This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE CCNC 2010 proceedings

Distributed Locality Sensitivity Hashing

Smita Wadhwa
smitaw@yahoo-inc.com
Video Platform, Yahoo!

Pawan Gupta
pawan_z@yahoo.com
Video Platform, Yahoo!

ABSTRACT

In this paper, we present DLSH Distributed Locality Sensitive Hashing , a similar-data search technology. The huge growth in the size of video content has broken the traditional multi-media index hosting and look-up solutions, these are not able to scale to the size of the current and projected index requirements. Distributed LSH (D-LSH) addresses this need of a highly scalable multi-media index. DLSH performs better for finding approximate near neighbors on extremely large scales , as DLSH distributes close points on single boxes, and far points on different boxes based on projections.

Index terms

duplicate video detection, video signature, video similarity, image similarity, large scale index with nearest neighbor search.

1. INTRODUCTION

Multimedia Content (Audio, Image & Video), matching techniques rely on content fingerprinting in which software identifies, extracts and then compresses characteristic components of a content , enabling that content to be uniquely identified by its resultant "fingerprint". For searching multimedia, content these fingerprints have to be stored in the indexes which can support quick lookup of similar content factoring both spatial and temporal changes from original content.

These fingerprint indexes should support huge amount of content covering movies, music, sport, news etc. The challenge is compounded by the fact that traditional hash look-up techniques cannot be used here as, a comparison for absolute equality may fail even when two video segments are perceptually identical. The current state of art for multi-media look-up technique is based on Locality Sensitive Hashing (LSH) which does a fuzzy match. LSH has a serious limitation in terms of index size as the in-memory index is constrained by the physical memory of the single box.

Based on an extensive evaluation of few indexing solutions, we have observed that there is a broad gap in the solutions available today. Solutions for similarity matches work well with limited index size say with few hundred hrs. of video index but fails as the index size grows to, large sizes, say 10,000+ hrs of video content which roughly translates into 2 Million fingerprints in the index. These limitations make the existing solutions unviable for the size and range of multimedia content Yahoo! handles.

The challenges in building a large scale index for similarity search can be broadly classified under three areas:

1. Similarity (Fuzzy) Matching
2. Scalability
3. Look up Time

Our solution takes a leave from the Distributed Hash Tables (DHT) approach, used in scaling large scale index in P2P, which distributes the index and lookup over the network. The challenge in using DHT approach in multi-media look-up has been that LSH is more of a fuzzy match (Nearest Neighbor match), as against an exact match in the DHT case.

Our solution utilizes a similar approach of distributing large index over the network, however in the context of fuzzy matches .Hence the name DLSH (Distributed - `Locality Sensitive Hashing`)..

2. EXISTING TECHNIQUES

A simple brute force approach could be used to iterate through each feature vector in a simple index and to calculate the distance to the query object. However, our index may contain billions of objects—each object described by a feature vector that contains hundreds of dimensions. Therefore, it is important that we find a solution that does not depend on linear search of the index. Existing methods to accomplish this search include trees and hashes.

2.1 Distributed Hash Table (DHT)

Distributed hash tables (DHTs) are a class of decentralized distributed systems that provide a lookup service similar to a hash table: (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. But as the hash returned values variate largely even in small deviation in input, i.e., a well designed hash function separates two symbols that are close together into different buckets. This makes a hash table a good means of finding exact matches but not in approximate matching. To find approximate (near) matches efficiently, we use k-d trees and locality sensitive hash.

978-1-4244-5176-0/10/$26.00 ©2010 IEEE

2.2 K-dimensional Tree (k-d tree):

By building a tree of objects, we can start at the top node when given a query, ask if our query object is to the left or to the right of the current node, and then recursively descend the tree. In a multidimensional space, this idea becomes the k-d tree algorithm. The problem with multidimensional algorithms such as k-d trees is that k-d trees are not suitable for efficiently finding the nearest neighbor in high dimensional spaces. We end up testing nearly all the nodes in the data set and the computational complexity grows to O(n1-1/k +m) time, where m is the number of the reported points, and k the dimension of the k-d tree.

2.3 Locality Sensitive Hashing (LSH)

LSH is based on the simple idea that, if two feature vectors are close together, then after a "projection / mapping / (dimensional reduction)" operation in k-dimensions, these two points will remain close together. The major drawback of the basic LSH indexing method is that it may require a large number of hash tables to cover most nearest neighbors. The size of each hash table is proportional to the data-set size, since each table has as many entries as the number of data objects in the dataset. When the space requirement for the hash tables exceed the main memory size, looking up a hash bucket may require a disk I/O, causing substantial delay to the query process. LSH, therefore, is not able to scale on large size of index, which is required by multimedia content.

Table 1. Comparison of Existing Techniques

Technology	Characteristics	Limitations
Distributed Hash Table	Distributed Scalable Fast Look up	No Support for Fuzzy Match
K dimensional Tree	Fuzzy Match Fast Look up	Reduced precision with large index size No support for higher dimension
Locality Sensitive Hashing	Fuzzy Match Fast Look up Support for higher dimension	Index size limited to physical memory in a box

3. DISTRIBUTED LOCALITY SENSITIVE HASHING

To address the above issues associated with the DHT exact matching, trees lookup time, LSH scalability issues, we propose a new method called DLSH, which uses a more systematic approach to explore hash buckets on a larger scale.

Distributed Locality-Sensitive Hashing (DLSH) is an algorithm for solving the (approximate/exact) Near Neighbor Search in high dimensional spaces. It is based on LSH (Locality Sensitive Hashing) which performs probabilistic dimension reduction of high-dimensional data. The basic idea is to hash the input items so that similar items are mapped to the same buckets with high probability (the number of buckets being much smaller than the universe of possible input items).Thus, DLSH is basically LSH performed on several machines which Yahoo! serves.

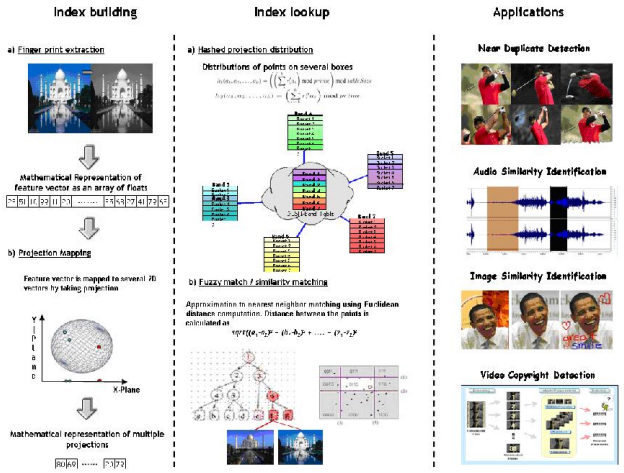

Figure 1. Distributed Locality Sensitivity Hashing workflow with applications supported.

DLSH, decentralized distributed systems, that provide a lookup service similar to a LSH. Responsibility for maintaining index and look up is now distributed across various machines/nodes. Mapping is initially done to reduce dimensions of fingerprints and is further used for node identification. Each node in turn maintains an index table and does lookup, in a second level of hashing, based on it. After determining the fingerprints which map to the same index, Euclidean distance between the points are calculated to find approximation of distance between two points. Based on the threshold distance, near neighbors are calculated. This allows DLSH to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

It provides a distributed and highly scalable, multi-media indexing and lookup solution to manage the huge size and large range of video content .

4. ALGORITHM OVERVIEW

The key idea of the DLSH is to use a projections/mappings to check multiple buckets that are likely to contain the nearest neighbors of a query object. Given the property of locality sensitive hashing, we know that if an object is close to a query object q but not hashed to the same bucket as q, it is likely to be in a bucket that is "close by" (i.e., the hash values of the two buckets

only differ slightly). So our goal is to locate these "close by" buckets, thus increasing the chance of finding the objects that are close to the query point q.

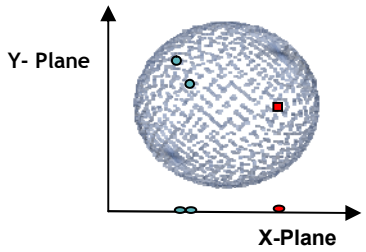

Figure 2. Projections of high dimensional near points getting mapped nearby in two dimensional plane, similarly high dimensional far points getting mapped far apart in two dimensional plane

LSH approach with large number of points in the index becomes a bottleneck in terms of time complexity and accuracy. To resolve this, DLSH provides multiple index levels and also divides the first index level into bands.

The first level of index is divided in bands, where each band represents the node for second level of indexing. The bands contain continuous h1 values, where number of bands is dependent on the scale of index and no of servers provided. At the first level of index, fingerprints are mapped and based on the values obtained; the computation is further delegated to second indexing level. On second level of indexing, another projection in singular dimension is used to index the points. The points falling far apart will be placed in different buckets, whereas near points falling in approximately same buckets.

$$h_1(a_1, a_2, \ldots, a_k) = \left(\left(\sum_{i=1}^{k} r_i' a_i \right) \bmod prime \right) \bmod tableSize$$

$h1(a_1,a_2,\ldots\ldots,a_k) = ((\Sigma^k$

Figure 3 .Functions to calculate first level of index and second level of indexing

Final nearest neighbor matches are found with finding the Euclidean distance between the two original points.

$$sqrt((a_1-a_2)^2 + (b_1-b_2)^2 + \ldots. + (z_1-z_2)^2$$

Figure 4 . Approximation to nearest neighbor matching using Euclidean distance computation.

$$h_2(a_1, a_2, \ldots, a_k) = \left(\sum_{i=1}^{k} r_i'' a_i \right) \bmod prime$$

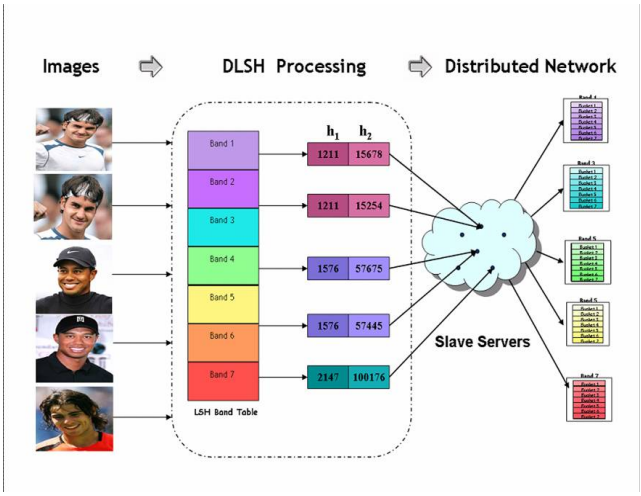

Figure 5. DLSH over the network.

DLSH maps similar fingerprints on the same band and different fingerprints on different bands, thereby enhances the search. DLSH method is much more time efficient and scalable .DLSH method can search and index millions of fingerprints to achieve the same search quality within in fraction of seconds.

5. EXPERIMENTAL RESULTS

All the algorithms and processing flow has been implemented on a Linux platform in C++. All the experiments were conducted on a single core with 2 GB RAM.

5.1 Test Suite

We have used two datasets in our evaluation. The two datasets are-

Image Data contains images crawled from the web. The total number of images picked is 12.5 million. For each image, we use the coorelogram tool to extract a 2080-dimensional coorelogram histogram.

Video Data - Data contains video crawled from the web. The total number of videos picked is 5.27 million. For each video, we used polemics video fingerprinting solution.

We simulate the web video duplicate cases and randomly produce zero to five duplicate copies of videos. These changes include the random combination of bit rate changes , frame rate changes, stitched videos, cropped videos . Following are the results of video data experimentation.

5.1.1 Look up time Results

It takes less than 10msec to give response to a query point request on the index of 6.5 million feature vectors.

5.1.2 Scalability Results

DLSH scales to even millions of index even when it is running on single machine .As there is no known practical limit, as the system is gracefully scaling out by adding more boxes on the network.

Table 1. Scalability Comparison with Existing Techniques

Techniques	Number of Feature Vectors *
DLSH	> 12,700,000**
LSH	~ 900,000
k-d tree	~ 210,000

* for above numbers ,the size of each feature vector was 2145 floats.

** no known limitations.

5.1.3 Accuracy Results

DLSH gives very high degree of accuracy even on the very large index size. The tests have been conducted for finding the nearest neighbors keeping original videos in DLSH index and taking edited videos as requested query video. The accuracy is same as LSH which are quoted with precision of 0.9 and recall of 0.85.

6. CONCLUSION

In this paper, we describe how DLSH is highly scalable than LSH approach, maintaining the accuracy levels of LSH with no known scaling limitations. This paper shows how DLSH solves the issue around fast similarity lookups on large multimedia fingerprint indexes by distributing the index over the network.

Our experimental results show that the DLSH method is highly scalable then the basic LSH to achieve desired search accuracy and query time with large scales desired by multimedia content.

REFERENCES

[1] A. Andoni and P. Indyk, "E2LSH 0.1 User Manual," Jun. 2005.[Online].Available:http://web.mit.edu/andoni/www/LSH.

[2] A. Andoni, M. Datar, N. Immorlica, and V.Mirrokni, "Locality-sensitive hashing using stabledistributions," in Nearest Neighbor Methods in Learning and Vision: Theory and Practice,

[3] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for near neighbor problem in high dimensions," in Proc. Symp. Foundations of Computer Science (FOCS'06), 2006.

[4] www.cs.umd.edu/~mount/ANN/

[5] A. Andoni, P. Indyk, and M. P˘atras¸cu. On the optimality of the dimensionality reduction method. Manuscript, 2006.

[6] S. Arya, D.Mount, N. Netanyahu, R. Silverman, and A.Wu. An optimal algorithm for approximate nearest neighbor searching. Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 573–582, 1994.

[7] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bounds for approximate nearest neighbor searching. Proceedings of the Symposium on Foundations of Computer Science, 2004.

[8] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. Proceedings of the Symposium on Foundations of Computer Science, 1998.

[9] J. H. Conway and J. A. Sloane. Soft decoding techniques for codes and lattices, including the golay code and the leech lattice. IEEE Trans. Inf. Theor., 32(1):41–50, 1986.