VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity

OSDI '23

Background – Vector Search

- Vector search
  - Find K nearest vectors in a dataset.
- High-dimension vector
  - Deep learning maps data into **high-dimension vectors** and achieve complex semantic analysis through similar queries of high-dimensional vectors.
  - Queries on high-dimensional vectors become the cornerstone for many important **online** services.
- Problem of online high-dimension vector search services
  - **Strict latency** conflicts with the inherently **high cost of exact search** algorithm.
  - Force users settle on **approximate query** results on high-dimensional vectors.
- Approximate Nearest Neighbor Search - ANNS
  - Find K most similar vectors in a dataset**, Top-K**.
  - **Sacrifice search accuracy** for lower latency.

Background – Index of Database or ANNS System

➢Index of database system

- Elements in indexes are **scalars**.
- Use **monotonic** index like B-tree, B+-tree and more.
- Traverse the data-set guided by the index monotonically along a certain direction.

➢Index of ANNS system

- Elements in indexes are **vectors.**
- Index are often organized as a graph or cluster-based **irreular structure**.
- Traversing such vector indexes does **not guarantee a strict monotonic order**.

Background – Database System + ANNS

- Top K search with filter conditions
  - Find the most similar K cups for less than ￥100

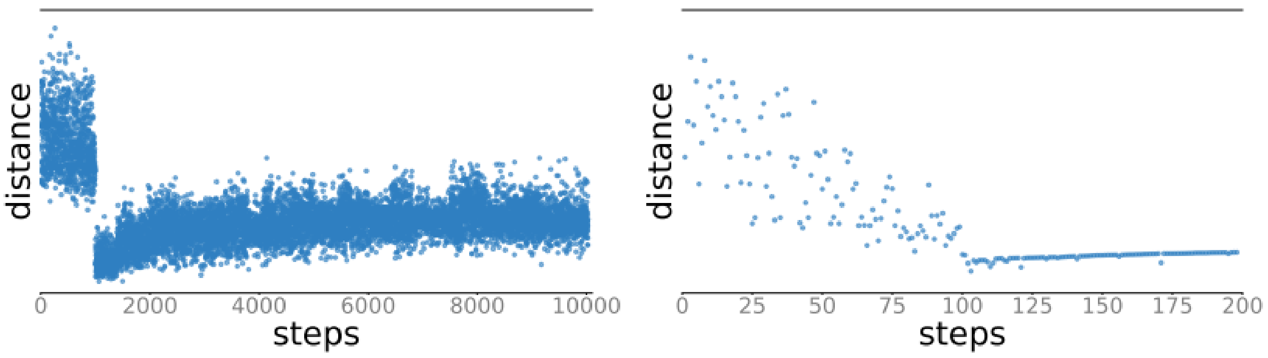
- Existing database systems that supports ANNS
  - First, set up a **tentative** index by using *TopK* interface of ANNS system.
  - Second, check the prices of $K`$ products in the index and filter out x products that meet the price below ￥100.

Problem

- It is difficult to predict the **right size of $\boldsymbol{K}$`** for the tentative index.
  - How to ensure $K` - x = K$?
  - choosing a **very large $\boldsymbol{K}$`** or perform **trial-and-error** with different sizes of $K$`.
  - Both methods can lead to excessive data access and computations.

Motivation

➢Well-designed vector indexes include a **two-phase** traversal pattern.



(a) FAISS IVFFlat (b) HNSW

Figure 1: Traversal patterns of two vector indices.

- Phase 1: approach the target vector region approximately in spite of **large oscillations**.
- Phase 2: stabilize and steadily departs from the target vector region in an approximate way.

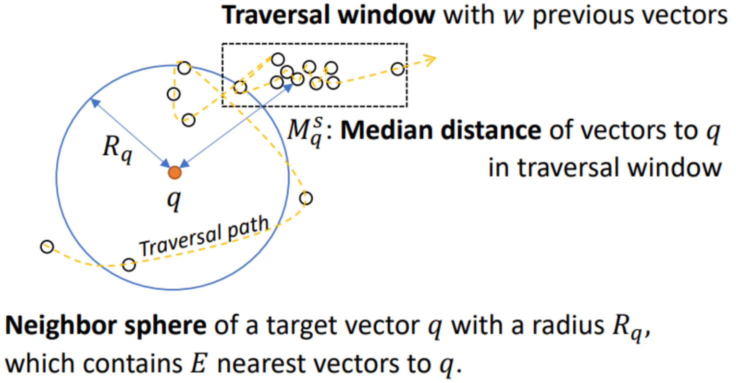
Design Overview

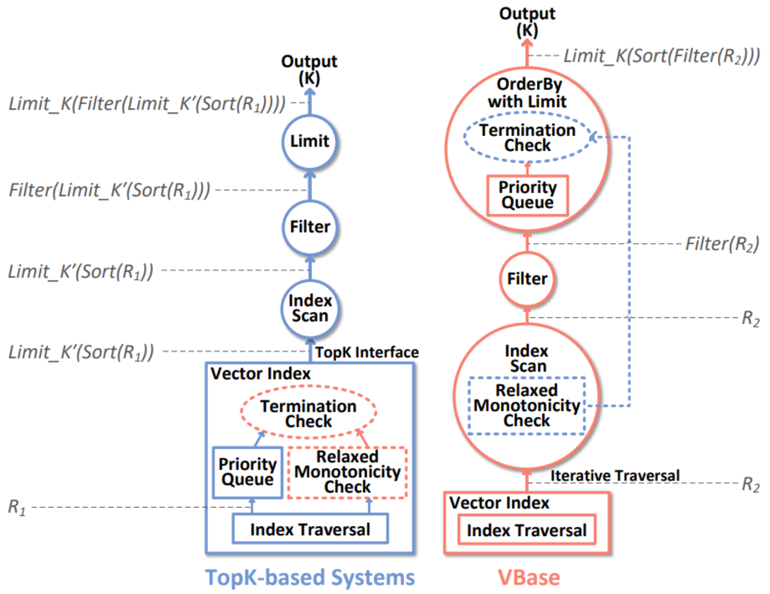
➢Relaxed Monotonicity

- As a termination condition, stop a query's execution timely.

➢Unified Query Execution Engine

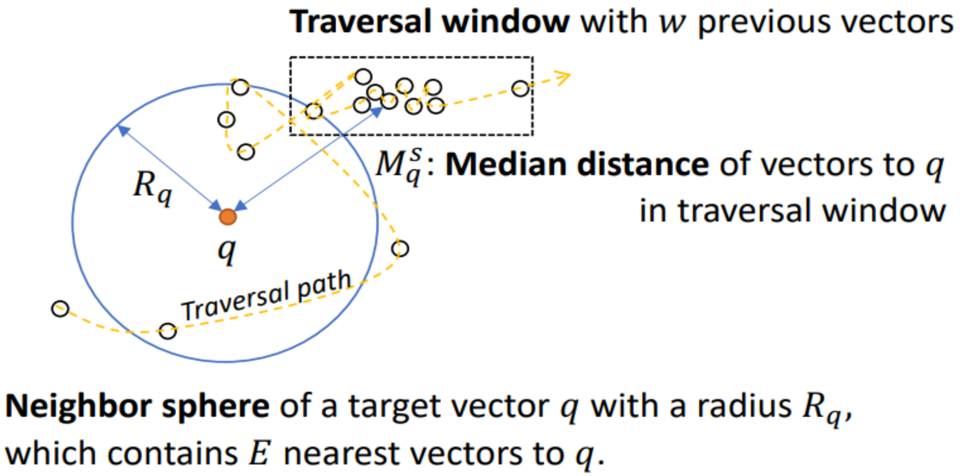
- support a wide range of queries on vector data in database system





Design1 - Relaxed Monotonicity

➢ The Formal Definition of Relaxed Monotonicity.

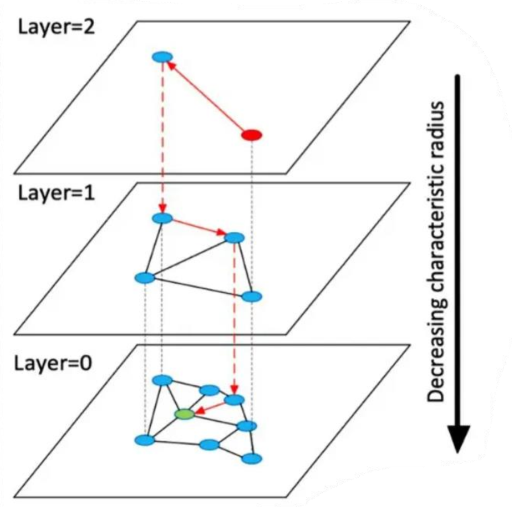


$$R_q = Max(TopE(\{Distance(q, v_j) | j \in [1, s-1]\})),$$

$$M_q^s = Median(\{Distance(q, v_i) | i \in [s-w+1, s]\})$$

$$\boxed{\exists s, M_q^t \geq R_q, \forall t \geq s.}$$

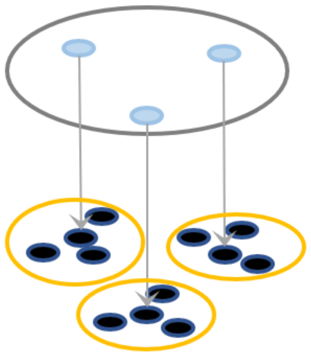
Design1 - Relaxed Monotonicity



graph-based

- Four general components for mainstream vector indexes
  - Index traversal to navigate the vector data-set;
  - Termination check to detect query termination signal;
  - **Monotonicity check** to determine if a query enters Phase 2;
  - Priority queue for keeping K nearest vectors so far.
- Graph-based vector indexes, such as HNSW
  - **Sorted candidate queue**, size that can abstract as E .
  - Compare the unvisited neighbors with vector in the queue, the abstract $w = 1$.
- Partition-based vector indexes, such as FAISS IVFFlat
  - Traverse over the centers, identify m closest clusters,

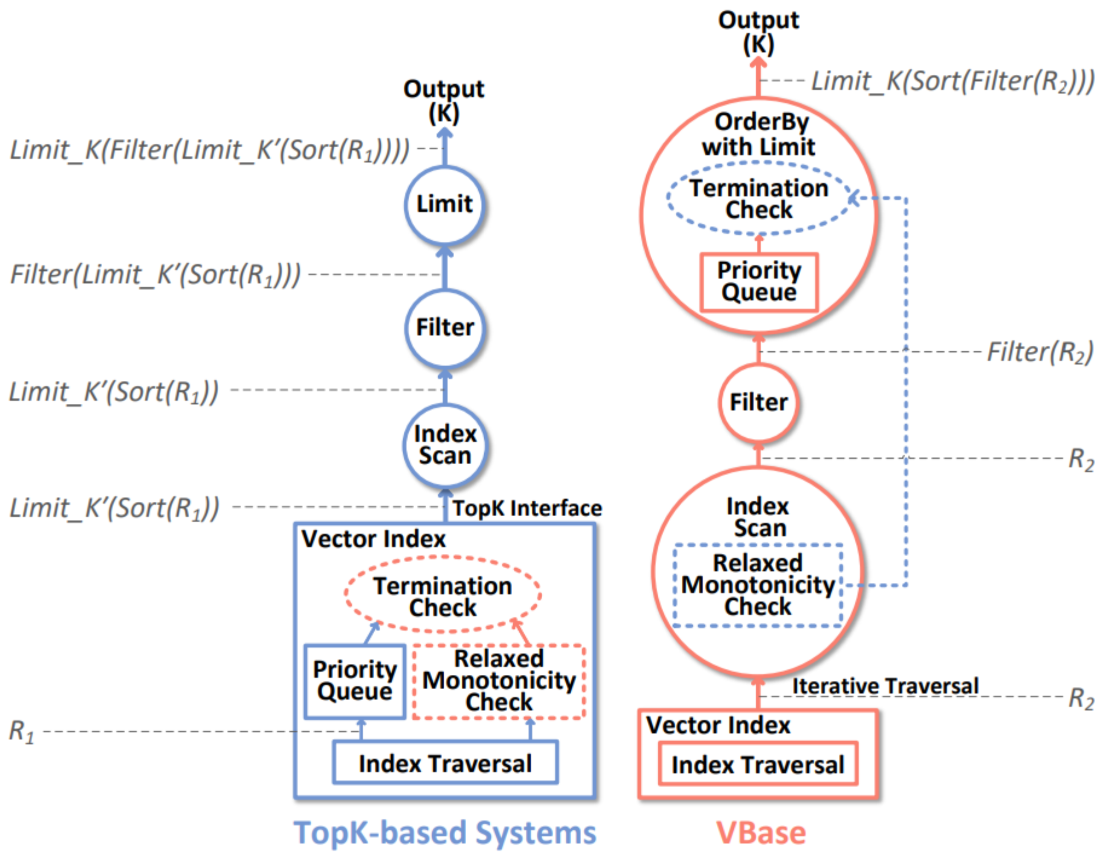
  the abstract $w =$ the number of total vectors in m clusters. $E = K$



partition-based

Design2 - Unified Query Execution Engine

- Traditional engine
  - Database : Volcano model (iterator model).
  - Vector Index: expose $TopK$ interfaces only.
- Unified Query Execution Engine
  - Modify the interface of vector indexes to support iterative traversal.
  - Modify the termination condition based on relaxed monotonicity.
  - Result Equivalence.



Evaluation

- Baseline systems
  - Milvus
  - Elasticsearch
  - PASE
  - PostgreSQL
- Vector similarity queries in SQL

- *Q1: Single-Vector* `TopK`.
- *Q2: Single-Vector* `TopK` + *Numeric Filter.*
- *Q3: Single-Vector* `TopK` + *String Filter.*
- *Q4: Multi-Column* `TopK`.
- *Q5: Multi-Column* `TopK` + *Numeric Filter.*
- *Q6: Multi-Column* `TopK` + *String Filter.*
- *Q7: Vector Range Filter.*
- *Q8: Join.*

Evaluation

Table 4: 8 Queries Result Overview (Latency: ms)

System	Q1:Single-Vector TopK				Q2:Single-Vector TopK+Numeric Filter				Q3:Single-Vector TopK+String Filter				Q4:Multi-Column TopK			
	Recall	Latency			Recall	Latency			Recall	Latency			Recall	Latency		
		average	median	99th		average	median	99th		average	median	99th		average	median	99th
PostgreSQL	1	2,980.1	3,021.7	3,133.6	1	1,108.3	1,124.1	2,286.2	1	4,322.2	3529.3	9,953.0	1	5,610.0	5,604.7	5,769.8
PASE	0.9949	**4.8**	**3.5**	**5.1**	0.9987	29.3	28.7	61.7	0.9982	13.2	10.7	17.9	-	-	-	-
Milvus	0.9949	9.4	9	12.7	0.9919	33.7	23.9	121.4	-	-	-	-	0.9041	6,696.4	8,349.3	9,299.0
Elasticsearch	0.9949	43.1	41.8	48.9	0.5010	97.9	98.1	118.1	0.8378	79.9	90.0	100.9	-	-	-	-
VBase	*0.9949*	*4.9*	*3.9*	*5.3*	***0.9989***	***11.7***	***6.3***	***51.7***	***0.9983***	***7.9***	***6.7***	***10.4***	***0.9696***	***19.8***	***18.4***	***46.4***

System	Q5:Multi-Column TopK+Numeric Filter				Q6:Multi-Column TopK+String Filter				Q7:Vector Range Filter				Q8:Join			
	Recall	Latency			Recall	Latency			Recall	Latency			Recall	Latency		
		average	median	99th		average	median	99th		average	median	99th		average	median	99th
PostgreSQL	1	1,192.9	1,234.4	2,343.6	1	6,543.2	5,996.3	16,734.6	1	8,244.9	8,212.6	8,641.6	1	129,051,273.9	-	-
PASE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Milvus	0.9691	12,637.9	5,617.4	36,887.9	-	-	-	-	-	-	-	-	-	-	-	-
Elasticsearch	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
VBase	***0.9805***	***35.8***	***24.9***	***160.7***	***0.9626***	***21.6***	***18.3***	***64.8***	***0.9840***	***10.8***	***2.2***	***168.9***	***0.9992***	***16,335.9***	-[1]	-[1]

- Compared to existing systems, VBase significantly improves the average latency, median latency, and tail latency of queries

Conclusion

Similarity Search and Relational Queries →

- Index
  - Elements
  - Structure
  - Monotonicity

  → Relaxed Monotonicity
- Execution Engine
  - Volcano Model
  - $TopK$ interface

  → Unified Query Execution Engine