Decentralized utility- and locality-aware replication for heterogeneous DHT-based P2P cloud storage systems

Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap

Department of Computer Engineering, Koç University, İstanbul, Turkey

{yhassanzadeh13, akupcu, oozkasap}@ku.edu.tr

July 30, 2019

Abstract

As a Distributed Hash Table (DHT), Skip Graph routing overlays are exploited in several peer-to-peer (P2P) services, including P2P cloud storage. The fully decentralized replication algorithms that are applicable to the Skip Graph-based P2P cloud storage fail on improving the performance of the system with respect to both the availability of replicas as well as their response time. Additionally, they presume the system as homogeneous with respect to the nodes' latency distribution, availability behavior, bandwidth, or storage. In this paper, we propose *Pyramid*, which is the first fully decentralized utility- and locality-aware replication approach for Skip Graph-based P2P cloud storage systems. *Pyramid* considers the nodes as heterogeneous with respect to their latency distribution, availability behavior, bandwidth, and storage. *Pyramid* is utility-aware as it maximizes the average available bandwidth of replicas per time slot (e.g., per hour). Additionally, *Pyramid* is locality-aware as it minimizes the average latency between nodes and their closest replica. Our simulation results show that compared to the state-of-the-art solutions that either perform good in utility-awareness, or in locality-awareness, our proposed *Pyramid* improves both the utility- and locality-awareness of replicas with a gain of about 1.2 and 1.1 times at the same time, respectively.

1   Introduction

A peer-to-peer (P2P) cloud storage system consists of a set of peers, i.e., computing nodes that are interconnected over the Internet (e.g., computers, mobile devices, resource-constrained devices). There exist two roles in such cloud storage systems; data owner and data requester. A data owner holds a set of data objects and aims to share them with a group of authorized peers that are named

data requesters. A data owner may also be a data requester for the other data owners.

Distributed Hash Table (DHT)-based P2P cloud storage [1–3] is a type of P2P cloud storage systems that operates over a structured overlay, where each peer is represented by a *node* in the overlay. Each node knows a logarithmic number of other nodes (i.e., neighbors) in the system, and keeps them as $(ID, address)$ pairs in its *lookup table.* Using its lookup table, each node is able to efficiently search and find other nodes in a fully decentralized manner. Skip Graph [4] is a DHT routing overlay that supports scalability, fast searching, and load balancing regarding message routing. Such features enable Skip Graph as a suitably structured overlay for DHT-based P2P cloud storage applications [5–14]. Furthermore, Skip Graph can be considered as an alternative to other DHT overlays (e.g., Chord [15]) in their DHT-based P2P services ranging from distributed storage systems [16] to online social networks [17] and search engines [18].

Nodes in P2P cloud storage systems are prone to churn, which is known as the dynamic arrival and departure behavior of the nodes [19]. Once a node arrives at the system, it is considered as available (online) until it departs the system (i.e., goes offline) or fails. A departed or failed node may join the system at a later time or may leave the system forever. Churn in the system results in data unavailability in P2P cloud storage [20], as upon departure or failure of a data owner, its data objects would no longer be available to the data requesters. In order to reduce the query load over the data owner, avoid the single point of failure, provide fault tolerance, and improve the data availability under churn, the data owner makes copies of its data objects on some other nodes of the system, which are called the data owner's replicas. The process of determining and managing the replicas is known as replication [21]. Adapting a replication policy on the P2P cloud storage systems defines a collaborative environment where nodes participate with their idle storage spaces, and are in charge of storing each others' data objects in exchange of having their data objects stored on some other nodes of the system.

The performance of P2P cloud storage is evaluated with respect to the accessibility of replicas [22] and their availability under churn [23]. The accessibility of replicas is correlated with their latency distribution in the underlying network, and is evaluated by the average access delay, which corresponds to the average latency between each data requester and its closest replica. The closest replica of each data requester is the one with the minimum Round-Trip-Time (RTT), and is called the *corresponding replica* of that data requester. The replication approaches that aim at minimizing the average access delay of replicas are known as *locality-aware* [12]. The availability of replicas is correlated with their churn behavior, and is evaluated as their average availability per time slot, e.g., the average number of available replicas at each hour. The replication approaches that aim at maximizing the average availability of replicas are known as *availability-aware* [10]. Following these performance-oriented goals, the best P2P cloud storage is the one that provides both the locality- and availability-awareness of replicas at the same time.

2

The locality- and availability-awareness of replicas have not been addressed together in a fully decentralized manner for DHT-based P2P cloud storage systems [20, 24]. By the full decentralization, we mean a strategy where the data owner is able to place its replicas without the need of any special node (e.g., supernodes) to handle the computation, storage, or communication that is needed for the replica placement. Our definition of full decentralization, hence, stands against the solutions such as [25–28], where a centralized entity is in charge of collecting the state of all nodes, executing the replication algorithm on behalf of the data owner, and deciding on the replica placement.

Considering the locality-awareness for Skip Graph-based P2P cloud storage systems, GLARAS [13] is the best fully decentralized one. However, it does not support availability-awareness of replicas. Likewise, the traditional fully decentralized availability-aware replications that are applicable to Skip Graphs do not consider the average access delay of replicas. Such availability-aware replications are classified into reactive and proactive ones. The reactive replications improve the data availability by reactively resolving the departure or failure of replicas [22, 29–34]. While the proactive ones aim at providing an average number of available replicas for a long period of time, e.g., providing three available replicas on average for one month [23, 35–45]. It is worth mentioning that some of the existing availability-aware replications aim at minimizing the average number of intermediate nodes between replicas and data requesters [22, 44]. However, as we experimentally demonstrated in [11], queries of the same path length show drastically different response time depending on the overlay connectivity. Hence, the number of intermediate nodes between the data requesters and replicas does not necessarily reflect the locality-awareness of replicas.

Evaluating the data availability of a P2P storage system by the availability-awareness of its replicas is only applicable to the homogeneous systems [23, 35, 46–48], where nodes participate with similar bandwidths. In such systems, a larger number of online replicas reflects a higher available bandwidth, and hence a better level of data availability for the data requesters. In the real world P2P systems, where nodes participate with heterogeneous bandwidths, a larger number of online replicas does not necessarily imply a better level of data availability. For example, two nodes with the available bandwidth of $100Kbps$ are not good replication candidates for serving 100 data requesters simultaneously. As each data requester only benefits from an average concurrent bandwidth of $2Kbps$. In the same example, a single node with an available bandwidth of $1Mbps$ provides an average bandwidth of $10Kbps$ for each data requester, which results in a better level of data availability. We conclude that addressing the data availability in the systems with heterogeneous bandwidth requires a stronger performance metric, which should consider both the availability of replicas as well as their bandwidth heterogeneity. To address this issue, we introduce the **utility-awareness** of replicas as a stronger performance metric (than their availability-awareness) for the bandwidth heterogeneous P2P cloud storage systems under churn. To define utility-awareness, we consider dividing a fixed-size periodic time interval ($FPTI$) into a set of identical time slots (ts), e.g., $FPTI$ as a day and ts as an hour. We then evaluate the utility-awareness of replicas as

3

the average available bandwidth of replicas during each ts where the average is taken over the corresponding data requesters of each online replica during each ts. Accordingly, we consider a replication mechanism as **utility-aware** if it aims at maximizing the utility-awareness of replicas. Similar to the availability-awareness, the utility-awareness is also correlated with the individual features of the nodes, i.e., availability behavior and bandwidth.

In this paper, to improve the data availability and data accessibility of the heterogeneous DHT-based P2P cloud storage systems, **we propose *Pyramid*, which is the first fully decentralized utility- and locality-aware replication algorithm for Skip Graph-based P2P cloud storage systems**. *Pyramid* considers the nodes as heterogeneous with respect to their latency distribution in the underlying network, availability behavior, bandwidth, and storage capacity. By employing *Pyramid*, a data owner can replicate its data objects in a fully decentralized manner with the maximized utility- and locality-awareness of its replicas is achieved. Since Skip Graphs can be used as alternatives to other DHTs in their P2P services, by employing *Pyramid*, any DHT-based application can benefit from a utility- and locality-aware replication service, e.g., DHT-based cloud storage systems [1–3, 15, 16], search engines [18], and social networks [17].

The original contributions of this paper are as follows:

- We propose *Pyramid*[1]: the first fully decentralized, proactive, utility-, and locality-aware replication algorithm for heterogeneous Skip Graph-based P2P cloud storage systems. *Pyramid* provides an optimization model that aims on maximizing both the utility- and locality-awareness of replicas.

- We extended the Skip Graph simulator, SkipSim [49], for simulating and evaluating the utility- and locality-awareness of replication algorithms. We implemented the best existing fully decentralized availability- and locality-aware replication algorithms on SkipSim, adapted them to our system model, and compared their performance against our proposed *Pyramid*.

- Based on our simulation results and compared to the best existing decentralized replications that either perform good in utility-awareness, or in locality-awareness, *Pyramid* improves the utility- and locality-awareness of the replicas with a gain of about 1.2 and 1.1 at the same time, respectively.

2 Preliminaries

Skip Graph: Skip Graph [4] is a DHT overlay where each node has two identifiers: a numerical ID and a name ID. Numerical IDs are non-negative

---

[1]*Pyramid* is the extension of our earlier work, Awake (*DOI: https://doi.org/10.1109/SmartCloud.2016.45*), which solely supports the availability awareness of replicas.

integers, and name IDs are binary strings. The basic operations in a Skip Graph overlay are the search for numerical ID [4] and search for name ID [12], which enable a (search initiator) node to look for the owner of a specific numerical ID or name ID, respectively. Having n nodes in a Skip Graph, a search initiator is able to perform both searches in a fully decentralized manner, and with the communication complexity of $O(\log n)$. As the result of a search for a numerical ID, (IP) address of the node with the largest numerical ID that is less than or equal to the search target is returned to the search initiator. As the result of a search for name ID, (IP) address(es) of the node(s) with the longest common **prefix** with the search target are returned to the search initiator. We elaborate more on the search for name ID in Section 4. In this paper, we define the *system capacity* as the maximum number of the registered nodes to the Skip Graph, denoted by n.

Locality-Aware Skip Graph: In a landmark-based Skip Graph [11] the overlay is virtually divided into a number of **regions**. Each region is recognized by a single landmark. **Landmarks** are not Skip Graph nodes, rather they are external components (e.g., servers) that are solely employed as reference points. Nodes use landmarks as some external reference points to ping, measure their RTT with respect to them, and share it with other nodes. Hence, using a landmark-based Skip Graph does not imply any sort of centralization at all, and the system is still administrated by the fully decentralized Skip Graph overlay of nodes. Studies like [50] also propose decentralized approaches to use the nodes themselves as the landmarks instead of relying on external ones. Each node of the Skip Graph belongs to the region of its closest landmark, which is the one with the minimum associated RTT. We denote the set of landmarks by L and assume that $|L| = O(\log n)$. We also assume that the set L is constant over time, and is known by every node of the system. In a landmark-based locality-aware Skip Graph overlay [13], the name IDs of the nodes are assigned in a way that the length of the common prefix in the name IDs of nodes is an inverse function of their RTT, i.e., a longer common prefix in name IDs of two nodes reflects their lower RTT in the underlying network. For example, in a locality-aware Skip Graph, a node with name ID of 0011 is expected to experience a lower latency to the node with name ID of 0001 than the node with name ID of 0111. Since the name ID of 0011 has a 2-bit common prefix length of 00 with 0001, while it has only a 1-bit common prefix length of 0 with 0111.

System-Wide Distribution of replicas (SWD): SWD is an independent module of the GLARAS replication algorithm [13], which approximates the optimal distribution of the replication degree among the regions of the system considering the locality-awareness of the replication. The replication degree denotes the number of replicas a data owner aims to place. A data owner executes SWD in a fully decentralized manner. Given the replication degree and the set of landmarks, SWD distributes the replication degree among the regions of the system based on some approximation on the data requesters' distribution in the Skip Graph overlay. As a result of SWD, each region of the system receives a sub-replication degree denoting the number of replicas should be placed in that

region of the system.

Churn Models: In P2P systems nodes are transient between online and offline states. The online and offline dynamics of the nodes is described by a churn model [19,51–53]. A churn model is specified by two distributions; session length, and inter-arrival time. The session length distribution characterizes the duration of the nodes' online states in the system. The inter-arrival time distribution characterizes the time interval between two consecutive arrivals of nodes to the system. Availability of a node is correlated with its session length, i.e., a longer average session length for a node corresponds its higher availability in the system. The availability of the system itself is correlated with the inter-arrival time distribution, i.e., a shorter average inter-arrival time corresponds a larger number of arrivals in the unit of time, which makes the system more available in the terms of the number of online nodes in the system.

3 System Model and Overview

System Model: Upon arrival to the system, a peer represents itself as a Skip Graph node by assigning its numerical ID as the hash value of its (IP) address, and its name ID by invoking a locality-aware name ID assignment scheme (e.g., LANS [13]), and joins the Skip Graph overlay using the join protocol in a fully decentralized manner [4]. Nodes use the Skip Graph overlay to discover each others' (IP) addresses by performing searches for each others' name IDs [12] or numerical IDs [4]. We assume that the nodes are heterogeneous with respect to their storage capacity, bandwidth, availability behavior, and latency distribution in the underlying network. Each online node frequently computes its utility state, which is a function of its available storage capacity, bandwidth, and availability behavior, and shares it with the other nodes using an aggregation mechanism. The aggregation scheme acts as a shared fully decentralized bulletin board, which keeps the aggregated utility state of the system.

In our system model, we assume that nodes depart the system arbitrarily by invoking the departure protocol of the Skip Graph in a fully decentralized manner [4]. The departure protocol keeps the connectivity of the overlay under churn. However, unnoticed departures can be handled via a decentralized churn stabilization algorithm (e.g., Interlaced [54]). Also, we assume the storage capacity of nodes is discrete in the unit of storage, and a data owner aims at utilization of one storage unit of a node for replication of its data object. Following this assumption, for example, if a node owns a free storage capacity of 3 units, it can be the replica of at most 3 data owners. Besides, we assume that all nodes are the data requesters of every data owner. Note that managing the access control of replicas is an orthogonal issue that, for example, can be handled by the data owner encrypting the data objects and sharing the key with the (authorized) data requesters.

Utility Vector: We model the utility state of a node at each time slot as a function of its available storage, bandwidth, and availability probability. We represent the utility state of the node i by the vector UV_i, named its *utility*

vector. Size of the utility vector corresponds to the number of time slots of length *ts* during one cycle of *FPTI* i.e., $|UV_i| = \frac{FPTI}{ts}$. For example, considering *FPTI* as a day and *ts* as an hour, utility vector of a node is of size 24. $UV_{i,t}$ represents the utility of the node i during the tth time slot of the *FPTI* (e.g., t^{th} hour of the day), and is computed as shown by Equation 1. In this equation, $p_{i,t}$ is the availability probability of node i during the tth time slot of *FPTI*, bw_i is its normalized bandwidth, and $rpLoad_i$ is its replication load.

$$UV_{i,t} = \frac{p_{i,t} \times bw_i}{rpLoad_i + 1} \qquad (1)$$

Node i computes $p_{i,t}$ by dividing the overall time it has been online during the tth time slot over the number of times that *FPTI* has cycled up to the computation time. For example, considering *FPTI* as a day and *ts* as an hour where 7 days have elapsed (i.e., *FPTI* has cycled 7 times), $p_{i,t}$ denotes the overall fraction of time that node i was online at the tth hour of the day, over the past 7 days. If the node i was online for 3 hours during the tth hour in the past 7 days, $p_{i,t} = \frac{3}{7} = 0.42$.

We define the *replication load* of node i (i.e., $rpLoad_i$) as the number of data owners that it has already been designated as their replica. The purpose of defining the replication load is two-fold. First, the replication load determines the amount of storage that a node devotes to serve as the replica. Second, one can compare the strength of the bandwidth that two replicas provide per data requester on the average by comparing their corresponding replication loads. For example, for two replica nodes with an identical bandwidth, a data requester is likely to get a better bandwidth from the one with lower replication load, as the loosely loaded replica likely has to respond to less number of data requesters' queries compared to the heavily loaded one.

By the normalized bandwidth, we mean the bandwidth of a node is mapped to a value in $[0, 1]$ by dividing that over the maximum bandwidth in the system. The maximum bandwidth is considered as a system-wide constant parameter of the protocol. The $+1$ in the denominator of Equation 1 is to prevent division by zero when node i has not been selected as a replica yet (i.e., $rpLoad_i = 0$). $UV_{i,t}$, hence, is taking a value in $[0, 1]$. The higher the $UV_{i,t}$ is, the higher is the utility node i provides in the replication process.

Replication with *Pyramid*: Having the aggregated utility state of the system, each data owner is able to invoke the *Pyramid* algorithm when it needs replication for its data. As the result, *Pyramid* determines the (IP) address of replicas in a fully decentralized manner considering the utility- and locality-awareness of the replication. The data owner then shares its list of replicas on the decentralized aggregation scheme, which makes them publicly available for its data requesters. Inputs to *Pyramid* are the utility states of the nodes in the system, the replication degree of the data owner, and the information about the landmarks. *Pyramid* first distributes the replication degree among the regions of the system, where each region receives a sub-replication degree. For each region, *Pyramid* then models the utility- and locality-awareness of replication with Integer Linear Programming (ILP), solves it and finds the placement of

replicas, accordingly. The challenge in designing *Pyramid* is that obtaining, storing, and operating on the utility vectors of all the nodes result in an asymptotic linear dependency of the communication, storage, and time complexities on the system capacity. Considering the data owner that executes *Pyramid* as a resource-constrained peer, these linear dependencies would cause performance degradation.

To improve the communication complexity, *Pyramid* is built upon the existence of an aggregation mechanism with $O(\log n)$ communication complexity. The aggregation scheme acts as a decentralized bulletin board, and enables each node to write its utility vector on it as well as to retrieve all the utility vectors of other nodes from it with $O(\log n)$ communication complexity. In this paper, we employ LightChain [55], as the underlying aggregation scheme. LightChain is a churn resilient DHT-based blockchain with $O(\log n)$ communication complexities on storing and retrieving the latest state a data object. Besides, LightChain distributes the storage of the utility vectors uniformly among the nodes, so no node is required to keep the entire blockchain database.

To improve the time and storage complexities, *Pyramid* squeezes the *original system* of size n nodes to a significantly smaller size system of size $\log n$ nodes that is called the **virtual system**. *Pyramid* provides an efficient many-to-one mapping functionality for each original node to find its corresponding virtual node. The utility vector of a virtual node represents the average of the utility vectors of all its corresponding original nodes. The set of utility vectors of all the virtual nodes is stored on the aggregation scheme and gets updated frequently by the original nodes. We call this set the *aggregated utility state of the system*. Each original node is able to read the latest aggregated utility state of the system as well as to update it, i.e., by integrating its own latest utility vector to the utility vector of its corresponding virtual node. Instead of operating on the individual utility vectors of the original nodes, which requires linear storage and time complexity in the system size, *Pyramid* receives the aggregated utility state of the system and operates solely on it. As we discuss in Section 7, mapping to the virtual system and operating on it results in the storage and time complexities of $O(\log n)$ and $O(\log^2 n)$ for *Pyramid*, respectively.

4 Details of *Pyramid*

4.1 Virtual System

As explained in Section 3, to perform the computation efficiently, *Pyramid* does not directly operate on the original system. Rather, it operates on the virtual system, which is the squeezed model of the original system. The virtual system has the same set of landmarks and regions as the original system. Except, multiple original nodes (i.e., nodes in the original system) are mapped into a single node of the virtual system. The size of the virtual system is denoted by vs_{size} and is assumed as a protocol parameter known by all the nodes. As we discuss later, we consider $vs_{size} = \log n$. The virtual nodes (i.e., nodes

in the virtual system) have an identifier length of $\lceil \log vs_{size} \rceil$ bits. All the original nodes that have $\lceil \log vs_{size} \rceil$ bits common prefix in their name IDs are represented by a virtual node. For example, assuming that $vs_{size} = 16$, virtual nodes have 4-bits identifiers, and all the original nodes with a name ID prefix of 0001 are mapped to the virtual node 0001. So, each original node is able to identify its corresponding virtual node by only taking the first $\lceil \log vs_{size} \rceil$ bits of its own name ID. We denote the associated virtual node of the original node j by *virtual(j)*.

4.2 Mapping to the virtual system

Mapping from the original system to the virtual system is done by the original nodes computing a utility vector for their corresponding virtual nodes in a fully decentralized manner by means of the underlying aggregation scheme of the system. The set of the utility vectors of the virtual nodes corresponds to the aggregated utility state of the system and is denoted by the table *UT*, which is named the *utility table of the system*. *UT* is a 3-dimensional table of size $|L| \times vs_{size} \times |UV|$. $UT_{l,i,t}$ represents the utility vector of the virtual node i in the region l of the virtual system within the tth time slot of *FPTI*. The utility vector of the virtual node i corresponds to the average of the utility vectors of all the original nodes that are mapped to it. *UT* keeps the utility vector of virtual node i as the number of its corresponding original nodes as well as the summation of their utility vectors. In this way, while the average utility vector corresponding to each virtual node is efficiently computable, the number of corresponding original nodes to each virtual node is also known. As we explain later, *Pyramid* employs such information on replica placement. *UT* is shared among the original nodes and maintained in a fully decentralized manner using the underlying aggregation scheme (i.e., LightChain blockchain [55]). *UT* is reset to zero at the beginning of each cycle of *FPTI*. Each online original node j that belongs to the region l of the original system and has free storage space, updates *UT* once during each cycle of *FPTI*. The update is done by including the updated utility vector of the original node j in the average utility vector of its corresponding virtual node, i.e., $UT_{l,virtual(j)}$. The underlying aggregation scheme also enables each node to efficiently retrieve the latest state of *UT* on demand.

4.3 *Pyramid* Algorithm

Inputs and outputs: *Pyramid* is represented by Algorithm 4.1, and is executed by a data owner to determine its replicas. The inputs to *Pyramid* are the set of landmarks' features (i.e., L), the replication degree (i.e., r), and the utility table of the system (i.e., *UT*). By the landmarks' features, we mean their (IP) addresses and pairwise latencies, which are assumed as public static information of the system. As the output, *Pyramid* returns *oRepSet*, which is the set of identifiers of replicas in the original system.

Algorithm 4.1: *Pyramid*

Input: Set of landmarks L, replication degree r, utility table of system UT

Output: Set of replicas in orginal system $oRepSet$

// System-wide distribution of replicas

1 $R = \text{SWD}(L, r)$;

2 **for** *each region $l \in L$* **do**

   // Computing the time slot coverage weights

3 $W_l = \text{TCWD}(UT_l)$;

   // Replicas placement in region l of virtual system

4 $vRepSet = vRepSet \ \cup \text{RWD}(UT_l, R_l, W_l)$;

5 **for** *each virtual replica $vRep \in vRepSet$* **do**

   // Finding best original node for virtual replica $vRep$

6 $oRep = searchForUtility(vRep)$;

7   add $oRep$ to $oRepSet$;

8 publish $oRepSet$ on the aggregation scheme;

System-Wide Distribution of replicas (SWD) (Algorithm 4.1, Line 1): On receiving the inputs, *Pyramid* distributes the replication degree among the regions of the system using SWD [13]. Given the set of landmarks and the replication degree, SWD returns the set of sub-replication degrees, denoted by R where R_l denotes the number of replicas that should be placed in the region l of the system considering only the locality-awareness of replicas. We skip the details of SWD for sake of space, as the details of SWD are not required for understanding our proposed *Pyramid*. The interested readers are referred to [13] for details about SWD.

Time slots Coverage Weight Distribution (TCWD) (Algorithm 4.1, Line 3): The utility of the virtual nodes may not be distributed uniformly among all the time slots. Some time slots may be covered with the majority of high-utility virtual nodes, while the rest may be covered by only a few high-utility ones, or even left uncovered. We call such time-slots that are covered with only a few nodes or even no node as *poorly covered time slots*. To maximize the utility of replicas during the poorly covered time slots, *Pyramid* identifies the poorly covered time slots of each region by assigning a weight to each of its time slots. The weight assignment is done by invoking the TCWD function of *Pyramid*. Invoking TCWD on the utility table of region l (i.e., UT_l) results in obtaining the vector W_l of the utility coverage weights for that region. $W_{l,t}$ is a number in $[0, 1]$ and represents the utility coverage weight of the tth time slot in region l of the system, and is computed by Equation 2. In this equation, $count_{l,t}$ is the number of virtual nodes of region l that their utility value at the tth time slot is less than or equal to the average utility value. The average is taken over the entire utility table of region l. A larger $W_{l,t}$ denotes a lower utility coverage of the tth time slot of region l.

$$W_{l,t} = \frac{count_{l,t}}{\sum_{t=1}^{|UV|} count_{l,t}} \qquad (2)$$

Region-Wide Distribution of replicas (RWD) (Algorithm 4.1, Line 4): Once the time slot utility coverage weights for the region l of the virtual system are determined, *Pyramid* invokes the RWD function. RWD determines the placement of replicas in region l of the virtual system considering the utility- and locality-awareness of replicas. To place the replicas, RWD constructs an ILP model that is represented by Equations 3-8. The presented ILP model is for region l of the virtual system, and hence l is constant. The only decision variables in this ILP are X and Y, and the rest are constant scalars. The decision variable X is a three-dimensional binary table of size $vs_{size} \times vs_{size} \times |UV|$. $X_{i,j,t} = 1$ if the ILP assigns the virtual node i as the corresponding replica for the virtual node j during the tth time slot of *FPTI*, otherwise, $X_{i,j,t} = 0$. Y is a one-dimensional decision variable of size vs_{size}. $Y_i = 1$ if the ILP decides to place a replica on the virtual node i, otherwise $Y_i = 0$. C is a three-dimensional table of the same size as X that is constructed by the RWD function. $C_{i,j}$ represents the common prefix length between the name IDs of the virtual nodes i and j. Name ID of the virtual node i corresponds to the binary representation of i in $\lceil \log vs_{size} \rceil$ bits. Since *Pyramid* operates on a locality-aware overlay, $C_{i,j}$ conveys an approximated inverse quantification of the latency between the nodes i and j in the virtual system, i.e., a higher $C_{i,j}$ value implies a lower latency.

$$\max \sum_{t=1}^{|UV|} \sum_{i=1}^{vs_{size}} \sum_{j=1}^{vs_{size}} X_{i,j,t} C_{i,j} UT_{l,i,t} W_{l,t} \quad \text{s.t.} \qquad (3)$$

$$\forall t \in [1, |UV|], i, j \in [1, vs_{size}] \quad Y_i \geq X_{i,j,t} \qquad (4)$$

$$\forall t \in [1, |UV|], i \in [1, vs_{size}] \quad \sum_{t=1}^{|UV|} \sum_{j=1}^{vs_{size}} X_{i,j,t} \geq Y_i \qquad (5)$$

$$\forall t \in [1, |UV|], j \in [1, vs_{size}] \quad \sum_{i=1}^{vs_{size}} X_{i,j,t} = 1 \qquad (6)$$

$$\sum_{i=1}^{vs_{size}} Y_i = R_l \qquad (7)$$

$$\forall t \in [1, |UV|], i, j \in [1, vs_{size}] \quad Y_i \in \{0,1\}, \quad X_{i,j,t} \in \{0,1\} \qquad (8)$$

Equation 3 shows the objective function of the *Pyramid*'s ILP that aims at maximizing both the utility- and locality-awareness of replicas. By maximizing the locality-awareness, we mean maximizing the summation of the name IDs' common prefix length between the data requesters and their corresponding replicas, which corresponds to minimizing the overall access delay of replicas. In Equation 3, when $X_{i,j,t} = 1$, virtual node j experiences an access delay inversely

11

proportional to $C_{i,j}$, while benefits from the (maximum) utilization proportional to $UT_{l,i,t}$. We consider the contribution of this replica assignment (i.e., virtual node i as the corresponding replica of the virtual node j) to both utility- and locality-awareness of the replication proportional to $C_{i,j} \times UT_{l,i,t}$ at every time slot t. To lead the ILP on maximizing the utility of the selected replicas at each time slot t, in the objective function, we project the utility of each virtual node with the associated utility coverage weight in that time slot, i.e., $W_{l,t}$. In this way, the ILP prioritizes to select the high-utility virtual nodes that also cover the poorly covered time slots, and improves the utility-awareness of replicas per time slot.

Equations 4 and 5 represent the replica assignment constraints of the *Pyramid*'s ILP. Equation 4 implies that a virtual node j should be assigned to the virtual node i as its corresponding replica during the tth time slot (i.e., $X_{i,j,t} = 1$) only if node i itself is selected as a replica (i.e., $Y_i = 1$). Equation 5 says that if virtual node i is selected as a replica (i.e., $Y_i = 1$), then it should be the corresponding replica of at least one other virtual node during at least one time slot.

Equation 6 represents the data requester constraint of the *Pyramid*'s ILP. It implies that at each time slot t, each node should be assigned to exactly one replica. This is done to lead the ILP towards finding the best replica that maximizes the objective function for the node. At each time slot t and for each virtual node j the summation of $X_{i,j,t}$ values over all i values determines the number replicas that node j is benefiting from during that time slot, which should be exactly one replica considering this constraint.

Equation 7 shows the constraint on the sub-replication degree of region l. The summation in this equation corresponds to the overall number of permissible replicas that the data owner grants *Pyramid* to place in the region l of the system, which should be exactly equal to the sub-replication degree of the region, i.e., R_l.

Equation 8 shows the constraint on the legitimate values of the decision variables. That is, the only values that elements of X and Y decision variables can take are either 0 or 1.

RWD solves the described ILP model and determines the placement of replicas in each region l of the virtual system. The identifiers of selected virtual nodes by RWD as replicas are collected into the *vRepSet*, which contains the union of the identifiers of replicas in all regions of the virtual system.

Virtual to original system mapping of replicas (Algorithm 4.1, Lines 6-8): Each virtual replica identifier in *vRepSet* corresponds to the name ID prefix of several nodes in the original system. Although replicating on any of these nodes satisfies the locality-awareness goal of *Pyramid*, not all of them may be suitable considering the utility-awareness goal. This follows the fact that the name IDs are assigned based on the locality information of the nodes, and do not reflect any utility attribute of them. Additionally, contacting all the original nodes with a name ID corresponding to a prefix and picking the one with the best utility requires linear time and communication complexities, and is not a scalable solution. To efficiently map a virtual replica to an original node

Strategy	Availability-Awareness	Locality-Awareness	Utility-Awareness
Replication on path [22, 31–33]	Reactive	✗	✓
Virtual nodes [34]	Reactive	✗	✓
Replication on neighbors [29, 48]	Reactive	✗	✓
Randomized [35–42]	Proactive	✗	✗
Power-of-choice [56, 57]	Proactive	✗	✓
Cluster-based [43, 44]	Proactive	✗	✓
Correlation-based [23, 45]	Proactive	✗	✓
GLARAS [13]	–	✓	✗
Pyramid	**Proactive**	✓	✓

Table 1: Comparison of various decentralized replication algorithms

with desirable utility, we develop a modified version of the search for name ID protocol [12], which we call the *search for utility*. Given a virtual replica identifier $vRep \in vRepSet$, our search for utility protocol traverses at most α original nodes with the name ID prefix of $vRep$, and returns the one with the maximum utility score as the corresponding original node to $vRep$. The utility score of each original node i corresponds to the second norm of its utility vector (i.e., $\|UV_i\|$). We assume α is a system-wide constant parameter of the search for utility protocol. We denote the output of search for utility by $oRep$, which is the corresponding original node of the virtual replica $vRep$. On receiving $oRep$ from search for utility, *Pyramid* adds it to the set of replicas in the original system, i.e., $oRepSet$.

Once the mapping to the original system is done for all the virtual replicas, *Pyramid* publishes the $oRepSet$ on the underlying aggregation scheme. This enables the data requesters to query the aggregation scheme with the identifier of the data owner and retrieve its set of replicas.

5 Related Works

5.1 Decentralized Availability-Aware Replication

Reactive replication first performs an initial placement of replicas, and then the replicas are frequently probed to detect the failure events. A replica that is not answering the probe message in a certain while is presumed as failed, and is substituted by a newly placed one [37]. Replication on path, virtual nodes, and replication on neighbors are the well-known reactive replication mechanisms. In replication on path [22, 31–33], nodes piggyback their query load on the search messages they route or initiate. The replication on search path then reacts to the failure of replicas by creating a new replica and relocates the existing replicas upon detection of a new query traffic hub. In the virtual nodes approach [34], several peers are mapped to a single virtual node on the DHT overlay. The corresponding peers of the same virtual node all keep the same set of data objects. Virtual nodes approach reacts to the replication load of the peers by frequently monitoring their loads, splitting the heavily loaded groups into two, and re-distributing the data objects among them. In replication on neighbors [29, 48], each node frequently checks its neighbors to find better ones in terms of replication load and availability, and relocates its replicas to its more available

and loosely loaded neighbors. Relocating replicas upon failure is the common drawback of the reactive replication [20]. Each reactive relocation applies a communication overhead to the system that is linear in the size of the replicated files.

Proactive replication aims at providing an average availability of replicas for a long period of time, for example, a few months. At the end of the scheduled period, a proactive mechanism should re-evaluate the placement of replicas based on the new state of the system and relocate the replicas if needed. Hence, proactive replication also does replica relocation but at a drastically slower pace. Power-of-choice is an enhanced version of randomized replication where the data owner randomly selects two replica candidates and replicates on the least loaded one [56,57]. Cluster-based replication [43,44] groups the nodes based on features like availability pattern, replication load, and bandwidth, and distributes the replication degree among the clusters. In correlation-based replication [23, 45], the replication is done on the pairs of anti-correlated nodes, i.e., pairs of nodes where the unavailability of one implies the availability of the other one with a high probability.

5.2   Decentralized Locality-Aware Replication

In [13] GLARAS is proposed and experimentally shown that it is the best in locality-awareness compared with the existing decentralized locality-aware replication algorithms that are applicable on Skip Graphs. Although GLARAS is scalable with the ILP size of $O(\log n)$, it does not consider the availability and utility of the nodes in replication. We skip surveying the other existing locality-aware replication algorithms for the sake of space and refer the interested readers to [12,13] for a detailed discussion of locality-aware replication schemes.

Table 1 summarizes a comparison between the existing decentralized replication algorithms and our proposed *Pyramid*. In this table, we mark an algorithm as utility-aware if it considers the bandwidth of the nodes in replica placement decision making. Compared to the existing solutions, *Pyramid* is the only proactive and fully decentralized replication algorithm for Skip Graph DHTs that provides utility- and locality-awareness of replicas simultaneously.

6   Simulation Setup

We extended the Skip Graph simulator, SkipSim [49] by adding the aggregation functionality. We also enabled SkipSim to consider both the utility and locality of the nodes in scenarios with heterogeneous bandwidth and storage capacity of the nodes.

Churn model: Among the existing churn models, the Bittorrent-based models in [19] are well-studied and parametrically clearer than the rest, e.g., [30, 58, 59]. We hence implemented their churn model in SkipSim with Weibull distributions [60] for the session length and the inter-arrival time. In our implementation, all nodes follow the same Weibull-based session length distribution.

However, to simulate different cliques of nodes with respect to the availability behavior, nodes of different regions follow distinct Weibull-based inter-arrival time distributions. Following our definition of regions in Section 2, this implies that nodes that are in the proximity of each other in the underlying network show a similar churn behavior. In our implemented churn model, considering the entire system, on average each node has a 2.7 hours of session length followed by an offline period of 2.8 hours.

Bandwidth and Storage Capacity: In our implementation, the bandwidth of the nodes follows an exponential distribution with an average of $2Mbps$. We extracted the shape of distribution from [61], and adapted its scale with the typical average bandwidth provided by the service providers for the household customers [62]. The storage capacity of the nodes is drawn from a uniform distribution in the range of $[1, 3]$. We found this range as the one that quickly makes nodes out-of-storage by being selected as replicas of multiple data owners under randomized replication of 10 data owners with replication degree of 14. Hence, we establish our simulation setups on this range as a challenging range to decide on the replica placement for all the algorithms of interest. In our simulation, the storage capacity of a node denotes the maximum number of replicas that it can take from different data owners, e.g., a storage capacity of 2 means that the node can be the replica of two data owners simultaneously. Once a node reaches its storage capacity, it simply stops sharing its utility vector as well as taking any further replication duty.

Topologies: We generated 5 random topologies where each topology corresponds to a distinct distribution of the nodes in the Skip Graph overlay as well as their overlay connectivity. Each topology consists of 4096 nodes, and 10 randomly chosen data owners. In SkipSim, one simulation step corresponds to an hour. *FPTI* and *ts* were set to a day and an hour, respectively. Each topology was simulated for a lifetime of 3 months. We spot a one-week learning phase for the data owners to learn the underlying utility behavior of the nodes. As we discuss in the followings the learning is done via aggregation and piggybacking. The data owners replicate at the end of this one-week learning period.

Replication algorithms: We implemented our proposed *Pyramid* in Skip-Sim. Additionally, for the sake of comparison, we also implemented GLARAS, randomized replication, power-of-choice, cluster-based replication, and correlation-based replication with the implementation details provided in Section 5. For our proposed *Pyramid*, SkipSim performs the aggregation of utility vectors every 24 time slots (i.e., hours). For the other algorithms, SkipSim provides $O(n_o^2)$ random searches per time slot, where n_o corresponds to the number of online nodes during that time slot. These random searches enable data owners to benefit from the piggybacked utility information of the nodes. By random searches, we mean the search initiator and the search target are chosen randomly among the set of online nodes. Each node on a search path piggybacks its utility vector to the search message, while it receives the updated piggybacked utility vectors of the previous nodes on the same path.

Virtual System Size: Based on our simulations for different values of the virtual system size in different system capacities, we found $vs_{size} = 1.33 \log n$

as a proper trade-off between the running time efficiency and performance of *Pyramid* in providing utility- and locality-awareness of replicas, which is consistent with our previous work [13]. Hence, during the simulations for the system capacity of 4096 nodes, we set $vs_{size} = 16$. Also, we simulated for different values of α in the search for utility procedure and found $\alpha = 3$ as a proper trade-off between the communication overhead of *Pyramid* and its performance in providing utility- and locality-awareness of replicas in system capacity of 4096 nodes. We skip the details for the sake of the page limit.

6.1   Algorithms used for comparison

For the sake of comparison with *Pyramid*, we choose the existing decentralized and proactive replications that are applicable to our system model, i.e., they do not change the operational complexity of the Skip Graph as well as its topology, do not require a centralized party, and are computationally efficient for a data owner peer to execute independently. For these algorithms, we implement piggybacking to enable each node disseminating its utility vector across the system efficiently. Also, as detailed in the followings, for all these algorithms except the randomized replication, we use the same utility scoring as our proposed *Pyramid*. Each node in the system piggybacks its utility vector and (IP) address on the search messages it routes or initiates so that the receivers of the search messages can benefit from its utility vector. It is worth to mention that *Pyramid* does not require the piggybacking and solely operates on the aggregated utility table of the nodes that is provided by the underlying aggregation scheme (i.e., LightChain blockchain [55]). We discuss the asymptotic complexity of interacting with LightChain later in this paper. We also choose GLARAS as the best existing fully decentralized locality-aware replication for Skip Graph, and implement it exactly as specified in [13]. Followings are the implementation details of the proactive replication algorithms.

Randomized Replication [35]: The data owner chooses its replicas uniformly from the set of obtained addresses, pings each, and replicates on the online one that has free storage capacity. The data owner does this procedure repeatedly until it satisfies the replication degree.

Power-of-choice [56,57]: Our implementation of power-of-choice is similar to that of the randomized replication, except that to place a replica, the data owner chooses two nodes randomly, and replicates on the one that has the higher second norm of the utility vector.

Cluster-Based Replication [43,44]: In our implementation of the cluster-based replication, the data owner performs an r-mean clustering [63] of the nodes based on their piggybacked utility vectors, where r is the replication degree of the data owner. The data owner then replicates on the nodes with the highest second norm of the utility vector from each cluster.

Correlation-Based Replication [23, 45]: In our implementation of the correlation-based replication, the data owner aims to find $\frac{r}{2}$ pairs of nodes with the minimum correlation. To do this, the data owner first finds $\frac{r}{2}$ nodes with the maximum value of the second norm of the utility vector from its set of
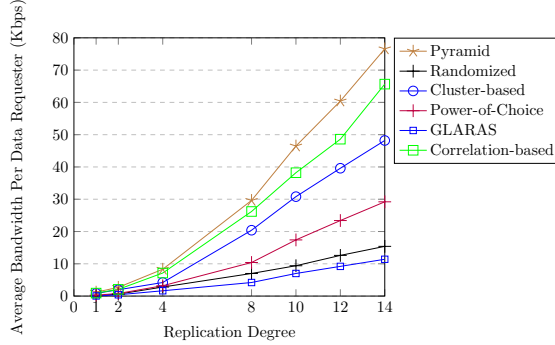
Figure 1: The performance of the replication algorithms with respect to the utility-awareness of replicas. The X-axis corresponds to the replication degree. The Y-axis represents the average available bandwidth of replicas per data requester at each time slot.

piggybacked utility vectors of nodes. For each of those nodes and in the same set of piggybacked utility vectors, the data owner finds the node with the highest fraction of the second norm of the utility vector over the correlation value, and replicates on both nodes. We measure the correlation between two nodes as the dot product of their utility vectors. The higher the dot product is, the higher the two nodes are correlated with each other.

7 Performance Results

7.1 Utility-Awareness

As the utility-awareness metric, we measure the average available bandwidth per data requester, where the average is taken over replicas of all the data owners, during all the simulation time slots, and over the topologies under simulation. For a single data requester, the corresponding replica with respect to a data owner is the one with the minimum RTT to it among the replicas of the data owner. We compute the average available bandwidth for each data requester node by dividing its closest replica's bandwidth by the number of corresponding data requesters of that replica. Figure 1 represents the utility-awareness performance of the replication algorithms. *For a specific replication algorithm, a point (x, y) in Figure 1 is interpreted as "placing x replicas for a single data owner using this algorithm results on the average available bandwidth of y Kbps per data requester node"*. As shown in the figure, by enforcing the replicas to be placed solely with respect to the latency distribution of the nodes in the underlying network, GLARAS performs even worse than the randomized distribution of the replicas in providing utility-awareness. Comparing the bandwidth of the two randomly chosen replication candidates, and replicating on the better one results in the power-of-choice strategy to outperform the randomized replication especially in the larger replication degrees. The performance gap among *Pyramid*, cluster-based and correlation-based, and the rest stems from the fact that these three replication algorithms have utility-based scoring. As the replication
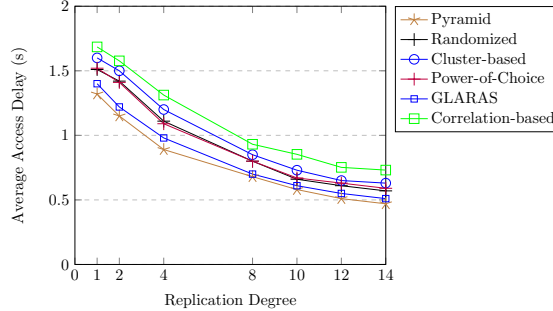
Figure 2: The performance of the replication algorithms with respect to the locality-awareness of replicas. The X-axis corresponds to the replication degree. The Y-axis shows the average latency between each data requester and its closest online replica at each time slot.

degree of the data owner increases, and the cluster-based replication divides the nodes into a larger number of cliques, the number of cliques with low availability also increases increases. This justifies the poor performance of the cluster-based replication compared to the correlation-based and *Pyramid* especially in higher replication degrees. Nodes in our simulations have the average online and offline duration of 2.7 and 2.8 hours, respectively. Mapping this to the uniform failure model results in the uniform failure probability of about 0.5 for each node at each time slot. This results in the appearance of many anti-correlated pairs of nodes in the system, which are exploited by the correlation-based replication algorithm, and makes it as the best among the existing applicable decentralized solutions on the Skip Graph. **Compared to the best existing solutions, our proposed *Pyramid* improves the utility of replicas with a gain of about 1.2 times on average.**

7.2 Locality-Awareness

Figure 2 shows the performance of replication algorithms with respect to the locality-awareness. At each time slot, we measure the locality-awareness as the average access latency between each data requester node and its closest online replica, which is performed for the replicas of every data owner at each time slot. We report the average access delay of replication per data owner at each time slot and over all the topologies under simulation. By purely considering the utility of the nodes, correlation-based replication that performs the second best in the utility-awareness, is the weakest one in the locality-awareness. As explained in Section 6, in our simulation setups, nodes with similar locality features show a similar availability behavior. Hence, by clustering the nodes into cliques based on their utility vectors -that also contains their availability behavior- and placing a replica in each clique, the cluster-based replication performs better than the correlation-based in locality-awareness. Cluster-based replication, however, is outperformed by the randomized and power-of-choice replications in locality-awareness. Since once the replica of a clique (temporarily) departs the system, the data requesters of that replica should benefit from the available replicas in the nearby cliques, which increases their average access delay. On the contrary,

18

since randomized and power-of-choice replications choose the replicas uniformly with respect to their latency distribution, any subset of their available replicas follows a uniform distribution with respect to the locality. Among the existing solutions, by purely considering the locality-awareness of the replicas, GLARAS provides the minimum average access delay. GLARAS, however, distributes the replicas totally regardless of the utility behavior of the nodes. This results in time slots where crowded regions of the system remain without an available replica and are imposed to use other regions' replicas, which results in an increase on the average access delay of the replication. In contrast to GLARAS, *Pyramid* aims at minimizing the locality-awareness per time slot while considering the utility of replicas. **Compared to GLARAS that acts as the best existing locality-aware replication, our proposed *Pyramid* reduces the average access delay of replicas under churn with a gain of about 1.1 times on average.** Concerning the scalability, we also simulated *Pyramid* in system sizes of 1024 and 2048 nodes and observed consistent performance results with the 4096 nodes setup in both the utility- and locality-awareness of replicas. We exclude the details of scalability results for the sake of space.

7.3 Asymptotic Complexity of *Pyramid*

Memory Complexity: Having the system capacity of n nodes, the data owner that executes an instance of *Pyramid* only needs to maintain the the set of landmarks' features with the memory complexity of $O(\log n)$, the replication degree with the memory complexity of $O(1)$, and the utility table of the system with memory complexity of $O(|L| \times vs_{size} \times |UV|)$. Considering $vs_{size} = \log n$ and $|UV|$ as a constant parameter of the protocol, the memory complexity of *Pyramid* is $O(\log^2 n)$.

Time Complexity: The asymptotic running time complexity of SWD is $O(\log^2 n)$ [13]. TCWD function of *Pyramid* iterates twice over a table of size $vs_{size} \times |UV|$, which causes a time complexity of $O(\log n)$. The ILP part of *Pyramid* that is represented by Equations 3-8 has both an objective function size and the number of constraints of $O(|UV| \times vs_{size}^2)$. Considering that $|UV|$ is a constant parameter, the ILP size of *Pyramid* is $O(\log^2 n)$ in terms of the size of the objective function and the number of constraints. Running on Intel i5 2.60 GHz CPU and 8 GB of RAM and using lpsolve [64] to solve the ILP model of *Pyramid*, a single execution of *Pyramid* in a 4096 nodes system takes the average computation time of about 3 minutes to determine the placement of 14 replicas for a single data owner.

Communication Complexity: By the communication complexity, we mean the round complexity (i.e., the number of rounds of the protocol), the message complexity (i.e., the total number of the transmitted messages), and the bit complexity (i.e., the total number of transmitted bits), altogheter. In our system model, a single node updates the utility table of the system by reporting its utility vector on the underlying aggregation scheme (i.e., LightChain) once during every cycle of *FPTI* (i.e., once every 24 hours based on our simulation setup), which causes the communication complexity of $O(\log n)$ [55]. As an in-

put to *Pyramid*, a data owner requires retrieving the latest state of the utility table of the system, which causes a communication complexity of $O(\log n)$ [55]. Having the replication degree of r, *Pyramid* performs r searches for utility to map the replicas from the virtual system to the original system. Each search causes a communication complexity of $O(\log n)$ [12]. Hence, placing r replicas for a single data owner using *Pyramid* costs an $O(r \times \log n)$ communication complexity.

8 Conclusion

To improve the response time of Skip Graph-based P2P cloud storage systems under churn, we proposed *Pyramid*, which is the first fully decentralized utility- and locality-aware replication algorithm for Skip Graph overlays. In replica placement, *Pyramid* considers the heterogeneity of the nodes with respect to their latency distribution in the underlying network, availability behavior, storage capacity, and bandwidth. *Pyramid* enables a data owner to place its replicas in a fully decentralized manner, and in a way that the utility- and locality-awareness of replicas are achieved under churn. *Pyramid* is utility-aware as it maximizes the average available bandwidth of replicas per time slot. Additionally, *Pyramid* is locality-aware as it minimizes the average response time of replicas. Our simulation results show that compared to the best existing replication algorithms that are applicable on a Skip Graph-based P2P overlay, *Pyramid* improves the utility- and locality-awareness of replicas with a gain of about **1.2 and 1.1 times**, respectively.

Acknowledgement

References

[1] Q. He, Z. Li, and X. Zhang, "Study on cloud storage system based on distributed storage systems," in *ICCIS*. IEEE, 2010.

[2] K. Hwang, S. Kulkareni, and Y. Hu, "Cloud security with virtualized defense and reputation-based trust mangement," in *DASC*. IEEE, 2009.

[3] K. Xu, M. Song, X. Zhang, and J. Song, "A cloud computing platform based on p2p," in *ITIME*. IEEE, 2009.

[4] J. Aspnes and G. Shah, "Skip graphs," *ACM TALG*, 2007.

[5] A. Goyal, S. Batra, N. Kumar, G. S. Aujla, and M. S. Obaidat, "Adaptive skip graph framework for peer-to-peer networks: Search time complexity analysis," in *GLOBECOM*. IEEE, 2018.

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

[7] S. Batra and A. Singh, "A short survey of advantages and applications of skip graphs," *IJSCE*, 2013.

[8] A. Singh and S. Batra, "P-skip graph: An efficient data structure for peer-to-peer network," in *IDC*.   Springer, 2015.

[9] T. Shabeera, P. Chandran, and S. Kumar, "Authenticated and persistent skip graph: a data structure for cloud based data-centric applications," in *ACM CSS*, 2012.

[10] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Awake: decentralized and availability aware replication for p2p cloud storage," in *Smart Cloud*.   IEEE, 2016.

[11] ——, "Locality aware skip graph," in *ICDCSW*.   IEEE, 2015.

[12] ——, "Laras: Locality aware replication algorithm for the skip graph." in *NOMS*.   IEEE, 2016.

[13] ——, "Decentralized and locality aware replication method for dht-based p2p storage systems," in *FGCS*.   Elsevier, 2018.

[14] S. Taheri Boshrooyeh and Ö. Özkasap, "Guard: Secure routing in skip graph," in *IFIP Networking*, 2017.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM TON*, 2003.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IEEE/IFIP Middleware*.   Springer, 2001.

[17] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *EuroSys*.   ACM, 2009.

[18] K.-H. Yang and J.-M. Ho, "Proof: A dht-based peer-to-peer search engine," in *Web Intelligence*.   IEEE, 2006.

[19] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *SIGCOMM*.   ACM, 2006.

[20] M. Rahmani and M. Benchaiba, "A comparative study of replication schemes for structured p2p networks," in *ICIW*.   IARIA, 2014.

[21] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*.   Prentice-Hall, 2017.

[22] H. Shen, "An efficient and adaptive decentralized file replication algorithm in p2p file sharing systems," *IEEE TPDS*, 2010.

[23] A. Kermarrec, E. L. Merrer, G. Straub, and A. Van Kempen, "Availability-based methods for distributed storage systems," in *IEEE SRDS*, 2012.

[24] W. Sun, V. Simon, S. Monnet, P. Robert, and P. Sens, "Analysis of a stochastic model of replication in large distributed storage systems: A mean-field approach," *POMACS*, 2017.

[25] G. Rodolakis, S. Siachalou, and L. Georgiadis, "Replicated server placement with qos constraints," *IEEE TPDS*, 2006.

[26] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Optimal placement of replicas in trees with read, write, and storage costs," *IEEE TPDS*, 2001.

[27] X. Tang and J. Xu, "Qos-aware replica placement for content distribution," *IEEE TPDS*, 2005.

[28] J. Wu, L. Chen, X. Wang, G. Jiang, S.-k. Lam, and T. Srikanthan, "Algorithms for replica placement and update in tree network," *The Computer Journal*, 2017.

[29] H. Shen and C.-Z. Xu, "Locality-aware and churn-resilient load-balancing algorithms in structured peer-to-peer networks," *TPDS*, 2007.

[30] A. Datta and K. Aberer, "Internet-scale storage systems under churn–a study of the steady-state using markov models," in *P2P Computing*. IEEE, 2006.

[31] Y. Koizumi, K. Watabe, and K. Nakagawa, "Reduction of response time by data placement reflecting co-occurrence structures in structured overlay networks," *PLOS ONE*, 2018.

[32] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," in *Distributed Computing Systems*. IEEE, 2004.

[33] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "Auto placer: Scalable self-tuning data placement in distributed key-value stores," *ACM TAAS*, 2015.

[34] J. Paiva, J. Leitao, and L. Rodrigues, "Rollerchain: A dht for efficient replication," in *NCA*. IEEE, 2013.

[35] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "Relaxdht: A churn-resilient replication strategy for peer-to-peer distributed hash-tables," *ACM TAAS*, 2012.

[36] J. Su and D. Reeves, "Replica placement algorithms with latency constraints in content distribution networks," in *Technical Report, ACM*, 2004.

[37] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, "Performance evaluation of replication strategies in dhts under churn," in *ACM MUM*, 2007.

[38] A. Harwood and D. E. Tanin, "Hashing spatial content over peer-to-peer networks," in *ATNAC, University of Melbourne*, 2003.

[39] Z. Xiaosu, W. Xiaolin, and H. Hao, "Caching, replication strategy and implementions of directories in dht-based filesystem," in *ICPCA*. IEEE, 2011.

[40] J. Paiva and L. Rodrigues, "On data placement in distributed systems," *ACM SIGOPS*, 2015.

[41] T. Pitoura, N. Ntarmos, and P. Triantafillou, "Replication, load balancing and efficient range query processing in dhts," in *EDBT*. Springer, 2006.

[42] P. Knežević, A. Wombacher, and T. Risse, "Dht-based self-adapting replication protocol for achieving high data availability," in *SITIS*. Springer, 2009.

[43] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE TPDS*, 2011.

[44] A. Pace, V. Quema, and V. Schiavoni, "Exploiting node connection regularity for dht replication," in *SRDS*. IEEE, 2011.

[45] S. Le Blond, F. Le Fessant, and E. Le Merrer, "Finding good partners in availability-aware p2p networks," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2009.

[46] B. Meroufel and G. Belalem, "Managing data replication and placement based on availability," *AASRI Procedia*, 2013.

[47] J.-J. Wu, Y.-F. Lin, and P. Liu, "Optimal replica placement in hierarchical data grids with locality assurance," *Journal of Parallel and Distributed Computing*, 2008.

[48] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE TNSM*, 2018.

[49] "Skipsim," https://github.com/yhassanzadeh13/SkipSim, accessed: 2019-05-20.

[50] S. T. Boshrooyeh, Ö. Özkasap, and B. Akgün, "Distributed landmark placement in p2p networks," in *SiU*. IEEE, 2018.

[51] R. Jiménez, F. Osmani, and B. Knutsson, "Connectivity properties of mainline bittorrent dht nodes," in *P2P Computing*. IEEE, 2009.

[52] J. L. J. Laredo, P. A. Castillo, A. M. Mora, J. J. Merelo, and C. Fernandes, "Resilience to churn of a peer-to-peer evolutionary algorithm," *IJHPSA*, 2008.

[53] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems," in *ACM SIGCOMM*, 2005.

[54] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Interlaced: Fully decentralized churn stabilization for skip graph-based dhts," *arXiv preprint arXiv:1903.07289*, 2019.

[55] ——, "Lightchain: A dht-based blockchain for resource constrained environments," *arXiv preprint arXiv:1904.00375*, 2019.

[56] V. Simon, S. Monnet, M. Feuillet, P. Robert, and P. Sens, "Splad: scattering and placing data replicas to enhance long-term durability," *Technical Document, Inria*, 2014.

[57] ——, "Scattering and placing data replicas to enhance long-term durability," in *NCA*. IEEE, 2015.

[58] F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in p2p protocols," in *Web content caching and distribution*. Springer, 2004.

[59] D. Wu, Y. Tian, K.-W. Ng, and A. Datta, "Stochastic analysis of the interplay between object maintenance and churn," *Computer communications*, 2008.

[60] H. Rinne, *The Weibull distribution: a handbook*. Chapman and Hall/CRC, 2008.

[61] J. A. Pouwelse, P. Garbacki, D. Epema, and H. J. Sips, "A measurement study of the bittorrent peer-to-peer file-sharing system," Technical Report, Delft University, Tech. Rep., 2004.

[62] J. F. Kurose, *Computer networking: A top-down approach*. Pearson Education India, 2016.

[63] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE ITPIDJ*, 2002.

[64] "lpsolve5.5," http://lpsolve.sourceforge.net/5.5/, accessed: 2019-04-12.