Fast Image Search using Distributed LSH

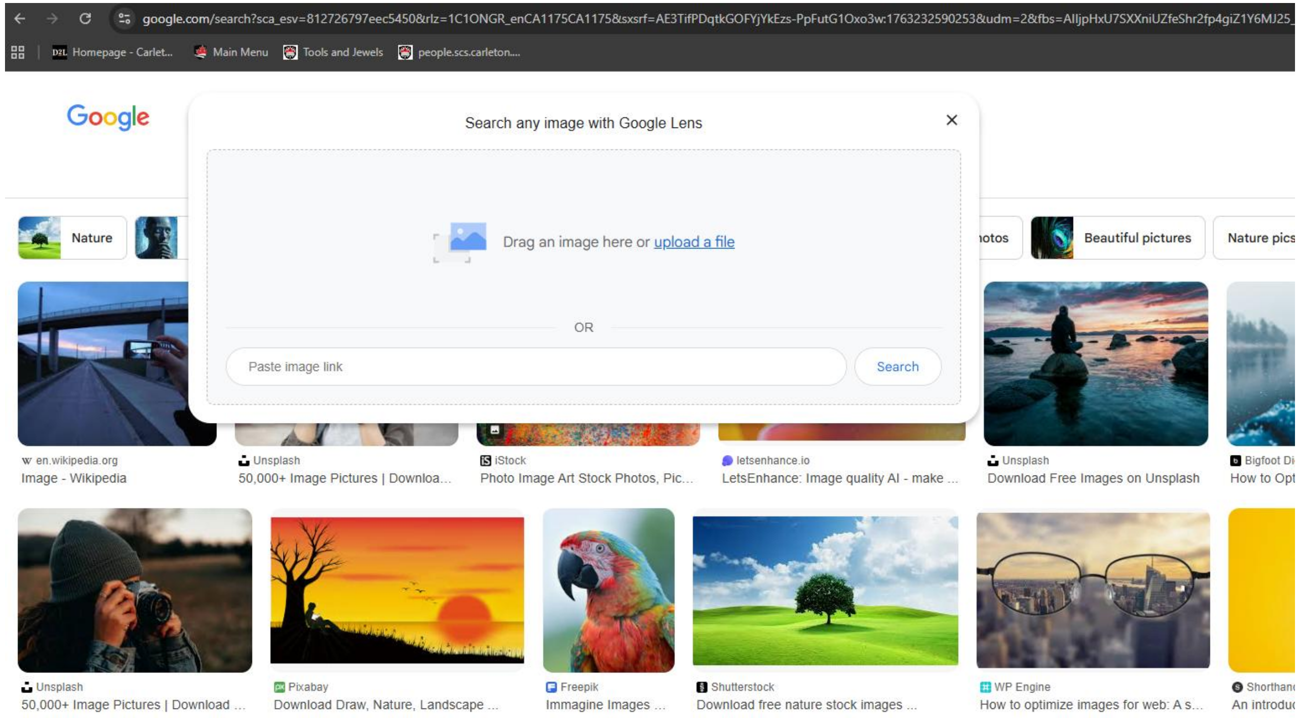
Osman Durmaz, Hasan Sakir Bilge

Motivation

- The current state of social media and digital interaction is increasingly visual
- A fewer number of platforms are processing massive volumes of image data to store and retrieve billions of images.
- The traditional methods of content-based retrieval systems relied on centralized tree structures which, for large scale databases, querying approaches O(n) time
- The state of the tech world requires scalability and efficiency which motivates a shift to distributed architectures.

Applications

- Reverse image search
- Copyright
- Content moderation
- Ai



Problem Statement

Input: Given a set of data $X \in \mathbb{R}^d$ where $X = \{x_1, x_2, \dots, x_N\}$, stored and distributed over a set m database nodes using family of hash functions $\mathcal{H} = \{h: x \rightarrow \{0,1\}\}$. A query image q.

Output: A set of the n most similar neighbors of q.

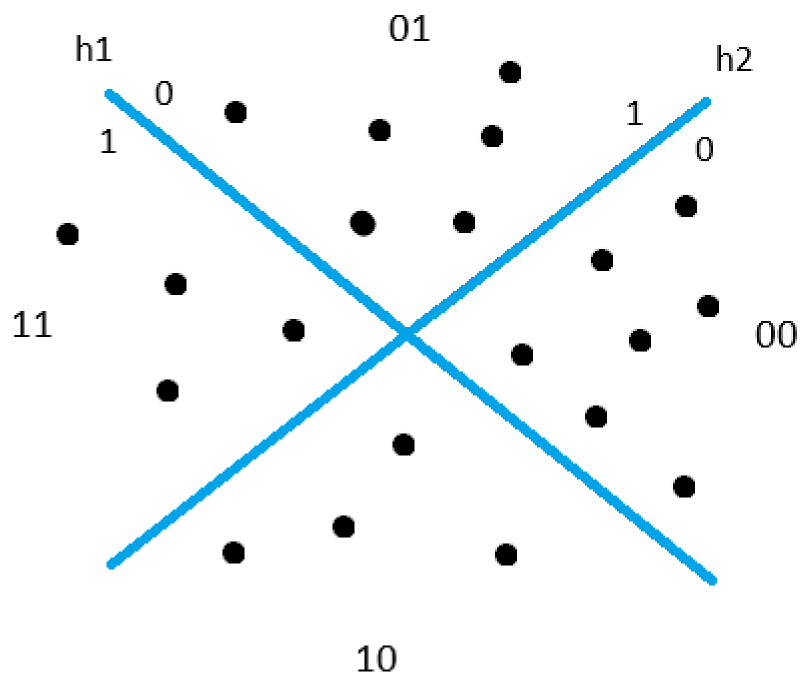
Preliminaries: Image representation and LSH

- Images are represented a vector in d-dimensional space
  - Different styles of encoding use different dimensions
- A single hash function h() ∈ $\mathcal{H}$ takes a d dimensional vector and produces a binary output
- To get accurate hash signatures, k hash functions from $\mathcal{H}$ are used to create a signature.
  - This hash signature is noted $\mathcal{H}_k(\cdot)$

<u>Assumptions:</u>

- The time complexity to compute a hash signature $\mathcal{H}_k(\cdot) = k * d$
- $\mathcal{H}$ is a sensitive family of function
  - $\mathcal{H}$ depends on the dimensional encoding of an image, the details of this will be abstracted from this LSH scheme

Preliminaries: Image LSH



$$x_1 = \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \mathcal{H}_2(x_1) = \begin{bmatrix} 0 & 1 \end{bmatrix}, \mathcal{H}_3(x_1) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$x_2 = \begin{bmatrix} 6 \\ 7 \\ 8 \end{bmatrix}, \mathcal{H}_2(x_2) = \begin{bmatrix} 0 & 1 \end{bmatrix}, \mathcal{H}_3(x_2) = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$$

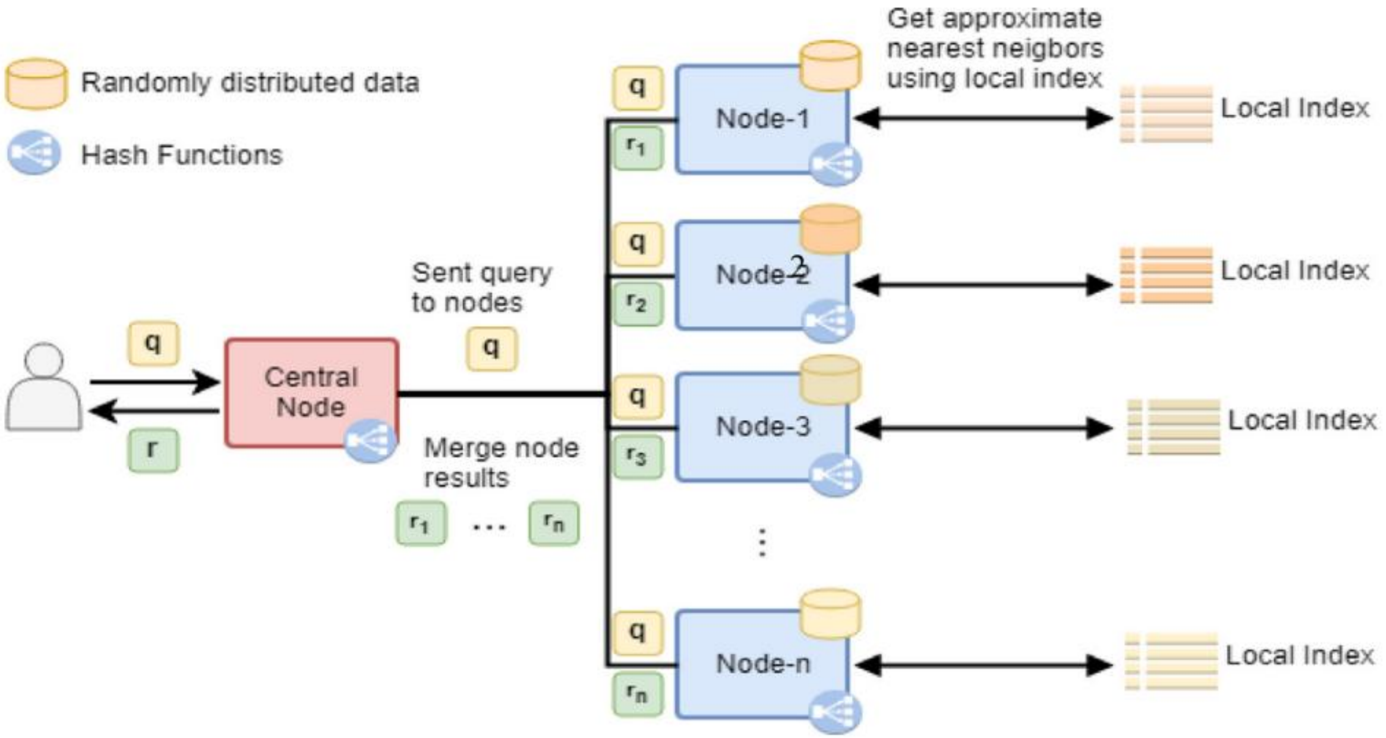
Preliminaries: Image LSH

Observation 1: The range of distinct hashing outcomes is 2^k

The number of ‘buckets’ an image can hash into doubles by the number of hash functions it undergoes.

Preliminaries: Distributed Environments

- There are m database nodes and 1 central node
- The central node can communicate and transmit data to the m database nodes
- Database nodes store the image vector x in the Local index pointed to by its hash signature $\mathcal{H}_k(x)$
- The database nodes do not communicate with each other
- The central node does not store data
- Transmission time to and from the central node is negligible
- All nodes will use the same family of hash functions $\mathcal{H}_k$



Randomized Distributed Hashing: Training Algorithm

Input: Set of N training data $X = \{x_1, x_2, \dots x_N\} \mid x_i \in \mathbb{R}^d$, Hash function set $\mathcal{H}$, Set of m database nodes, pre-determined value k.

1. Divide X into m equal parts so: $X = X_1 \cup X_2 \cup \cdots \cup X_m$
2. For i = 1 to m do
3. send X_i to i'th node

   In Parallel:
4. For j = 1 to | X_i| do
5. Store image vector $X_{i,j}$ in bucket $\mathcal{H}_k(X_{i,j})$

RDH: Query Algorithm

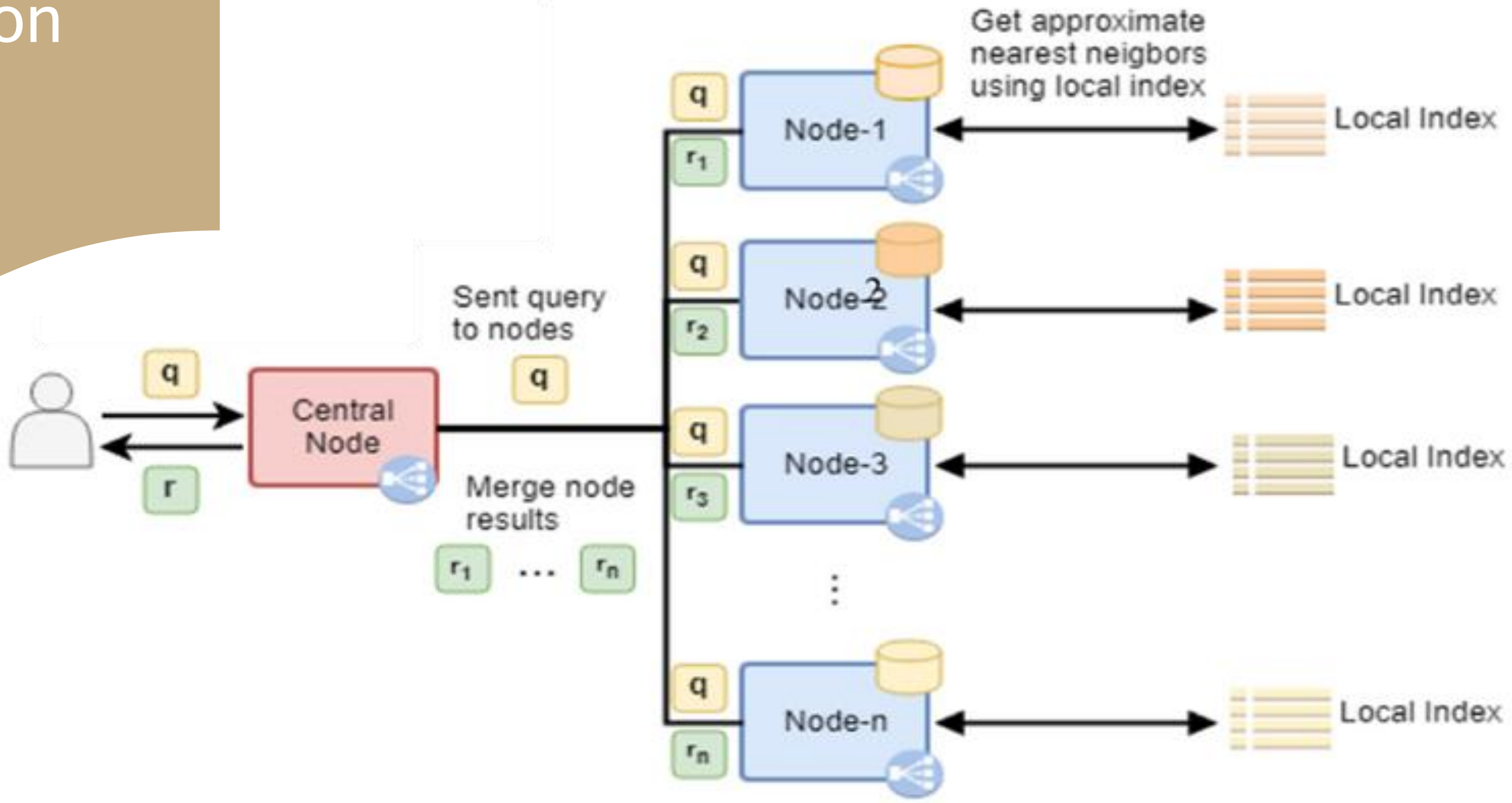
Input: Query sample q ∈ $\mathbb{R}^d$, Pre-determined number n

1. Send q to m nodes

   In Parallel:

2. Compute $\mathcal{H}_k(q)$
3. In bucket $\mathcal{H}_k(q)$, find the n approximate nearest neighbors to q
4. Merge m*n returned results
5. Find the final neighbor(s)

Querying Visualization



Time analysis against monolithic LSH

In monolithic LSH:

Claim: If a data set X containing N d-dimensional items, and k hash functions are used; Finding the nearest neighbor takes $dk + dN/2^k$.

Proof:

1. Total number of possible buckets are 2^k
   - Per Observation 1
2. Time to find the appropriate bucket = time to compute $\mathcal{H}_k(q)$ = dk
   - Per image hashing assumption
3. Cost to compare d-dimensional items = d
4. Average number of items in a bucket = N/2^k
5. Average cost to search a bucket = dN/2^k

Time analysis against monolithic LSH ctd.

In monolithic LSH:

6. Together, the cost to find the n nearest neighbors is time to find the bucket + time to search through the bucket = dk + dN/2^k
7. Set k = $\log_2 N$
8. Substitute:

$$\mathrm{d}(\log_2 N) + dN/2^{\log_2 N}$$
$$\mathrm{d}(\log_2 N) + d(1)$$
$$\mathrm{d}(\log_2 N + 1)$$
$$\equiv O(\log_2 N)$$

Note: d is a constant set by the encoding of the image.

Time analysis against monolithic LSH ctd.

In RDH:

Claim: If data is randomly distributed and stored over m independent nodes; Finding the nearest neighbor takes $\mathrm{dmn} + dk + \frac{dN}{m}/2^k$.

Proof:

1. Total number of possible buckets are 2^k
2. Time to find the appropriate bucket = time to compute $\mathcal{H}_k(q)$ = dk
3. Cost to compare d-dimensional items = d
4. Average number of items in a bucket in one node= $\frac{\mathrm{N}}{\mathrm{m}}/2^k$
5. Average cost to search the bucket = $\frac{\mathrm{dN}}{\mathrm{m}}/2^k$

Time analysis against monolithic LSH ctd.

In distributed LSH:

6. The number of images returned to the central node = $\mathrm{n} \cdot m$
7. Time to search central node for final neighbors = $\mathrm{d} \cdot \mathrm{n} \cdot m$
8. Together, the cost to find the n nearest neighbors is time to find the bucket + time to search through the bucket = $\mathrm{dmn} + dk + \frac{dN}{m}/2^k$
9. Set k = $\log_2 \frac{N}{m}$
10. Substitute:

$$\mathrm{d} \cdot \mathrm{n} \cdot m + \mathrm{d}(\log_2 \frac{N}{m}) + \frac{dN}{m}/2^{\log_2 \frac{N}{m}}$$

$$\mathrm{d} \cdot \mathrm{n} \cdot m + \mathrm{d}(\log_2 \frac{N}{m}) + d(1)$$

$$\mathrm{d}(\log_2 \frac{N}{m} + \mathrm{n} \cdot m + 1)$$

$$\equiv O(\log_2 \frac{N}{m})$$

Note: d, n, and m are constants set before the setup

Correctness

Claim: The images produced from RDH will be the same as in a monolithic LSH scheme.

Correctness: distributed hash function sensitivity

- Let S be the set of all inputs, and U be the universe of hash values

A family of hash functions $\mathcal{H} = \{h: S \rightarrow U\}$ is (r_1, r_2, p_1, p_2) sensitive if for any points $q, v \in S$ and $\mathrm{h}(q), h(v) \in U$:

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

If S is divided into two equally sized subsets S_1 and S_1, the family of hash functions would hash $\mathcal{H}_1 = \{h_1: S_1 \rightarrow U_1\}$ and $\mathcal{H}_2 = \{h_2: S_2 \rightarrow U_2\}$ and would be (r_1, r_2, p_1, p_2) and (r_1, r_2, p_1, p_2) sensitive if for any query point q and all other points $v_1 \in S_1$ and $v_2 \in S_2$:

$$if\ dist(q,\ v_1)\ \leq\ r_1\ then\ Pr_H(h_1(q)\ =\ h_1(v_1))\ \geq\ p_1$$
$$if\ dist\ (q,\ v_1)\ >\ r_2\ then\ Pr_H(h_1(q)\ =\ h_1(v_1))\ \leq\ p_2$$
$$if\ dist(q,\ v_2)\ \leq\ r_3\ then\ Pr_H(h_2(q)\ =\ h_2(v_2))\ \geq\ p_3$$
$$if\ dist\ (q,\ v_2)\ >\ r_4\ then\ Pr_H(h_2(q)\ =\ h_2(v_2))\ \leq\ p_4$$

Correctness: distributed hash function sensitivity ctd.

If v_1 is equivalent to v_2 and $\mathcal{H}_1$ and $\mathcal{H}_2$ are the same hash functions, then:

- $p_1 = p_3$
- $p_2 = p_4$
- $r_1 = r_3$
- $r_2 = r_4$
- $U_1 \cup U_2 = U$

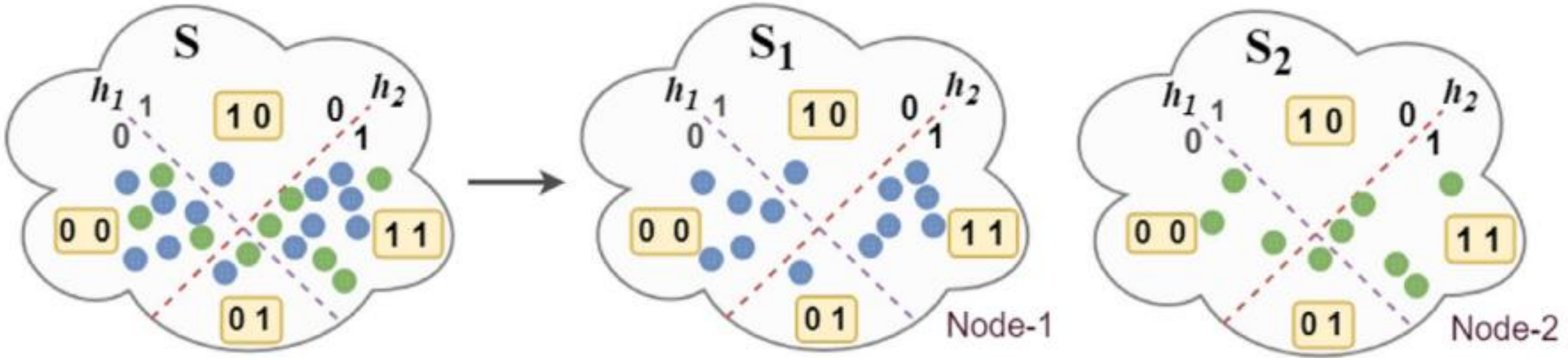
It follows; that if the same hash function is used, even at different nodes, the bucket in which h(v) and h(q) result to will be the same.

- The number of elements in each bucket will decrease, the number of buckets will increase

<u>Distributed sensitive hash family principle:</u>

If the hashing function done on separate nodes is the same, the probability h(v) = h(q) is the same on both nodes:
Pr(h(v)=h(q))$_{node\ 1}$ $=$ Pr(h(v)=h(q))$_{node\ 2}$

Correctness: distributed hash function sensitivity ctd.



Correctness

Proof: The images produced from RDH will be the same as in a monolithic LSH scheme.

Consider the set X* of the n images returned by LSH and the following observations:

1. Each image will have the same hash signature as $\mathcal{H}_k(q)$
2. The maximum distance from any image x* in X* is at most the distance from any other image in the $\mathcal{H}_k(q)$ bucket.

   $\max(dist(x^*, q) | x^* \in X^*) \leq \min(dist(x, q) | x \in \mathcal{H}_k(q) \setminus X^*)$

1. In RDH the n images will all hash to the same bucket $\mathcal{H}_k(q)$ in each of the m nodes.

   Per Distributed sensitive hash family principle

3. When querying the $\mathcal{H}_k(q)$ bucket in a node, the n images with the minimum dist(x,q) is returned to the central node

   Per RDH algorithm

Correctness ctd.

4. The n*m candidate neighbors sent to the central node will include the N*
   Per observations 1 and 2
5. The central node will parse the n*m images and return N*
   Per RDH algorithm

Experiments

Table 3
Query time (milliseconds) results on Corel-10K.

Method	Hash Tables	8-bit	16-bit	24-bit	32-bit	64-bit
LSH	50	488.57	5.44	0.37	0.23	0.44
LSH	100	901.22	11.29	1.22	0.48	1.72
LSH	150	1224.79	20.82	1.47	0.80	2.73
LSH	200	1857.07	28.68	1.65	1.25	2.71
RDH	50	39.53	2.53	0.31	0.22	0.41
RDH	100	71.16	3.73	0.59	0.45	0.97
RDH	150	88.66	4.61	0.85	0.69	1.72
RDH	200	107.64	5.45	1.25	1.05	2.52

- m = 10 node were used in RDH experiments
- The Corel-10k image data set contained 9902 images with 80 dimesnions
- The ‘bit’ refers to the length of the hash signatures
- RDH query times fall between 8% and 5% of equivalent LSH times in 8-bit hash codes

Conclusion

- Multiple hash tables can be used to create a probabilistic amplification effect for increased accuracy
- RDH provides higher efficiency querying with similar accuracy to monolithic method
- RDH is scalable to add and remove database nodes as the node an image is stored in is independent from the computations done on another node
  - Additional nodes can be added and to the m database nodes and query time (in the database nodes) will remain the same until <N/m images are added to the new node
- Inter node communication is not required to achieve a competitively accurate solution to the monolithic architecture
- Experiments show that RDH increases query efficiency by a factor of 8 against LSH;
  - Diminishing returns as hash signature length and number of hash tables increase.

References

- Osman Durmaz, Hasan Sakir Bilge, Fast image similarity search by distributed locality sensitive hashing, Pattern Recognition Letters, Volume 128, 2019, Pages 361-369. https://doi.org/10.1016/j.patrec.2019.09.025
- Gallas, A., Barhoumi, W., Kacem, N. and Zagrouba, E. (2015), Locality-sensitive hashing for region-based large-scale image indexing. IET Image Processing, 9: 804- 810. https://doi.org/10.1049/iet-ipr.2014.0910
- Anil Maheshwari, (2025), Locality-Sensitive Hashing. Notes on Algorithm Design: 183-204. https://people.scs.carleton.ca/~maheshwa/Notes/DAA/notes.pdf