Scalable and Efficient Querying Methods for Learning to Hash

JC1704

LIU Jie 1155047039

SONG Yang 1155046870

Outline

- A Quick Review of Previous Work
- Problems in Handling Large Dataset
- Our Solution
- Experiments Results

A Quick Review of Previous Work

Recap: Approximate Nearest Neighbor Search

Nearest Neighbor Search:

Given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q.

Approximate Nearest Neighbor (ANN) Search:

A generalization of this problem is a k-NN search, finding the k closest points.

ANN - Hashing-based methods

Preprocess: read in item vector.

Item A

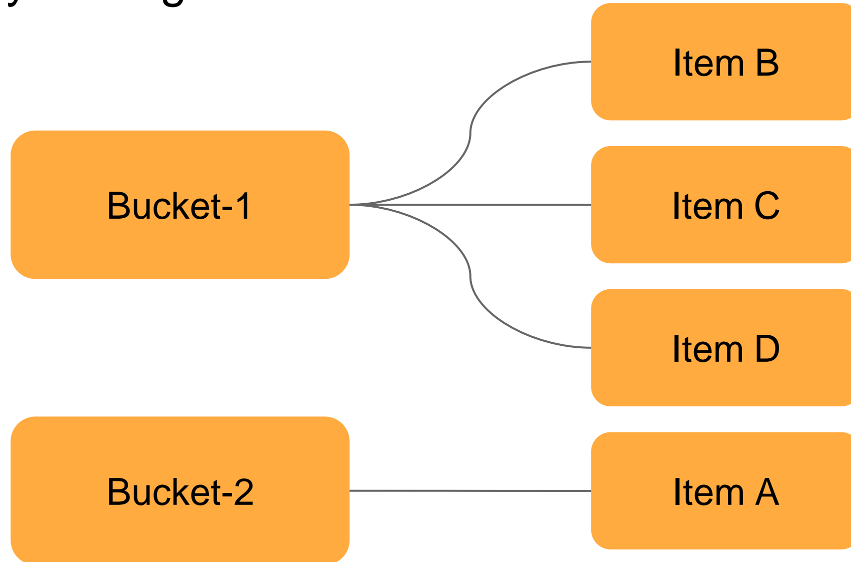
Item B

Item C

Item D

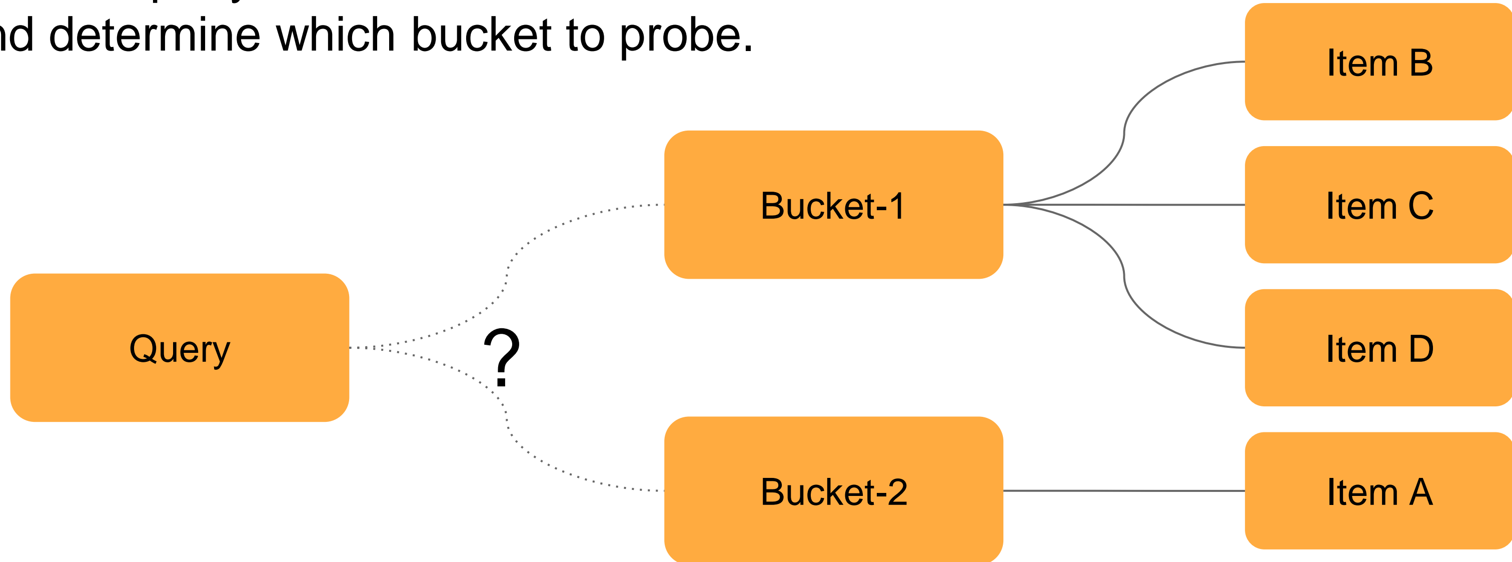
ANN - Hashing-based methods

Preprocess:
Send item vector to hash functions to determine which bucket they belong to.



ANN - Hashing-based methods

Querying:
Read in query vector. Send it to hash functions and determine which bucket to probe.



ANN - Hashing-based methods

There are generally two categories of hashing-based methods.

- Locality Sensitive Hashing (LSH)
- Learning to Hash (L2H)

The **key difference** between them is whether the hash functions are **dataset-dependent** or not.

ANN - Locality Sensitive Hashing (LSH)

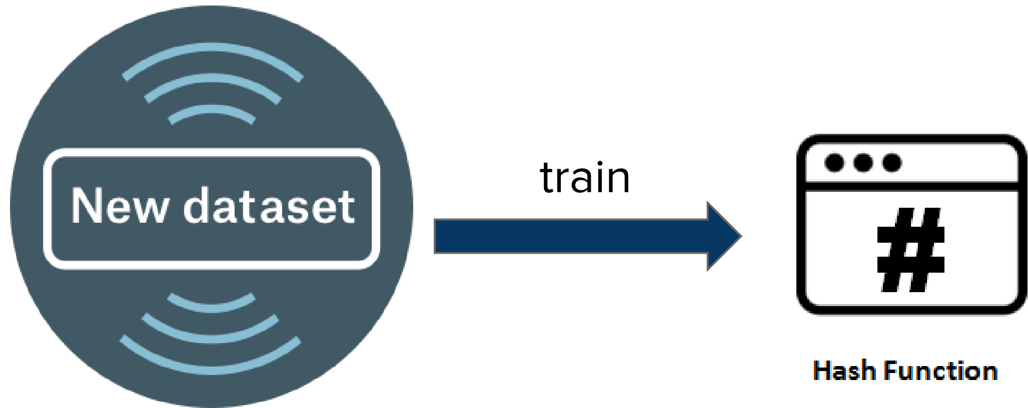
Main Idea

Use a family of **predefined** hash functions

Similar items are hashed to the same bucket with higher probability

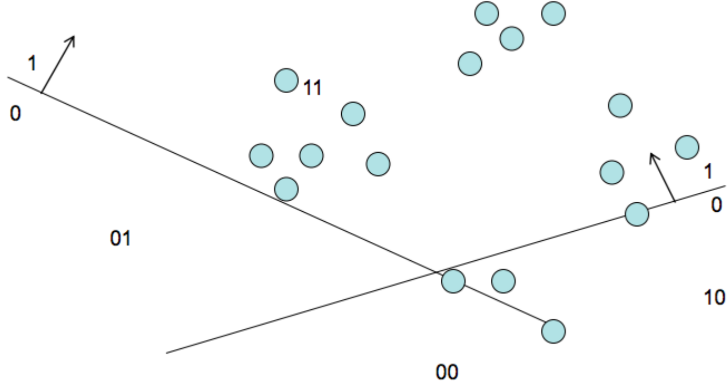
Only a small number of buckets need to be checked for items similar to a query.

ANN - Learning to Hash (L2H)

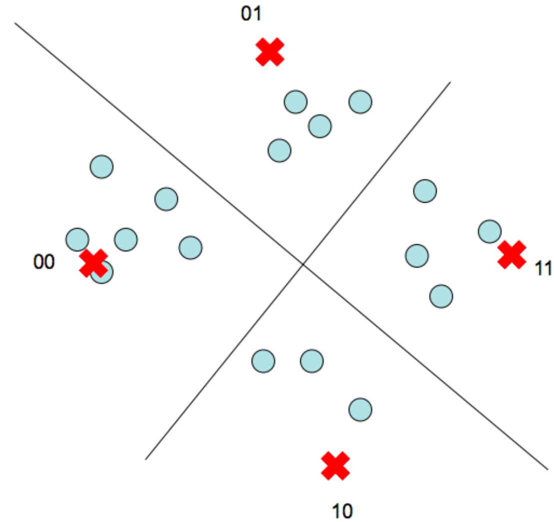


L2H learns tailored hash functions for the given dataset

ANN - LSH vs L2H

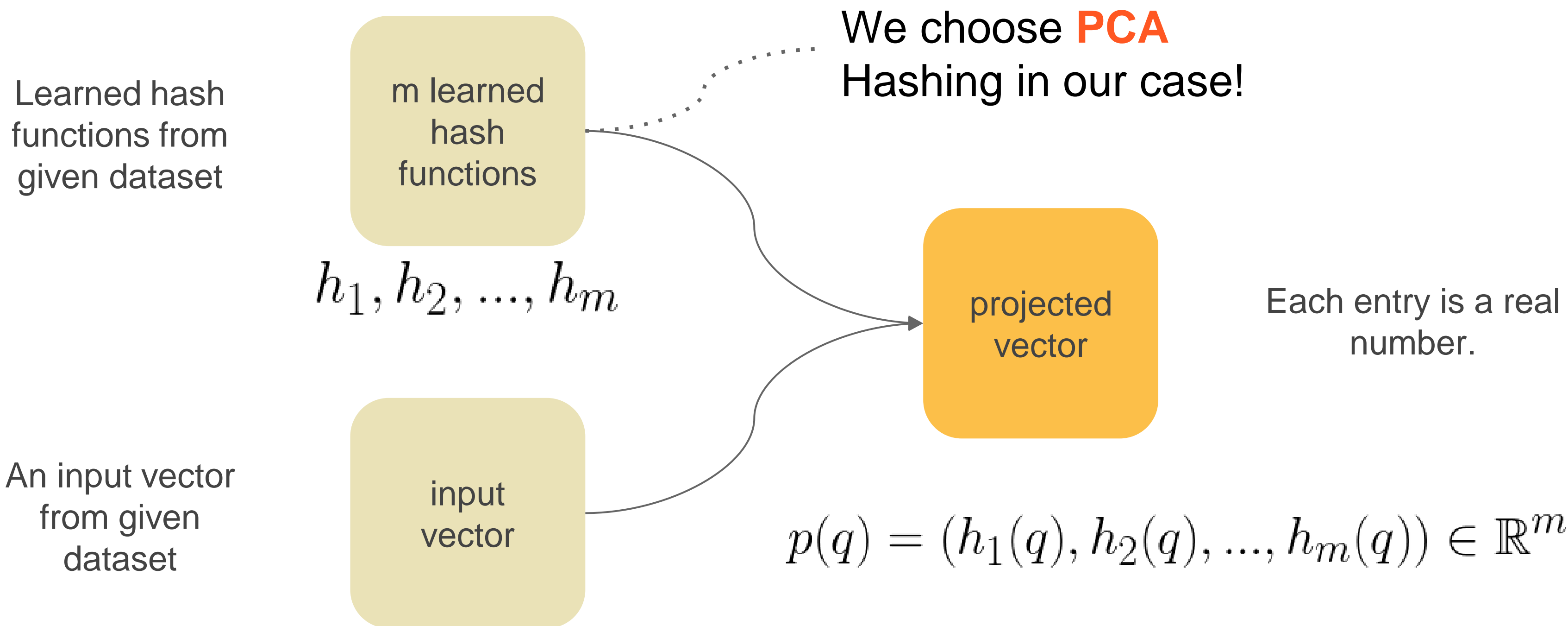


LSH:
Completely blind, not looking at the data at all

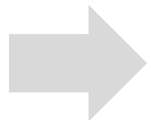


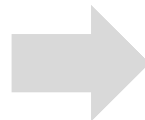
L2H:
Pick the best rotation of the data (or of the hypercube) to minimize quantization errors

Find the right bucket: Hashing



Find the right bucket: Quantization

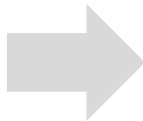




Assign this item vector to the corresponding bucket

Quantization

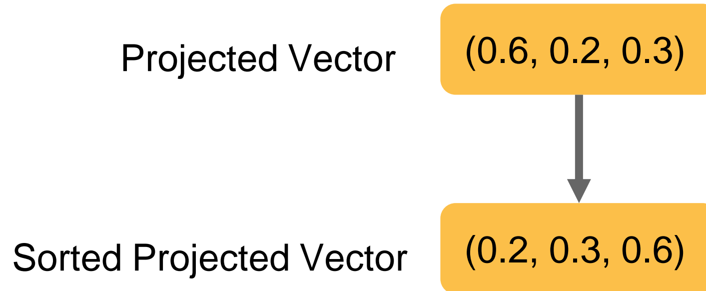
p = (0.1, -0.5, 0.4, 0.3)



c(p) = (1, 0, 1, 1)

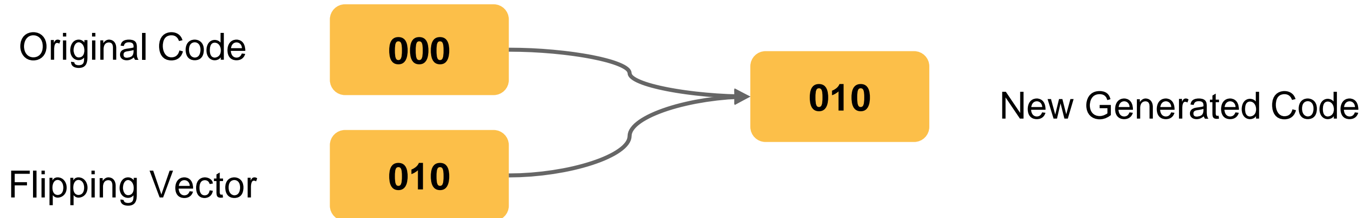
Related Concepts

Sorted Projected Vector

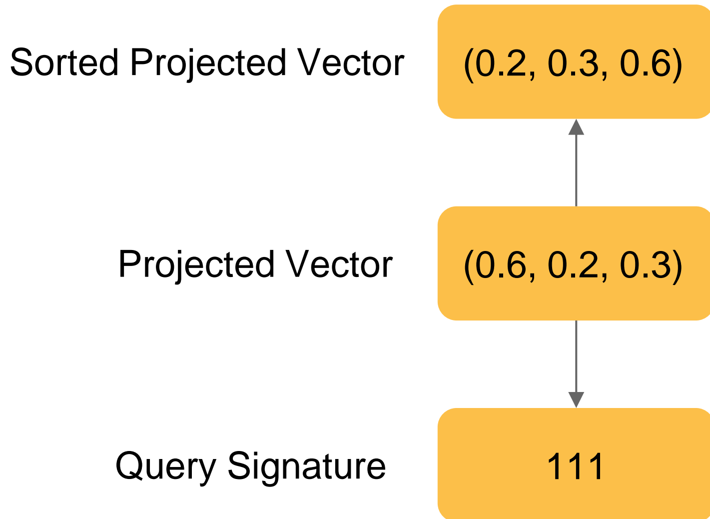


Flipping Vector

A binary code help us to convert one signature to another.



Related Concepts

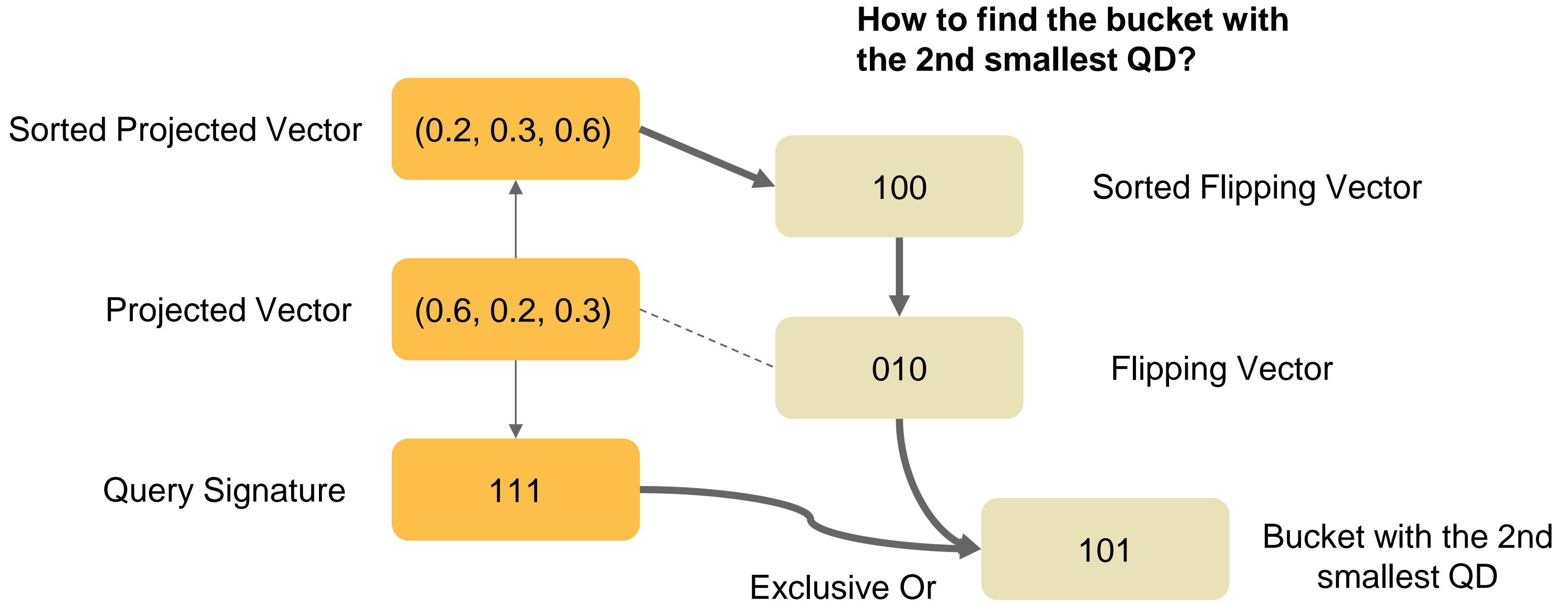


Easy to find the bucket with the smallest QD.

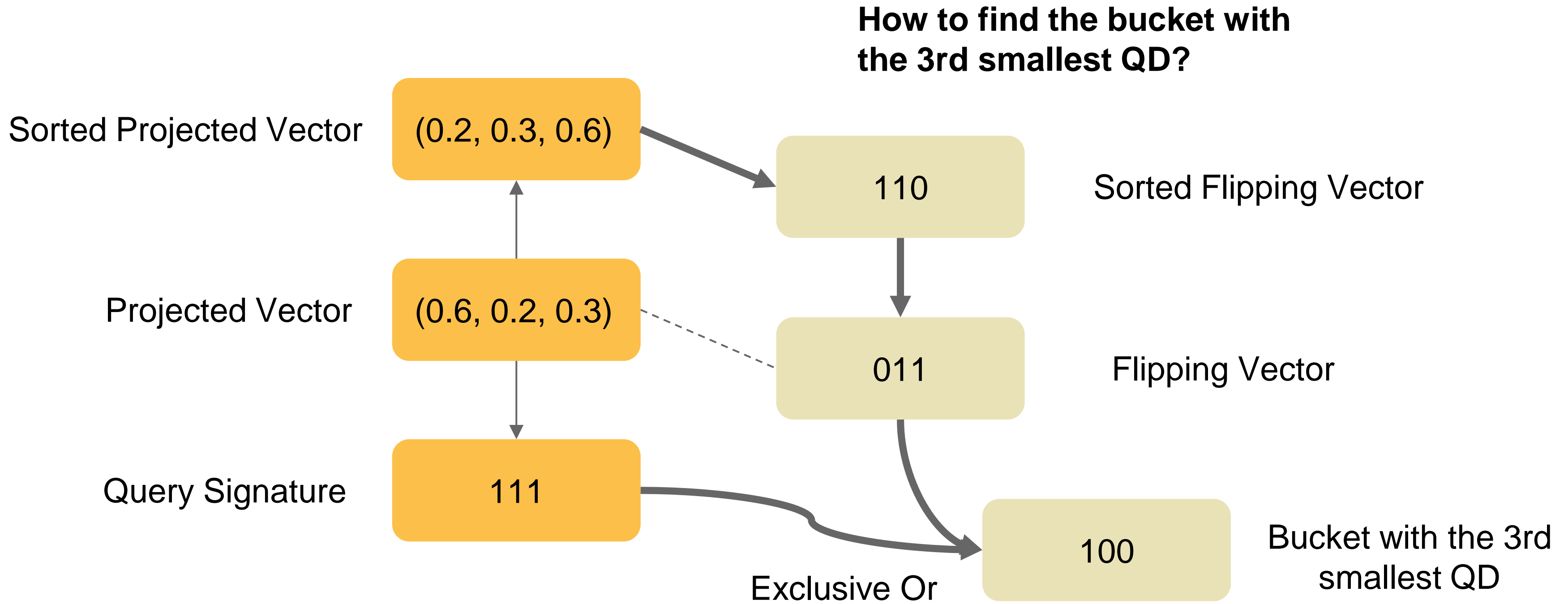
111

Bucket with smallest QD
QD = 0

Related Concepts



Related Concepts



Related Concepts

Question:

How can we generate the sequence of sorted flipping vector efficiently?

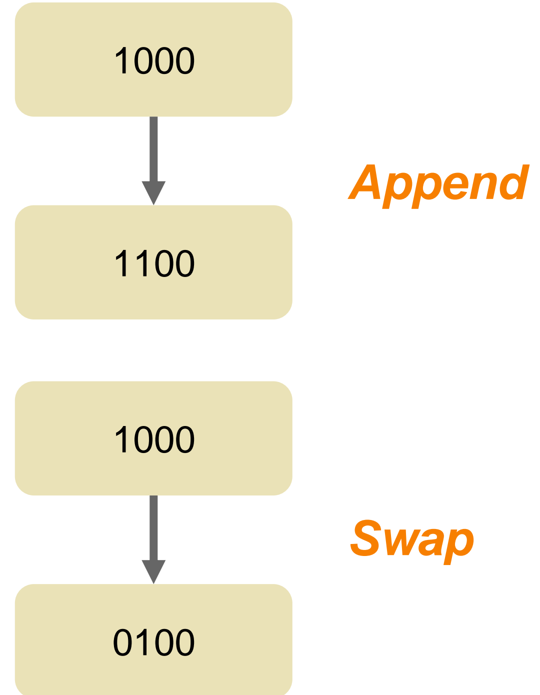
100	Sorted Flipping Vector
110	Sorted Flipping Vector
001	Sorted Flipping Vector

Bucket Generation

Append & Swap

Append adds a 1 to the right-hand side of the rightmost 1 of the vector.

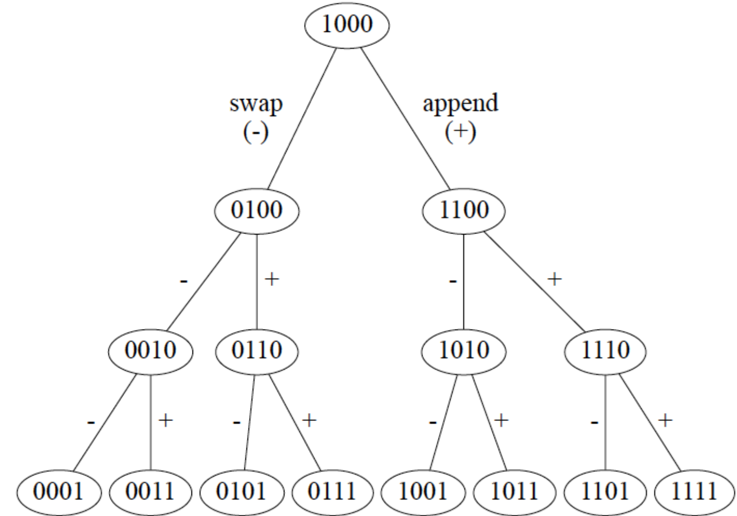
Swap exchanges the positions of the rightmost 1 and 0 on its right hand side.



Bucket Generation

Generation Tree

1. ***Root*** node: (1, 0, …, 0).
2. ***Swap*** parent node get the left child node.
3. ***Append*** parent node get the right child node.



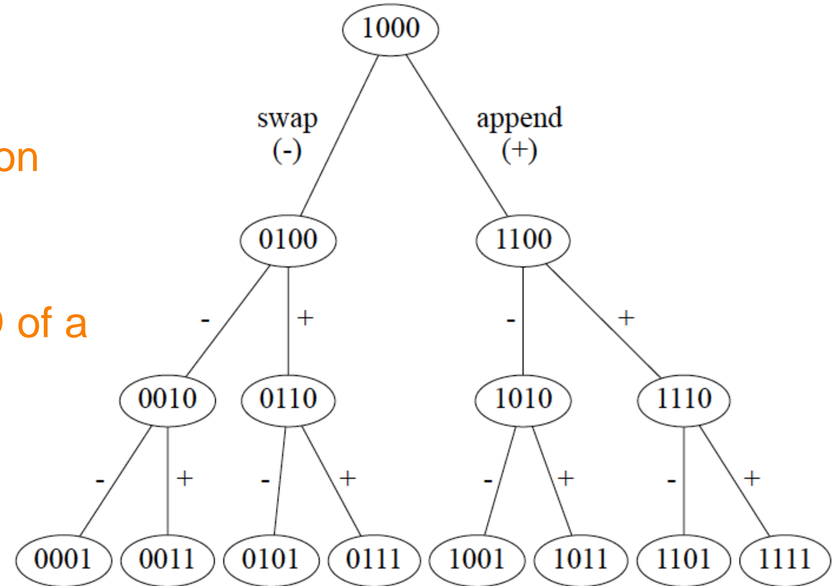
Bucket Generation

Generation Tree - Properties

Property 1. All possible sorted flipping vectors can be obtained from the generation tree exactly once.

Property 2. In the generation tree, the QD of a child is always bigger than its parent.

QD of the child node can be obtained by a simple calculation from its parent node.



Bucket Generation

Outcome:

1. **Generate each bucket exactly once.**
2. **Generate buckets in the order of their QD.**
3. **Do not need to calculate QD for each bucket at once.**

Problems in Handling Large Dataset

The Era of Big Data

How big is big ?

Data sets grow rapidly

In 2012, every day 2.5 exabytes (2.5×10^{18}) of data are generated

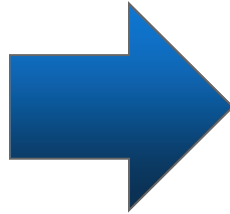
“Big data” refers to analytics methods that extract value from data

- real-word nearest neighbor search applications:
    - Pattern recognition, Recommendation systems, DNA sequencing ...

No single computer can process that much data

Single Machine VS Distributed System

How to turn a “toy” into a powerful tool used in practice



“Scale out” : a problem is divided into many tasks, each of which is solved by one or more computers, which communicate with each other by message passing

Major Challenges in designing and implementing distributed algorithms

~ dividing the problem into relatively independent subproblems

~ coordinating the behavior of the independent parts of the algorithm

Single-Probing VS Multi-Probing in Distributed Implementation

Query processing with multi-probing LSH is complicated to be distributed

Single-probing is straight-forward

For multi-probing, there exists some dependencies between jobs

- without info sent from Item, we cannot conduct Query-side evaluation
- without info sent from Query, we cannot conduct Item-side computation

Therefore, we need to do iterative query processing

Difficulities in Using General Computing Framework

- most of the works used the batched processing system MapReduce and adopted external-memory implementations
- iterative nature of our algorithms does not perfectly match the MapReduce framework
- there is a lack of a programming framework specially designed for LSH algorithms on existing general-purpose distributed frameworks
- need to define a complicated dataflow consisting of many steps (lead to much performance tuning efforts)

Higher level abstraction !

Our Solution: Distributed GQR

Abstraction: Query, Bucket and Item

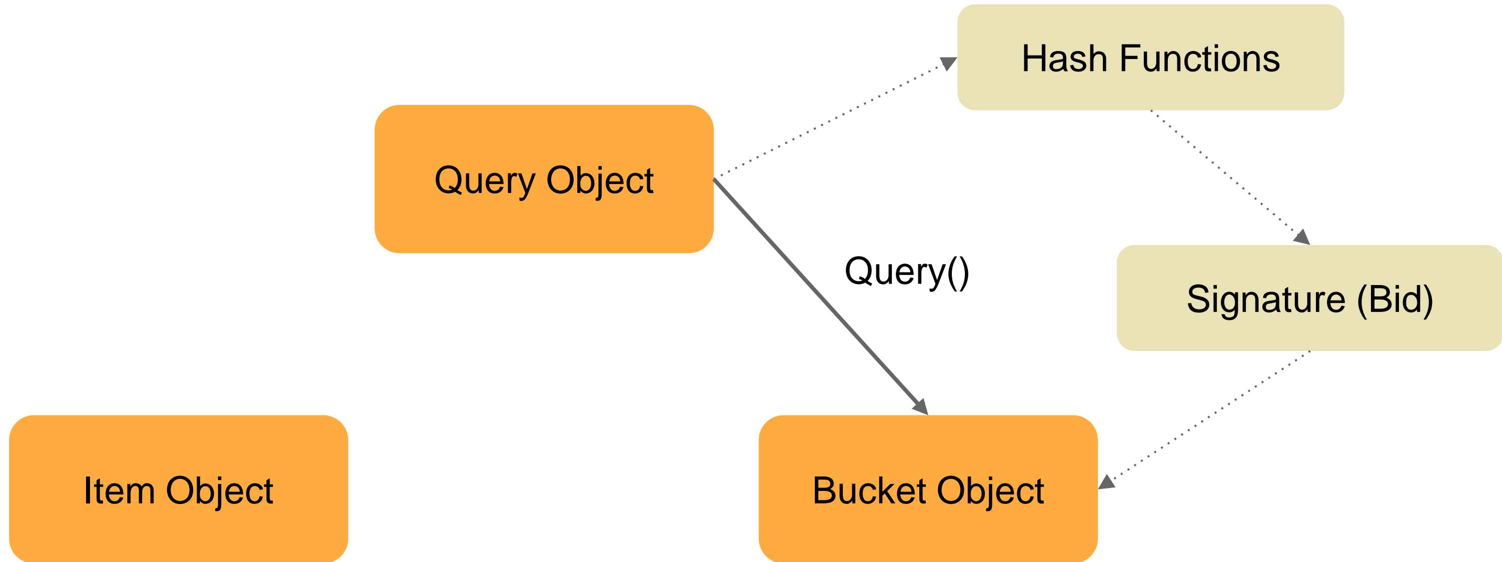
Query-and-Answer

Query Object

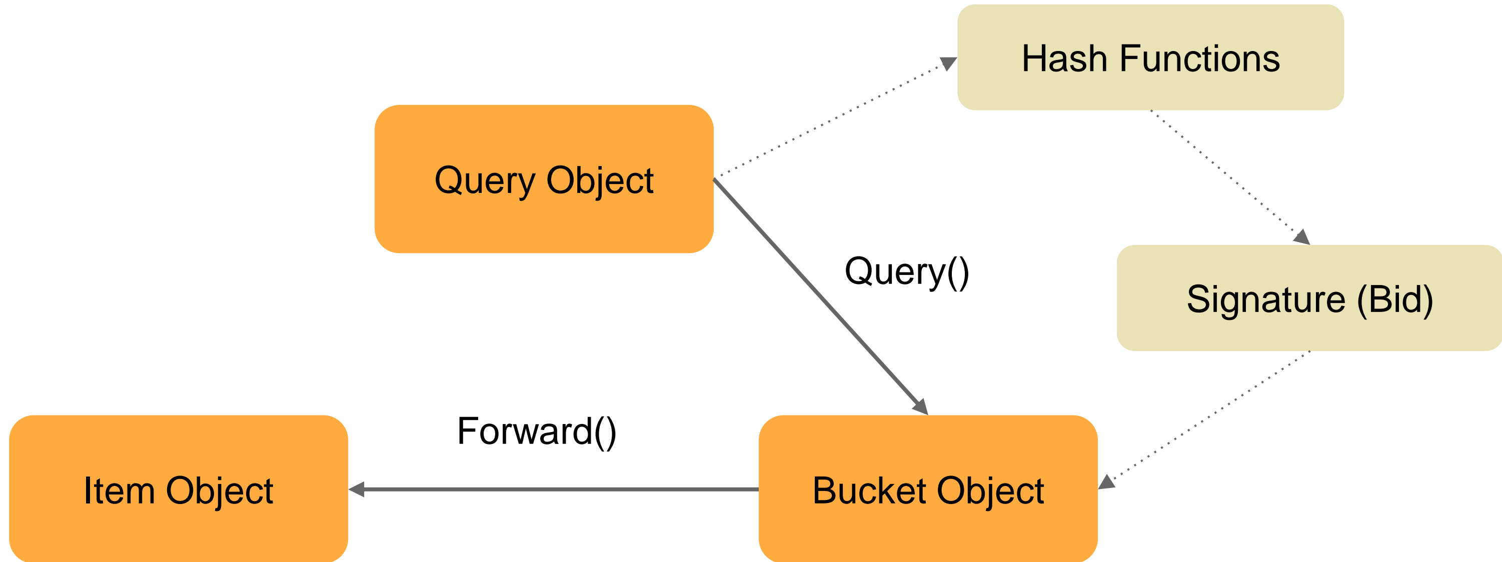
Item Object

Bucket Object

Abstraction: Query, Bucket and Item

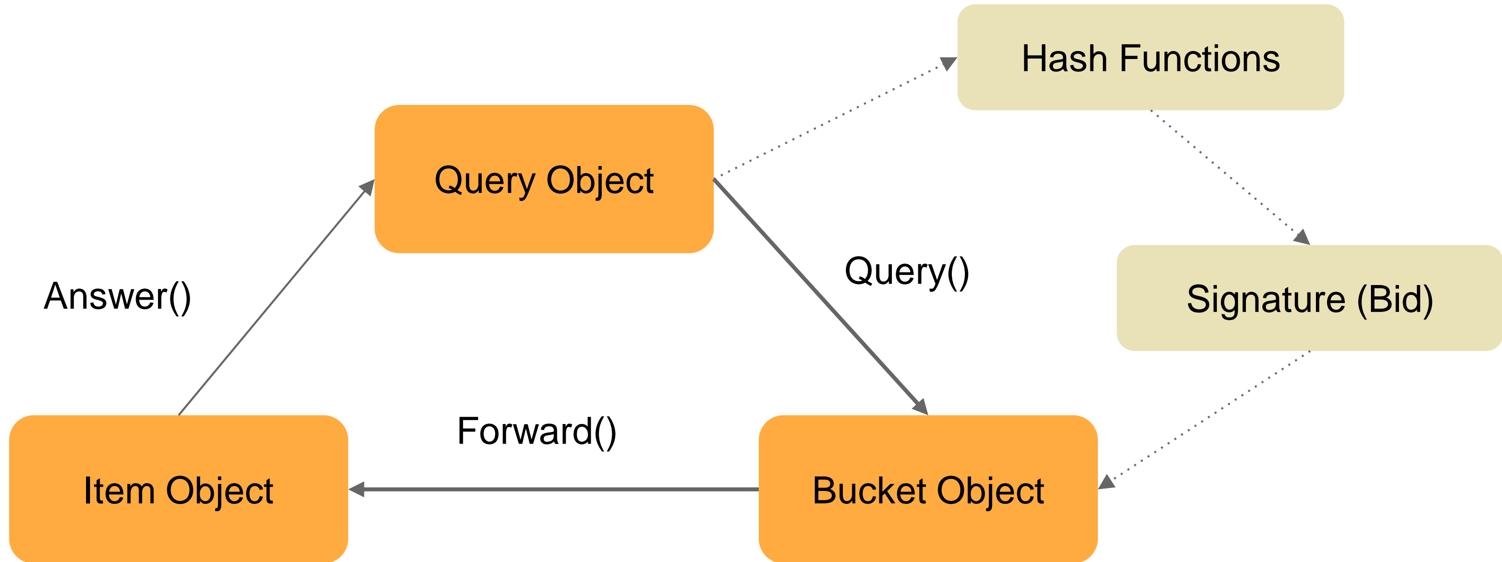


Abstraction: Query, Bucket and Item



Each bucket object has a list, storing all the item ids belonging to this bucket.

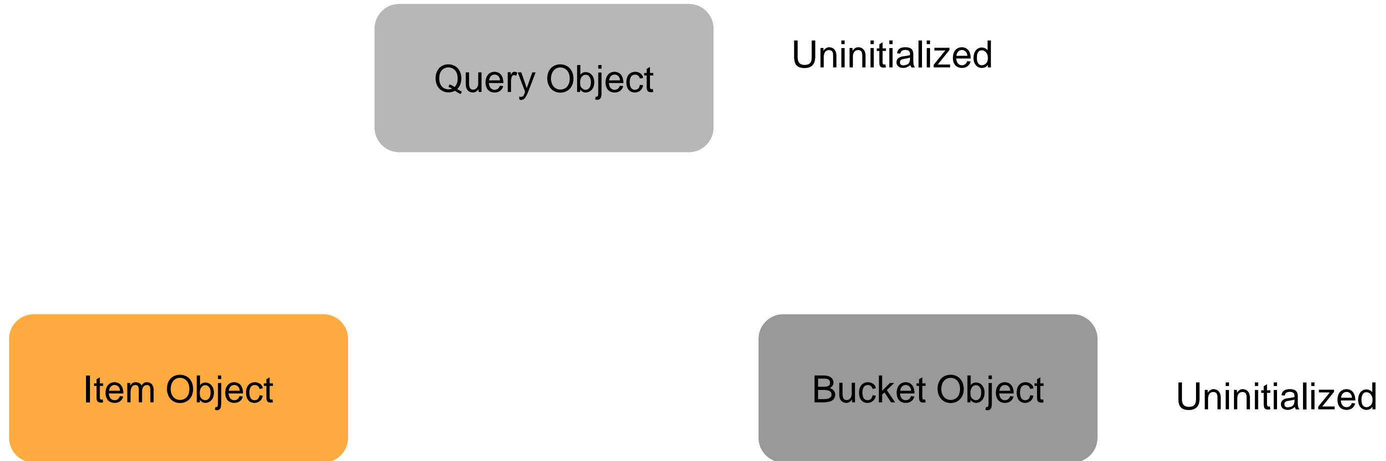
Abstraction: Query, Bucket and Item



Calculate **similarity score** between **item** and **query**, return the score as well as its ID to query object.

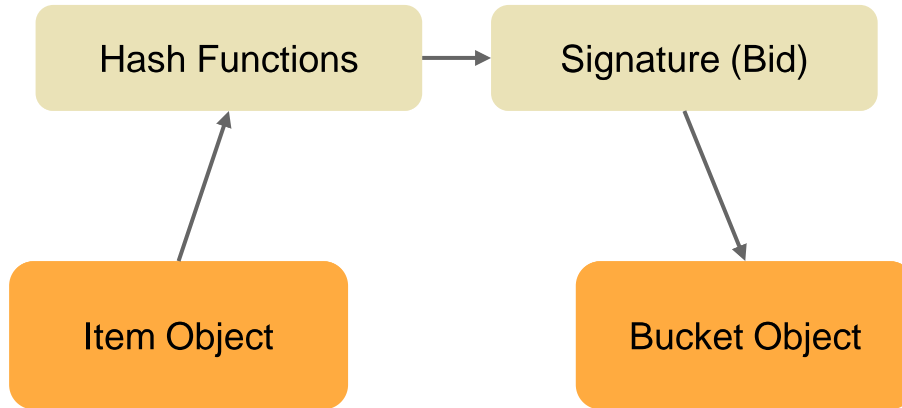
Each bucket object has a list, storing all the item ids belonging to this bucket.

Details in Initialization Stage



Read in **item** dataset from HDFS. Creates item objects. Item Object are distributed among different workers.

Details in Initialization Stage



Create bucket objects according to the generated **signatures**.
Bucket objects are also distributed among different workers.

Each bucket object maintains a list, storing all the item ids belonging to this bucket.

Details in Initialization Stage

Query Object

Read in **query** dataset from HDFS. Creates query objects. Query object are also distributed among different workers.

Item Object

Bucket Object

Details in Initialization Stage

Query Object

Broadcast query objects.
Each worker maintains an query object list, containing all the query vector.

Item Object

Bucket Object

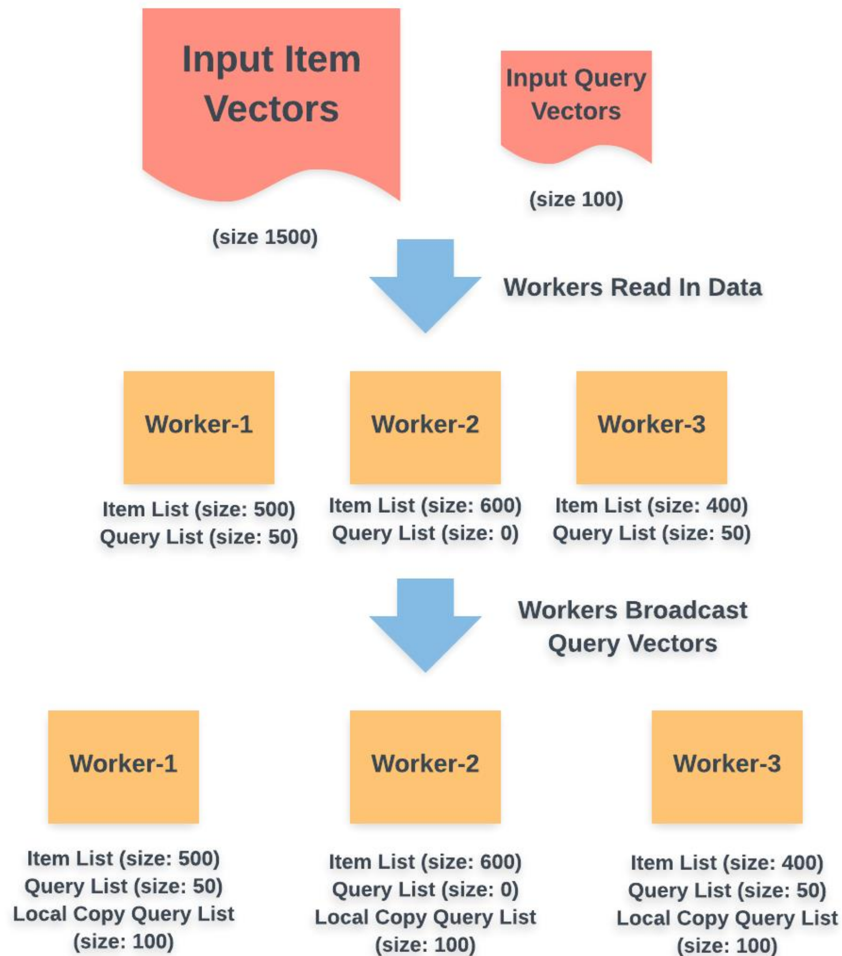
Details in Initialization Stage

Broadcast query vectors

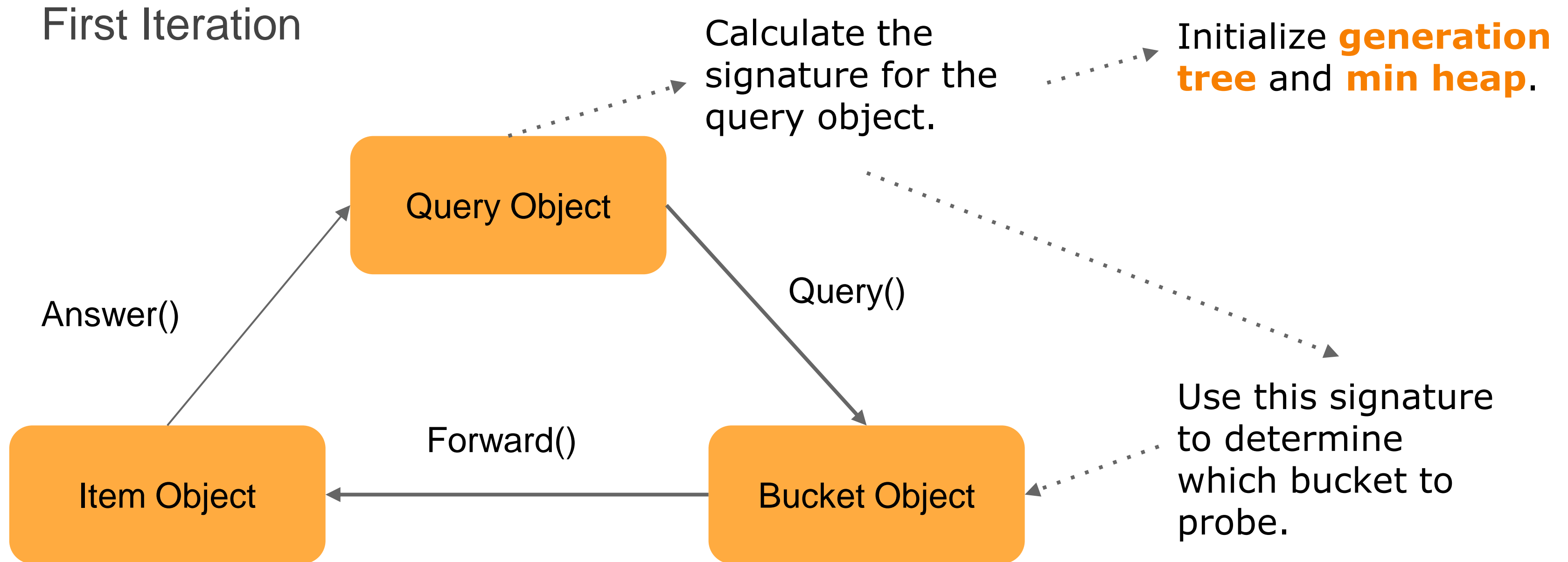
Pros:

1. No need to pass item vectors or query vectors among network.

2. Only need to pass query object ID to calculate similartiy.



Implement Multi-Probing in Iterations



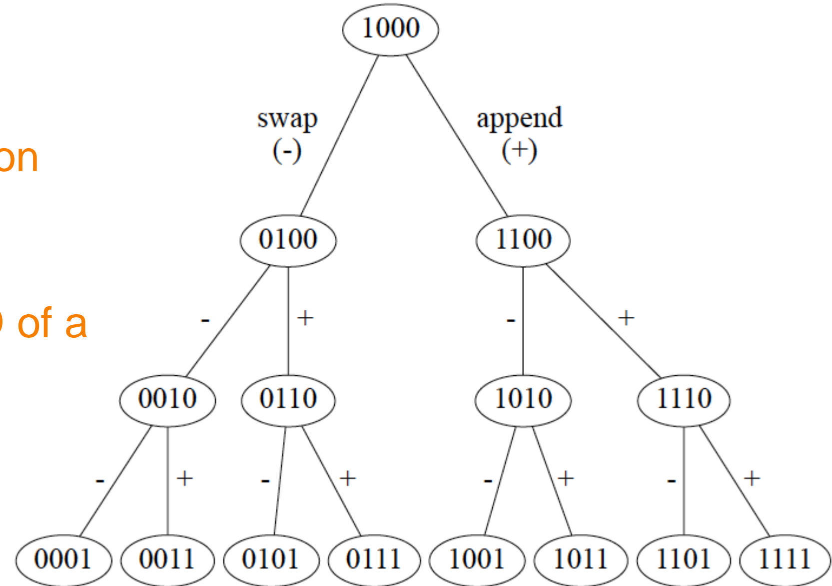
Bucket Generation

Generation Tree - Properties

Property 1. All possible sorted flipping vectors can be obtained from the generation tree exactly once.

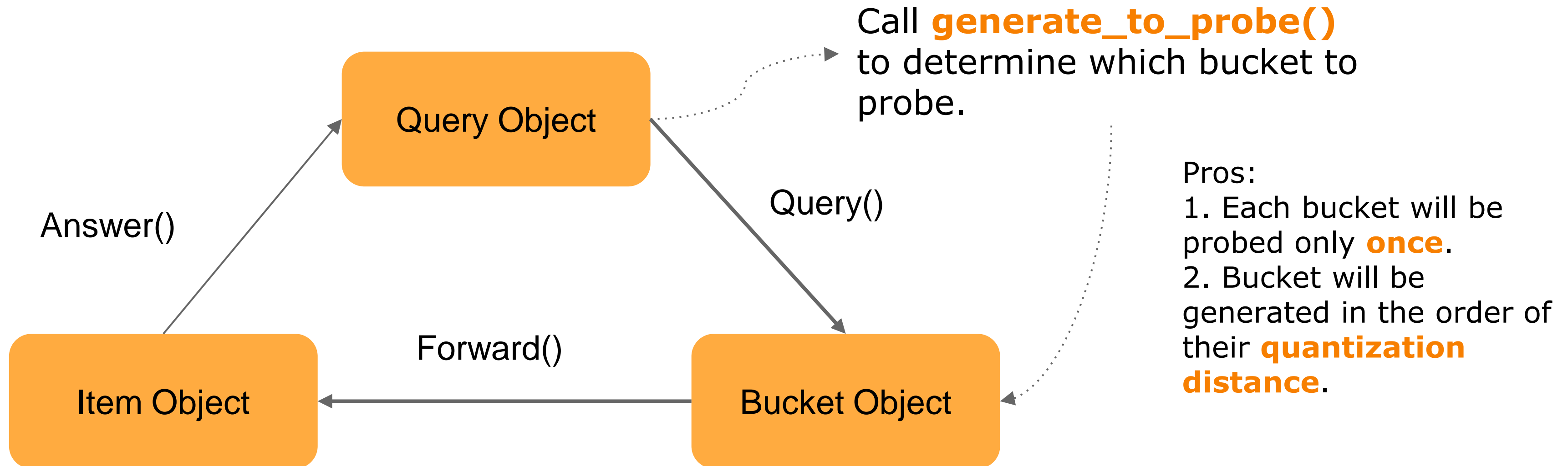
Property 2. In the generation tree, the QD of a child is always bigger than its parent.

QD of the child node can be obtained by a simple calculation from its parent node.

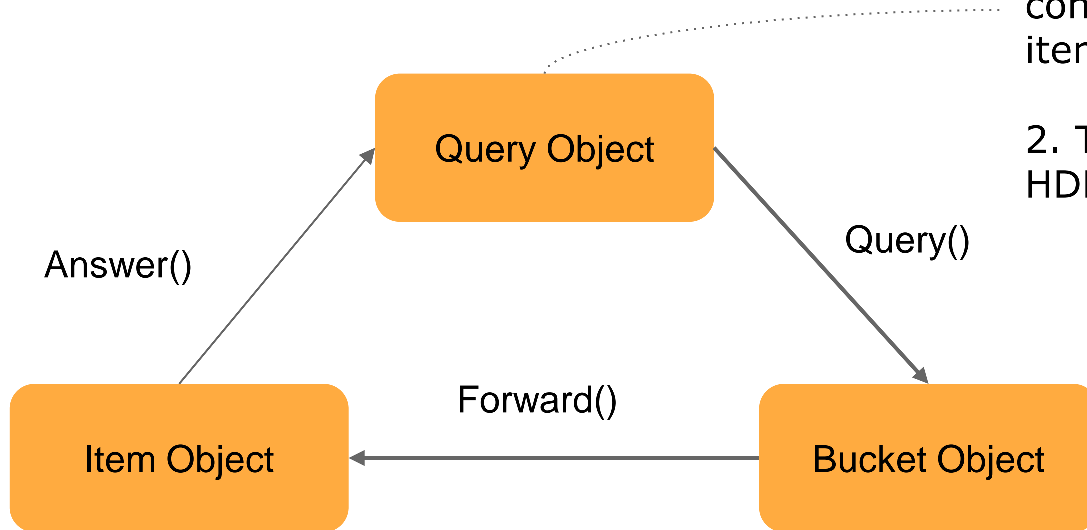


Implement Multi-Probing in Iterations

Later Iterations



Implement Multi-Probing in Iterations



1. Each query object will maintains a **set** which contains the **top-K** nearest item to this query.

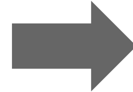
2. The result will be written to HDFS in the last iteration.

Implement Multi-Probing in Iterations

Message Reduction: Combiner

Src: Item A, Worker 1
Dst: Query X, Worker 2
Content: (Item A id, sim score)

Src: Item B, Worker 1
Dst: Query Y, Worker 2
Content: (Item B id, sim score)



Src: Worker 1
Dst: Worker 2
Content: [(Query X, (Item A id, sim score)), (Query Y, (Item B id, sim score))]

Experiments Results

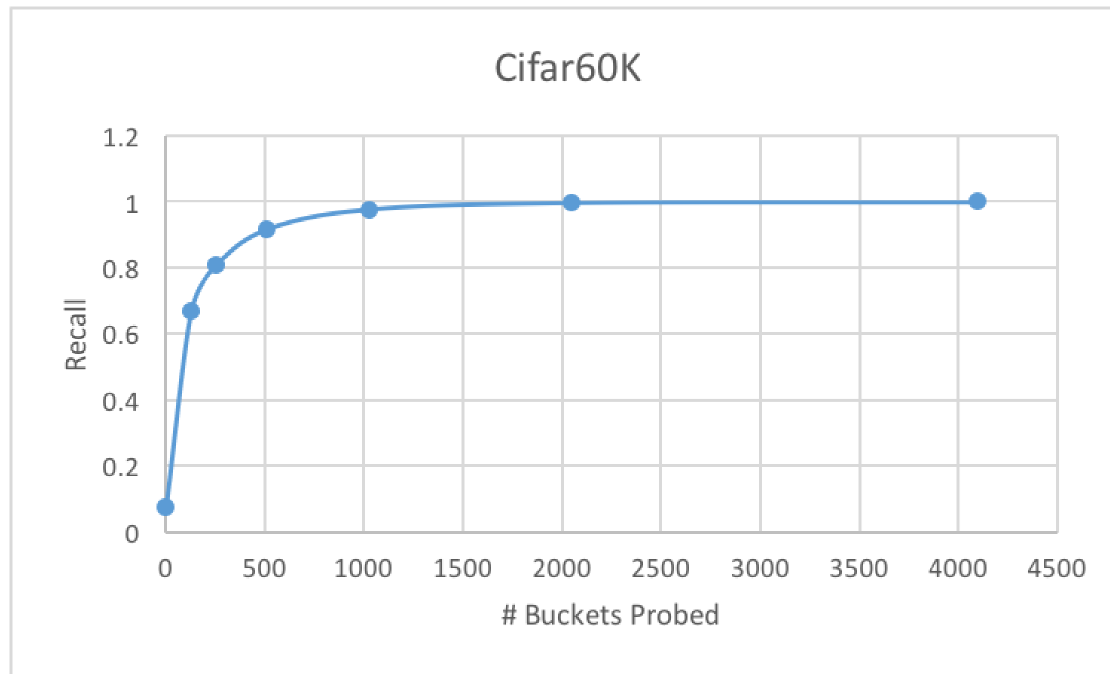
Datasets and Cluster setting

1. We ran our experiments on 6 machines and each machine ran 20 threads. Each thread is processed as a worker.

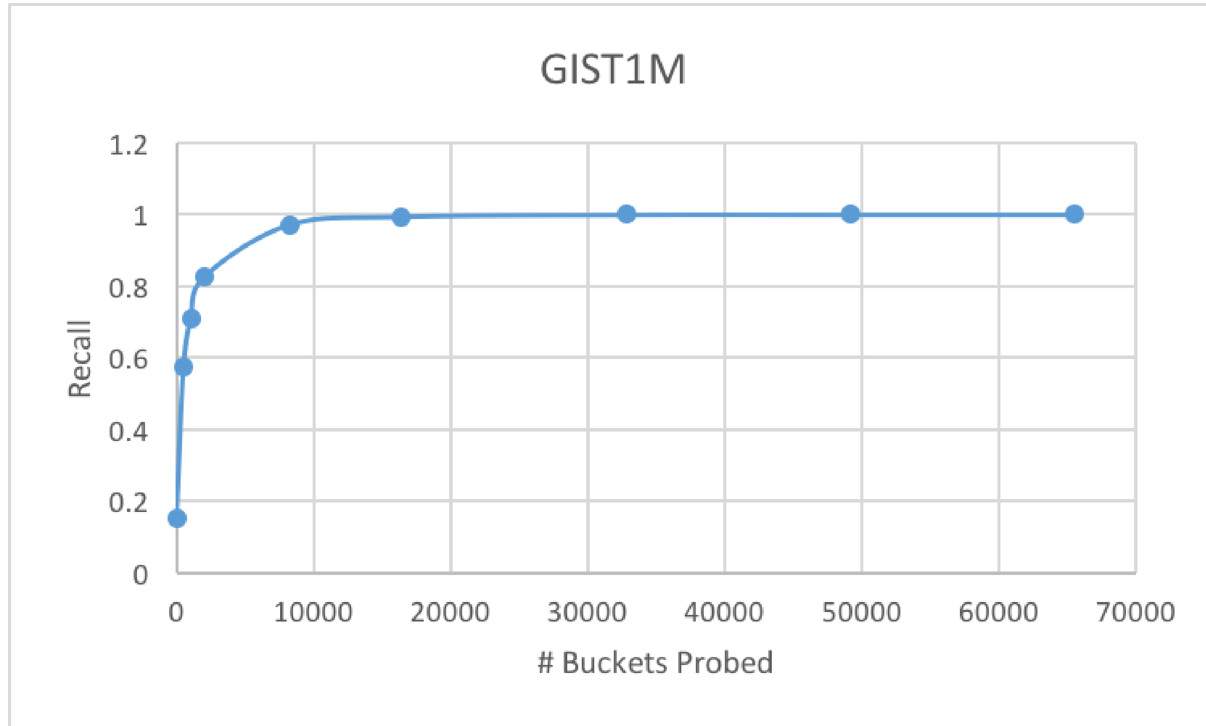
2. Dataset:

Dataset	Dimension Number	Item Number
CIFAR60K	512	60,000
GIST1M	960	1,000,000
TINY5M	384	5,000,000

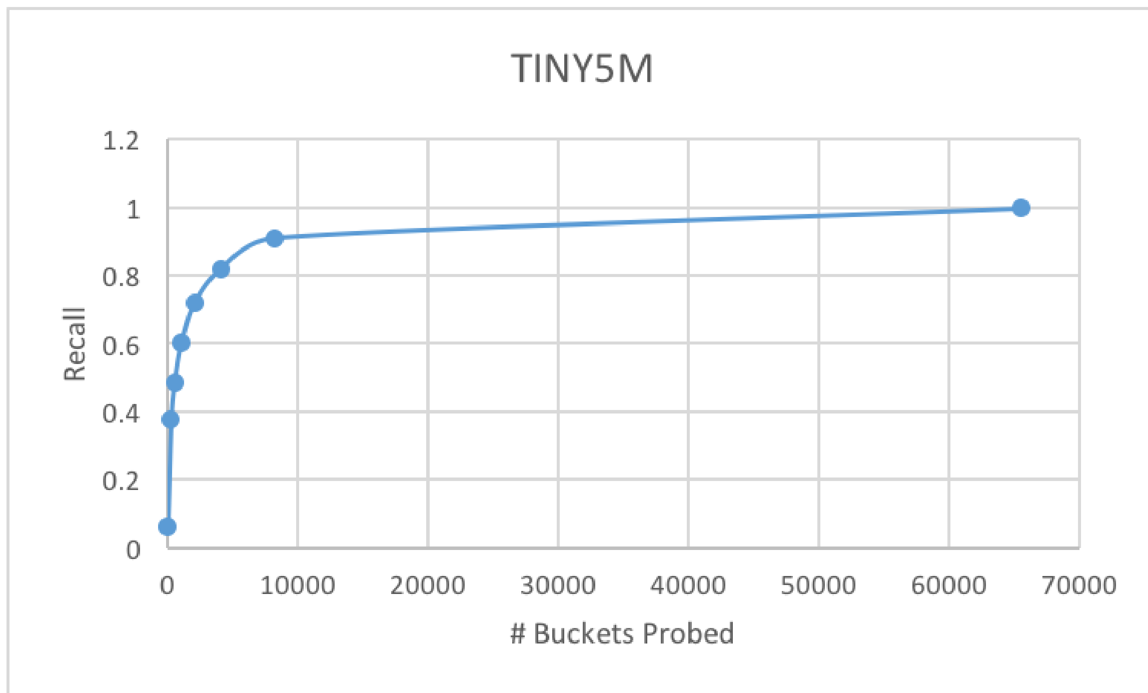
Experiments Results



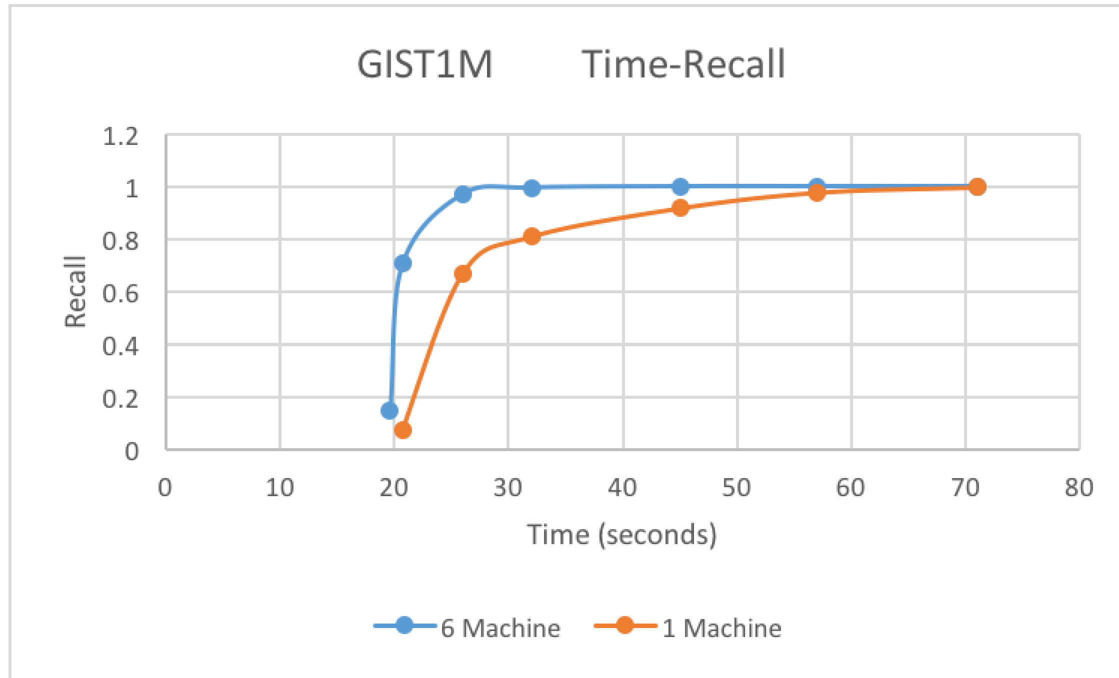
Experiments Results



Experiments Results



Experiments Results



References

1. Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. volume 36, pages 2227–2240, 2014.
2. Jinfeng Li, James Cheng, Fan Yang, Yuzhen Huang, Yunjian Zhao, Xiao Yan, and Ruihao Zhao. Losha: A general framework for scalable locality sensitive hashing. In SIGIR, pages 635–644, 2017.
3. Kaiming He, Fang Wen, and Jian Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In CVPR, pages 2938–2945, 2013.
4. Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In CVPR, pages 2946–2953, 2013.
5. Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In NIPS, pages 1753–1760, 2008.
6. Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In CVPR, pages 817–824, 2011.
7. Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In VLDB, pages 950–961, 2007
8. Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In Proceedings of the 21st ACM international conference on Information and knowledge management, pages 2174–2178. ACM, 2012.
9. Jinfeng Li, Xiao Yan, Jian Zhang, An Xu, James Cheng, Jie Liu, Kelvin K. W. Ng, Ti-chung Cheng. A General and Efficient Querying Method for Learning to Hash. In SIGMOD , pages 1333-1347, 2018

Acknowledgement

This is a joint work together with Jinfeng Li, Xiao Yan and Prof. James Cheng. We would like to express our special thanks of gratitude to them who gave us this opportunity to join this research work!

Q&A

Thank you!