Distributed Similarity Search in High Dimensions Using Locality Sensitive Hashing*

Parisa Haghani
EPFL
Lausanne, Switzerland
parisa.haghani@epfl.ch

Sebastian Michel
EPFL
Lausanne, Switzerland
sebastian.michel@epfl.ch

Karl Aberer
EPFL
Lausanne, Switzerland
karl.aberer@epfl.ch

ABSTRACT

In this paper we consider distributed K-Nearest Neighbor (KNN) search and range query processing in high dimensional data. Our approach is based on Locality Sensitive Hashing (LSH) which has proven very efficient in answering KNN queries in centralized settings. We consider mappings from the multi-dimensional LSH bucket space to the linearly ordered set of peers that jointly maintain the indexed data and derive requirements to achieve high quality search results and limit the number of network accesses. We put forward two such mappings that come with these salient properties: being locality preserving so that buckets likely to hold similar data are stored on the same or neighboring peers and having a predictable output distribution to ensure fair load balancing. We show how to leverage the linearly aligned data for efficient KNN search and how to efficiently process range queries which is, to the best of our knowledge, not possible in existing LSH schemes. We show by comprehensive performance evaluations using real world data that our approach brings major performance and accuracy gains compared to state-of-the-art.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Distributed High Dimensional Search

Keywords

knn search, range query, high dimensionality, p2p

1. INTRODUCTION

The rapid growth of online information, triggered by the popularity of the Internet and the huge amounts of user-generated content from Web 2.0 applications, calls for efficient management of this data to improve usability and enable efficient and accurate access to the data. User-generated data today range from simple text snippets to (semi-) structured documents and multimedia content. To enable rich representation and avoid loss of information, the number of features extracted to represent the data is very often high. Furthermore, as the data sources are naturally distributed in large-scale networks, traditional centralized indexing techniques become impractical. To address the demanding needs caused by this rapidly growing, large-scale, and naturally distributed information ecology, we propose in the following an efficient, distributed, and scalable index for high-dimensional data enabling efficient and accurate similarity search.

Peer-to-Peer (P2P) overlay networks are well-known to facilitate the sharing of large amounts of data in a decentralized and self-organizing way. These networks offer enormous benefits for distributed applications in terms of efficiency, scalability, and resilience to node failures. Distributed Hash Tables (DHTs) [30, 28], for example, allow efficient key lookups in logarithmic number of routing hops but are typically limited to exact or range queries. Similarity search in high dimensional data has been a popular research topic in the last years [9, 17, 32, 8, 14]. Distributed processing of such queries is even more complicated, but is unavoidable due to the inherently distributed way data is generated in the Web. Existing approaches to the similarity search problem in high dimensional data either focus on centralized settings, as cited above, rely on preprocessing data centrally, assume data ownership by peers in a hierarchical P2P setting or fail at providing both high quality search results and a fair load balance in the network [16, 29, 15].

In this paper we consider similarity search over high dimensional data in structured overlay networks. Inspired by the idea of Locality Sensitive Hashing (LSH) technique [17, 14] which probabilistically assigns similar data to the same bucket in a hash table, we first investigate the difficulties of directly applying this method to a distributed environment and then devise two locality preserving mappings which satisfy the identified requirements of bucket placement on peers of a P2P network. The first requirement, placing buckets which are likely to hold similar data on the same peer or its neighboring peers, aims at minimizing the number of network hops necessary to retrieve the search results, causing

a decrease in both network traffic and the overall response time. The second requirement considers load balancing and is satisfied by harnessing estimates of the distribution of resulting data (bucket) mapping. The basic ideas for our approach appeared in a preliminary short paper at a workshop [19]. This current work substantially extends the prior work in the following ways: we capture the notion of buckets likely to hold similar data, enabling more elaboration on suitability of different mapping scheme. We propose another novel mapping schemes which satisfies our identified requirements. We also address range queries which are difficult to process in LSH-based indexing techniques and show how our proposed mappings can be used to derive estimates of the range of necessary peers to be visited. In addition, we extended our experimental evaluation by considering more datasets and comparing against state-of-the-art.

1.1 Problem Statement and System Overview

In similarity search objects are characterized by a collection of relevant features and are represented as points in a high dimensional space. In some applications the objects are considered in a metric space where only a distance function is defined among them and the features of the objects are unknown. However, with the advances in metric space embedding (*cf.*[1]) a vector space assumption is valid and realistic. Given a collection of such points and a distance function between them, similarity search can be performed in the following two forms:

- *K-Nearest Neighbor (KNN) query:* Given a query point q the goal is to find the K closest (in terms of the distance function) points to it.

- *range query:* Given a query point q and a range r the goal is to find all points within a distance r of q.

In many applications returning the approximate KNN of a point, instead of the exact ones, suffices. The approximate version is even more desirable when the data dimensionality is high, as similarity search is very expensive in such domains. Here, the goal is to find K objects whose distances are within a small factor $(1+\epsilon)$ of the true K nearest neighbors' distances. The quality of similarity search is measured by the number of returned results, as well as the distances to the query for the K points returned compared to the corresponding distances of the true K nearest objects for KNN queries.

We consider similarity search in structured peer-to-peer networks, where N peers $P_1, ..., P_N$ are connected by a DHT that is organized in a cyclic ID space, such as in Chord [30]. Every node is responsible for all keys with identifiers between the ID of its predecessor node and its own ID. Our underlying similarity search method is probabilistic and relies on building several indices of data to achieve highly accurate query results. We assume each of these data indices is maintained by a subset of size n of all peers where n is an order of magnitude smaller than N. Each of these subsets form a *local* DHT among themselves. For each replica of the data we deterministically select a subset of peers to hold the corresponding index, i.e., not all peers hold a share of the index from the beginning. These initially selected peers are gateway peers to the corresponding local DHTs. The number of peers/nodes inside each local DHT might grow/shrink over time depending on the load of the system. The locations (IDs) of the gateway peers is global knowledge based on a deterministic sampling process with fixed seed value. However, not the peers are known but only their IDs in the underlying network. We explain this process in more detail

in Section 4. If a gateway peer is not accessible, the peer currently holding the gateway peer ID is asked to join the local DHT using one of the other gateway peers. These dynamics are handled by the underlying network and are beyond the scope of this paper.

Our goal is to map the high dimensional data to the peers in a way that assures fair load balancing in the local DHTs and at the same time enables efficient and accurate KNN and range query processing.

1.2 Contribution and Outline

With this work we make the following contributions:

- We discuss the difficulties of distributing existing LSH schemes and derive requirements to distribute them in a way that assures fair load balance and efficient and accurate similarity search processing.

- We present two novel mapping schemes which satisfy the mentioned requirements.

- We present a top-K algorithm to efficiently process distributed KNN queries.

- We show, relying on our mapping schemes, how otherwise-difficult-to-process range queries can be efficiently processed in our setting by presenting a novel sampling-based method which utilizes our estimated range of peers necessary to contact.

- We experimentally evaluate the efficiency and effectiveness of our approach using two real-world data sets.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the requirements of distributing LSH indices to the linear peer domain and puts forward two mappings which satisfy those requirements. Section 4 concentrates on the creation of the local DHTs. Section 5 presents the KNN query processing algorithms. Section 6 addresses the challenges in processing range queries and presents an approach based on range sampling to overcome these constraints. Section 7 presents the experimental evaluation of our approach. Section 8 concludes the paper.

2. RELATED WORK

Similarity search in high dimensional spaces has been the focus of many works in the database community in the recent years. The problem has been intensively researched on in the centralized setting, for which the approaches can be divided into two main categories. *Space partitioning* methods form the first category consisting of all *tree-based* approaches such as R-tree [18] and K-D trees [7], which perform very well when data dimensionality is not high but degrade to linear search for high enough dimensions [9]. The Pyramid [8] and iDistance[32] techniques map the high dimensional data to one dimension and partition/cluster that space to answer queries by translating them to the one dimensional space. The second category consists of *hash-based* approaches which trade accuracy for efficiency, by returning approximate closest neighbors of a query point. LSH [17] is an approximate method, which uses several locality preserving hash functions to hash the data, such that with high probability close points are hashed to the same bucket. While this method is very efficient in terms of time, tuning such hash functions depends on the distance of the query point to its closest neighbor. Several follow-ups of this method exist which try to solve the problems associated with it [6, 14, 23, 25, 2]. Very recently [4] proposed

a method based on distance hashing which has the advantage of not depending on any specific distance measure, but involves some off-line tuning of parameters. Approximate range search in high dimensions has also been the focus of some works such as [3, 12] which return points in $(1 + \epsilon)r$ of the query point, instead of retrieving points which have exactly distance smaller or equal to r to the query point. In this work we do not consider approximate ranges queries.

With the emergence of the P2P paradigm [30, 28], there has been a tendency to leverage the power of distributed computing by sharing the cost incurred by such methods over a set of machines. A number of P2P approaches, such as [13, 10, 11] have been proposed for similarity search, but they are either dedicated to one dimensional data or do not consider very high dimensional data. MCAN [16] uses a pivot-based technique to map the high dimensional metric data to an N-dimensional vector space, and then uses CAN [28] as its underlying structured P2P system. The pivots are chosen based on the data, which is preprocessed in a centralized fashion and then distributed over the peers. SWAM [5] is a family of *Small World Access Methods*, which aims at building a network topology that groups together peers with similar content. In this structure peers can hold a single data item each, which is not well-suited for large data sets. SkipIndex [33] and VBI-tree [21] both rely on *tree-based* approaches which do not scale well when data dimensions are high. In pSearch [31], the well known information retrieval techniques *Vector Space Model*(VSM) and *Latent Semantic Indexing*(LSI) are used to generate a semantic space. This Cartesian space is then directly mapped to a multi-dimensional CAN which basically has the same dimensionality of the Cartesian space (as high as 300 dimensions). Since the dimensionality of the underlying peer-to-peer network depends on the dimensionality of the data (or the number of reduced dimensions) different overlays are needed for various data sets with different dimensionality. This dependency and centralized computation of LSI make this approach less practical in real applications. In [29] the authors follow pSearch by employing VSM and LSI, but map the resulting high dimensional Cartesian space to a one dimensional Chord. Unlike pSearch this method is independent of corpus size and dimensionality. This is the closest work in state of the art to us, since it considers high dimensional data over a structured peer-to-peer system. We compare our approach to this work and explain it in more depth in Section 2.1. Recently, SimPeer [15] was proposed, which uses the principle of iDistance [20] to provide range search capabilities in a hierarchical unstructured P2P network for high dimensional data. In this work also, the peers are assumed to hold and maintain their own data. On the contrary, we consider efficient similarity search over structured P2P networks, which guarantees logarithmic lookup time in terms of network size, and leverage on *LSH-based* approaches to provide approximate results to KNN search efficiently, even in very high dimensional data. Our approach also enables efficient range search which is difficult in LSH-based approaches.

2.1 The Approach by Sahin et al.

Here we briefly describe the approach of [29]. A globally known list $R = r_1, r_2, ..., r_v$ of reference data points is considered. These are either randomly chosen from the data set or are the cluster representatives of a clustered sample set. In order to index a data point v, it is compared against all reference points and a sorted list of references in increasing distance to v is constructed. The first j references, which are the reference points closest to v are used to make the Chord key for it: the binary representations of the ID's of these j references are concatenated, with the highest relevant refer-

ence as the high order bits. If there are any remaining bits in the Chord bit representation, they are filled with zeros. The intuition behind this approach is that points which are close to each other, share common top references and will therefore be stored at the same peer. In order to increase the probability of this event, multiple Chord keys are made for each data point, choosing j different reference points, or different permutations of them. At query time, the query point is similarly mapped to the Chord ring and the corresponding peers are scanned with the K closest points returned and merged at the peer issuing the query. Figure 1 shows an example of indexing a data point. This approach of mapping data points to the Chord ring focuses on placing close points on the same peer and does not exploit nearby peers. For example a data point q close to v might have (r_6, r_4) as its two first closest references, corresponding to the Chord key 1001100000, resulting in placing it on a far peer from where v is placed (*i.e.* 1101000000). In our approach we aim at placing close points on the same peer *or* neighboring peers to exploit the linear order of peers in Chord style DHTs and avoid high number of DHT lookups.
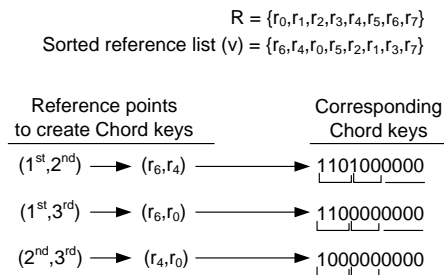
$$R = \{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$$

Sorted reference list $(v) = \{r_6, r_4, r_0, r_5, r_2, r_1, r_3, r_7\}$

Reference points to create Chord keys		Corresponding Chord keys
$(1^{st}, 2^{nd}) \longrightarrow$	$(r_6, r_4) \longrightarrow$	1101000000
$(1^{st}, 3^{rd}) \longrightarrow$	$(r_6, r_0) \longrightarrow$	1100000000
$(2^{nd}, 3^{rd}) \longrightarrow$	$(r_4, r_0) \longrightarrow$	1000000000

Figure 1: Illustration of mapping a data point v to the Chord peer space by Sahin. The number of references is 7, the Chord range is $(0, 2^{10})$, j is 2 and each data point is replicated three times.

2.2 Revisiting LSH

The basic idea behind the *LSH-based* approaches is the application of *locality sensitive hashing* functions. A family of hash functions $H = \{h : S \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive if the following conditions are satisfied for any two points $\mathbf{q}, \mathbf{v} \in S$:

- if $dist(\mathbf{q}, \mathbf{v}) \leq r_1$ then $Pr_H(h(\mathbf{q}) = h(\mathbf{v})) \geq p_1$
- if $dist(\mathbf{q}, \mathbf{v}) > r_2$ then $Pr_H(h(\mathbf{q}) = h(\mathbf{v})) \leq p_2$

where S specifies the domain of points and *dist* is the distance measure defined in this domain.

If $r_1 < r_2$ and $p_1 > p_2$, the salient property of these functions results in more similar objects being mapped to the same hash value than distant ones. The actual indexing is done using LSH functions and by building several hash tables to increase the probability of collision (*i.e.* being mapped to the same hash value) for close points. At query time, the KNN search is performed by hashing the query point to one bucket per hash table, scanning that bucket and then ranking all discovered objects by their distance to the query point. The closest K points are returned as the final result.

In the last few years, the development of locality sensitive hash functions has been well addressed in the literature. In this work, we consider the family of LSH functions based on p-stable distributions [14]. This family of LSH functions

are most suitable when the distance measure between the points is the l_p norm. Given a point $\mathbf{v} = v_1, \ldots, v_2$ in the d-dimensional vector space, its l_p norm is defined as: $||\mathbf{v}||_p = (|v_1|^p + \ldots + |v_d|^p)^{1/p}$.

Stable Distribution: A distribution D over $\mathbb{R}$ is called p-stable, if there exists $p \geq 0$ such that for any n real numbers $r_1 \ldots r_n$ and i.i.d. variables $X_1 \ldots X_n$ with distribution D, the random variable $\sum_i r_i X_i$ has the same distribution as the variable $(\sum_i |r_i|^p)^{1/p} X$, where X is a random variable with distribution D. p-stable distributions exist for $p \in (0, 2]$. The Cauchy and Normal distributions are respectively 1-stable and 2-stable.

In the case of p-stable LSH, for each d-dimensional data point $\mathbf{v}$ the hashing scheme considers k independent hash functions of the following form :

$$h_{\mathbf{a},B}(\mathbf{v}) = \lfloor \frac{\mathbf{a} \cdot \mathbf{v} + B}{W} \rfloor \qquad (1)$$

where $\mathbf{a}$ is a d-dimensional vector whose elements are chosen independently from a p-stable distribution, $W \in \mathbb{R}$, and B is drawn uniformly from $[0, W]$. Each hash function maps a d-dimensional data point to an integer. With k such hash functions, the final result is an integer vector of dimension k of the the following form:

$$g(\mathbf{v}) = (h_{\mathbf{a_1},B_1}(\mathbf{v}), \ldots, h_{\mathbf{a_k},B_k}(\mathbf{v})) \qquad (2)$$

In this work we assume the distance function is the widely used l_2 norm (Euclidean distance) and use the Normal distribution as our p-stable distribution.

In LSH-based schemes, in order to achieve high search accuracy, multiple hash tables need to be constructed. Experimental results [17] show that the number of hash tables needed can reach up to over a hundred. In centralized settings this causes space efficiency issues. While this constraint is less visible in a P2P setting, a high number of hash tables results in another serious issue arising specifically in this environment. In order to visit all hash tables (which is needed to answer the KNN query with good accuracy) a large number of peers may need to be contacted. Solutions to this shortcoming in centralized settings [23, 25] suggest investigating more than one bucket in each hash table, instead of building many different hash tables. The main idea is that we can guess which buckets other than the bucket which the query hashes to, are more likely to hold data that is similar to the query point. In our envisioned P2P scenario, jumping from one bucket to another can potentially cause jumping from one peer to another, which induces $O(log\, n)$ network hops in a network of n peers. In the following section, we discuss and introduce mapping schemes which allow us to significantly reduce the number of incurred network hops during query time by grouping those buckets which are likely to hold similar data on the same peer, while effectively balancing the load in the network.

3. MAPPING LSH TO THE PEER IDENTIFIER SPACE

Given the output of the p-stable LSH, which is a vector of integers, we consider a mapping to the peer identifier space, denoted as $\xi : \mathbb{Z}^k \to \mathbb{N}$.

Different instances of the mapping function ξ come with different characteristics w.r.t. to load balancing and the ability to efficiently search the index. In terms of network bandwidth consumption and number of network hops, clearly, a mapping of all data to just one peer is optimal. Obviously, this mapping suffers from a huge load imbalance. The other extreme is to assign each hash bucket to a peer using

a pseudo-uniform hash function that provides perfect load balancing but steals any control on grouping similar buckets on the same peer, therefore causing an excessive number of DHT lookups. More formally, ξ should satisfy the following two conditions:

- *Condition 1:* assign buckets likely to hold similar data to the same peer.

- *Condition 2:* have a predictable output distribution which fosters fair load balancing.

Figure 2 shows an illustration of the overall mapping from the d-dimensional space, to the k-dimensional LSH buckets, to finally the peer identifier space using ξ.
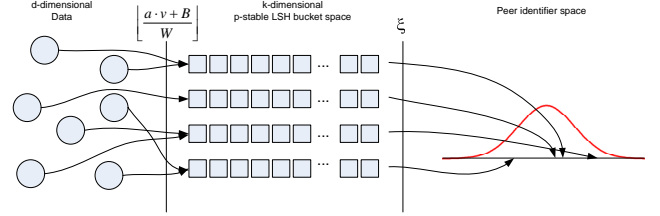

Figure 2: Illustration of the two level mapping from the d-dimensional space to the peer identifier space.

We first try to capture the semantics of similar buckets: *buckets likely to hold close data*. The first condition of the LSH definition states that close points are more likely to be mapped to the *same* hash value. However, it is not clear which hash buckets are more probable to hold similar data. This has been discussed also in [23, 25] in a query-dependent way. However we need a more general view, as mapping buckets to peers should be independent of queries. We show that using hash functions of the form of Equation 1 close points have a higher probability of being mapped to *close* integers, that is, integers with small l_1 distance. This is more general than the LSH definition, i.e. being hashed to the *same* value. Since bucket labels are concatenations of such integers, we argue that the l_1 distance can capture the distance between buckets, *buckets likely to hold close data have small l_1 distance to each other.* We prove the following theorems following the above argument.

Theorem 1 For any three points $\mathbf{v_1}, \mathbf{v_2}, \mathbf{q} \in S$ where $||\mathbf{q} - \mathbf{v_1}||_2 = c_1$ and $||\mathbf{q} - \mathbf{v_2}||_2 = c_2$ and $c_1 < c_2$ the following inequality holds:

$$pr(|h(\mathbf{q}) - h(\mathbf{v_1})| \leq \delta) \geq pr(|h(\mathbf{q}) - h(\mathbf{v_2})| \leq \delta)$$

Proof. Let $s(c, \delta) := pr(|h(\mathbf{q}) - h(\mathbf{v})| \leq \delta)$ and $t(c, \delta) := pr(|h(\mathbf{q}) - h(\mathbf{v})| = \delta)$ where $||\mathbf{q} - \mathbf{v}||_2 = c$. Then $s(c, \delta)$ can be written as $s(c, \delta) = t(c, 0) + \cdots + t(c, \delta)$. We want to show that for any fixed δ, $s(c, \delta)$ is monotonically decreasing in terms of c. We first derive $t(c, \delta)$. Our argument is similar to that of [14]. Since elements of the random vector $\mathbf{a}$ are chosen from a standard Normal distribution, $\mathbf{a.q} - \mathbf{a.v}$ is distributed according to cX where X is a random variable drawn from a Normal distribution. Therefore the probability distribution of $|\mathbf{a.q} - \mathbf{a.v}|$ is $\frac{1}{c} f(\frac{x}{c})$ where $f(x)$ denotes the probability density function of the absolute value of the standard Normal distribution (i.e., the mean is zero and the variance is one) :

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ \frac{2}{\sqrt{2\pi}} e^{-x^2/2} & \text{if } x \geq 0 \end{cases}$$

For $\delta = 0$, in order to have $|h(\mathbf{q}) - h(\mathbf{v_1})| = \delta$, $|\mathbf{a.q} - \mathbf{a.v}|$ has to be in $[0, W)$. Depending on the exact value of $|\mathbf{a.q} - \mathbf{a.v}|$ different range of values for B can leave $|h(\mathbf{q}) - h(\mathbf{v_1})| = 0$ or change it to $|h(\mathbf{q}) - h(\mathbf{v_1})| = 1$. For example if $|\mathbf{a.q} - \mathbf{a.v}|=0$, all values of $B \in [0, W)$ keep the desired values $|h(\mathbf{q}) - h(\mathbf{v_1})| = 0$. The size of this range of values for B decreases linearly as $|\mathbf{a.q} - \mathbf{a.v}|$ increases inside $[0, W)$, until it reaches 0 for $|h(\mathbf{q}) - h(\mathbf{v_1})| = W$. Since B is drawn uniformly at random from $[0, W]$, the following can be derived:

$$t(c, 0) = \int_0^W \frac{1}{c} f(\frac{u}{c})(1 - \frac{u}{W}) du$$

The case for $\delta > 0$ is similar. However this time, a bigger range of values $([(\delta - 1)W, (\delta + 1)W))$ for $|\mathbf{a.q} - \mathbf{a.v}|$ can satisfy $|h(\mathbf{q}) - h(\mathbf{v_1})| = \delta$. The argument regarding B is similar to above. Therefore the following can be seen:

$$t(c, \delta) = \int_{(\delta - 1)W}^{(\delta)W} \frac{1}{c} f(\frac{u}{c})(\frac{u}{W} - (\delta - 1)) du$$

$$+ \int_{(\delta)W}^{(\delta + 1)W} \frac{1}{c} f(\frac{u}{c})((\delta + 1) - \frac{u}{W}) du \qquad (3)$$

Summing up all values of $t(c, d)$ for $d \leq \delta$, we arrive at the following for $s(c, \delta)$:

$$s(c, \delta) = \int_0^{(\delta + 1)W} \frac{1}{c} f(\frac{u}{c}) du + \int_{(\delta)W}^{(\delta + 1)W} \frac{1}{c} f(\frac{u}{c})(\delta - \frac{u}{W}) du$$

With a change of variable $v = \frac{u}{c}$ we can eliminate all occurrences of c inside the integrals and take the derivative of $s(c, \delta)$ in terms of c. Given that $\int u f(u) du = -f(u)$, this will lead us to:

$$\frac{\partial s(c, \delta)}{\partial c} = \frac{1}{W}(f(\frac{(\delta + 1)W}{c}) - f(\frac{(\delta)W}{c}))$$

which is smaller than 0 for all values of $c > 0$, as $f(x)$ is monotonically decreasing. Therefore for any fixed δ, $s(c, \delta)$ is monotonically decreasing in terms of c. $\qquad \square$

Theorem 2 For any two points $\mathbf{q}, \mathbf{v} \in S$, $pr(|h(\mathbf{q}) - h(\mathbf{v})| = \delta)$ is monotonically decreasing in terms of δ.

Proof. Let $||\mathbf{q} - \mathbf{v}||_2 = c$. From Theorem 1, $pr(|h(\mathbf{q}) - h(\mathbf{v})| = \delta)$ is equal to Eq. 3. It is easy to see that if we take the derivative from this equation in terms of δ we arrive at the following:

$$\frac{\partial t(c, \delta)}{\partial \delta} = - \int_{\frac{(\delta - 1)W}{c}}^{\frac{(\delta)W}{c}} f(u) du + \int_{\frac{(\delta)W}{c}}^{\frac{(\delta + 1)W}{c}} f(u) du$$

which is smaller than zero, as the range of the two integrals is equal, the term under both is the same and is monotonically decreasing and non negative for the values under comparison. $\qquad \square$

The above two theorems indicates that l_1 can capture the distance between buckets in terms of probability of holding close data: Given bucket labels $\mathbf{b_1}, \mathbf{b_2}$ and $\mathbf{b_3}$ which are integer vectors of dimension k, if $||\mathbf{b_1} - \mathbf{b_2})||_1 < ||\mathbf{b_1} - \mathbf{b_3}||_1$, then $\mathbf{b_1}$ and $\mathbf{b_2}$ have a higher probability to hold similar data than $\mathbf{b_1}$ and $\mathbf{b_3}$.

Having a better understanding of the semantics of similar buckets, we now discuss two mappings which satisfy the two conditions mentioned above.

3.1 Linear Mapping based on Sum

We propose $\xi_{sum}(\mathbf{b}) = \sum_{i=1}^k b_i$ as an appropriate placement function which can be used to map the k-dimensional vector of integer $\mathbf{b}$, as the output of p-stable LSH, to the one dimensional peer identifier space. The intuition is that the sum treats all bucket label parts b_i equally and that minor differences in the b_i values are smoothed out by the sum leading to close $\xi_{sum}()$ values for "close" bucket labels. In the following, we show how relying on p-stable LSH and its characteristics, it satisfies both conditions above.

We first investigate condition 1. As discussed in the previous section, buckets which are likely to hold similar data have small l_1 distance to each other. Given ξ_{sum} as our mapping function we have: $|\xi_{sum}(\mathbf{b_1}) - \xi_{sum}(\mathbf{b_2})|=|(b_{11} - b_{21}) + \cdots + (b_{1k} - b_{2k})| \leq ||\mathbf{b_1} - \mathbf{b_2}||_1$. Which means if buckets with labels $\mathbf{b_1}$ and $\mathbf{b_2}$ are likely to hold similar data, $\xi_{sum}(\mathbf{b_1})$ and $\xi_{sum}(\mathbf{b_2})$ will also be close. Given the assignment of data to peers in Chord-style overlays, this results in assigning the two buckets to the same or neighboring peers in the Chord ring with high probability.

As for the second condition, assume d-dimensional points, $\mathbf{a}$ and $\mathbf{v_1}$. If elements of $\mathbf{a}$ are chosen from a Normal distribution with mean 0 and standard deviation 1, denoted as $N(0,1)$, $\mathbf{a} \cdot \mathbf{v_1}$ is distributed according to the Normal distribution $N(0, ||\mathbf{v_1}||_2)$. For not too large W, $h_{\mathbf{a}, B}(\mathbf{v_1})$ is distributed according to the Normal distribution $N(\frac{W}{2W}, \frac{||\mathbf{v_1}||_2}{W})$ where h is function of the form Eq. 1. Therefore $g(\mathbf{v_1})$ will be a k-dimensional vector, whose elements follow the above distribution. We can now benefit from a nice property of the Normal distributions under summation: $\xi_{sum}(g(\mathbf{v_1}))$ is distributed according to the Normal distribution

$$N(\frac{k}{2}, \frac{\sqrt{k} ||\mathbf{v_1}||_2}{W})$$

The global picture consisting of all data points $\mathbf{v_1}, \ldots, \mathbf{v_M}$ first projected using p-stable LSH and then mapped to $\mathbb{Z}$ by ξ_{sum}, following the Normal distribution

$$N(\frac{k}{2}, \frac{\sqrt{k \sum_i ||\mathbf{v_i}||_2^2}}{\sqrt{M} W})$$

We can therefore predict the distribution of the output of ξ_{sum}, having an estimate of the mean of data points' l_2 norm. We assume that we know the mean norm of available data, but as we will later see, this assumption is only relevant for the start-up of the system where gateway peers are inserted into the hash tables. Calculating statistics, like in our case the mean, over data distributed in large-scale systems has been well addressed in the literature (cf., e.g., [22]). In Section 4 we show how this can be used to balance the load in the network.

3.2 Linear Mapping based on Cauchy LSH

As another instance of ξ we propose the LSH function based on Cauchy distribution (1-stable) which offers a probabilistic placement of similar buckets on the same peer. More formally, for a bucket label $\mathbf{b}$,

$$\xi_{lsh_{\mathbf{a}', B'}}(\mathbf{b}) = \lfloor \frac{\mathbf{a}' \cdot \mathbf{b} + B'}{W_2} \rfloor$$

where elements of $\mathbf{a}'$ are chosen independently from a standard Cauchy distribution with probability distribution function

$$cr(x; x_0, \gamma) = \frac{1}{\pi\gamma} \frac{1}{1 + (\frac{x-x_0}{\gamma})^2}$$

where the *location* parameter $x_0 = 0$, *scale* parameter $\gamma = 1$, $W_2 \in \mathbb{R}$, and B' is chosen uniformly from $[0, W_2]$. Note that $\mathbf{b}$ denotes a bucket label and is a k dimensional vector of integers which is itself the output of an LSH function applied on a d-dimensional data point. Given that this LSH function is most suitable for the l_1 norm and that l_1 captures the dissimilarity among buckets, $\xi_{lsh_{\mathbf{a}', B'}}$ probabilistically satisfies condition 1.

The output distribution of this function is similarly predictable. Assume a bucket label $\mathbf{b_1}$. Given the characteristics of p-stable distributions, $\mathbf{a}' \cdot \mathbf{b_1}$ follows the distribution $||\mathbf{b_1}||_1 X$ where X is a Cauchy distribution. For not too large W_2, $h_{\mathbf{a}', B'}(\mathbf{b_1})$ is distributed with the probability distribution function

$$cr(x; \frac{W_2}{2W_2}, \frac{||\mathbf{b_1}||_1}{W_2})$$

a Cauchy distribution with *location* parameter $\frac{1}{2}$ and *scale* parameter $\frac{||\mathbf{b_1}||_1}{W_2}$. Now, considering all bucket labels, $\mathbf{b_1}, \dots, \mathbf{b_P}$ mapped to $\mathbb{Z}$ by $\xi_{lsh_{\mathbf{a}', B'}}$, the output follows the Cauchy distribution with *location* parameter $\frac{1}{2}$ and *scale* parameter $\frac{\sum ||b_i||_1}{PW_2}$. In this case, to be able to predict the distribution of the output, we need the mean of l_1 norms of all possible bucket labels. Since the initial hash functions $h_{\mathbf{a}, B}$ are known to all peers, this again boils down to the problem of distributed statistics calculation [22].

4. LOCAL DHT CREATION

To map a particular domain of integer values to a (subset of) peers, it is important to know the size and distribution of the domain. As discussed in Sections 3.1 and 3.2 the values generated by ξ_{sum} and ξ_{lsh} follow known distributions. We briefly describe how this information can be utilized to create local DHTs which as described in Section 1.1 maintain the data index, for a more detailed description see [19].

Consider a linear bucket space of M buckets in which we want to distribute the values generated by the ξ_{sum} mapping. The case for ξ_{lsh} follows similarly. Let μ_{sum}, σ_{sum} be the mean and the standard deviation of the values generated by ξ_{sum} (cf. Section 3). We choose the first bucket (at position 1) to be responsible for $\mu_{sum} - 2 * \sigma_{sum}$ and the last bucket (at position M) to be responsible for $\mu_{sum} + 2 * \sigma_{sum}$. We restrict ourselves to the span of two standard deviation to avoid overly broad domains and map the remaining data to the considered range via a simple modulo operation:

$$\psi(value) := (\frac{value - (\mu_{sum} - 2 * \sigma_{sum})}{4 * \sigma_{sum}} * M) \bmod M \quad (4)$$

As mentioned in Section 1.1 we want to maintain each particular LSH hash table (which is an index of the whole data points) by a subset of peers that is usually some orders of magnitude smaller than the global set of peers. To limit the responsibility of maintaining one hash table to a subset of peers, we dynamically form separate local DHTs for each hash table as follows: At system startup, we place γ peers at predefined positions (known by all peers) based on the normal distribution $N(\mu_{sum}, \sigma_{sum})$ by sampling γ values from $N(\mu_{sum}, \sigma_{sum})$ and mapping them to buckets in the range of $\{1, .., M\}$ using ψ.

For a particular number of initial peers and the sampled

values, we consider

$$\rho(value, l) := (\psi(value) + hash(l)) \bmod |G|$$

as the mapping of a (value, l)-pair to the global set of peers G, where l is a hash table id. ρ consists of two components, the previously described ψ function and $hash(l)$ as an offset for global load balancing. The peers responsible for these ρ values are invited to join (create) the particular DHTs.

The case for ξ_{lsh} is similar, just we use the *location* and *scale* parameters of the predicted Cauchy distribution in Equation 4 since mean or standard deviation are not defined for Cauchy distributions. Also at start up, peers are sampled from the predicted Cauchy distribution.

The number of peers dynamically grows inside each local DHT by overloaded peers issuing requests on the global DHT to find peers to join the local DHT on a particular position (bucket). In case of access load problems, the gateway peers can call for a global increment of the number of gateway peers, i.e., increase the number of possible gateway peers that will subsequently be hit by requests and hence invited to join the local DHTs maintaining the LSH hash tables. We can benefit from the rich related work on load balancing techniques over DHT, such as the work by Pitoura et al [26], that replicates "hot" ranges inside a Chord style DHT and then lets peers randomly choose among the replicated arcs.

4.1 Handling Churn

We will now discuss possible ways to handle churn, in particular, peers leaving the system without prior notice, but leave any detailed analysis and in particular the impact of the low level (DHT based) churn handling mechanisms to the overall performance as future work.

To handle churn it is common practise to introduce a certain degree of replication to the system. One such replication based mechanisms has been already introduced above when presenting the concept of multiple gateway peers per local DHT which solves the following problem: If a peer lookup on one of the predefined positions leads to a peer that is not currently in the local DHT as a gateway peer, that peer issues a lookup on one of the other entry points and joins the particular hash table it belongs to. We can furthermore add two more ways of replication: (i) replication of complete local DHTs and/or (ii) replication within particular local DHTs. While approach (i) is straight forward to implement it is extremely coarse and more suitable for handling access load problems (hot peers) rather than handling churn in an efficient way. Approach (ii) seems to be more suitable for handling churn: neighboring peers within each local DHT could maintain also the index of their immediate neighbors and in case of a peer failure transmit the replicas to the new peer joining the free spot. Both approaches certainly cause higher load on the system not only in terms of storage but in particular in terms of message exchanges to keep the replicas in sync. We will investigate the impact of replicated local DHTs in our future work and for this paper concentrate on the actual indexing mechanisms. Note that in addition to the replication mechanisms presented above, the underlying global DHT might use replication of routing indices as well, which is treated by us as a black box.

5. KNN QUERY PROCESSING

Given l LSH hash tables, a query point $\mathbf{q} = (q_1, ..., q_d)$ is first mapped to buckets $g_1(\mathbf{q}) \dots g_l(\mathbf{q})$ using the p-stable LSH method. The query initiator then uses one randomly selected gateway peer per local DHT as an entry to that local DHT. Subsequently, the responsible peer P for main-

taining the share of the global index that contains $g_i(\mathbf{q})$ is determined by mapping $g_i(\mathbf{q})$ to the peer identifier space using $\xi(g_i(\mathbf{q}))$, as defined above. The query is passed on to P that executes the KNN query locally using a full scan and passes the query on. We restrict the local query execution to a simple full-scan query processing since we do not want to intermingle local performance with global performance. The local query execution strategy is orthogonal to our work. For the query forwarding (i.e., routing), we consider two possible options: (i) incremental forwarding to neighboring peers or (ii) forwarding based on the multi probe technique [23]. The results return by all l hash tables are aggregated at the query initiator and the K closest points to q are returned.

5.1 Linear Forwarding

We will now define a stopping condition for the linear forwarding method. Let τ denote the distance of the Kth object w.r.t. the query object $\mathbf{q}$, obtained by a local full scan KNN search. Peer P will pass the query and the current rank-K distance τ to its neighboring peers P_{pred} and P_{succ}, causing each one single network hop. Upon receiving the query, P_{pred} and P_{succ} will issue a local full scan KNN search and compare their best result to τ (cf. Algorithm 1). If the distance d_{best} of the best document is bigger than τ, the peer will not return any results and will stop forwarding the query to its neighbor (depending on the direction, successor or predecessor). The stopping condition can be relaxed by introducing a parameter α and stop forwarding if $d_{best} > \tau/\alpha$. α allows for either a more aggressive querying ($\alpha > 1$) of neighboring peers or for an early stopping ($\alpha < 1$).

Input: query q, threshold τ, P_{init}, direction
result[] = localIndex.executeLocalKnn(q);
if *result[0].distance>τ/α* **then**
  | done;
else
  resultSet = $\emptyset$;
  for *(index=0; index<K; index++)* **do**
    if *results[index].distance<τ/α* **then**
      | resultSet.add(results[index]);
    else
      τ' = resultSet.rankKDistance();
      sendResults(resultSet, P_{init});
      forwardQuery(this.predecessor() or/and this.successor(), τ', q, P_{init}, pred or/and succ);
    end
  end
end

Algorithm 1: Top-K Style Query Execution based on the locality sensitive mapping to the linear peer space by passing the query on to succeeding or preceding peers.

5.2 Multi-Probe Based Forwarding

The multi-probe LSH method [23] slightly varies the integers in $g(\mathbf{q})$ and produces bucket ID's which are likely to hold close elements to $\mathbf{q}$. For each of these modifications, the method then probes the resulting bucket for new answerers. We adapt this technique as an alternative to the successor/predecessor based forwarding as follows: after the full scan, the peer generates a list of buckets to probe next, considering the maximum number of extra buckets. It is very likely that some of these buckets have already been visited, thus they are removed from the list. For a generated bucket $g(\mathbf{q})$ with $\xi_{sum}(g(\mathbf{q})) \notin]P.pred().id, P.id]$, the peer

issues a lookup in the local DHT and forwards the query and bucket list to the peer responsible for $\xi(g(\mathbf{q}))$. The peer that receives the query, issues a full scan, removes visited buckets from the list and forwards the query (cf. Algorithm 2).

Input: Local DHT L, query q, bucketlist, P_{init}
result[] = localIndex.executeLocalKnn(q);
while *(bucketlist.hasElement()* **do**
  b =bucketlist.removeBucket();
  bucketId =$\xi(b)$;
  if *bucketId $\in$]P.pred().id, P.id]* **then**
    | nothing to do;
  else
    P_{new} = L.lookup(bucketId);
    sendResults(P_{init}, results);
    forwardQuery(P_{new}, ,bucketlist, P_{init});
    break;
  end
end

Algorithm 2: Multi Probe based Variant of the KNN query processing.

The multi probe algorithm relies on the parameter that specifies the maximum number of probes, whereas the linear forwarding algorithm has a clear defined stopping condition. The relaxation parameter α is optional.

6. RANGE QUERY

While LSH provides a nice solution to KNN query processing in high dimensional data, unlike other methods, it is difficult to extend it to range queries. This is because specific LSH functions are designed to map points with certain distance from each other to the same hash value. The parameter r_1 in the definition of LSH functions indicates this distance (cf., Section 2.2). Therefore different indices should be made for different parameters r_1 to satisfy range search for different values of the range radius. However it is impractical to construct a different index for each possible range.

With mapping similar buckets to the same peer or to neighboring peers, we can support also range queries. Several buckets, which are likely to hold similar data to the query are investigated.

6.1 Range Query Processing

Range query processing over the linearly mapped data is different to the KNN queries as the overall objective is to return *all* items within a particular range, i.e., the stopping condition of the linear forwarding algorithm needs to be adapted. The startup phase of the query processing is the same as described in Section 5 for the KNN query algorithms: for a given query we determine the peer responsible for the bucket to which this point is mapped.

Once the starting peer is known and receives the query, it will first execute the query locally using a full scan and return all matching items to the query initiating peer, i.e., send all items within range $< r$. Subsequently, it will forward the query to its neighboring peers. The forwarding stops at a peer that does not have a single matching item.

While this processing seems to be intuitive it has the following serious drawback caused by an observation illustrated in Figure 3. The range of the query is indicated by the long arrow, the spot in the middle represents the query point. Now, the three circles indicate the responsibility of peers for the data items. Obviously, the data in the inner circle can

be processed since it is maintained by the initial peer hosting also the query point. Then, however, due to the often naturally clustered data, the following up peer does not maintain any items in the desired range hence causing the algorithm to stop when having covered the range to the second inner circle, thus missing relevant items.

To overcome this problem of "empty" ranges inside the query range, we opt for sampling the range, i.e., starting separate range queries at predefined position.

The idea is to sample s points in the range $peer_l, ..., peer_u$

Unfortunately, the data distribution inside this range is not known a priori. Furthermore, it depends on the query point and on the range itself, which makes it not tractable to pre-compute. This in particular means that even though the actual data distribution inside the hash tables is known (to the peers maintaining it) it cannot be used to predict the query dependent range of peers to visit.

In absence of any knowledge of data distributions, we employ a simple sampling method that divides the range in equally sized sub-ranges. For each sub-range, the query is forwarded to one responsible peer. Subsequently, each peer starts the range query processing and as described above, each processing thread stops when (i) an already queried peer is met or (ii) a peer does not have any single item within the specified range. We will now describe the process of range estimations.
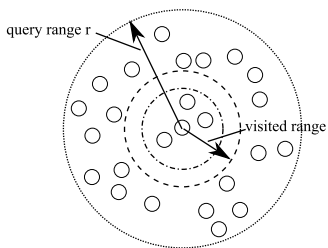

Figure 3: Illustration of the problem we face when processing range queries over the linearly mapped data. The "empty" ranges cause the algorithm to stop before the full range has been explored.

6.2 Range Prediction

We will now try to derive query dependent lower and upper bounds for the values generated by ξ. As we will later see, these estimates can be used to enable efficient parallelized in-hash-table query processing.

Recall from Section 2.2 that the output of the considered LSH hash function is a k-dimensional vector of integer values where each value corresponds to one of the k hash functions of the form of Equation 1.

Assume a query point $q = (q_1, ..., q_d)$ and a range r. Let furthermore be $a^{(+)} = argmax_j\{a_{ij}\}$ and $a^{(-)} = argmin_j\{a_{ij}\}$ the positions of the largest and smallest values of elements of one of the k vectors $\mathbf{a}_i$. The idea is to select samples from the d dimensional space that are in distance r of q and produce a bucket label with maximum l_1 difference from $g_{\mathbf{a},b}(\mathbf{q})$ from Equation 2. These samples will be mapped into the linear space using $h_{\mathbf{a},b}$ and ξ.

To construct these samples, we repeat the following for all k vectors of $\mathbf{a}_i$. We add to the query vector at the maximal and minimal value positions of vector $\mathbf{a}_i$ the query range r. More formally, we generate the upper range point as $\mathbf{q_i}^{(+)} := \mathbf{q} + \mathbf{j}_{a_i^{(+)}} * r$ and the lower range point as $\mathbf{q_i}^{(-)} := \mathbf{q} - \mathbf{j}_{a_i^{(-)}} * r$ where $\mathbf{j}_i$ is the ith unit vector.

	Flickr	Corel
#data points	1,000,000	60,000
#dimensions	282	89
#peers in Global DHT (N)	1,000,000	100,000
#peers per Local DHT (n)	1000	100

Table 1: Data Sets and Overlay setup

Using these generated samples of points in distance r we apply the standard techniques using LSH hashing and mapping through ξ to determine the upper and lower bound ξ values

$$upper(\mathbf{q}, r) := argmax_i\{\xi(g(\mathbf{q}^{(+)}))\}$$

$$lower(\mathbf{q}, r) := argmin_i\{\xi(g(\mathbf{q}^{(-)}))\}$$

Assume $peer_u$ and $peer_l$ to be the two peers responsible for the above two values. According to condition 1 of an appropriate ξ function, peers which fall between these two peers in the Chord style ring, are most likely to hold data in range r of the query point $\mathbf{q}$. Since the output distribution of ξ is known the above values can be used to estimate the number of peers which should be contacted to answer a query with range r.

7. EXPERIMENTS

7.1 Experimental Setup

We have implemented a simulation of the proposed system and algorithms using Java 1.6. The simulation runs on a 2x2.33 GHz Quad-Core Intel Xeon CPU with 8GB RAM. The data is stored in an Oracle 11g database.

Data Sets and Overlay setup

Flickr: We used a crawl of Flickr consisting of $1,000,000$ images represented by their MPEG7 visual descriptors. The total number of dimensions per image is 282 and contains descriptors such as *Edge Histogram Type* and *Homogeneous Texture Type*. For the global DHT we considered a population of $1,000,000$ peers and each replica of the data set is maintained by a local DHT of 1000 peers.

Corel: For the second data set we experimented on $60,000$ photo images from the Corel data set as previously used in, e.g. [24] [1]. Each image has 89 dimensions in this data set. In this case we assumed a global DHT of $100,000$ peers and 100 peers per local DHT.

For both datasets, we use the Euclidean distance to measure the distances between points, treating all dimensions equally and without preprocessing the data. As query points we chose 100 points randomly from each of the datasets. All performance measures are averaged over 100 queries. $K = 20$ in all KNN experiments. Table 1 summarizes the data set and overlay setup parameters.

Methods under Comparison

To evaluate our data placement methods, we distribute the data among peers once with ξ_{sum} and once with ξ_{lsh}. We experimented with different values of LSH parameters: k, W and W_2 and here report the best performances achieved. In the results, unless otherwise stated, the default values are $k = 20$, $W = 5$ and $W_2 = 3$ for the Flickr data set and $k = 20$, $W = 50$ and $W_2 = 1.25$ for the Corel data set. For each of these mapping functions we consider the following query processing methods:

Simple: This is the baseline query processing algorithm. At query time, the whole local index of the peer which is

---

[1] available under:
http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures

responsible for the mapped LSH bucket using ξ is scanned without further forwarding. This is used both for KNN and range queries.

MProbe: At KNN query time we use the multi-probing based algorithm as described in Section 5.2, fixing the number of probes to 100.

Linear: At query time the linear forwarding algorithm, Section 5.1, is used with appropriate stopping conditions for KNN or range search.

Sample: This is the sampling-based method described in Section 6.1 which is dedicated to range query processing.

Sahin: To compare with state of the art, we have implemented the method described in Section 2.1 by Sahin et. el [29]. In order to fairly compare against this method, we follow the same index creation of Section 4 where replicas of the data are maintained by smaller rings. We have experimented with different number of reference vector sizes and different number of references for publishing indices. We report here the best performance results which achieve a fair load balance as well. The reference vector size is set to 32 and reference points are selected uniformly at random from the whole data set. To achieve a fair load balance, we employed multi-level reference sets as described in the initial work, however increasing the number of references used for publishing the indices proved to be more effective. Therefore only one level of references is used and the number of references used for publishing indices is set to 4. The initial work of [29] employs the *Simple* query processing method as explained in 2.1. In addition to that we also experimented processing queries with our *Linear* method. The *MProbe* method is not applicable here, as it depends on the LSH buckets. Processing range queries are not discussed in [29]; we also experimented only the KNN queries with this method.
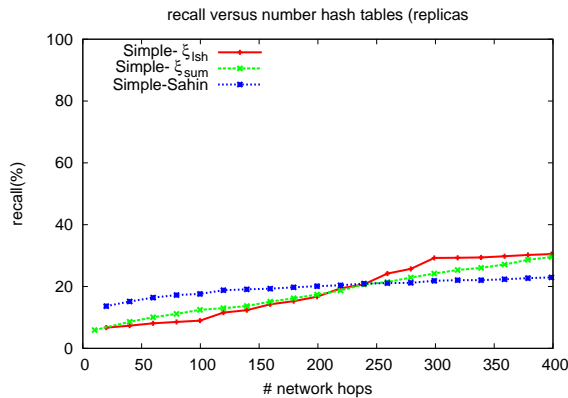

Figure 4: Recall versus number of DHT lookups for different data placement methods employing *Simple* query processing for the Flickr data set.

Measures of Interest

Gini Coefficient: As for a measure of load imbalances we consider the Gini coefficient of the load distribution, that is defined as $G = 1 - 2\int_0^1 L(x)dx$ where $L(x)$ is the Lorenz curve of the underlying distribution. Pitoura et al [27] show that the Gini coefficient is the most appropriate statistical metric for measuring load distribution fairness. The Gini coefficient, apart from the other three measures, is query in-
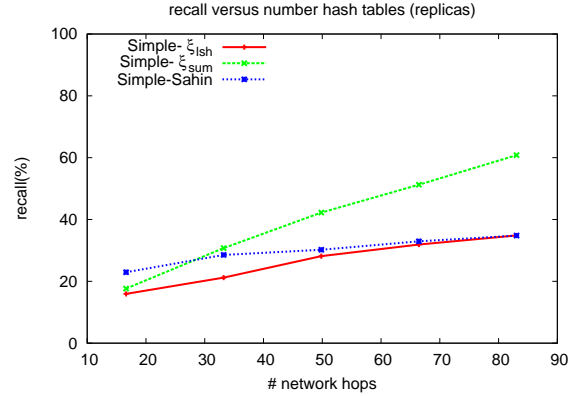

Figure 5: Recall versus number of DHT lookups for different data placement methods employing *Simple* query processing for the Corel data set.

dependent and measured once for each benchmark to report on the storage load distribution.

Number of Network Hops: We count the number of network hops during the query execution. Network hops are one of the most critical parameters in making distributed algorithms applicable in large-scale wide-area networks. Each DHT lookup causes $logn/2$ or $logN/2$ network hops (i.e., local or global DHT). Hence, we count the number of local and global lookups and translate this to the overall number of network hops. The cost for local query execution is considered to be negligible in our scenario, as the network cost is clearly the dominating factor: One single network hop in a wide-area costs in average around $100ms$, which overrules the I/O cost, induced by a standard hard disk, with approximately $8ms$ for disk seek time plus rotation latency and $100MB/s$ transfer rate for sequential accesses, in case of local disk access.

Relative Recall: For the effectiveness metric, we report on the relative recall, i.e., the number of relevant data points among returned data points. The relevance is defined by the full-scan run over the entire data set to determine the K nearest points to a query point for KNN queries. For range queries, all data points in range r of the query point are relevant. It should be noted that since we are ranking all candidate objects and returning only the top K in KNN queries and only the points within distance r of the query point in range queries, precision is equal to relative recall and we do report it separately.

Error Ratio: Given that LSH is an approximate algorithm, we also measured the *Error Ratio* which measures the quality of approximate nearest neighbor search as defined in [17]. $\frac{1}{K}\sum_{i=1}^{K}\frac{d_{LSH_i}}{d_{true_i}}$ where d_{LSH_i} is the distance of query point to its i-th nearest neighbor found by LSH and d_{true_i} is its distance to its true i-th nearest neighbor. Since this measure does not add new insight over relative recall and due to space constraints we do not report it here.

7.2 Experimental Results

We first investigate the effect of employing ξ_{sum} and ξ_{lsh} on the load distribution and compare this against the *Sahin* data placement. As seen in Table 2 for both Flickr and Corel data sets, the Gini coefficients of all different load distributions fall in the range of $[0.4, 0.6]$ which is a strong indicator of a fair load distribution [26].

Data set	Sahin	ξ_{sum}	ξ_{lsh}
Flickr	0.42	0.52	0.41
Corel	0.40	0.46	0.57

Table 2: Gini Coefficient when distributing 2 replicas of the data sets
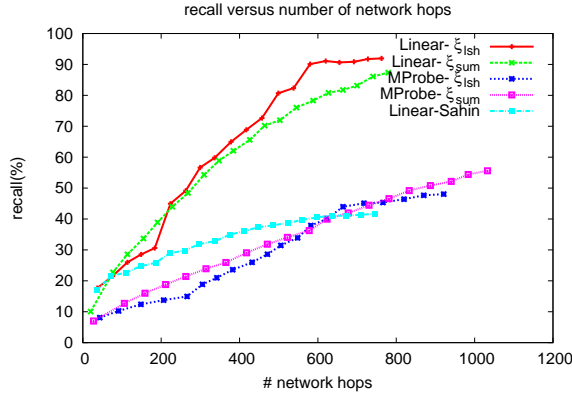

Figure 6: Recall versus number of DHT lookups for the Flickr data set representing *Linear* with the three different placement methods and *MProbe* with ξ_{sum} and ξ_{lsh}.

7.2.1 KNN query Results

We now show the results obtained for the KNN search. Figures 4 and 5 show the obtained recall when queries are processed by the *Simple* method. We have varied the number of hash tables (or respectively replicas for *Sahin*) from 2 to 40 for the Flickr data set and from 2 to 10 for the Corel data set. For the Corel dataset ξ_{sum} achieves better recall compared to *Sahin* and ξ_{lsh}. ξ_{sum} and ξ_{lsh} obtain better recalls for number of replicas more than 24 for the Flickr data set which has higher dimensionality . We observe that the obtained recall for all three placement methods with the *Simple* query processing algorithm is quite low even when increasing the number of hash tables (replicas). It should be noted that while employing the *Simple* method for processing queries, the number of incurred network hops for answering each query is equal to number of hash tables (replicas) times $logN/2$, where N is the number of peers in the global DHT. In this case, only one peer is visited in each local DHT maintaining a hash table (replica) of the data set.

Figures 6 and 7 show recall versus number of network hops for three placement methods, this time using *Linear* and *MProbe* processing algorithms respectively for the Flickr and Corel data sets. We see a big increase in recall compared to when processing queries using *Simple* for both data sets and all three placement methods. We have varied the number of hash tables (replicas) exactly like the previous experiment, from 2 to 40 for Flickr, and from 2 to 10 for Corel. We show on the x-axis the number of network hops incurred which corresponds to the number of hash tables (replicase) and number of times the query is forwarded in each local DHT maintaining a hash table (replica). As can be seen for both data sets, the combination of ξ_{sum} and ξ_{lsh} with the *Linear* processing algorithm achieves the best recall while incurring not many network hops. This confirms that ξ_{sum} and ξ_{lsh} preserve the locality, i.e., they group buckets with similar content to the same or neighboring peers. ξ_{sum} and ξ_{lsh} achieve similar recall in both data sets, while *Sahin*'s quality of result degrade drastically when processing the Flickr
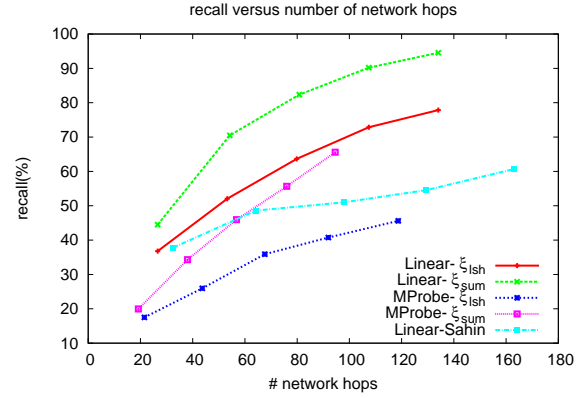

Figure 7: Recall versus number of DHT lookups for the Corel data set representing *Linear* with the three different placement methods and *MProbe* with ξ_{sum} and ξ_{lsh}.

data set. This observation again shows better scalability of our algorithm with respect to data dimensionality, which is due to LSH characteristics. *MProbe* achieves better recall compared to *Simple*, but does not perform as well as *Linear*: number of incurred network hops is more, as each forward in a local DHT maintaining a hash table results in $logn/2$ hops, where n is the number of peers maintaining that ring. We have also summarized these results in Tables 3 and 4 to better compare these methods with respect to network load (number of times the data is replicated in the network). For example let us consider the 13th and 15th rows of Table 3. With *Linear* the ξ_{lsh} data placement achieves more than twice better recall at the expense of only 0.01 more number of network hops while having the same network load as *Sahin*. The best achieved recall is shown in bold in both tables.

placement	l	Relative Recall in % (#Network Hops)		
		Simple	**MProbe**	**Linear**
Sahin	2	13.65% (19)	-	17.00% (34)
ξ_{sum}	2	6.90%(19)	8.50%(27)	**18.65% (19)**
ξ_{lsh}	2	6.70% (19)	8.10% (43)	17.65% (36)
Sahin	10	17.60% (99)	-	25.80% (185)
ξ_{sum}	10	12.45% (99)	21.40% (262)	**38.90% (190)**
ξ_{lsh}	10	8.95% (99)	14.95% (266)	30.60% (183)
Sahin	20	20.10% (199)	-	34.95% (375)
ξ_{sum}	20	17.40% (199)	34.05% (522)	62.05% (384)
ξ_{lsh}	20	16.70% (199)	28.60% (471)	**64.95% (378)**
Sahin	30	21.85% (298)	-	39.70% (559)
ξ_{sum}	30	24.20% (298)	46.60% (781)	78.30% (587)
ξ_{lsh}	30	29.25% (298)	43.95% (664)	**90.10% (579)**
Sahin	40	22.95% (398)	-	41.65% (746)
ξ_{sum}	40	29.55% (398)	55.60% (1032)	87.35% (779)
ξ_{lsh}	40	30.50% (398)	48.05% (921)	**91.95% (762)**

Table 3: Measuring recall and number of network hops for different number of hash tables, for different placement and processing methods the Flickr data set.

7.2.2 Range query Results

We now report the results obtained for range searches. For each data set the radius of the range query is chosen such that the number of possible results are reasonable, i.e. for Flickr this ranges from 27 to 562, while for Corel it ranges from 17 to 194. Figure 8 shows a general view of the effect of varying the range on recall. As discussed also in Section

placement	l	Relative Recall in % (#Network Hops)		
		Simple	MProbe	Linear
Sahin	2	22.95% (16)	-	37.85% (32)
ξ_{sum}	2	17.65% (16)	19.95% (19)	**44.49% (26)**
ξ_{lsh}	2	15.95% (16)	17.50% (21)	36.80% (26)
Sahin	10	34.80% (83)	-	60.70% (163)
ξ_{sum}	10	60.79% (83)	65.35% (94)	**94.55% (134)**
ξ_{lsh}	10	34.85% (83)	45.60% (118)	77.84% (134)

Table 4: Measuring recall and number of network hops for different number of hash tables, for different placement and processing methods the Corel data set.
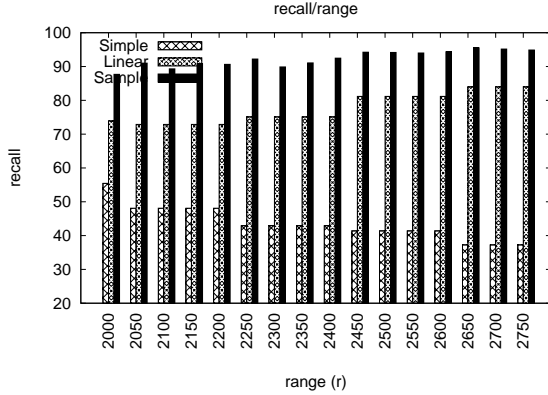

recall/range

Figure 8: The effect of varying the range on recall. The results are shown for #hash tables=20 and ξ_{sum} as the placement function for the Corel data set.

6.1 recall in the *Simple* method can fall when the radius of range increases. An example of this effect is shown in Figure 8. However, our Sampling method proves to be effective at maintaining high recall as the radius changes. Tables 5 and 7 show the results for ξ_{sum}, while Tables 6 and 8 report on ξ_{lsh} for the two data sets. Clearly, *Sample* performs very well at achieving good recall for different choices of the range in both data sets. Our *Linear* method obtains smaller recall compared to *Sample*, however the number of network hobs incurred by this method is considerably less than *Sample*. The best achieved recalls are shown in bold.

range	l	Relative Recall in % (#Network Hops)		
		Simple	Linear	Sample
2000	2	60.98% (17)	77.55% (24)	**80.09% (41)**
2000	10	81.04% (83)	95.95% (119)	**95.00% (200)**
2150	2	49.48% (17)	70.17% (25)	**75.26% (43)**
2150	10	74.10% (83)	96.01% (124)	**95.47% (211)**
2300	2	42.68% (17)	72.10% (26)	**78.02% (47)**
2300	10	68.87% (83)	95.82% (130)	**96.54% (224)**

Table 5: Measuring recall and number of network hops for different range radius' and number of hash tables, for different processing methods under comparison with ξ_{sum} as the placement function for the Corel data set.

8. CONCLUSIONS

We have presented a robust and scalable solution to the distributed similarity search problem over high dimensional data. Having investigated the characteristics of the existing centralized LSH based methods, we have devised an algorithm to distribute the *p*-stable LSH method considering the requirements that arise in distributed settings. Our proposed locality preserving mapping, brings together two *con-*

range	l	Relative Recall in % (#Network Hops)		
		Simple	Linear	Sample
2000	2	51.88% (17)	63.22% (24)	**70.73% (42)**
2000	10	66.95% (83)	82.86% (116)	**93.16% (210)**
2150	2	40.05% (17)	54.03% (24)	**61.52% (44)**
2150	10	54.31% (83)	77.98% (120)	**91.99% (222)**
2300	2	33.23% (17)	50.39% (25)	**61.40% (48)**
2300	10	49.36% (83)	76.46% (124)	**94.13% (239)**

Table 6: Measuring recall and number of network hops for different range radius' and number of hash tables, for different processing methods under comparison with ξ_{lsh} as the placement function for the Corel data set.

range	l	Relative Recall in % (#Network Hops)		
		Simple	Linear	Sample
200	2	79.38% (20)	80.67% (26)	**81.32% (53)**
200	10	80.82% (100)	85.00% (138)	**86.54% (264)**
200	20	82.02% (199)	87.06% (275)	**89.98% (530)**
200	30	84.03% (299)	89.96% (414)	**93.04% (795)**
200	40	86.45% (399)	92.92% (553)	**95.88% (1060)**
225	2	61.82% (20)	63.14% (27)	**64.02% (54)**
225	10	64.85% (100)	70.44% (141)	**72.75% (269)**
225	20	67.82% (199)	75.42% (282)	**79.09% (540)**
225	30	70.66% (299)	80.70% (423)	**85.63% (812)**
225	40	74.29% (399)	84.85% (566)	**89.99% (1086)**
250	2	41.63% (20)	44.10% (29)	**45.45% (56)**
250	10	44.58% (100)	53.51% (149)	**57.45% (282)**
250	20	47.69% (199)	60.30% (296)	**66.15% (567)**
250	30	51.26% (299)	66.95% (449)	**73.89% (857)**
250	40	55.20% (399)	72.73% (600)	**80.16% (1147)**
275	2	22.00% (20)	26.36% (34)	**28.70% (66)**
275	10	25.59% (100)	40.21% (177)	**43.99% (330)**
275	20	29.81% (199)	49.33% (351)	**55.87% (659)**
275	30	33.77% (299)	58.03% (530)	**67.12% (999)**
275	40	37.47% (399)	64.36% (707)	**74.62% (1335)**
300	2	11.80% (20)	18.94% (46)	**22.98% (89)**
300	10	14.92% (100)	36.50% (234)	**43.14% (434)**
300	20	18.85% (199)	48.42% (466)	**57.06% (870)**
300	30	23.16% (299)	59.27% (707)	**68.77% (1329)**
300	40	27.55% (399)	66.62% (947)	**76.55% (1784)**

Table 7: Measuring recall and number of network hops for different range radius' and number of hash tables, for different processing methods under comparison with ξ_{sum} as the placement function for the Flickr data set.

tradictory conditions of efficient and high quality similarity search in distributed se1ttings: Enabling probabilistic placement of similar data on the same peer or neighboring peers, while achieving a fair load balancing. We have shown how to create the index, leveraging our proposed mapping and its characteristics. We have theoretically proved the locality preserving properties of our mapping and devised efficient algorithms for both K-nearest neighbor and range queries. To our knowledge this is the first work enabling similarity range queries over LSH indices. Our experimental evaulation shows major performance gains compared to state-of-the-art. We believe that our approach is thus well-positioned to become a fundamental building block towards applying LSH based methods in real world, distributed applications.

9. REFERENCES

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Advances in metric embedding theory. In *STOC*, 2006.
[2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 2006.
[3] Sunil Arya and David M. Mount. Approximate range searching. *Comput. Geom.*, 2000.
[4] Vassilis Athitsos, Michalis Potamias, Panagiotis

		Relative Recall in % (#Network Hops)		
range	l	Simple	Linear	Sample
200	2	78.90% (20)	81.44% (27)	**81.49% (51)**
200	10	79.66% (100)	84.08% (134)	**84.58% (256)**
200	20	81.06% (199)	86.89% (270)	**88.10% (518)**
200	30	84.75% (299)	90.59% (409)	**92.54% (780)**
200	40	85.89% (399)	92.07% (542)	**93.92% (1034)**
225	2	61.31% (20)	64.55% (28)	**65.34% (53)**
225	10	61.94% (100)	67.05% (136)	**68.82% (261)**
225	20	64.63% (199)	73.50% (276)	**76.44% (531)**
225	30	69.53% (299)	80.54% (418)	**84.36% (800)**
225	40	70.41% (399)	82.75% (552)	**86.98% (1058)**
250	2	41.59% (20)	47.53% (31)	**48.36% (57)**
250	10	42.62% (100)	51.36% (144)	**54.56% (272)**
250	20	45.43% (199)	59.17% (294)	**65.20% (558)**
250	30	52.92% (299)	69.43% (448)	**78.37% (845)**
250	40	54.72% (399)	72.25% (586)	**80.91% (1112)**
275	2	22.45% (20)	29.58% (36)	**31.82% (65)**
275	10	24.26% (100)	35.38% (159)	**40.07% (300)**
275	20	27.09% (199)	45.42% (335)	**51.63% (628)**
275	30	35.82% (299)	62.10% (522)	**71.02% (970)**
275	40	37.76% (399)	64.63% (670)	**73.61% (1256)**
300	2	11.61% (20)	21.90% (46)	**24.41% (80)**
300	10	13.46% (100)	31.17% (194)	**35.61% (364)**
300	20	17.22% (199)	45.34% (420)	**52.43% (791)**
300	30	25.04% (299)	64.39% (671)	**73.05% (1235)**
300	40	27.22% (399)	67.00% (846)	**76.10% (1574)**

Table 8: Measuring recall and number of network hops for different range radius' and number of hash tables, for different processing methods under comparison with ξ_{lsh} as the placement function for the Flickr data set.

Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, 2008.

[5] Farnoush Banaei-Kashani and Cyrus Shahabi. Swam: a family of access methods for similarity-search in peer-to-peer data networks. In *CIKM*, 2004.

[6] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW*, 2005.

[7] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Symposium on Computational Geometry*, 1990.

[8] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. The pyramid-technique: towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27(2), 1998.

[9] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *ICDT*, 1999.

[10] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM*, 2004.

[11] Erik Buchmann and Klemens Böhm. Efficient evaluation of nearest-neighbor queries in content-addressable networks. In *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments*, pages 31–40, 2005.

[12] Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Comput. Geom.*, 39(1):24–29, 2008.

[13] Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-ring: an efficient and robust p2p range index structure. In *SIGMOD*, 2007.

[14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, 2004.

[15] Christos Doulkeridis, Akrivi Vlachou, Yannis Kotidis, and Michalis Vazirgiannis. Peer-to-peer similarity search in metric spaces. In *VLDB*, 2007.

[16] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. A content-addressable network for similarity search in metric spaces. In *DBISP2P*, 2005.

[17] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[18] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[19] Parisa Haghani, Sebastian Michel, Philippe Cudré-Mauroux, and Karl Aberer. LSH at large - distributed knn search in high dimensions. In *WebDB*, 2008.

[20] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang 0003. idistance: An adaptive b^+-tree based indexing method for nearest neighbor search. *TODS*, 30(2), 2005.

[21] H. V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, Rong Zhang, and Aoying Zhou. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *ICDE*, 2006.

[22] Márk Jelasity, Alberto Montresor, and Özalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3), 2005.

[23] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *VLDB*, 2007.

[24] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Kriengkrai Porkaew, Sharad Mehrotra, and Thomas S. Huang. Supporting ranked boolean similarity queries in mars. *IEEE Trans. Knowl. Data Eng.*, 10(6), 1998.

[25] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, 2006.

[26] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Replication, load balancing and efficient range query processing in dhts. In *EDBT*, 2006.

[27] Theoni Pitoura and Peter Triantafillou. Load distribution fairness in p2p data management systems. In *ICDE*, 2007.

[28] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM*, 2001.

[29] Ozgur D. Sahin, Fatih Emekçi, Divyakant Agrawal, and Amr El Abbadi. Content-based similarity search over peer-to-peer systems. In *DBISP2P*, 2004.

[30] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[31] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, 2003.

[32] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the distance: An efficient method to knn processing. In *VLDB*, 2001.

[33] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. Technical report, Department of Computer Science, Princeton University, May 2004.