted by $checkAvailability$ is 4.18 where the average is taken over 100 randomly generated systems each with 4096 nodes.

4.3. Comparison with our previous work

In our previous work DPAD [33] the dynamic prefix of landmarks comes from the Huffman coding of their latency toward one of the selected landmarks. Ignoring the pairwise delays between landmarks disadvantages the locality awareness of DPAD's dynamic prefixes in cases where landmarks have similar latency toward the selected one, but different latency patterns toward the rest. Likewise, in DPAD the latency toward the ith landmark determines the ith bit of the name ID's body part, which makes the locality awareness of name IDs forcefully dependent to the numbering of landmarks. To preserve the locality awareness in larger system scales, DPAD requires the optimal numbering of landmarks.

5. Growing Locality Aware Replication Algorithm for Skip Graph (GLARAS)

5.1. Algorithm overview

GLARAS works on top of a Skip Graph that is furnished with a landmark-based locality aware name ID (e.g., *LANS*). By executing our dynamic and fully decentralized *Growing Locality Aware Replication Algorithm for Skip Graph (GLARAS)*, a data owner places its replicas in a way that the *close to minimum* average access delay of replication is achieved. We define the average access delay as the average of pairwise latency between each data requester and its closest replica. The average is taken over all data requesters. The exact minimum average access delay is achievable by collecting all pairwise delays among the nodes of the system, which enforces heavy traffic load and is not feasible in large scale systems. On the other hand, the data owner who runs *GLARAS* needs to know only the landmarks' dynamic prefixes and replication degree i.e., the number of replicas. In the private replication case, the data owner also needs to know the name IDs of its data requesters. We assume the set of landmarks' dynamic prefixes as a publicly-known constant. Replication degree is either a system-wide constant or is determined based on parameters such as data owner's preferences and its membership tier.

To achieve the minimum average access delay of replication, *GLARAS* applies two levels of replica distribution known as system-wide and region-wide distributions. In the **system-wide distribution (SWD)**, *GLARAS* distributes the total replication degree to the regions of the system. The distribution is done based on the approximated number of data requesters inside each region, pairwise latencies among the landmarks, and the number of nodes in the neighborhood of each region. As the result, each region receives a non-negative sub-replication degree. After the system-wide distribution is done, *GLARAS* executes the region-wide distribution of replicas for each region. In **region-wide distribution (RWD)**, *GLARAS* models the locality aware replication as an integer linear programming (ILP) that uses the assigned sub-replication degree of the region. As the result of RWD, the placement of replicas in each region is determined. Once RWD is executed for all the regions, the placement of replicas across the system is accomplished.

RWD's ILP model considers all possible name IDs inside a region as possible replica candidates. This results in the quadratic dependency of problem size on the expected maximum number of nodes in the system i.e., the system capacity. By the problem size, we mean the number of decision variables as well as the number of constraints of ILP. To tackle this situation and work efficiently, *GLARAS* first maps the original system to a significantly smaller size system, which is called **virtual system**. *GLARAS* then models and solves ILP for each region of the virtual system, and maps the determined replicas from the virtual system to the original one. Once the mapping is done, the data owner receives its replicas' name IDs and responses to its data requesters queries with the list of replicas' name IDs. Each data requester node finds its closest replica from the list and queries that replica instead of contacting the data owner directly. Since name IDs are supposed to be locality aware, for a data requester the closest replica is the one with the longest name ID's common prefix. From the *GLARAS* point of view, the data owner can be selected as a replica itself only if RWD decides so. Otherwise, the data owner does not have to hold its data objects after the replication is done.

A single run of *GLARAS* determines the set of replicas for the data owner who executes it. If a data owner wants to share different sets of data objects with different sets of data requesters, it needs to execute *GLARAS* once for each set of data requesters separately.

5.2. Algorithm description

As shown by Algorithm 5.1, inputs of *GLARAS* are the set all landmarks denoted by L, the replication degree denoted by R, the set of data requesters' name IDs denoted by K^O, virtual system size denoted by $size_{vs}$, and the maximum size of an ILP model that the data owner can solve efficiently based on its computational power denoted by $maxSize$. In the case of public replication, K^O is an empty set, and hence *GLARAS* considers the whole possible name IDs as tentative data requesters. In private replication case, however, K^O represents the name ID set of data requesters. The output of *GLARAS* is the set of replicas name IDs, which is denoted by $repSet$.

5.2.1. System-Wide distribution (SWD) (Alg-5.1, Line 1)

In the first step, *GLARAS* invokes SWD as a sub-routine. On receiving L, R, and K^O as inputs, SWD distributes the replication degree among the regions and returns r, which is a vector of size $|L|$ where r_i denotes the sub-replication degree associated with the region i of the system (Algorithm 5.1, Line 1). r_i represents the number of replicas that should be placed in region i of the system. The sub-replication degrees are non-negative integers, which sums up to the original replication degree R of data owner.

We studied the optimal patterns of system-wide distribution by examining all possible combinations of replica distribution (in small scale) in 100 randomly generated topologies of 4096 nodes. We realized that the optimal system-wide distribution follows a cyclic behavior on a permutation of regions (i.e., landmarks). Once the permutation is fixed, starting from the first region, the corresponding sub-replication degree of regions in the permutation is increased by one in a cyclic manner, until the original replication degree is completely distributed among the regions. Following this conclusion, SWD finds the permutation of regions that the cyclic distribution results in close to the optimal distribution of replicas among regions. If the original replication degree R of the data owner is less than the number of regions, then some regions toward the end of the permutation receive 0 sub-replication degree. Otherwise, once each region is assigned a replica, SWD restarts assigning the remaining ones from the beginning of the permutation.

Algorithm 5.1: *GLARAS*

Input: set of landmarks L, replication degree R, set of data requesters K^O, virtual system size $size_{vs}$, maximum size of ILP model *maxSize*
Output: replicas set *repSet*

// distributing the replication degree among the regions, and obtaining the array r of sub-replication degrees
1 r = SWD(L, R, K^O);
// mapping the original system to the virtual system, and obtaining the set of data requesters as well as name IDs in the virtual system
2 (K^v, I^v) = toVirtual(n, L, K^O, $size_{vs}$);
// placing the replicas in each region of the virtual system based on its assigned sub-replication degree
3 **for** *each region* $q \in$ *virtual system* **do**
4 $bestScore = 0$;
5 $bestRepSet = \varnothing$;
6 **do**
// solving the locality aware replication ILP and obtaining the set of replicas in the region q of the virtual system
7 $vRepSet$ = RWD(r_q, K_q^v, I_q^v);
// mapping replicas from region q of the virtual system to the nodes in the original system
8 $(accuracy, oRepSet) = toOriginal(vRepSet)$;
9 **if** $accuracy \times size_{vs} > bestAccuracy$ **then**
10 $bestRepSet = oRepSet$;
11 $bestScore = accuracy \times size_{vs}$;
// removing the bad replication candidates based on the obtained accuracy
12 $(I_q^v, status) = remBadCan(oRepSet, vRepSet, size_{vs},$
13 $maxSize)$;
14 **if** $status == growth$ **then**
// growing the virtual system size of region q and extending its available name IDs accordingly
15 $size_{vs} = size_{vs} \times 2$;
16 $I_q^v = expandNameIDs(I_q^v)$;
17 **while** $status \neq terminate$;
18 $repSet = repSet \cup bestRepSet$;

To generate the close to optimal permutation of regions, SWD places the landmark with the minimum total latency regarding the rest of landmarks as the first in the permutation. The pairwise latency of landmarks is obtained by contacting each landmark and asking for its latency-based coordinate. The rest of the permutation is based on the positioning of regions with respect to each other, their number of data requesters, and their latency with the landmarks of regions that have been already placed inside the permutation. For each landmark j, which has not been yet placed in the permutation, SWD computes a score denoted by $score_j$. As shown by Eq. (2), $score_j$ is the summation of $dataReq_j$, $minLatency_j$, and $closestCov_j$ weighted by w_1, w_2, and w_3, respectively.

$$score_j = (w_1 \times dataReq_j) + (w_2 \times minLatency_j) + (w_3 \times closestCov_j). \quad (2)$$

Eq. (2) is described in detail as follows.

Number of data requesters: $dataReq_j$ corresponds to the number of data requesters in region j, which is normalized with the total number of data requesters $|K^O|$. If the replication is private, the number of data requesters inside region j is efficiently computable by counting the name IDs inside K^O that start with the dynamic prefix of region j. Since the dynamic prefixes of regions are supposed to hold the prefix property, the corresponding region of each data requester is uniquely identified. In the case of public replication, every node inside a region is a tentative data requester. In this case, number of data requesters inside each region is equal to the number of nodes that region contains. However, since obtaining the exact size of each region charges a noticeable communication cost, and is also timely inefficient, SWD makes an approximation of the number of data requesters based on the dynamic prefix length of landmarks. In a landmark-based locality aware name ID assignment (e.g., *LANS*) longer dynamic prefix of landmarks is a sign of more populated corresponding regions for them. Therefore in public replication case $dataReq_j$ corresponds to the dynamic prefix length of landmark j, which is normalized with the total dynamic prefix lengths of all landmarks in the system.

Minimum latency to the selected regions: $minLatency_j$ denotes ratio of the minimum latency between landmark j and the already selected landmarks in permutation, over the maximum pairwise latency of landmarks.

Number of nearby regions: If the replication is public, then $closestCov_j$ denotes the ratio of number of regions that have region j as their closest region, over the total number of regions in the system i.e., $|L|$. In the case of private replication, $closestCov_j$ denotes the total number of data requesters inside the regions that have region j as their closest region, which is normalized by the total number of data requesters $|K^O|$.

In Eq. (2), w_i corresponds to normalized hard-coded weights. Benefiting from Eq. (2), SWD iteratively assigns a score to each of the landmarks that have not been placed in the permutation, picks the landmark with maximum score, places it into the next empty place of permutation, and updates the score for the rest of not yet selected landmarks based on their possibly changed *minLatency* values.

5.2.2. Mapping to the virtual system (Alg-5.1, Line 2)

In large-scale systems, during the determination of the sub-replication degree by SWD, there exists a large number of replication candidates inside each region. The ILP size of *GLARAS* 's region-wide optimization presented in Section 5.2.3 has a quadratic dependency on the expected number of nodes inside a region. Processing a large number of nodes and deciding where to replicate among them is timely inefficient and costly. To improve the running time, *GLARAS* constructs a significantly smaller size model of the original system, which is called the virtual system. Original and virtual system share the same set of landmarks. Hence, regions and dynamic prefix of landmarks that correspond to each region are identical in both systems. The only difference between the original and virtual system is the body size of their name IDs, which for the virtual system is shorter than the original one. The intuition behind the mapping is to stratify all possible name IDs inside each region of the original system into cliques based on their similarity and map each clique to the most similar name ID in that region of the virtual system. By similarity between a clique in the original system and a name ID in the virtual system, we mean the common prefix length between the common prefix of nodes inside the clique and the mapped name ID. For example, considering the name ID size of original system as 4 bits and the virtual system as 2 bits, the name IDs 1100, 1101, 1110, and 1111 in a region, which all have the prefix of 11 in common, are mapped to the name ID of 11 in the same region of the virtual system. In addition to improving the running time, working on the virtual system lets the ILP model of *GLARAS* to be solved efficiently without the need for the relaxation.

GLARAS does the mapping by invoking the *toVirtual* function, which on receiving the system capacity n, landmarks set L, set of data requesters' name IDs K^O, and virtual system size $size_{vs}$, creates a virtual system with the same set of landmarks but the lower capacity of $size_{vs}$. It is worth to mention that $size_{vs}$ is a local parameter of the data owner, which is adjusted by the data owner based on its computational power. Output of *toVirtual* is the set of data requesters in the virtual system, which is denoted by $K^v = \{K_1^v, K_2^v, \ldots, K_{|L|}^v\}$ where K_q^v is the set of data requester's name IDs in region q of the virtual system, as well as the set of all possible name IDs in each region of the virtual system, which is denoted by $I^v = \{I_1^v, I_2^v, \ldots, I_{|L|}^v\}$ where I_q^v is the set of all possible name IDs in the region q of the virtual system (Algorithm 5.1, Line 2).

If the replication is public, for each region q of the virtual system, $K_q^v = I_q^v$, as each possible name ID is a possible data requester. In the private replication case, however, *toVirtual* function maps each data requester from K^O to a name ID in the virtual system by removing rightmost bits of the input name ID's body part iteratively until the size of body part becomes identical to the size of name ID's body part of the virtual system. With the capacity of n and $size_{vs}$ nodes, sizes of the name ID's body parts of the original and virtual systems are $\lceil \log n \rceil$ and $\lceil \log size_{vs} \rceil$, respectively. Hence, to map a data requester node's name ID from the original system to the virtual system, *toVirtual* function removes its rightmost $\lceil \log n \rceil - \lceil \log size_{vs} \rceil$ bits.

5.2.3. Region-Wide distribution (RWD) (Alg-5.1, Line 7)

After mapping to the virtual system is done, *GLARAS* invokes RWD function on each region of the virtual system. Inputs of RWD are the sub-replication degree of region q of the virtual system r_q, set of region q's data requesters' name IDs K_q^v, and set of all possible name IDs inside the region q of virtual system I_q^v. RWD aims to achieve the minimum average access delay of replication by properly placing the replicas inside a region (Algorithm 5.1, Line 7). The number of replicas for each region corresponds to the sub-replication degree of that region, which is computed by SWD.

The placement of replicas in RWD is done based on ILP, which models the locality aware replication. Eqs. (3)–(8) show the RWD's ILP model for region q of the virtual system. The only two decision variables of the ILP model are X and Y. The X is a $size_{vs} \times |K_q^v|$ binary matrix where $X_{i,j} = 1$ if and only if node i is assigned as the corresponding replica for node j in the virtual system, otherwise $X_{i,j} = 0$. Likewise, Y vector has size of $size_{vs}$ where $Y_i = 1$ if and only if node i is selected as a replica, otherwise $Y_i = 0$. In the RWD's ILP model C is a $size_{vs} \times |K_q^v|$ table where $C_{i,j}$ denotes the length of common prefix in the name IDs of nodes i and j. In RWD's ILP model, body part of node i's name ID in the virtual system is generated by the binary representation of i in $\lceil \log size_{vs} \rceil$ bits. By employing a locality aware name ID assignment strategy (e.g., *LANS*), $C_{i,j}$ reflects the pairwise latency between nodes i and j. Finally, r_q is the sub-replication degree of region q, which is obtained from SWD.

$$\min \sum_{i=1}^{size_{vs}} \sum_{j=1}^{|K_q^v|} C_{i,j} X_{i,j}, \quad \text{s.t.} \tag{3}$$

$$\forall i \in I_q^v, j \in K_q^v \quad Y_i \geq X_{i,j} \tag{4}$$

$$\forall j \in K_q^v \quad \sum_{i=1}^{size_{vs}} X_{i,j} = 1 \tag{5}$$

$$\forall i \in I_q^v \quad \sum_{j=1}^{|K_q^v|} X_{i,j} \geq Y_i \tag{6}$$

$$\sum_{i=1}^{size_{vs}} Y_i = r_q \tag{7}$$

$$\forall i \in I_q^v, j \in K_q^v \quad Y_i \in \{0, 1\}, X_{i,j} \in \{0, 1\}. \tag{8}$$

Eq. (3) shows the objective function of the ILP model for region q of the virtual system, which minimizes the average access delay of replication in the region q. We define the average access delay of replication as the average access delay between data requester nodes and their closest replicas. When node i is assigned as the corresponding replica of the data requester node j (i.e., $X_{i,j} = 1$), it contributes to the total access delay of region q by an approximation of $C_{i,j}$. The inner summation of Eq. (3) represents the total latency between a single replica and its corresponding set of data requesters. The outer summation represents the total latency between all selected replicas and their corresponding data requesters in region q of virtual system. RWD minimizes the total latency between each node and its closest replica, which results in minimizing the average access delay of replication.

Eq. (4) denotes the assignment's constraint, which conveys that if node i is assigned to the data requester j as its corresponding replica (i.e., $X_{i,j} = 1$), then node i should be designated as a replica.

Eq. (5) represents the data requester's constraint, which means that a data requester node should be exactly assigned to one replica. In column j of binary matrix X, summing up the $X_{i,j}$ values over all is results in the number of replicas that data requester j benefits from, which should exactly be equal to 1.

Eq. (6) describes the replica's constraint, which states that if node i is designated as a replica, it should be assigned to at least one data requester node as its corresponding replica. In binary matrix X, summing up the $X_{i,j}$ values over all js results in the number of data requesters that are assigned to the node i as their corresponding replica. If node i is a replica, then $Y_i = 1$. Eq. (6) tells that if node i is a replica, the number of data requester nodes which are assigned to it should be at least one.

Eq. (7) shows the sub-replication degree constraint for region q. In region q of the virtual system, $Y_i = 1$ if and only if node i is deputized as a replica. The number of 1s on the Y vector hence corresponds to the number of replicas in the region q, which should be exactly equal to the sub-replication degree of that region.

Eq. (8) denotes the allowed values for decision variables. The only allowed values for Y_i and $X_{i,j}$ are either 0 or 1. $Y_i = 1$ if and only if node i is selected as a replica, otherwise $Y_i = 0$. Likewise, $X_{i,j} = 1$ if and only if node i is assigned as the corresponding replica of the data requester j, otherwise $X_{i,j} = 0$.

After RWD solves the ILP model for region q of the virtual system, it returns the replica set for that region denoted by v*RepSet* (Algorithm 5.1, Line 7). The replica set is the set of all nodes i with $Y_i = 1$. Each $Y_i = 1$ corresponds to a replica in the region q of the virtual system. The name ID of node i in the region q of the virtual system is determined as the concatenation of dynamic prefix of the landmark q followed by the binary representation of i in $\lceil \log size_{vs} \rceil$ bits.

5.2.4. Accuracy-based iterations (Algorithm 5.1, Lines 8–17)

Upon receiving v*RepSet* from RWD, *GLARAS* invokes *toOriginal* function on it. *toOriginal* function returns the most similar name ID in the original system to each name ID in v*RepSet*, as well as the accuracy of this mapping, which are denoted by *oRepSet* and *accuracy*, respectively (Algorithm 5.1, Line 8). To construct the *oRepSet*, *toOriginal* conducts a search for name ID of each node i from v*RepSet*. The searches are conducted in the original system. The search result for node i's name ID is node i' that has the most similar name ID in the original system to the name ID of node i. Likewise, *toOriginal* function computes the accuracy of mapping node i from virtual system to node i' in original system as

$\frac{commonPrefix(i,i')}{|nameID_i|}$ where $commonPrefix(i, i')$ corresponds to the length of common prefix between nodes i and i' name IDs. The name ID size of the virtual system is shorter than the original system. Therefore, the maximum similarity between the name IDs of nodes i and i' is equal to the length of node i's name ID. Eq. (9) shows the mapping accuracy of *vRepSet* to *oRepSet*, which is denoted by *accuracy* and defined as the minimum accuracy of nodes inside the *vRepSet*.

$$accuracy = \min_{i \in vRepSet} \{\frac{commonPrefix(i, i')}{|nameID_i|}\}. \quad (9)$$

GLARAS considers a score of $accuracy \times size_{vs}$ for mapping *vRepSet* to *oRepSet*. If the mapping score is the greatest score that is met, *oRepSet* is kept as the best set of replication candidates for region q of the virtual system (Algorithm 5.1, Lines 9–11). To improve the mapping accuracy of replicas, *GLARAS* invokes the *remBadCan*, which on receiving *oRepSet*, *vRepSet*, $size_{vs}$, and *maxSize*, finds the replication candidates with accuracy less than one, and marks them as bad candidates of replication. The bad candidates are removed from the set of name IDs in region q of virtual system i.e., I_q^v. By calling *remBadCan* function, *GLARAS* obtains the updated set of I_q^v with bad candidates removed (Algorithm 5.1, Line 12). Additionally, in the updated set of I_q^v, for each bad candidate node $i \in I_q^v$, *remBadCan* removes all nodes $j \in I_q^v$ with $commonPrefix(i, j) \geq commonPrefix(i, i')$ where i' is the node in original system with most similar name ID to node i. Since there exists no name ID in the original system with greater than $commonPrefix(i, i')$ bits common prefix length with node i, all name IDs with more than $commonPrefix(i, i')$ bits common prefix length with node i are removed from the virtual system.

GLARAS repeats RWD on the updated I_q^v, obtains the new set of replicas for region q of the virtual system, maps the replicas from virtual system to the original system, and updates I_q^v by removing the bad candidates. If a noticeable number of bad replication candidates are removed, *GLARAS* expands the virtual system size to support a bigger and more precise model of the original system while preserving the computation time. If after refining the bad candidates, size of I_q^v breaks down to less than the half of its full size (i.e., $|I_q^v| < \frac{size_{vs}}{2}$), *GLARAS* doubles the size of the virtual system, and extends the existing name IDs in I_q^v by invoking *expandNameIDs* (Algorithm 5.1, Lines 13–15). *expandNameIDs* replaces each name ID in I_q^v with two name IDs: one with a suffix of 0 and the other with a suffix of 1. The decision of expanding the virtual system size is taken by *remBadCan* function. To announce this decision, in addition to I_q^v, *remBadCan* outputs the *status* variable, which can take one of the possible values of *continue*, *growth*, or *terminate*. A value of *continue* tells *GLARAS* to repeat the RWD on the refined set of I_q^v without any expansion. A value of *growth* signals *GLARAS* that $|I_q^v| < \frac{size_{vs}}{2}$, and it should expand the virtual system size. A value of *terminate* tells that there is no bad candidate to refine, or the ILP was infeasible for RWD, and *GLARAS* should conclude the replication for region q. The infeasibility occurs when due to the refinements the number of replication candidates becomes less than the sub-replication degree of region q that is $|I_q^v| < r_q$. Lastly, if $|I_q^v| > maxSize$ then *remBadCan* decides on termination, where *maxSize* is the maximum size of ILP model that the data owner can solve efficiently.

Once the replica selection for all regions is accomplished, *GLARAS* returns *repSet* to the data owner. *repSet* represents the finalized replica set for the data owner and contains the concatenation of *bestRepSet*s that are collected from each region. On receiving the replicas set from *GLARAS*, the data owner replicates on the nodes of replicas list. After the replication is done, any data requester can communicate the data owner asking for the replica list. In response, the data owner sends back the list of replica addresses as well as name IDs. The data requester node then finds the replica with the most similar name ID and contacts it instead of contacting the data owner. Since *GLARAS* works on top of a locality-aware name ID assignment, the replica with the most similar name ID to a data requester is the replica with minimum access delay to it.

5.3. Complexity

SWD part of *GLARAS* is done only once for the entire lifetime of the system and consists of two nested loops each iterating $|L|$ times. Therefore, the running time of SWD is $O(|L|^2) = O(\log^2 n)$ where n is the maximum number of nodes in the system. For a fixed $size_{vs}$, size of RWD's ILP of *GLARAS* that is presented by Eqs. (3)–(8) is $O(size_{vs}^2)$. In our simulations with 100 randomly generated topologies each with $n = 4096$ nodes, starting from the $size_{vs} = 4$ nodes, the virtual system size expansion occurred on average 2.72 times. This means that, on the average, RWD's ILP is modeled for the virtual system sizes of 4, 8, and 16 nodes. It is worth mentioning that the initial value of $size_{vs} = 4$ is crucial for *GLARAS* to function properly with the size expansions.

In our storage system, we consider $maxSize = O(\log n)$ for all the data owners that invoke *GLARAS* . *maxSize* is the computational bound of the data owner on solving the RWD's ILP model efficiently, and *GLARAS* always considers $size_{vs} \leq maxSize$. In public replication case, $|K_q| = size_{vs}$ i.e., all the name IDs of region q of the virtual system are considered as data requesters. In private replication, however, a subset of name IDs inside each region q of the virtual system is considered as data requesters i.e., $|K_q^v| \leq size_{vs}$. Combining these two cases makes $|K_q^v| = O(size_{vs})$ and the size of RWD's ILP objective function shown by Eq. (3) as $O(size_{vs}^2)$. Similarly, the ILP model has $O(size_{vs}^2)$ constraints of the type shown by Eq. (4), $O(size_{vs})$ constraints of the types shown by Eqs. (5) and (6), one constraint of the type shown by Eq. (7), and $O(size_{vs}^2)$ constraints of the type shown by Eq. (8), which concludes the size of RWD's ILP model as $O(size_{vs}^2)$. Since $size_{vs} \leq maxSize$ and $maxSize = O(\log n)$, the RWD's ILP model has the size of $O(\log^2 n)$. This discussion derives that by providing $O(\log^2 n)$ ILP size, the asymptotic running time of RWD remains polynomial in n even when the ILP solver algorithm running time is exponential in n. In our simulations, average $size_{vs}$ was $1.33 \times \log n$. In general, RWD can replace the ILP model with its corresponding relaxed LP version. However, with the shown behavior of $size_{vs}$, the problem size is small enough that makes ILP plausible and consistent with the RWD criterion.

The overall running time of *GLARAS* depends on the number of RWD's ILPs to be solved, which corresponds to the number of while-loop iterations of *GLARAS* over RWD. The number of iterations, in turn, depends on the number times that virtual system size is expanded and the number of bad candidates that are removed. In the worst case, the virtual system size experiences a growth from 4 to *maxSize* as powers of two, with *remBadCan* function removing only one bad replication candidate from I_q^v at each execution. Since virtual system size expansion follows a geometric pattern with base 2, the number of times it expands until it crosses the bound of *maxSize* is $O(\log maxSize) = O(\log \log n)$. For a fixed virtual system size $size_{vs}$, in the worst case *remBadCan* removes one bad candidate per execution, which enforces *GLARAS* to iterate $\frac{size_{vs}}{2}$ times before the next virtual system size expansion. This forms a geometrical series with the ratio of 2 and initial value of 4, which after $O(\log \log n)$ expansions sums up to total $O(\log n)$ iterations over RWD.

5.4. Comparison with our previous work

In our previous work, we proposed LARAS [33] that in contrast to *GLARAS*, distributes the replication degree among the regions merely based on their expected number of data requesters. This results in the concentration of replicas in condensed regions of data requesters while making the scattered regions empty of the replica. Likewise, in LARAS there exist no accuracy-based iterations, which increases the average access delay of replication due to the mismatch of virtual and original systems' views of name IDs distribution. Also, LARAS shrinks the original system size adaptively based on the expected number of data requesters. This makes some regions of the virtual system as big as the original system, which negatively affects the running time of replication.

6. Simulation setup

6.1. Simulator

We developed a new version of the open source Skip Graph simulator, SkipSim [34]. In comparison to the older version, the new version is fully object-oriented with improvements in both CPU and memory utilization. Each simulation consists of 100 randomly generated topologies. The topologies are generated once, stored, and loaded for each simulation. In SkipSim, each topology is generated in a 7000×7000 points environment where the pairwise latency between each pair of nodes is identical to their Euclidean distance.

Each name ID assignment strategy is simulated with all 100 random topologies, and evaluated by the average latency of nodes with their lookup table neighbors in the Skip Graph, as well as the average end-to-end latency of search queries. To measure the average end-to-end latency of searches, a number of random searches for name IDs, as well as numerical IDs are initiated for each topology. The randomness of a search is achieved by the randomized selection of its search initiator and search target. The average end-to-end latency of the searches for each topology is computed, and the average over all the topologies is reported.

Each replication algorithm is likewise simulated with all 100 random topologies. For all replication simulations, name IDs of nodes are generated by employing the *LANS* algorithm, which performs as the best-decentralized locality aware name ID assignment for Skip Graph. Each replication algorithm is simulated in two modes: public replication and private replication. In public replication, all nodes are considered as data requesters. In private replication, for each topology, a fixed set of data requesters is chosen randomly once, stored, and loaded with that topology. In both public and private replications, for each topology, one data owner is chosen randomly once, stored, and loaded alongside the topology. The data owner executes the specified replication algorithm. After the replicas are determined, the average access delay between each data requester and its closest replica in each topology is computed. As the replication algorithm's performance, the average access delay over all topologies is reported.

6.2. Algorithms used for comparison

6.2.1. Name ID assignments

We implemented the best known name ID assignment algorithms and compared their performance with *LANS* in terms of providing locality awareness and end-to-end latency of search queries in Skip Graph. Implementation details of these algorithms are as follows:

LAND: In LAND, name IDs are $\lceil \log n \rceil$ bits string, which are chosen uniformly at random.

LDHT: The prefix part of nodes' name ID in LDHT is the prefix of the closest landmark to the node. Each landmark in LDHT defines an ASN by a fixed size randomly assigned prefix. The body part of name IDs are $\lceil \log n \rceil$ bits string, which are chosen uniformly at random.

Hierarchical: Similar to *LANS*, prefix part of nodes' name ID is the dynamic prefix of the closest landmark to the node. The body part of name IDs is chosen randomly as $\lceil \log n \rceil$ bits binary strings.

LMDS: A *Distance* matrix of size $n \times |L|$ is generated where $Distance[i][j]$ corresponds to the pairwise latency between node i and landmark j. The LMDS is applied on the *Distance* matrix, which results in a single LMDS value for each node. Nodes are sorted based on their LMDS value in ascending order. Name ID is the $\lceil \log n \rceil$ bits binary representation of node's rank in the sorted list.

Dynamic Prefix LMDS (DPLMDS): To compare with our proposed *LANS*, we implemented an improved version of LMDS, that is called DPLMDS. In DPLMDS, the prefix part of nodes' name IDs is the dynamic prefix of their closest landmark. The body part of name IDs is obtained in the same way as LDMS.

6.2.2. Replication algorithms

We implemented the best known decentralized locality-based replication algorithms and compared their performance in term of average access delay with *GLARAS* . Implementation details of these algorithms are as follows where in all cases the data owner is chosen uniformly at random:

Randomized replication: A data owner is chosen uniformly at random, and the replicas are chosen randomly from the set of nodes including the data owner until the replication degree is met.

Replication on neighbors: A data owner is chosen randomly from the set of nodes that have neighbors greater than or equal to the replication degree. The replicas are chosen randomly from the set of data owner's neighbors until the replication degree is met.

Replication on path: Data requesters start searching for the data owner's numerical ID or name ID. On each search query, replicas are placed on the path from the data requester node to the data owner including both the data requester and the data owner, until the replication degree is met.

Replication adaptively on path: We developed an improved version of replication on the path for the sake of comparison, which is called adaptive on path. In this improved version, the data owner records the search path of all the search queries that are destinated to it, and distributes its replication degree on the nodes that contribute to the maximum number of search paths. During the simulation of this algorithm, the data owner waits until all its data requesters perform a search for name ID or numerical ID of the data owner. The data owner picks the nodes with the maximum number of contributions to the search paths and replicates on them.

7. Performance results

7.1. Name ID assignments

Fig. 2 shows the performance of name ID assignment algorithms in providing locality awareness for Skip Graph in different scales of system capacity. As the locality awareness metric, we consider the average latency of each node to its neighbors in Skip Graph. The reported average is taken over all nodes of Skip Graph i.e., the system capacity. LAND and LMDS that do not employ the prefix of landmarks, act as the weakest ones. As the system capacity scales up and hence the number of landmarks increases, the name ID assignment algorithms that are based on the prefix of landmarks, provide a finer grained hierarchical distinction among the nodes,

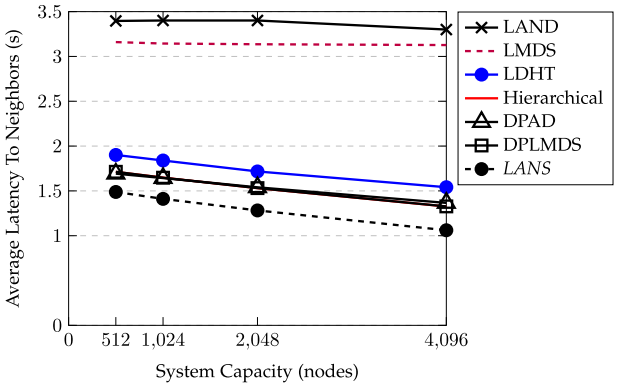

Fig. 2. Average latency of nodes to their neighbors in Skip Graph vs. system capacity. *Y*-axis: average latency in seconds. *X*-axis: system capacity.

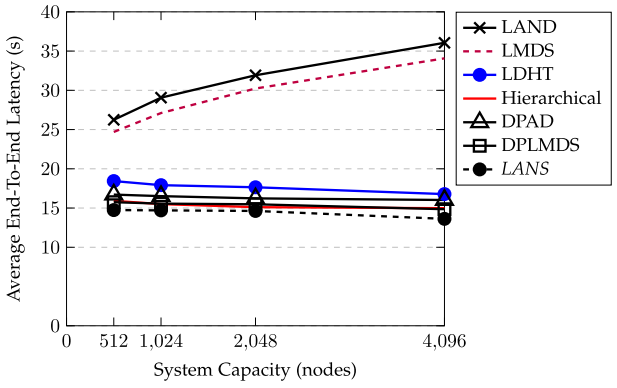

Fig. 3. Average end-to-end latency of search for numerical ID in Skip Graph vs. system capacity. *Y*-axis: average end-to-end latency in seconds. *X*-axis: system capacity.

which results in improvement of locality awareness. LDHT considers a fixed size static prefix for the landmarks that is independent of their latency-based coordinate. This has the disadvantage for LDHT in providing locality awareness among the landmark-based name ID assignment approaches. Compared to DPAD that acts as the best existing decentralized approach, *LANS* improves the locality awareness of Skip Graph's nodes with the gain of about **19%**.

Fig. 3 and 4 show the average end-to-end latency of search for numerical ID and search for name ID that are resulted by employing each of the name ID assignment strategies in the Skip Graph in different scales of system capacities, respectively. We consider the end-to-end latency of searches as the total pairwise latency of consecutive nodes on the search path from the search initiator to the search target. For each topology, $256 \times n$ random searches for numerical ID and name ID are initiated separately where n denotes the system capacity. The average end-to-end latency of searches is recorded for each topology, and the average of all topologies is reported. Growing system capacity comes with the growing number of nodes on the search path. Following the behavior of algorithms in providing locality awareness (Fig. 2), as illustrated by Figs. 3 and 4, since average pairwise latency of nodes in LAND and LMDS are almost independent of the system capacity, by increasing the number of nodes on the search paths, their end-to-end latency of searches increases. On the other hand, the landmark-based name ID assignments have a decreasing pairwise latency of nodes. Hence, in larger system capacities, although the number of nodes on the search paths increases, the decreasing pairwise latency of nodes on the search path results in improving their end-to-end latency of searches. Compared to the hierarchical approach that performs as the best existing solution in the search for numerical ID, *LANS* improves the end-to-end latency of search for numerical ID with a gain of about **5%**. In the case of the search for name ID, compared to the DPAD that performs as the best among the existing solutions, *LANS* has a gain of about **10%**.

7.2. Replication

Fig. 5 shows the public replication's average access delay of replication algorithms in the system capacity of 4096 nodes as the replication degree is scaled up. In a locality aware Skip Graph, the majority of a node's neighbors are the ones with the lowest pairwise latency. This causes the replication on the neighbors to place the majority of replicas around the data owner, which results in higher access delay for the far away data requester nodes. Due to the locality awareness of Skip Graph, the expected pairwise latency between consecutive nodes on a search path is low, which

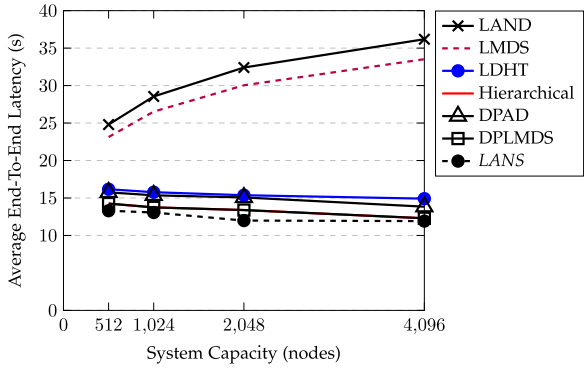

Fig. 4. Average end-to-end latency of search for name ID in Skip Graph vs. system capacity. *Y*-axis: average end-to-end latency in seconds. *X*-axis: system capacity.

causes the compression of replicas in some parts of the system and enforces higher access delay for rest of system. In comparison to LARAS that acts as the best existing decentralized solution, *GLARAS* improves the average access delay of public replication with a gain of about **13%**. We also simulated the replication algorithms in the private replication mode with the fixed number of 400 randomly chosen data requesters and observed a similar performance of the replication algorithms as in the case of the public replication. Compared to LARAS that acts as the best existing decentralized solution in the private replication case, *GLARAS* improves the average access delay of private replication with the gain of about **17%**.

Fig. 6 illustrates the average number of replicas needed for the replication algorithms in private replication mode to achieve the maximum 1.5 s average access delay of replication in the system capacity of 4096 nodes as the number of data requesters is scaled up. As shown in this figure, almost all replication algorithms are acting independently of the system capacity with *GLARAS* acting as the best. Compared to LARAS that acts as the best existing decentralized solution, *GLARAS* needs about **21%** less number of replicas to provide the maximum average access delay of 1.5 s in private replication.

7.3. Running time

Running on Intel i5 2.60 GHz CPU and 8 GB of RAM, a single execution of *LANS* in a system with 4096 nodes takes the average computation time of about 1 s. To validate a generated name ID,

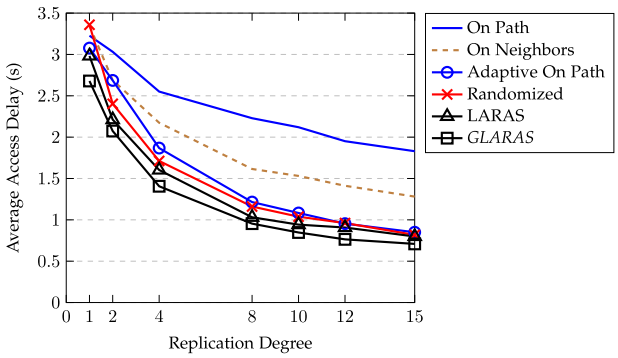

Fig. 5. Performance of replication algorithms in public replication mode. Original system capacity is 4096 nodes. *Y*-axis: average access delay between each node and its closest replica in seconds. *X*-axis: replication degree.

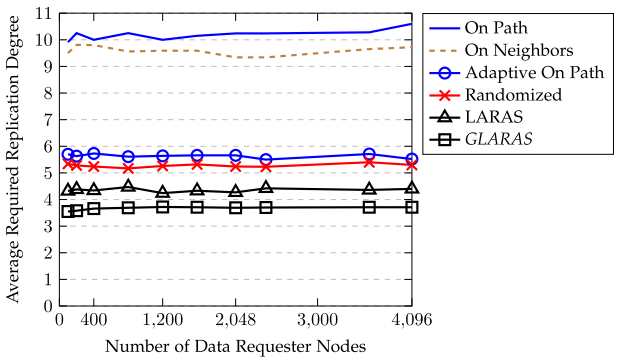

Fig. 6. Number of replicas needed to achieve the maximum 1.5 s private replication average access delay vs. number of data requester nodes in private replication mode. Original system capacity is 4096 nodes. *Y*-axis: replication degree. *X*-axis: number of data requester nodes.

LANS conducts 4.18 searches for name IDs on the average. Based on our simulation results, a single search for name ID in a system with 4096 nodes takes about 12 s on the average, which makes the total communication time of *LANS* about 50 s. Having the computation time of about 1 s and communication time of about 50 s, an arriving node to the system can obtain its locality aware name ID in less than one minute.

Likewise, using the lpsolve 5.5 [63] to solve the ILP model of *GLARAS*, in a system with 4096 nodes and 12 landmarks the average computation time of *GLARAS* to place a replica in a region is about 5 s. *GLARAS* performs 4 searches for name IDs on the average to place a single replica. Hence placing a replica in a region costs the computation time of about 5 s and communication time of about 48 s, which makes a data owner waiting for about 10 min to place all its 12 replicas in the system. On a quad core Intel i5, which can support 4 concurrent executions of *GLARAS* 's RWD, this total average running time can be reduced to about 2 min by performing the placement of replicas in parallel for the regions.

7.4. Comparison to our previous work

In the scale of 4096 nodes, compared to our previously proposed DPAD name ID assignment algorithm [33], *LANS* is about 19% better in providing the locality awareness of name IDs and improves the average end-to-end latency of search queries with the gain of about 13%. However, to assign a single name ID, *LANS* is about 2 times slower compared to DPAD and charges the system with 3 more searches for name IDs on the average. Compared to our previously proposed LARAS replication approach [35], *GLARAS* is about 200 times faster. In the scale of 4096 nodes based on *LANS* name ID assignment approach, *GLARAS* has the gain of about 13% and 17% over LARAS in the public and private replications, respectively. However, in contrast to LARAS which only needs one search for name ID to place each replica, *GLARAS* needs the average of 4.21 searches to place a single replica. The average access delay of replication for a Skip Graph-based P2P storage system that is based on *GLARAS* and *LANS* is about **2.7** times better in comparison to a Skip Graph-based P2P storage system that is based on our previously proposed LARAS and DPAD.

8. Conclusion

To improve the performance of Skip Graph based P2P storage systems in terms of query processing and response time, we proposed *GLARAS*, a dynamic and fully decentralized locality aware replication algorithm for Skip Graph. Using *GLARAS*, any data owner can place its replicas where minimum average access delay of replication is achieved. We define the average access delay of replication as the average latency between each data requester and its closest replica.

GLARAS works on the top of a Skip Graph that uses a landmark-based locality aware name ID assignment. We define the locality awareness of Skip Graph as assigning the name IDs such that the latency between each pair of nodes in the system corresponds to the common prefix length of their name IDs where longer common prefix denotes a lower latency. As an independent contribution, we proposed a dynamic fully decentralized locality aware name ID assignment called *LANS*.

To evaluate the performance of our proposed algorithms, we optimized the Skip Graph simulator, SkipSim [34] to be scalable up to 4096 nodes. We implemented *LANS*, *GLARAS* as well as state-of-the-art identifier assignments and decentralized replications. Compared to our previously proposed LARAS [35], which acts as the best known existing decentralized locality aware replication for Skip Graph, *GLARAS* improves the average access delay of public and private replications with a gain of about **13%** and **17%**, respectively. Likewise, in contrast to the best existing solutions on DHTs that rely on the static explicit assumptions about the distribution of nodes, *GLARAS* acts independently of the nodes' distribution in the underlying network.

In comparison to our previously proposed DPAD [33], which acts as the best existing decentralized locality aware name ID assignment for Skip Graph, *LANS* improves the locality awareness of name IDs, and end-to-end latency of searches with the gains of about **19%**, **8%**, respectively. As our future work, we aim to minimize the average access delay of replication in a Skip Graph under the churn considering the load and bandwidth constraint of nodes.

Acknowledgments

The authors thank Türk Telekom for their support. We also thank Amin Alizadeh for his contributions to the SkipSim implementation.

References

[1] Q. He, Z. Li, X. Zhang, Study on cloud storage system based on distributed storage systems, in: ICCIS, IEEE, 2010.
[2] K. Hwang, S. Kulkareni, Y. Hu, Cloud security with virtualized defense and reputation-based trust mangement, in: DASC, IEEE, 2009.
[3] K. Xu, M. Song, X. Zhang, J. Song, A cloud computing platform based on p2p, in: ITIME, IEEE, 2009.

[4] J. Aspnes, G. Shah, Skip graphs, Acm TALG (2007).
[5] M. Uddin, R. Stadler, A bottom-Up approach to real-time search in large networks and clouds, in: NOMS, 2016 IEEE/IFIP.
[6] A. Singh, S. Batra, P-skip graph: An efficient data structure for peer-to-peer network, in: Intelligent Distributed Computing, Springer, 2015.
[7] T. Shabeera, P. Chandran, S. Kumar, Authenticated and persistent skip graph: a data structure for cloud based data-centric applications, in: ICACCI, ACM, 2012.
[8] Y. Hassanzadeh-Nazarabadi, A. Küpçü, Ö. Özkasap, Awake: decentralized and availability aware replication for p2p cloud storage, in: Smart Cloud, IEEE, 2016.
[9] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, ACM SIGCOMM (2001).
[10] W. Wu, J.C. Lui, Exploring the optimal replication strategy in p2p-vod systems: Characterization and evaluation, IEEE Trans. Parallel Distrib. Syst. (2012).
[11] G. Liu, H. Shen, H. Chandler, Selective data replication for online social networks with distributed datacenters, IEEE Trans. Parallel Distrib. Syst. (2016).
[12] M. Almashor, I. Khalil, Z. Tari, A.Y. Zomaya, S. Sahni, Enhancing availability in content delivery networks for mobile platforms, IEEE Trans. Parallel Distrib. Syst. (2015).
[13] P. Matri, M.S. Pérez, A. Costan, L. Bougé, G. Antoniu, Keeping up with storage: Decentralized, write-enabled dynamic geo-replication, Future Gener. Comput. Syst. (2017).
[14] N.K. Gill, S. Singh, A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers, Future Gener. Comput. Syst. (2016).
[15] H. Mustafa, B. Baveja, S. Vijayan, S. Merchant, U.B. Desai, Replicating the geographical cloud: Provisioning omnipresence, omniscience and omnipotence, Future Gener. Comput. Syst. (2015).
[16] J. Su, D. Reeves, Replica placement algorithms with latency constraints in content distribution networks, in: Technical Report, ACM, 2004.
[17] S. Ktari, M. Zoubert, A. Hecker, H. Labiod, Performance evaluation of replication strategies in dhts under churn, in: MUM, ACM, 2007.
[18] A. Harwood, D.E. Tanin, Hashing spatial content over peer-to-peer networks, in: ATNAC, University of Melbourne, 2003.
[19] Z. Xiaosu, W. Xiaolin, H. Hao, Caching, replication strategy and implementions of directories in dht-based filesystem, in: ICPCA 2011. IEEE.
[20] J. Paiva, L. Rodrigues, On data placement in distributed systems, Oper. Syst. Rev. (2015).
[21] T. Pitoura, N. Ntarmos, P. Triantafillou, Replication, load balancing and efficient range query processing in dhts, in: Advances in Database Technology-EDBT, Springer, 2006.
[22] P. Knežević, A. Wombacher, T. Risse, Dht-based self-adapting replication protocol for achieving high data availability, in: Advanced Internet Based Systems and Applications, Springer, 2009.
[23] J. Paiva, L. Rodrigues, Policies for efficient data replication in p2p systems, in: IEEE ICPADS, 2013.
[24] T. Chang, M. Ahamad, Improving service performance through object replication in middleware: a peer-to-peer approach, in: IEEE P2P 2005.
[25] O. A.-H. Hassan, L. Ramaswamy, J. Miller, K. Rasheed, E.R. Canfield, Replication in overlay networks: A multi-objective optimization approach, in: Collaborative Computing: Networking, Applications and Worksharing, Springer, 2009.
[26] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Keleher, Adaptive replication in peer-to-peer systems, in: Distributed Computing Systems 2004. IEEE.
[27] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM CSUR (2004).
[28] S. Luo, M. Hou, S. Zhan, M. Lyu, M. Li, Consistency maintenance in replication: A novel strategy based on diamond topology in cloud storage, Chin. J. Electron. (2017).
[29] B. Cohen, Incentives build robustness in bittorrent, in: Workshop on Economics of Peer-To-Peer Systems, 2003.
[30] Y.-Y. Teing, A. Dehghantanha, K.-K.R. Choo, L.T. Yang, Forensic investigation of p2p cloud storage services and backbone for iot networks: Bittorrent sync as a case study, Comput. Electr. Eng. (2017).
[31] R. Chaabouni, P. Garcia-Lopez, M. Sanchez-Artigas, S. Ferrer-Celma, C. Cebrian, Boosting content delivery with bittorrent in online cloud storage services, in: P2P, IEEE, 2013.
[32] P. Maymounkov, D. Mazieres, Kademlia: A peer-to-peer information system based on the xor metric, in: International Workshop on Peer-To-Peer Systems, Springer, 2002.
[33] Y. Hassanzadeh-Nazarabadi, A. Küpçü, Ö. Özkasap, Locality aware skip graph, in: IEEE ICDCSW, 2015.
[34] SkipSim: https://gitlab.com/yhassanzadeh13/skipsim-distribution-bundle.
[35] Y. Hassanzadeh-Nazarabadi, A. Küpçü, Ö. Özkasap, Laras: Locality aware replication algorithm for the skip graph, in: IEEE NOMS 2016.
[36] I. Abraham, D. Malkhi, O. Dobzinski, Land: Locality aware networks for distributed hash tables, TR 2003-75, Leibnitz Center, The Hebrew University, Tech. Rep.
[37] I. Abraham, D. Malkhi, O. Dobzinski, Land: stretch (1+ epsilon) locality-aware networks for dhts, in: SODA, 2004.
[38] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, X. Li, Ldht: locality-aware distributed hash tables, in: IEEE ICOIN 2008.
[39] G. Huston, Exploring autonomous system numbers, Int. Protoc. J. (2006).
[40] S. Zhou, G.R. Ganger, P.A. Steenkiste, Location-based node ids: Enabling explicit locality in dhts, Technical Report, Carnegie Mellon University, 2003.
[41] T. Toda, Y. Tanigawa, H. Tode, Autonomous and distributed construction of locality aware skip graph, in: CCNC, IEEE, 2017.
[42] V. De Silva, J.B. Tenenbaum, Sparse multidimensional scaling using landmark points, Stanford, Tech. Rep., 2004.
[43] S. Lee, S. Choi, Landmark mds ensemble, Pattern Recognit. (2009).
[44] U. Brandes, C. Pich, Eigensolver methods for progressive multidimensional scaling of large data, in: Graph Drawing, Springer, 2007.
[45] F. Araújo, L. Rodrigues, Geopeer: A location-aware peer-to-peer system, in: 3rd IEEE NCA, 2004.
[46] D.-T. Lee, B.J. Schachter, Two algorithms for constructing a Delaunay triangulation, Int. J. Comput. Inf. Sci. (1980).
[47] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM CSUR (1991).
[48] A. Allan, R. Humphrey, G.D. Fatta, Non-euclidean internet coordinates embedding, in: 13th ICDMW, 2013. IEEE.
[49] V. Martins, E. Pacitti, P. Valduriez, Survey of data replication in p2p systems, HAL, 2006.
[50] R. Mokadem, A. Hameurlain, Data replication strategies with performance objective in data grid systems: a survey, Int. J. Grid Util. Comput. (2014).
[51] M. Shorfuzzaman, P. Graham, R. Eskicioglu, Allocating replicas in large-scale data grids using a qos-aware distributed technique with workload constraints, Int. J. Grid Util. Comput. (2012).
[52] F. Ben Charrada, H. Ounelli, H. Chettaoui, An efficient replica placement strategy in highly dynamic data grids, Int. J. Grid Util. Comput. (2011).
[53] A. Pace, V. Quéma, V. Schiavoni, Exploiting node connection regularity for dht replication, in: IEEE SRDS, 2011.
[54] Y. Chen, R.H. Katz, J.D. Kubiatowicz, Dynamic replica placement for scalable content delivery, in: Peer-To-Peer Systems, Springer, 2002.
[55] S.-Q. Long, Y.-L. Zhao, W. Chen, Morm: A multi-objective optimized replication management strategy for cloud storage cluster, J. Syst. Archit. (2014).
[56] A.S. Vijendran, S. Thavamani, An efficient algorithm for clustering nodes, classifying and replication of content on demand basis for content distribution in p2p overlay networks, Int. J. Comput. Commun. Technol. (2013).
[57] D. Yang, Y.-x. Zhang, H.-k. Zhang, T.-Y. Wu, H.-C. Chao, Multi-factors oriented study of p2p churn, Int. J. Commun. Syst. (2009).
[58] R. Pecori, L. Veltri, 3akep: Triple-authenticated key exchange protocol for peer-to-peer voip applications, Comput. Commun. (2016).
[59] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, Commun. ACM (1990).
[60] T. Crain, V. Gramoli, M. Raynal, No hot spot non-blocking skip list, in: 33rd ICDCS 2013. IEEE.
[61] M.T. Goodrich, R. Tamassia, A. Schwerin, Implementation of an authenticated dictionary with skip lists and commutative hashing, in: DISCEX 2001. IEEE.
[62] J. Qin, W. Fu, H. Gao, W.X. Zheng, Distributed k-means algorithm and fuzzy c-means algorithm for sensor networks based on multiagent consensus theory, IEEE Trans. Cybern. (2017).
[63] lpsolve5.5: http://lpsolve.sourceforge.net/5.5/.

Yahya Hassanzadeh-Nazarabadi is a Ph.D. Candidate in the Department of Computer Engineering at Koç University. His research interests are distributed cloud storage, P2P systems, replication and security.

Alptekin Küpçü received his Ph.D. degree from Brown University Computer Science Department in 2010. Since then, he has been working as an assistant professor at Koç University, and leading the Cryptography, Security & Privacy Research Group he has founded. His research mainly focuses on applied cryptography, and its intersection with cloud security, privacy, peer-to-peer networks, and game theory and mechanism design. Dr. Küpçü has various accomplishments including 3 international patents granted, 7 funded research projects (for 5 of which he was the principal investigator), 2 European Union COST Action management committee memberships, 2 Koç University Teaching Innovation Awards, Science Academy Young Scientist Award (BAGEP), Turkish Academy of Sciences Outstanding Young Scientist Award (GEBİP), and the Royal Society of UK Newton Advanced Fellowship.

Öznur Özkasap received the M.S. and Ph.D. degrees in Computer Engineering from Ege University, Izmir, Turkey, in 1994 and 2000, respectively. From 1997 to 1999, she was a Graduate Research Assistant with the Department of Computer Science, Cornell University, Ithaca, NY, USA, where she completed her Ph.D. dissertation. She is currently an Associate Professor with the Department of Computer Engineering, Koç University, Istanbul, Turkey, which she joined in 2000. Her research interests include distributed systems, multicast protocols, peer-to-peer systems, bioinspired distributed algorithms, mobile ad hoc networks, energy efficiency, cloud computing, and computer networks. She serves as an Area Editor of the Future Generation Computer Systems journal, Elsevier Science. She also served as an Area Editor of the Computer Networks journal, Elsevier Science, and as a Management Committee Member of the European COST Action IC0804: Energy efficiency in large-scale distributed systems.