LSH At Large – Distributed KNN Search in High Dimensions[*]

Parisa Haghani [†] , Sebastian Michel [†] , Philippe Cudré-Mauroux [‡] , Karl Aberer [†]

[†] EPFL
Lausanne, Switzerland
firstname.lastname@epfl.ch

[‡] MIT
USA
pcm@csail.mit.edu

ABSTRACT

We consider K-Nearest Neighbor search for high dimensional data in large-scale structured Peer-to-Peer networks. We present an efficient mapping scheme based on p-stable Locality Sensitive Hashing to assign hash buckets to peers in a Chord-style overlay network. To minimize network traffic, we process queries in an incremental top-K fashion leveraging on a locality preserving mapping to the peer space. Furthermore, we consider load balancing by harnessing estimates of the resulting data mapping, which follows a normal distribution. We report on a comprehensive performance evaluation using high dimensional real-world data, demonstrating the suitability of our approach.

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, nearest neighbor, high dimensionality, p2p

1. INTRODUCTION

We are currently witnessing a rapid growth of online information, triggered by the popularity of the Internet and the huge amounts of user-generated contents from Web 2.0 applications. As it becomes increasingly easy to create and publish information on the Internet, the need for efficiently managing user-generated data gets more and more pressing. User-generated data today range from simple text snippets to (semi-) structured documents and multimedia content. The number of features that can be considered for indexing those rich and heterogeneous pieces of data is growing dramatically. Furthermore, as the data sources are naturally distributed in large-scale networks, traditional centralized indexing technique 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 exact as well as similarity search over this data.

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) [24, 22], for example, allow efficient key lookups in logarithmic number of routing hops but are typically limited to exact or range queries. K-Nearest Neighbor (KNN) search in high dimensional data has been a popular research topic in the last years [5, 12, 25, 4, 9]. Given a set of data points represented in high dimensional space and a distance measure between them, the goal is to efficiently return the K-Nearest neighbors of a query point. Existing approaches to the problem either focus on centralized settings, as cited above, or rely on preprocessing data centrally or assume data ownership by peers in a hierarchical P2P setting [11, 23, 10].

In this paper we consider KNN search over high dimensional data in structured overlay networks. Inspired by the idea of Locality Sensitive Hashing (LSH) technique [12, 9] which probabilistically assigns similar data to the same bucket in a hash table, we devise a locality preserving mapping to place buckets likely to hold similar data on the same peer. We minimize the number of network hops required to retrieve the neighbors to decrease both network traffic and the overall response time. Furthermore, we consider load balancing by harnessing estimates of the resulting data (bucket) mapping, which follows a normal distribution.

1.1 Contribution and Outline

With this work we make the following contributions: (i) We review existing LSH schemes, investigate the difficulties in distributing them and devise new means to apply them in distributed settings. (ii) We present a novel technique to map LSH buckets to peers in a large-scale network, considering load balancing issues during the data placement. (iii) We present a top-K algorithm to process distributed KNN queries by harnessing our locality preserving mapping of buckets to peers. (iv) We experimentally evaluate the efficiency and effectiveness of our approach using real-world data.

The rest of this paper is organized as follows. Section 2

[*] The work presented in this paper was partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and by the European project NEPOMUK No FP6-027705.

Proceedings of the 11th International Workshop on Web and Databases (WebDB 2008), June 13, 2008, Vancouver, Canada

discusses related work. In Section 3, we extend LSH techniques to distributed settings by introducing a linear mapping between the buckets and the domain of the peers. Section 4 concentrates on the creation of the distributed index. We present the query processing algorithm in Section 5. Section 6 is devoted to the experimental evaluation of our approach. We conclude the paper in Section 7.

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-base* approaches such as the R-tree [13] and K-D trees [3], which perform very well when data dimensionality is not high but degrade to linear search for high enough dimensions [5]. The Pyramid [4] and iDistance[25] 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 [12] 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 [2, 9, 17, 19].

With the emergence of the P2P paradigm [24, 22], there has been a tendency to leverage the power of distributed computing by sharing the costs incurred by such methods over a set of machines. A number of P2P approaches, such as [8, 6, 7] have been proposed for similarity search, but they are either dedicated to one dimensional data or do not consider very high dimensional data. MCAN [11] uses a pivot-based technique to map the high dimensional metric data to an N-dimensional vector space, and then uses CAN [22] 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 [1] 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 [26] and VBI-tree [15] both rely on *tree-based* approaches which do not scale well when data dimensions are high. Recently, SimPeer [10] was proposed, which uses the principle of iDistance [14] 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.

3. DISTRIBUTING LSH

3.1 Locality Sensitive Hashing

The basic idea behind the *LSH-based* approaches is the application of *locality sensitive hashing* functions. The salient property of these functions is that they map, with high probability, similar objects (represented in the d-dimensional vector space) to the same hash bucket, i.e., related objects are more probable to have 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 for close points. At query time, the KNN search is performed by hashing the query point to one bucket per hash table and then to rank all discovered objects by their distance to the query point. The closest K points are returned as the final result.

In the last years, the development of locality sensitive hash function has been well addressed in the literature. In this work, we consider the family of LSH functions based on p-stable distributions [9] which are most suitable for l_p norms. In this case, for each data point $\mathbf{v}$, the hashing scheme considers k independent hash functions of the form $h_{\mathbf{a},b}(\mathbf{v}) = \lfloor\frac{\mathbf{a}\cdot\mathbf{v}+B}{W}\rfloor$ 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 chosen uniformly from $[0, W]$. Each hash function maps a d-dimensional data point onto the set of integers. With k such hash functions, the final result is a vector of length k of the form $g(\mathbf{v}) = (h_{\mathbf{a_1},B_1}(\mathbf{v}), ..., h_{\mathbf{a_k},B_k}(\mathbf{v}))$. We use the Normal distribution as our p-stable distribution, which is 2-stable and is appropriate for the Euclidean distance function.

In order to achieve high search accuracy, multiple hash tables need to be constructed. Experimental results [12] 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, that is, in order to visit all hash tables (which is needed to answer the KNN query) a lot of peers may need to be contacted. Solutions to this shortcoming in centralized settings [17, 19] suggest investigating more than one bucket in each hash table. The main idea is that we can guess which other buckets 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 cause $O(log\, n)$ network hops. In the following subsection, we discuss and introduce a mapping scheme, which allows 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 nicely balancing the load in the network.

3.2 Mapping LSH to the peer identifier space

We consider a network of n peers $P_1, ..., P_n$ connected by a DHT that is organized in a cyclic ID space, such as in Chord [24]. Every node is responsible for all keys with identifiers between the ID of its predecessor node and its own ID.

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 $\xi : I^k \rightarrow \mathbb{N}$.

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.

We choose $\xi_{sum}(\mathbf{x}) = \sum_{i=1}^{k} x_i$ as ξ, and present how, relying on p-stable LSH and its characteristics, it satisfies both conditions above. Figure 1 shows an illustration of the overall mapping from the d-dimensional space, to the k-dimensional LSH buckets, to finally the peer identifier space.

We first investigate condition 1. As discussed in [17] for p-stable LSH, buckets which are likely to hold similar data have small l_1 distance to each other. Given ξ_{sum} as our mapping function this 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 be close which given the assignment of data to peers in Chord-style overlays, results in assigning them to the same peer with high probability.

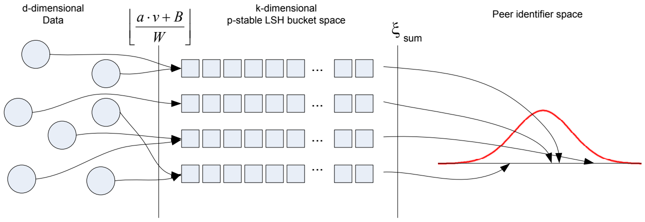
d-dimensional Data
$\left\lfloor \frac{a \cdot v + B}{W} \right\rfloor$
k-dimensional p-stable LSH bucket space
ξ sum
Peer identifier space
...

Figure 1: Illustration of the two level mapping from the d-dimensional space to the peer identifier space.

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 $N(0,||\mathbf{v_1}||)$, where $||\mathbf{v_1}||$ is the Euclidean norm of $\mathbf{v_1}$. For not too large W, $h_{\mathbf{a},B}(\mathbf{v}_1) = \lfloor \frac{\mathbf{a} \cdot \mathbf{v_1} + B}{W} \rfloor$ is distributed according to the Normal distribution $N(\frac{W}{2W}, \frac{||\mathbf{v_1}||}{W})$ where W and B satisfy the conditions given in 3.1. Therefore $g(\mathbf{v_1})$ will be a k-dimensional vector, whose elements follow the Normal distribution $N(\frac{1}{2}, \frac{||\mathbf{v_1}||}{W})$. We can now benefit from a nice property of the normal distributions under summation: ξ_{sum}(g($\mathbf{v_1}$)) is distributed according to the Normal distribution $N(\frac{k}{2}, \frac{\sqrt{k}||\mathbf{v_1}||}{W})$. 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 $\mathbb{R}$ by ξ_{sum}, following the Normal distribution $N(\frac{k}{2}, \frac{\sqrt{k \sum_i ||\mathbf{v_i}||^2}}{\sqrt{M}W})$. We can therefore predict the distribution of the output of ξ_{sum}, having an estimate of the mean of data points' Euclidean 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., [16]). In the next section we show how this can be used to balance the load in the network.

4. INDEX 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. For instance, mapping integers from the possible span of 32bit values to peers fails in our case, since the values generated by ξ_{sum} are not spread across the whole domain. However, the values generated by ξ_{sum} follow a normal distribution as discussed above. Thus we can benefit from a nice property of this distribution known as the *68-95-99.7* rule. This rule says that 68% of the data lies within one standard deviation from the mean. 95% of the data lies within two standard deviations, and 99.7% of the data lies within three standard deviations.

Consider a linear bucket space of M buckets in which we want to distribute the values generated by the ξ_{sum} mapping. Let μ_{sum}, σ_{sum} be the mean and the standard deviation of the values generated by ξ_{sum} (cf. Section 3.2). 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:

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

We want to maintain one particular hash table by a subset of peers that is usually some orders of magnitude lower than the global number of peers. To limit the responsibility of maintaining one hash table to that subset of peers, we dynamically form separate distributed hash tables as follows (cf. Algorithm 1): 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

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

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 resonsible for these ρ values are invited to join (create) the particular DHTs.

```
Input: Global DHT G, number of gateways $\gamma$
init($N(\mu_{sum}, \sigma_{sum})$);
for (tableId=0; tableId<l;tableId++) do
    sampleSet = $\emptyset$;
    for i=0; i<$\gamma$; i++ do
        sample = $N(\mu_{sum}, \sigma_{sum}).nextRandom()$;
        sampleSet.add(sample);
        P = G.lookup($\rho(sample, tableId)$);
        if tableId==0 then
            P.createDHT();
        else
            $P'$ = G.lookup($\rho(sampleSet.getRandom(),$
            $tableId)$);
            P.join($P'$);
        end
    end
end
```

Algorithm 1: Initial Algorithm to build up the l hash tables that contain the gateway peers, drawn from the global peer population

Algorithm 1 shows the initial algorithm to build up the distributed LSH index. The most important property is the usage of so called gateway peers (similar to the ones used in [18]) that are initially placed in each of the LSH hash tables. These peers can be determined using the lookup method of the global DHT. If a lookup on one of the predefined positions fails, i.e., leads to a peer that is not currently in the LSH hash table, that peer issues a lookup on one of the other entry points and joins the particular hash table it belongs to. In case of a successful access to one of the gateway peers, the query initiator (or data indexing peer) gains access to the LSH hash table.

The number of peers dynamically grows inside each LSH hash table by overloaded peers issuing requests on the global DHT to find peers joining the hash table 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 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 [20], that replicates "hot" ranges inside a Chord style DHT and then let peers randomly choose among the replicated arcs.

5. QUERY PROCESSING

Given a query object $\mathbf{q} = (q_1, ..., q_d)$, the object is mapped to a bucket $g(\mathbf{q})$, i.e., a vector of length k as described above, using the p-stable LSH method. The query initiator uses one randomly selected gateway peer per LSH hash table as an entry point to the LSH hash buckets. Subsequently, the responsible peer P for maintaining the share of the global index that contains $g(\mathbf{q})$ is determined by mapping $g(\mathbf{q})$ to the peer identifier space using $\xi_{sum}(g(\mathbf{q}))$, 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 [17].

5.1 Linear Forwarding

Let τ denote the distance of the K-th object w.r.t. the query object $\mathbf{q}$, obtained by the 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 2). 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). That stopping condition can be relaxed by introducing a parameter α and stop the processing if $d_{best} > \tau/\alpha$. α allows for either a more aggressive querying ($\alpha > 1$) of neighboring peers or for an early stopping ($\alpha < 1$).

5.2 Multi-Probe Based Forwarding

Due to the way we are mapping hash buckets to peers, P maintains all data points that map to a sum that falls into its responsibility, i.e., all values that are in $]P.pred().id, P.id]$. The multi-probe LSH method [17] 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 the to the peer responsible for $\xi_{sum}(g(\mathbf{q}))$. The peer that receives the query, issues a full scan, removes visited buckets from the list and forwards the query (cf. Algorithm 3).

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

Algorithm 2: 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.

```
Input: Local DHT L, query o, bucketlist, $P_{init}$
result[] = localIndex.executeLocalKnn(o);
while (bucketlist.hasElement() do
    b =bucketlist.removeBucket();
    bucketId =$\xi_{sum}(b)$;
    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 3: 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. EXPERIMENTS

6.1 Experimental Setup

We have implemented a simulation of the proposed system and algorithms using Java 1.6. The simulation run on a 2x2.33 GHz Quad-Core Intel Xeon CPU with 8GB RAM. The data is stored in Oracle 11g. As data set, we used a flickr collection consisting of 200,000 MPEG7 visual descriptors with 282 dimensions per image such as the *Edge Histogram* or *Homogeneous Texture Types*. We uses the Euclidean distance to measure the distances between images, treating all dimensions equally and without preprocessing the data. We chose 100 points randomly from the above as query points. $K = 20$ in all experiments. For the global DHT we considered a population of 100,000 peers and we considered 100 peers to maintain each LSH hash table.

We implemented and tested the following four methods:

SimpleRand: This is the baseline algorithm that uses a pseudo-random mapping as ξ. At query time, the whole local index of the peer which is responsible for the mapped LSH bucket is scanned without further forwarding.

SimpleSum: This is when the p-stable LSH buckets are distributed among peers using ξ_{sum}. Query processing is done like the previous method.

MultiProbeSum: In this method also the distribution is done with ξ_{sum}, while at query time we use the multi-probing based algorithm as described in Section 5.2.

LinearSum: Here also we use ξ_{sum} to distribute the buckets among peers while at query time the linear forwarding algorithm, Section 5.1, is used.

We report on the following measures:

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 [21] show that the Gini coefficient is most appropriate statistical metric for measuring load distribution fairness. The Gini coefficient, other than the other three measures, is query independent 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 work in large-scale wide-area networks.

Relative Recall: For the effectiveness metric, we report on the relative recall, i.e., the number of relevant documents among returned documents. The relevance is defined by the full-scan run over all data points to determine the K highest ranked points for a given query, where similarity is measured based on l_2. It should be noted that since we are ranking the all candidate objects and returning only the top K, we do not need to consider precision here.

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 [12]. $\frac{1}{K}\sum_{i=1}^{K}\frac{d_{LSH_i}}{d_{true_i}}$ 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.

All performance measures are averaged over 100 queries.

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.

6.2 Experimental Results

We first investigate the effect of ξ_{sum} on the load distribution, which is shown in Figure 2 for different parameter settings. As seen in Table 1 the Gini coefficients of all four load distributions fall in the range of [0.4,0.6] which is a strong indicator for a fair load distribution [20].

k=20,W=10	k=20,W=1	k=100,W=10	k=100,W=1
0.47	0.52	0.53	0.57

Table 1: Gini Coefficient when distributing 10 hash tables using ξ_{sum}

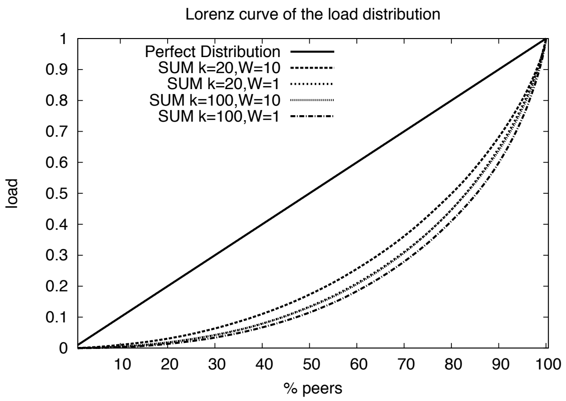

Figure 2: The load distribution under different parameters in the case of 10 LSH hash tables.

Figure 3 shows the results obtained by varying the number of hash tables from 2 to 20. SimpleSum is obtaining up to twice better recall compared to SimpleRand when using the same number of hash tables or having the same number of network hops. The number of network hops in these two cases clearly reflects the number of hash tables used; as queries are not forwarded we have one lookup in the global DHT per hash table. This confirms that ξ_{sum} preserve the locality, i.e., it groups buckets with similar content to the same peer. The results are shown for different values of W and k and show the robustness of the method against parameter changes.

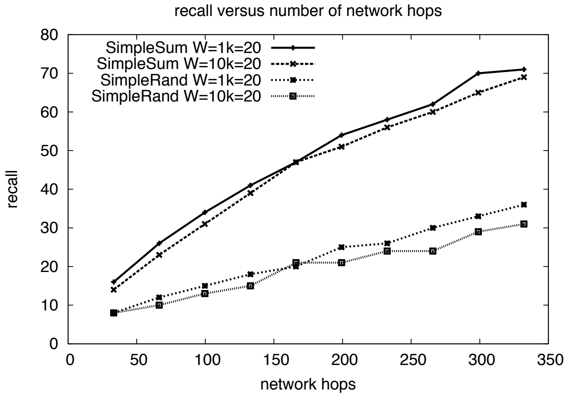

Figure 3: Recall versus number of DHT lookups.

We report in Table 2 the recall and number of network hops for the four different methods under comparison. All methods using ξ_{sum} are superior to SimpleRand in terms of recall and number of network hops. MultiProbeSum and LinearSum achieve better recall compared to SimpleSum at the expense of a small increase in the number of network hops. To better compare SimpleSum and LinearSum, we use information from Figure 3. Observing the curve for $k = 20$ and $W = 1$, for SimpleSum to achieve a recall of 61%, 265 network hops are necessary, while for the same parameters LinearSum needs 193 network hops (cf. 8th row in Table 2). This also reflects the number of required hash tables which is in this case 16 for SimpleSum and 10 for LinearSum, i.e., 60% more redundancy in the network.

The comparison between MultiProbeSum and LinearSum shows that when they have equal recall (cf. 3rd row of Table 2) LinearSum needs less network hops. In other cases LinearSum achieves up to 18% more recall at the expense of around 15% more network hops.

			Relative Recall in % (#Network Hops)			
#tables	**W**	**k**	**Simple Sum**	**MProbe Sum**	**Linear Sum**	**Simple Rand**
2	10	20	14% (33)	17% (43)	18% (38)	8% (33)
10	10	20	47% (166)	56% (218)	58% (194)	19% (166)
20	10	20	69% (332)	79% (438)	79% (338)	28% (332)
2	10	100	14% (33)	16% (38)	19% (38)	9% (33)
10	10	100	47% (166)	52% (189)	58% (194)	20% (166)
20	10	100	69% (332)	75% (438)	80% (388)	31% (332)
2	1	20	16% (33)	16% (34)	20% (38)	8% (33)
10	1	20	47% (166)	48% (171)	61% (193)	21% (166)
20	1	20	71% (332)	72% (341)	82% (387)	38% (332)
2	1	100	15% (33)	16% (33)	22% (38)	8% (33)
10	1	100	49% (166)	50% (168)	63% (193)	20% (166)
20	1	100	70% (332)	70% (336)	82% (386)	36% (332)

Table 2: Measuring recall and number of network hops for different configurations of W, k and number of hashtables for different methods under comparison. Number of probes for MultiProbeSum is 100.

7. CONCLUSIONS

We have presented a robust and scalable solution to the distributed K-Nearest Neighbor 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 *contradictory* conditions of efficient and high quality KNN search in distributed settings: Enabling probabilistic placement of similar data on the same peer, while achieving a fair load balancing. We have presented how to create the index, leveraging our proposed mapping and its characteristics. We have devised two algorithms, considering an incremental top-K processing over neighboring peers and as an alternative approach, an algorithm that uses multi-probe LSH to find peers that need to be included in the query processing. The presented experimental results indicate that the proposed algorithms are stable w.r.t. parameter choices and provide high search quality. Preparatory experiments with LSH based methods showed that existing centralized algorithms are very sensitive to the tunable parameters. Suboptimal parameters can be expected in dynamic, large-scale distributed systems and 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.

8. REFERENCES

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

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

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

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

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

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

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

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

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

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

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

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. *VLDB*, 1999.

[13] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Conference*, 1984.

[14] 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. *ACM Trans. Database Syst.*, 30(2), 2005.

[15] 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. *ICDE*, 2006.

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

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

[18] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Minerva$_{\text{infinity}}$: A scalable efficient peer-to-peer search engine. *Middleware*, 2005.

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

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

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

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

[23] Ozgur D. Sahin, Fatih Emeki, Divyakant Agrawal, and Amr El Abbadi: Content-Based Similarity Search over Peer-to-Peer Systems. *DBISP2P*, 2004.

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

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

[26] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. TR, Dept of CS, Princeton University, 2004.