### SUMMARY
To realize an information-centric networking, IPFS (InterPlanetary File System) generates a unique ContentID for each content by applying a cryptographic hash to the content itself. Although it could improve the security against attacks such as falsification, it makes difficult to realize a similarity search in the framework of IPFS, since the similarity of contents is not reflected in the proximity of ContentIDs. To overcome this issue, we propose a method to apply a locality sensitive hash (LSH) to feature vectors extracted from contents as the key of indexes stored in IPFS. By conducting experiments with 10,000 random points corresponding to stored contents, we found that more than half of randomly given queries return a non-empty result for the similarity search, and yield an accurate result which is outside the $σ$ confidence interval of an ordinary flooding-based method. Note that such a collection of random points corresponds to the worst case scenario for the proposed scheme since the performance of similarity search could improve when points and queries follow an uneven distribution.

**key words:** IPFS, similarity search, locality sensitive hash, Kademlia DHT

### 1. Introduction
IPFS (InterPlanetary File System) is a hypermedia protocol proposed by J. Benet, and is being developed by Protocol Labs, with the aim of realizing a global distributed file system over the Internet [4]. IPFS has several interesting features such as Git-like version control, blockchain-like security protocol, and efficient swarm-based resource sharing as in BitTorrent. One of the most notable features of IPFS is that the resources are accessed in a content-based manner, whereas HTTP, which is the basis of the WWW, is a typical location-based protocol. Because of these features, IPFS is expected to be applied to various application domains including smart contract and identity overlay networks.

In IPFS, each node is given a unique identifier (NodeID), and the index of the content holder (i.e., list of NodeIDs) is stored on a node whose NodeID is close to the identifier of the content (ContentID) which is obtained by applying an appropriate cryptographic hash such as SHA to the content. Access to the indexes is realized through a message routing in $O(\log N)$ time for $N$ nodes [18]. Therefore, if the content is modified even slightly, the resulting ContentID will have a completely different value, which guarantees the robustness against tampering as in digital signatures, but does not guarantee the similarity of contents stored in the same and/or nearby nodes. The hash value is also used to check the authenticity of acquired file blocks as in BitTorrent. On the other hand, since there are many practical situations in which we should keep the same ContentID even if the content is updated as in the case of web pages, IPFS provides an option called IPNS (InterPlanetary Name System) which associates content with NodeID of the content owner, where the access for an IPNS address is done with /ipns/prefix to emphasize the use of IPNS.

In this paper, we consider the problem of content retrieval in IPFS. As described above, IPFS allows us to retrieve the index to the content holder by specifying ContentID we want to retrieve. However, this method poses a dilemma such that the content to be retrieved must be in hand to calculate the ContentID. IPFS itself provides no specific search mechanism including keyword search, and to overcome such a situation, researchers proposed several search engines for IPFS. For example, in YaCy†, each node in IPFS obtains key-value pairs through crawling, so that each pair has a word in the document as the key and ContentID as the value, and merge them with key-value pairs obtained by the other nodes on a distributed hash table (DHT). Similarly, ipfs-search** aggregates these key-value pairs on a centralized server and performs an intensive document search using Elasticsearch***. PeARSearch**** is based on an idea similar to the keyword search in unstructured P2Ps such as Gnutella[21]. That is, each node locally creates a pod, which collects its favorite web pages, to create a collection of web pages tagged with keywords. Resulting collection of web pages is shared on the network and is updated by the users. Similarly, IPSE (InterPlanetary Search Engine) adds tags to each content to enable keyword search, where tagging is autonomously done by each user, with the aid of an incentive mechanism to reward tagging. Unfortunately, to the best of the author’s knowledge, there is no previous work concerned with the similarity search in IPFS except for centralized scheme based on Elasticsearch.

In this paper, we propose a similarity search scheme for IPFS. As mentioned above, IPFS applies a cryptographic hash to the content (e.g., to the text itself in the case of text files) to obtain a unique ContentID. Thus the similarity of contents is not reflected in the proximity of ContentIDs. On

---

*a) E-mail: fujita@hiroshima-u.ac.jp
DOI: 10.1587/transinf.2020EDP7198

---

| DOI: 10.1587/transinf.2020EDP7198 |
|-----------------------------------|
|DOI: 10.1587/transinf.2020EDP7198 |
the other hand, IPFS stores key-value pairs on DHT so that ContentID is the key and the index of the content is the value by default. However, in principle, it is possible to store any key-value pair, as long as it satisfies a certain format as in IPNS. In fact, [16] recently proposed an effective content retrieval scheme for conjunctive queries with the notion of result cache[22]. In this paper, we propose a method which uses hash value obtained by applying a locality sensitive hash (LSH) to feature vectors of media content as the key of key-value pairs. Although LSH has been widely used as a dimension reduction technique for image retrieval and data mining, to the best of our knowledge, this is the first proposal concerned with the application of LSH to the retrieval in a large-scale distributed storage such as IPFS. In addition, since IPFS intrinsically stores data across multiple computers in a distributed manner, it does make sense to develop a distributed similarity search scheme without involving indexes of the data to a single server, as in Elasticsearch.

Similarity search is a general technique which has been widely used in many important domains such as document retrieval and image retrieval. On the other hand, recent advances in the metric space embedding [2] enable us to make the vector space assumption valid and realistic, which leads to a more general framework for the similarity search. Along with this direction, we specifically focus on the nearest neighbor search in a high-dimensional vector space. Although we could extend it to several directions as a future work, which includes the k-nearest neighbor queries, which return k points close to the query point, and range queries, which return all data within a certain range, we merely consider the most basic query, which returns a point close enough to the query point. In other words, we evaluate the goodness of schemes in terms of the distance in the vector space.

The remainder of this paper is organized as follows. Section 2 reviews related work concerned with the similarity search in P2P systems. Section 3 describes an overview of IPFS protocol. Section 4 describes the basic idea of LSH. Section 5 explains the proposed similarity search scheme and Sect. 6 summarizes the results of evaluations. Finally, Sect. 7 concludes the paper with future work.

2. Related Work

Existing methods for the similarity search in P2P systems try to implement sophisticated data structures similar to kd-tree as an overlay network, while we could take another option to extend the matching condition in usual flooding based exact matching schemes so as to take into account the similarity condition [13]. The method in [9], [25] connects nodes holding similar contents by logical links to organize clusters in an overlay. Although Self-Chord proposed in [10] has exactly the same structure as the usual Chord overlay [24], it improves the efficiency of content search by employing a method in which multiple swarm agents patrol the overlay and conduct relocation of content IDs in such a way to organize clusters of similar contents. With this mechanism, when a content ID stored on a node is accessed by a user, the user could find IDs of other contents which are semantically similar to the accessed content on the same node. P-Grid (Peer-Grid) [1], which uses binary tree structure as the overlay topology, naturally realizes a similarity search based on the kd-tree [15]. As domain specific approaches, [5] focuses on the edit distance among keywords to generate several queries similar to a given keyword within a small edit distance, and [6] uses Gray code as a method to keep the proximity in embedding content IDs to the P2P overlay. However, there is a persistent suggestion such that structured P2P such as DHT should not be used for the similarity search in P2P due to the lack of flexibility in structured P2Ps [19].

Recently, there has been done a lot of work concerned with the decentralization of the LSH-based similarity search [27]. Kraus et al. [17] proposed nearbucket LSH which integrates LSH and cosine similarity metric into a content addressable network (CAN) to improve the network efficiency of similarity search. Haghani et al. [12] proposed similarity search schemes for Kademlia DHT by using the LHS with the notion of stable distribution, similar to our proposed method. The main contribution of Haghani et al. was how to properly map LSH hashes to the linear address space of Kademlia so as to cover the similarity of stored data by expanding the target for content retrieval from a single bucket to a collection of consecutive buckets. In other words, it becomes a useful alternative only if the mapping to the address space of Kademlia can be modified by the designer of the similarity search scheme. Karapiperis et al. [14] proposed a distributed similarity search engine called LSHDB, which supports parallelization of query processing by providing an abstract layer to hide the LSH mechanism from the user. Qi et al. [20] proposed a distributed LSH mechanism for service recommendations from multi-source data to protect user privacy and improve the scalability of recommendations in cross-cloud platforms. Chuag et al. [7] applied LSH to a distributed retrieval system for large scale and mobile radio networks (DSL) and showed that similar documents can be distributed to the same region by geography-based routing.

3. IPFS

IPFS is a P2P (Peer-to-Peer) system consisting of nodes with equal roles. IPFS nodes store IPFS objects, such as text and media files, in their local storage and transfer them to each other over the network. Access to those shared objects is conducted in a content-based manner, not on a location base with the network address of file servers. The IPFS protocol consists of several sub-protocols, overviewed as follows.

3.1 Transport Layer

Each IPFS node communicates with hundreds of nodes in the network in a continuous manner. IPFS supports several
transport layer protocols, which enables us to switch different protocols depending on the use case; e.g., data channel of WebRTC [26] for the browser connection and another high throughput protocol for LEDBAT [23]. In addition, if the underlying network layer is unreliable, we can use a secure protocol such as SCTP to ensure the reliable communication, and for the NAT traversal, it adopts the Interactive Connectivity Establishment (ICE) protocol. As for the security issues, it checks the integrity by using a hash checksum and the authenticity by HMAC (Hash-based Message Authentication Code) with the sender’s public key.

3.2 Kademlia Routing Protocol

In IPFS, query messages are routed to their specified nodes and objects through Kademlia DHT [18], which has the following characteristics:

1. Average number of hops to the destination is $O(\log N)$, where $N$ is the number of nodes in the network.
2. It exchanges few control messages between nodes with small overhead.
3. It achieves a high resilience against churn and attacks by prioritizing nodes to have a long stay time during the maintenance of the overlay.

In the following, the information necessary to contact node $u$, such as NodeID, IP address, and port number, is called node information of $u$ with node $u$.

Routing table in Kademlia is a two-dimensional array of $n$ rows and $k$ columns, and each cell in the table contains (at most) one node, where $n$ and $k$ are predetermined parameters. Let us call the most significant bit of NodeID the $0^\text{th}$ bit. The $i^{\text{th}}$ row of the routing table for node $u$ stores (at most $k$) nodes such that the $0^\text{th}$ to the $(i-1)^{\text{st}}$ bits are the same as $u$ and the $i^{\text{th}}$ bit is different. Suppose that node $u$ discovers a node $v$ and wants to store it in its local table. If the row in which $v$ should be stored is full, nodes in the row are checked for survival from head to tail, and if it finds a node $w$ which is not alive, then it replaces $w$ by $v$, and otherwise (i.e., if there is no such $w$), it keeps $v$ in the replacement cache, which will be used if all nodes in the table stops responding.

Messages used in Kademlia protocol are as follows:

- **PING**: For survival confirmation.
- **STORE(key, value)**: To store the given key-value pair in the receiving node.
- **FIND_NODE(key)**: The receiver selects the $k$ closest nodes to key from its table and returns them.
- **FIND_VALUE(key)**: The receiver returns the corresponding value when it has key.

In any case, the sender of a request assigns a random value $v$ to the message, so that the receiver of a reply can determine which request corresponds to the reply.

3.3 Procedure for Node Search

Let $\alpha$ be a parameter representing the parallelism of node search ($\alpha = 3$ for default). Let $v$ be the destination node of a query and $S$ be a local variable of source node $u$. At first, $u$ selects $\alpha$ closest nodes to $v$ from its table, puts them in $S$, and repeats the following steps until $S$ is no longer updated, where a timeout could be set for FIND_NODE to continue processing even if the receiver of the message has departed the system:

1. Send FIND_NODE($v$) to each node $w$ in $S$;
2. $w$ selects $k$ closest nodes to $v$ from its table and returns them to $u$; and
3. $u$ selects $\alpha$ closest nodes $v$ (from at most $\alpha k$ candidates) and lets them be new $S$.

By mapping resources to the same ID space as nodes, resource discovery (including the recovery of index with a specific key) can be realized in the same manner. We can improve the churn tolerance of the above scheme by storing replica of the resource at nearby nodes. In addition, we can reduce the search time while avoiding hotspots by each requester caching retrieved resources.

If node $u$ has several queries to be issued, the forwarding of all queries to their destination can be realized as follows: 1) prepare set $S_i$ for each query $q_i$, 2) send a set of queries to node $w$ instead of a single query, and 3) the receiver $w$ calculates the candidate for the next node for each query received from $u$ and returns the results to $u$. Note that although the above modification increases the time required for the local computation, it is much smaller than the transmission time of a message to the next node which is generally the magnitude of tens of milliseconds.

4. Revisit LSH

To make this paper self-contained, this section describes the basic idea of locality-sensitive hash, which plays a crucial role in the proposed similarity search scheme. Consider a (probabilistic) function $h$ from $m$-dimensional Euclidean space $\mathbb{R}^m$ to the set of real numbers $\mathbb{R}$. Function $h$ is called **Locality-Sensitive Hash (LSH)** if two points in close proximity are mapped to the same value with high probability. Although the above condition could be simply written so that: for any $u, v \in \mathbb{R}^m$,

$$d(u, v) \leq r \implies Pr(h(u) = h(v)) \geq p$$

using two parameters $r$ and $p$, since it does not represent what happens to distant points, it is common to refine the definition by using four parameters $r_1$, $r_2$, $p_1$ and $p_2$ satisfying $r_1 < r_2$ and $p_1 > p_2$, in the following manner: for any $u, v \in \mathbb{R}^m$,

$$d(u, v) \leq r_1 \implies Pr(h(u) = h(v)) \geq p_1 \text{ and }$$
$$d(u, v) \geq r_2 \implies Pr(h(u) = h(v)) \leq p_2.$$
where function $h$ satisfying the above condition is said to be $(r_1, r_2, p_1, p_2)$-sensitive.

4.1 Simple LSH Based on $p$-Stable Distribution

The first LSH discovered by Datar et al. [8] is based on the notion of stable distribution. Let $p$ be a real number greater than or equal to zero, and consider $m+1$ random variables $X_1, X_2, \ldots, X_m$ and $X$ which are independently selected from $\mathbb{R}$ according to a probability distribution $\Omega$. Then $\Omega$ is said to be $p$-stable if $\sum c_i X_i$ has the same distribution with $(\sum |c_i|^p)^{1/p} X$ for any $m$ real numbers $c_1, c_2, \ldots, c_m$. It is known that normal distribution is 2-stable. Note that the above definition implies: 1) the distribution of the linear sum of $m$ random variables is a constant multiple of the original distribution, and 2) the constant is equal to the $L_p$ norm of the coefficient vector.

Consider a vector $a$ of $m$ random variables $a_1, a_2, \ldots, a_m$ selected from $\mathbb{R}$ according to a normal (i.e., 2-stable) distribution $\Omega$ and a real number $b$ which is uniformly selected from closed interval $[0, W]$. Let $h$ be a function from from $\mathbb{R}^m$ to $\mathbb{R}$ defined as follows

$$h(v) = \left[ \frac{a \cdot v + b}{W} \right]$$

where $W$ is a positive real. Since normal distribution is 2-stable, the projection of the difference vector of two vectors $u$ and $v$ on vector $a$ also follows a normal distribution, which implies that the resulting $h$ is certainly LHS. Note that the idea of projecting a set of points to a certain vector has been used in the design of kd-tree which is widely used for the efficient neighborhood search. However, there are two differences in kd-tree: 1) it is restricted to the bi-partitioning of the point set, and 2) the target of projection is limited to axial vectors.

4.2 Refinement of Accuracy

In actual similarity search, it is rare to use the above function $h$ as an LHS as it is, and we generally combine several $h$’s with different $a$, $b$ to generate a more accurate LHS [8]. The intention of combination is to: 1) increase the probability of nearby points being mapped to the same value, and 2) decrease the probability of remote points being mapped to the same value. More concretely, we prepare a function $g$ which outputs a $k$ dimensional vector $g(v)$ by using $k$ different $h$’s as

$$g(v) = (h_1(v), h_2(v), \ldots, h_k(v)),$$

and to use $L$ such $g$’s for the similarity test. In the similarity test, point $v$ is judged to be similar to query $q$ if at least one function $g$ (among $L$ candidates) falls in the same bucket as $q$; namely, it uses logical-OR in this part. In addition, two $k$-dimensional vectors are considered to fall in the same bucket by a function $g$ if all $k$ elements generated by the function are identical; namely, it uses logical-AND in this part.

The accuracy of the resulting similarity test is evaluated as follows. Let $q$ be a query and assume that each $h$ is a $(r, r(1 + \epsilon), p_1, p_2)$-sensitive function; namely: 1) the probability that a point whose proximity to $q$ is at most $r$ is mapped to $h(q)$ is at least $p_1$, and 2) the probability that a point whose proximity to $q$ is at least $r(1 + \epsilon)$ is mapped to $h(q)$ is at most $p_2(< p_1)$. Note that this condition is satisfied for sufficiently large $\epsilon$ if the distance to $q$ with respect to the value of function $h$ follows a normal distribution.

If each $h$ is $(r, r(1 + \epsilon), p_1, p_2)$-sensitive, in the sense of $k$-dimensional vector, the probability that a point $v$ whose distance from $q$ is $r$ or less is mapped to $g(q)$ is at least $p_1^k$, and the probability that a point whose distance from $q$ is $r(1 + \epsilon)$ or greater is mapped to $g(q)$ is at most $p_2^k$. The ratio $(p_1/p_2)^k$ can be as large as possible by increasing value $k$, and the value of $p_1^k$ can also be increased as much as possible by increasing value $L$. In addition, this refinement technique [8] can be applied to any $(r_1, r_2, p_1, p_2)$-sensitive function $h$.

5. Proposed Method

5.1 Basic Flow

In this section, we describe the details of the proposed method. Recall that in the default setting of IPFS, a cryptographic hash such as SHA-1 is applied to the content itself to generate a ContentID. The first idea of the proposed method is to use the notion of feature vectors which have been widely used in a variety of fields; e.g., word2vec and TF-IDF are common techniques to generate feature vectors of text files, and in the case of media files, genre, artist, and musical idea are used for the feature vectorization. In addition to those definitions of feature vectors, various distance metrics have been used in the literature, such as the angular distance and the Jaccard distance, in addition to the Euclidean distance used in the proposed scheme.

Although the notion of feature vectors allows to numerically represent the similarity of contents, unfortunately, since direct application of cryptographic hash to feature vectors does not allow the vector proximity to be reflected in the identity of the cryptographic hash values (i.e., high probability of mapping to the same value), we shall introduce the second idea described below: Feature vector $v$ is converted to a value $g(v)$ by LSH and a cryptographic hash is applied to $g(v)$ to generate a ContentID. By the definition of LSH, two vectors which are close together in the feature space have a high probability of being mapped to the same value, which results in similar contents being assigned to the same node with a high probability, allowing a similarity search within the IPFS framework. The reader should note that although existing schemes for the similarity search in P2P systems often assume that similar content is mapped to neighbor-
search for a given query content $X$ in the following manner (see Fig. 1 for illustration):

**Similarity Search($X$)**

1. Generate a $d$-dimensional feature vector $v(X)$ from $X$.
2. Apply LSH functions to $v(X)$ to generate $L$ hash values $g_1(v(X)), g_2(v(X)), \ldots, g_L(v(X))$. Then apply cryptographic hash to each of them to generate IPFS addresses corresponding to query $X$, and request to get the index list matching the query. Such requests can coexist with normal requests in IPFS by designating an appropriate prefix such as /lsh/.
3. Let $S_i$ be the response to $g_i(v(X))$ and let $S = \bigcup_{i=1}^L S_i$. To simplify the exposition, in the following we call each $S_i$ a bucket hit by the query. For each $Y \in S$, calculate the proximity of $v(Y)$ to $v(X)$, sort them in a descending order of the proximity, and output $Y$’s with the highest proximity as the result of similarity search.

Additional cost necessary for the proposed scheme is evaluated as follows, where a quantitative analysis will be given in the next section: Although it needs to generate feature vector $v(X)$ and its $L$ hash values for each stored content $X$ in the preprocessing phase, actual retrieval of index list can be done in a similar way to the retrieval with inverted indexes which is a technique commonly used for the keyword search in DHTs. Such a calculation can be executed locally provided that all parameters necessary for the calculation of $v(X)$ and $g_i(v(X))$’s have been notified to all nodes, although it takes a (slightly) longer time than the original IPFS. Similarly, the storage cost for the scheme is at most $L$ times of the original IPFS. The reader should note again that the proposed method is probabilistic, and it does not guarantee the retrieval of contents closest to the query content $X$.

5.2 Implementation Issues and Solutions

In the proposed method, the above basic flow is extended in various directions to improve the efficiency and accuracy of the similarity search. An exhaustive evaluation of those extensions is left as a future work, while experimental evaluations of the basic flow will be conducted in the next section.

1. We can sort each subset $S_i$ in a descending order of the proximity to the query and merge them to have a sorted $S$. Such a behavior can be realized by attaching $v(X)$ to the query sent to an address designated by $g_i(v(X))$. This is particularly effective to bound the amount of data exchanged among nodes if we are given an upper limit on the number of contents to be returned, since it allows nodes to reply only few elements in $S_i$ to the requester.
2. The merge of buckets received from other nodes can be asynchronously done, as in the trickle ICE (Interactive Connectivity Establishment) which is widely used for the NAT traversal.
3. In the original scheme, two vectors are considered to be similar if they match with at least one hash value generated by $L$ LSH functions. However, as will be evaluated later, it significantly increases the bucket size in the worst case. To overcome this issue, we propose to refine the definition of similarity so that two vectors are judged to be similar if they match with more than $\alpha$ hash values for some $\alpha \geq 0$. This idea can be realized by extending the merge process so as to take into account the multiplicity of elements contained in the buckets, and to use the multiplicity in the resulting $S$ as a filter.
4. The fourth direction is to extend the length of vectorized hash value generated by function $g$. The analysis given in the previous section indicates that the dimension $k$ in $g$ plays an important role in mapping only those points that are really close to the query $X$ to the same value as $X$ with high probability; i.e., the larger $k$ the better. However, too large $k$’s yield many empty buckets. Although such a weak point could be partially supplemented by increasing $L$, a proper setting of parameter $k$ is still a tricky problem. This problem can be overcome by adaptively adjusting $k$ in the following manner: The most extreme case is to prepare IPFS address for each $h$ value, and to collect those values for calculating function $g$. As an intermediate implementation, we can divide $k$ into several segments, and associate (conjunctive) value corresponding to a segment to an IPFS address.

6 Evaluation

6.1 Performance as a Similar Search Method

At first, we evaluate the performance as a similarity search scheme. In the following experiments, we vary the dimension of the vector space from $d = 20$ to $100$ and distribute $N = 10000$ random points beforehand in such a way that the value of each element is uniformly selected from $[0, 1]$. The reader should notice that the accuracy of the similarity
search becomes higher for larger \( N \), since it increases the bucket size hit by the query. Other parameters are set up as follows: vector \( a \) is selected from \([0, 1]^d\) so that the value of each dimension follows a normal distribution with mean 0.0 and standard deviation 1.0, and parameter \( W \) is fixed to 5.0, according to the result of preliminary evaluations.

As the first experiment, we evaluate the impact of parameter \( L \) to the hitting probability of random query by letting \( k = 12 \) and varying \( L \) from 5 to 60, since \( L \) directly affects the cost of similarity search. The result is summarized in Fig. 2, where the vertical axis is the number of queries hitting a bucket of a designated size among \( M = 1000 \) random queries. In the figure, cases in which hitting bucket has size one is painted gray, cases in which hitting bucket has a size from two to ten is painted red, and so on. When \( L = 5 \), 90\% of queries hit empty bucket, but the number of non-empty cases increases as \( L \) increases, while it hits bucket of size one in almost a half of non-empty cases. In other words, when \( L \geq 30 \), more than one third of random queries hit buckets containing several candidates for the similarity search. From this result, in the following, we will let \( L = 60 \) unless otherwise stated.

Next, we focus on cases in which query \( q \) hits non-empty bucket, and evaluate the proximity of the closest point in the bucket to \( q \). Figure 3 summarizes the results. The horizontal axis is the size of the bucket hit by \( q \) and the vertical axis is the distance from \( q \) to the closest point in the bucket, which is normalized by the distance to the closest point \( p^* \) among \( N \) points in such a way that: 1) the distance between \( p^* \) and \( q \) corresponds to 0\% and 2) the average distance from \( q \) over \( N \) points corresponds to 100\% (in other words, the expected similarity of a simple random selection corresponds to 100\%). Although it is not designated in the figure, the standard deviation of the random sampling is not large and the \( \sigma \) confidence interval lies down around 70 to 80\%.

From the figure, we can observe that if the bucket size hit by the query is moderate (e.g., ten or more), the proposed scheme outputs data outside \( \sigma \) confidence interval with high probability. Although this phenomenon thanks to the property of the underlying LSH, it is worth noting that it realizes a significant improvement of conventional flooding-based approaches for the similarity search, since in a high-dimensional vector space, the volume of a \( d \)-dimensional sphere significantly reduces by decreasing the radius of the sphere; namely, points close to \( q \) are rare for any \( q \), as long as \( N \) points are uniformly distributed over \([0, 1]^d\). Such a proximity of the resulting point could not be improved by simply increasing \( L \), while it certainly increases the number of candidates as was shown in Fig. 2.

In many practical scenarios, we are requested to find several data points as the output of a similarity search. To evaluate the performance of the scheme under such a scenario, we measure the amount of reduction of the average distance to the query by restricting the target for the selection from \( N \) points to points in the bucket hit by the query. Figure 4 summarizes the results, where the vertical axis is the average distance of points in the bucket normalized by the average distance to \( N \) points. We can observe that it converges to 100\% as the bucket size increases, and in the experiment, there certainly exists a big bucket containing more than 2500 points which occupies one fourth of \( N \) points.

### 6.2 Storage Cost

Next, we evaluate the storage cost of the proposed scheme. The scheme applies function \( h \) to each \( d \)-dimensional feature vector to generate \( k \times L \) scalar values, and applies function \( g \) to the resulting \( k \)-dimensional vectors to generate \( L \) scalar values. It then applies a cryptographic hash to each scalar value to generate an IPFS address to store the index of the content at a node corresponding to the calculated address. Thus, the number of indexes generated from a content is at most \( L \) (resp. \( k \times L \)) if the outcome of function \( g \) (resp. \( h \)) is used for calculating an IPFS address, which means that
the storage cost is $O(N)$ for $N$ contents as long as $k$ and $L$ are constant (i.e., no exponential explosion occurs). Since the ID space of Kademlia has a 160 bit length, it is large enough for storing all indexes generated by the proposed scheme. In addition, the load of nodes concerned with the index storage could be balanced with the aid of cryptographic hash used in Kademlia.

Since the proposed scheme takes advantage of collisions of hash values to realize an efficient similarity search, the kind of hash values actually generated by functions $g$ and $h$ is expected to be significantly smaller than $L \times N$ and $k \times L \times N$, respectively. Table 1 summarizes the number of buckets with a designated size generated by applying 1000 different $h$’s to 10000 random feature vectors, where value 74.2 in the table indicates that 74.2% of vectors (i.e., 7420 vectors) are mapped to a bucket of size one. In the experiment, we vary parameter $W$ to control the size of buckets generated by function $h$. In fact, 1) when $W = 1.0$, almost $N$ vectors are mapped to different hash values (i.e., collision of hash values rarely occurs); 2) as $W$ increases, the number of cases with no collision decreases as 74.2, 23.9, and 8.3; and 3) for $W = 5.0$, which is the value actually used in the experiments described in the last subsection, only 3.5% of vectors are mapped to a bucket of size one, and the total number of buckets generated by $h$ is 8.4% of $N$ vectors.

### Table 1

| $W$  | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| size 1 | 99.9 | 74.2 | 23.9 | 8.3 | 3.5 | 1.8 | 1.0 | 0.6 |
| size 2–10 | 0   | 10.3 | 13.0 | 6.7 | 3.5 | 2.0 | 1.2 | 0.8 |
| size 11– | 0   | 0   | 1.4 | 1.7 | 1.4 | 1.1 | 0.8 | 0.7 |

6.3 Number of Messages

Finally, we evaluate the performance of the proposed scheme as a search engine for IPFS. As mentioned previ-

ously, there are no existing methods to conduct similarity search targeting IPFS. Although it would be possible to realize a similarity search of text files stored in IPFS by using a centralized approach with the aid of Elasticsearch as in ipfs-search, a fair comparison with the proposed scheme is difficult due to the long latency of data crawling and the existence of single point of failure at the centralized server. On the other hand, although several recent works try to decentralize the LSH-based similarity search since they focus on the load balancing issues, it is still difficult to compare with them as a similar search method on IPFS. In addition, although there is a previous work which applies LSH to Kademlia [12], it could not be applied to the similarity search in IPFS since it uses specific mapping from LSH hashes to the address space of Kademlia.

The query response time achieved by the proposed scheme can be evaluated as follows. Since it uses Kademlia as the underlying network, at most $O(\log P)$ messages are sequentially issued for the routing of a given query to the target node, where $P$ is the number of nodes in the overlay (note that it is not a function of the number of contents stored in the network). Of course, the actual query response time depends on the size of bucket hit by the query held by the target node, but as was shown previously, it is very rare to generate such a huge bucket. In addition, the processing time in a node is usually much shorter than the transmission time of a query to the next node on the overlay, which generally takes at least 10 to 100 [ms] in a typical setting for IPFS. The evaluation of the amount of increase of the query-response time due to the proposed similarity search scheme in actual IPFS is left as a future work.

On the other hand, most of existing methods for the similarity search in unstructured P2Ps are based on the clustering of nodes according to the similarity of contents held by the nodes and the preferential forwarding of queries into clusters to realize an efficient similarity search in a heuristic manner. In such schemes, query propagation is generally controlled by the TTL of flooding, which implies that the maximum number of query hops can be easily bounded by the maximum TTL. Thus for example, by limiting the maximum TTL to 3, we can easily realize a similarity search within 3 hops, and outperform the proposed scheme with respect to the length of sequential hops. However, since the total number of messages is proportional to the number of nodes within a designated distance from the requester, it increases exponentially with respect to the average degree in the network, which plagues us by the trade-off between the quality and the cost of similarity search. In addition, such a clustering-based method is based on a strong assumption such that the content held by each node reflects the interests of the node, which makes difficult to conduct a quantitative analysis of the accuracy of similarity search.

7. Concluding Remarks

This paper proposes a similarity search scheme for IPFS based on the notion of locality sensitive hash. By conduct-
ing experiments, we found that more than half of randomly given queries return a non-empty result for the similarity search, and yield an accurate result which is outside the \( \sigma \) confidence interval of an ordinary random selection method. Future work includes the evaluation of extensions and construction of concrete applications.

References

[1] K. Aberer, P. Cadré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Puncetea, and R. Schmidt, "P-Grid: A self-organizing structured P2P system," ACM SIGMOD Rec., vol.32, no.3, pp.29–33, 2003.

[2] I. Abraham, Y. Bartal, and O. Neiman, "Advances in metric embedding theory," Proc. 38th Annual ACM Symposium on Theory of Computing, pp.271–286, 2006.

[3] I. Baumgart and S. Mies, "S/Kademia: A practicable approach towards secure key-based routing," Proc. Int'l Conf. on Parallel and Distributed Systems, vol.2, pp.1–8, IEEE, 2007.

[4] J. Benet, "IPFS - Content addressed, versioned, P2P file system," arXiv:1407.3561 [cs.NI].

[5] T. Bocek, E. Hunt, D. Hausheer, and B. Stiller, "Fast similarity search in peer-to-peer networks," Proc. IEEE Network Operations and Management Symposium, pp.240–247, 2008.

[6] Z. Cai, I. Lee, S.-C. Chu, and X. Huang, "SimSim: A service discovery method preserving content similarity and spatial similarity in P2P mobile cloud," J. Grid Computing, vol.17, no.1, pp.79–95, 2019.

[7] Y.-T. Chuang, C.-Y. Yu, and Q.-W. Wu, "DLSM: A decentralized search for large and mobile networks," J. Supercomputing, vol.74, no.2, pp.738–767, 2018.

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," Proc. Symposium on Computational geometry, pp.253–262, 2004.

[9] C. Doulkeridis, A. Vlachou, K. Nørvag, Y. Kotidis, and M. Vazirgiannis, "Efficient search based on content similarity over self-organizing P2P networks," Peer-to-Peer Networking and Applications, vol.3, no.1, pp.67–79, 2010.

[10] A. Forestiero, E. Leonardi, C. Mastroianni, and M. Meo, "Self-chord: A bio-inspired P2P framework for self-organizing distributed systems," IEE/ACM Trans. Netw., vol.18, no.5, pp.1651–1664, 2010.

[11] M.J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing content publication with coral," NSDL vol.4, p.18, 2004.

[12] P. Haghani, S. Michel, and K. Aberer, "Distributed similarity search in high dimensions using locality sensitive hashing," Proc. 12th Int'l Conf. on Extending Database Technology: Advances in Database Technology (EDBT ’09), pp.744–755, 2009.

[13] P. Kalnis, W.S. Ng, B.C. Ooi, and K.-L. Tan, "Answering similarity queries in peer-to-peer networks," Information Systems, vol.31, no.1, pp.57–72, 2006.

[14] D. Karapiperis, A. Gkoulalas-Divanis, and V. Verykios, "LSHDB: A parallel and distributed engine for record linkage and similarity search," Proc. 16th Int'l Conf. on Data Mining Workshops (ICDMW), pp.1–4, IEEE, 2016.

[15] M. Karnstedt, K.-U. Satller, M. Hauswirth, and R. Schmidt, "Cost-aware processing of similarity queries in structured overloads," Proc. 6th IEEE Int'l Conf. on Peer-to-Peer Computing, pp.81–89, 2006.

[16] N. Khudhr and S. Fujita, "Siva - The IPFS search engine," Proc. CANDAR 2019, pp.150–156, 2019.

[17] N. Kraus, D. Carmel, I. Keidar, and M. Orenbach, "Nearbucket-LSH: Efficient similarity search in P2P networks," Proc. Int'l Conf. on Similarity Search and Applications, Lecture Notes in Computer Science, vol.9939, pp.236–249, Springer, Cham, 2016.

[18] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer infor-