GaussDB-Vector:

A Large-Scale Persistent Real-Time Vector Database for LLM Applications

Ji Sun
Tsinghua University
sunji@tsinghua.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

James Pan
Tsinghua University
jamesjpan@tsinghua.edu.cn

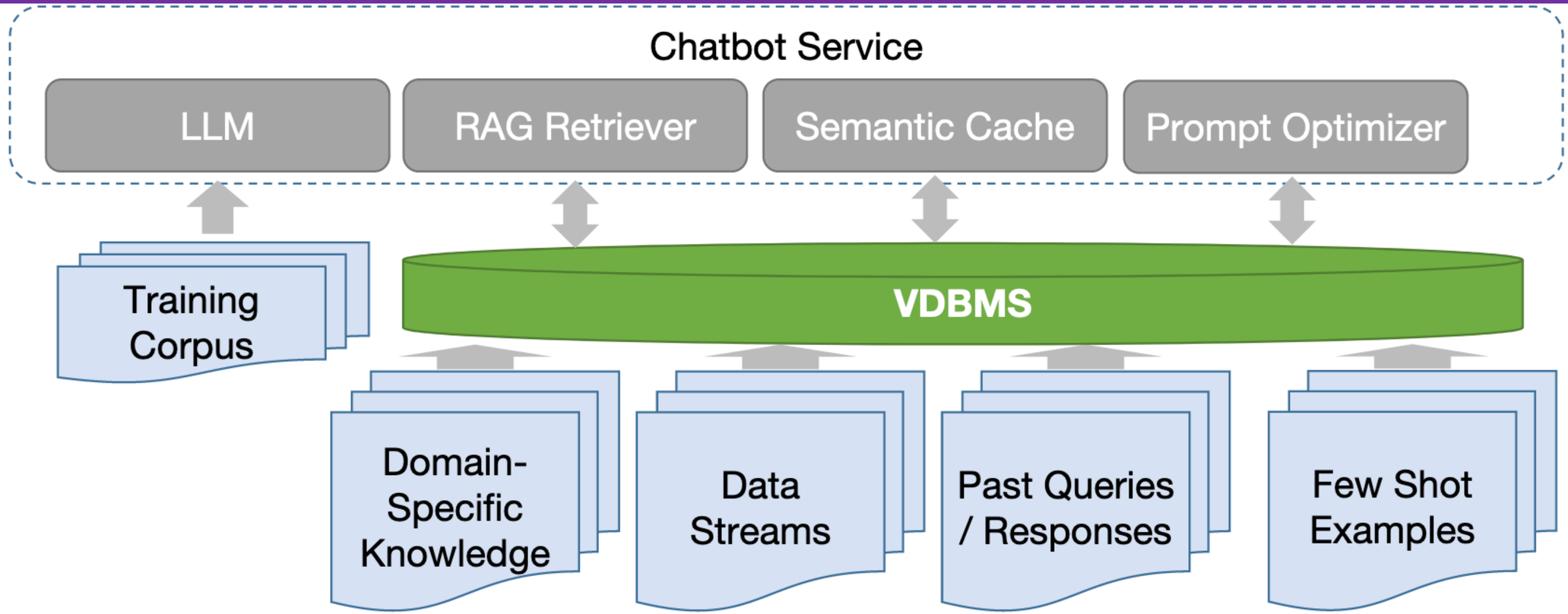
Jiang Wang
Huawei
wangjiang16@huawei.com

Yongqing Xie
Huawei
xieyongqing1@huawei.com

Ruicheng Liu
Huawei
liuruicheng1@huawei.com

Wen Nie
Huawei
niewen2@huawei.com

Introduction - VDBs and LLMs



Search Capabilities

- Support various queries, e.g. k-NN, Range NN, Predicated (Filtered) Search

Performance

- Low latency queries
- High throughput
- Memory efficient

Quality

- High recall
- High semantic relevance

System-Level Features

- Scalable
- Available
- Elastic
- Consistent
- Persistent

Vector Databases: Key Challenges

More and more applications rely on embedding vectors that are retrieved via similarity search

Goal: Store embeddings in a DB and retrieve for whatever downstream task

1) **Size: Existing hardware not for large data units**

2) **Structure: Lack of structure (e.g. attrs) makes embeddings hard to index**

Performance

- **Latency**
  - Fundamentally high I/O due to more frequent paging
  - Naive disk-resident HNSW is not I/O optimal
  - High latency of filtered search

Quality

- **Recall**
  - Accuracy degradation of IVF due to data drift following updates

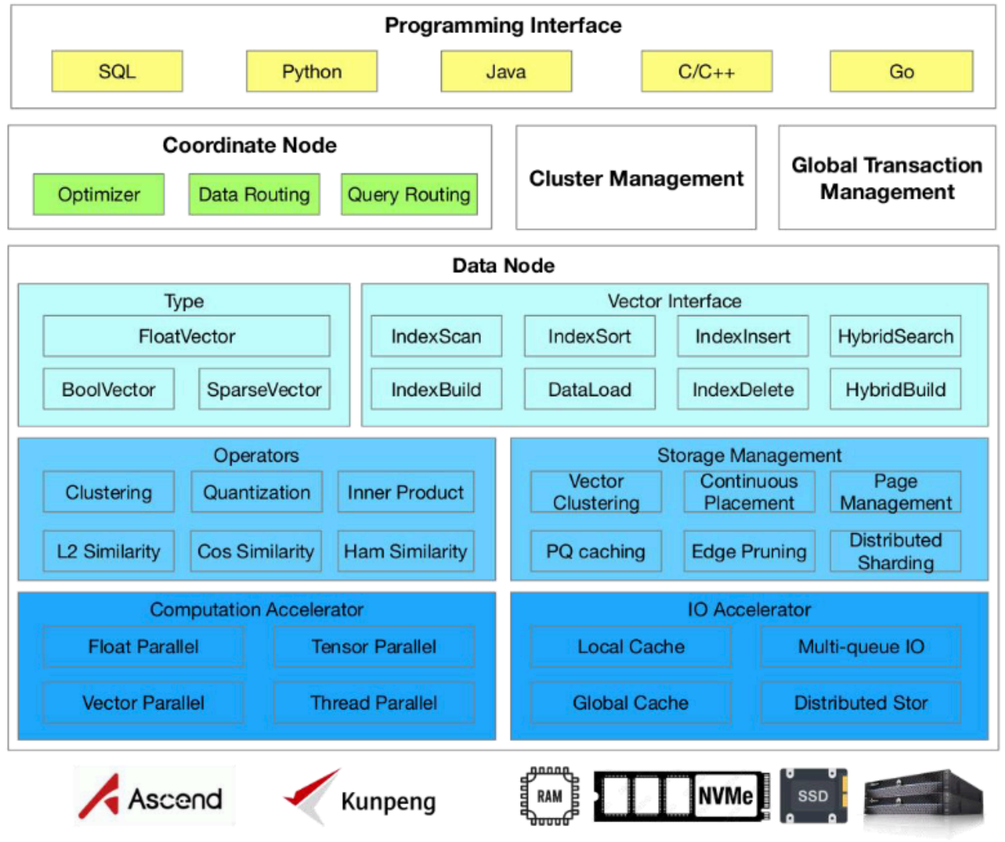
System-Level Features

- **Scalability**
  - Unclear sharding and query routing strategy for distributed deployments

GaussDB-Vector

Build a distributed persistent real-time database based on IVF & DiskANN

- **Architecture:** Two-Tiered Distributed Database to provide scalability / availability
- **Techniques:** IVF & DiskANN but make it easier to build, update, and search



Index Storage & Construction

- Two-Phase Neighborhood Pruning
- **Tailored Storage Structures to reduce I/O**
- **Data Sharding with Drift Detection & Auto Rebalancing**
- **Balanced Tree Hybrid Index**

Search

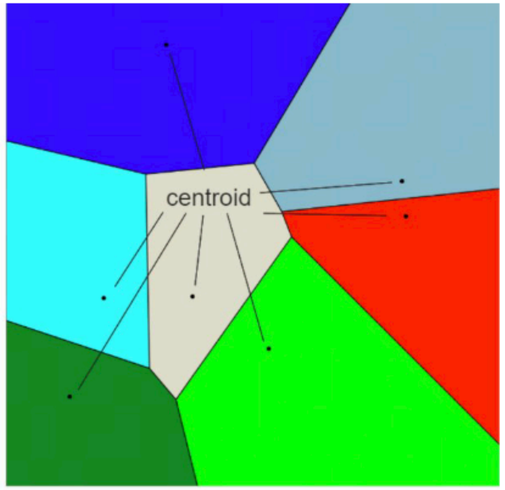
- Hot Nodes Buffering
- **Cardinality-Based Query Routing for Distributed Search**
- I/O Efficient Search Algorithms

Update

- Asynchronous Delete
- **Incremental Index Updates for data freshness**

IVF ANN INDEX

1



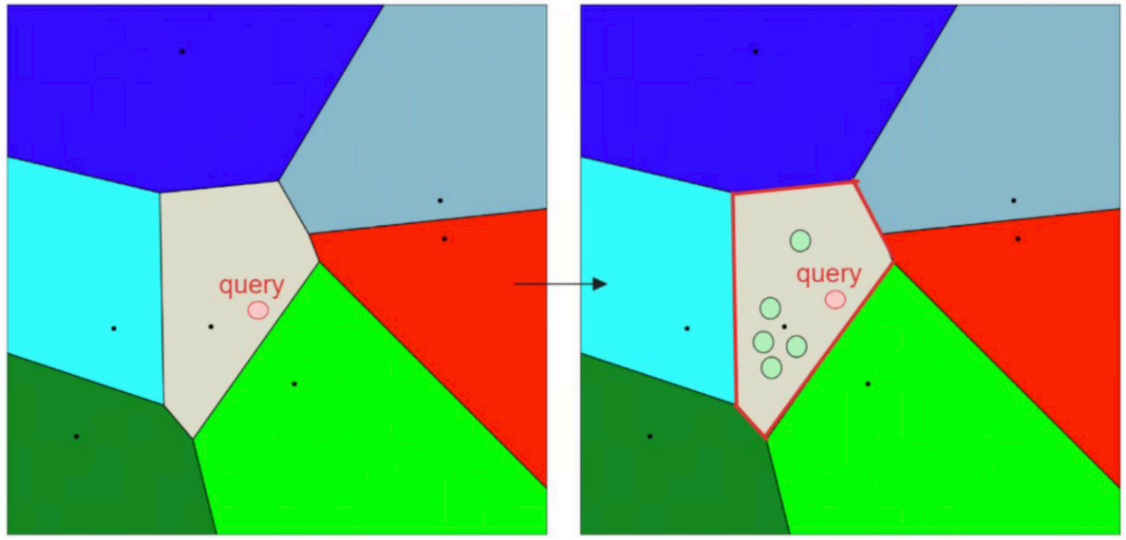
Group vectors into clusters, each represented by a **centroid**

2

centroid	vectors
centroid_1	[vector_1, vector_2, ...]
centroid_2	[vector_3, vector_4, ...]
centroid_3	[vector_5, vector_6, ...]
...	...

Create index to map centroid to its cluster's vectors

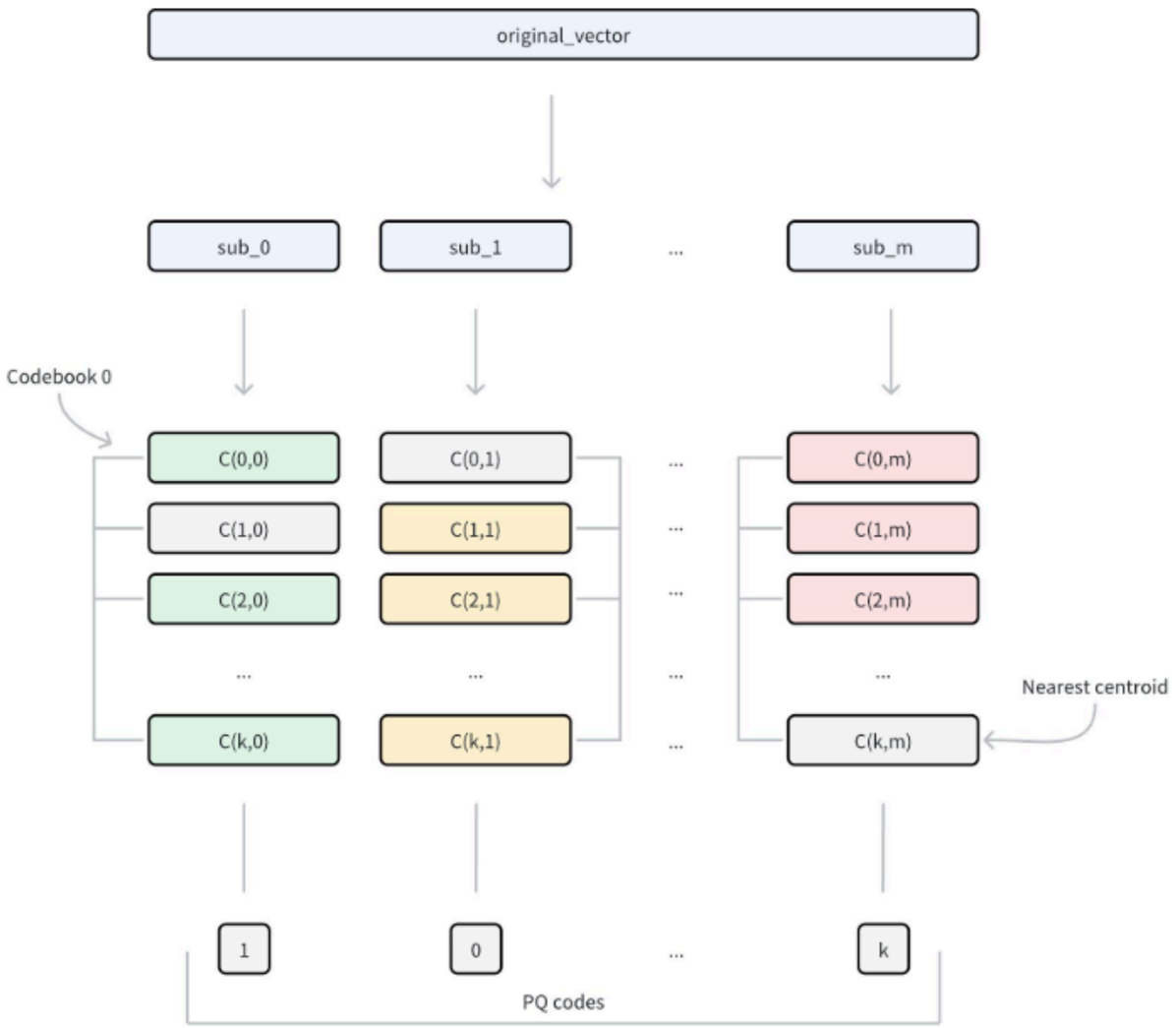
3



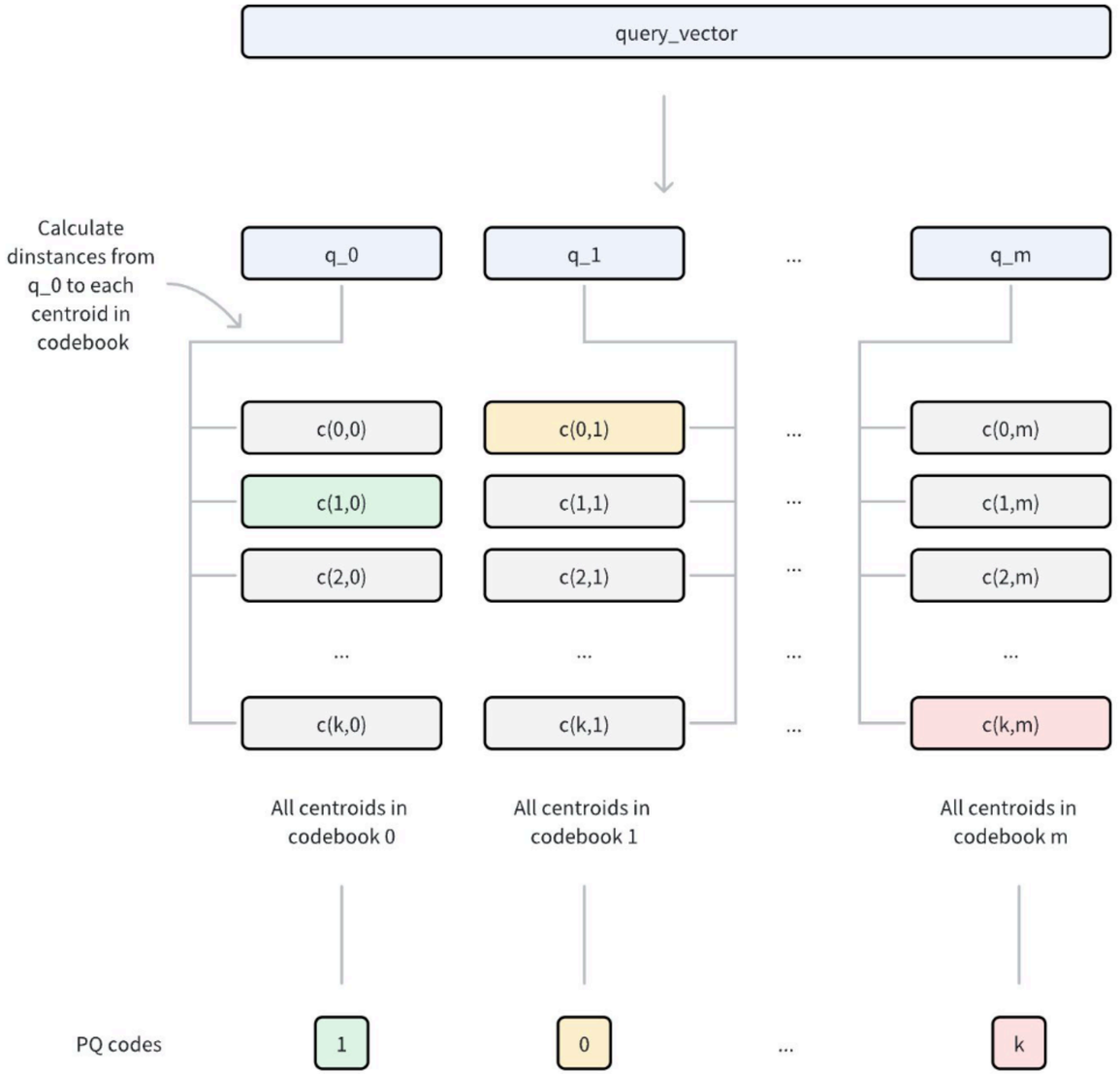
When a query vector is provided:

- Calculate its distance to each centroid to find the nearest clusters.
- Retrieve vectors from these clusters using the inverted index.
- Perform similarity computations only within the retrieved vectors to identify the most similar results.

IVF-PQ



- **k**: Number of centroids per subspace, k = 2^nbits.
- **m**: Number of sub-vectors the original vector is split into.
- **nbits**: Bits used to encode each centroid's index.



STRUCTURE OF IVF ANN INDEX

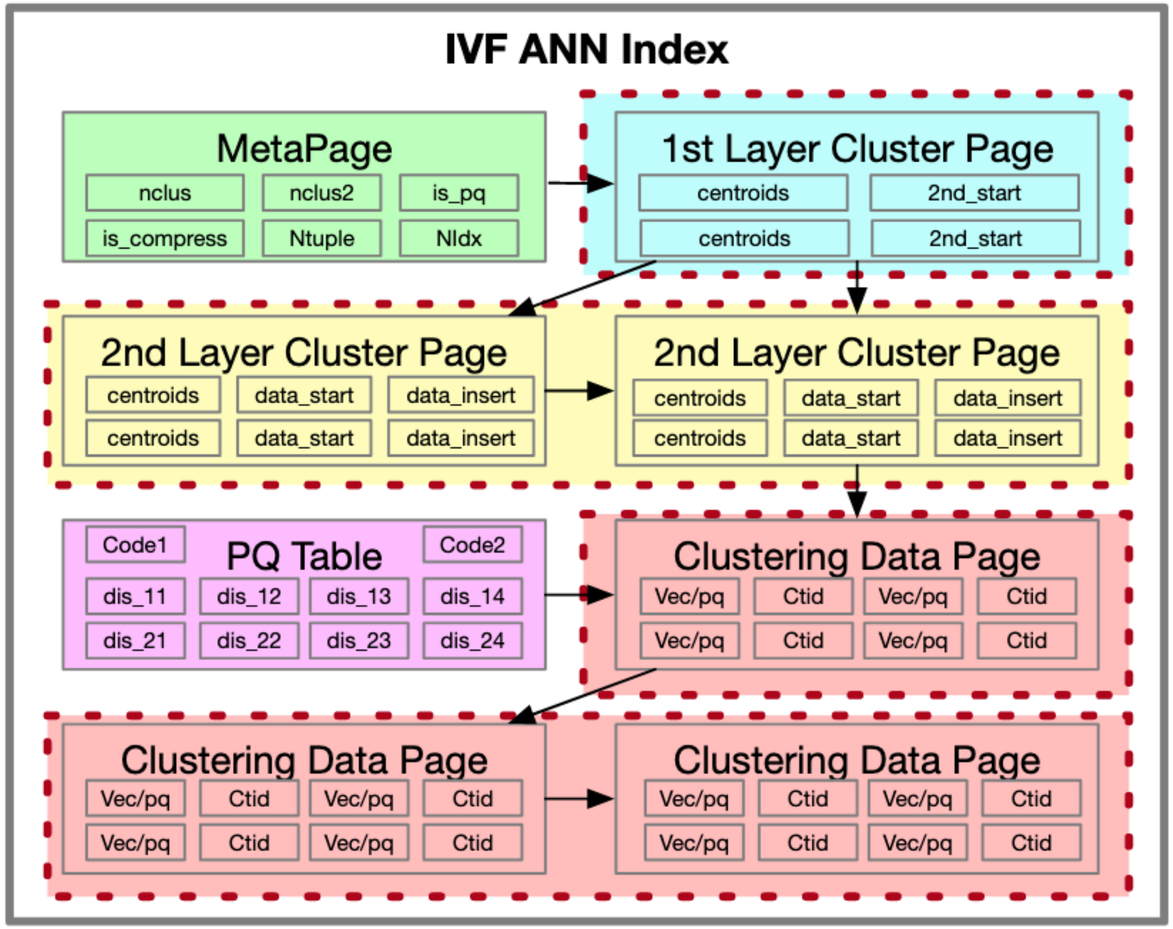


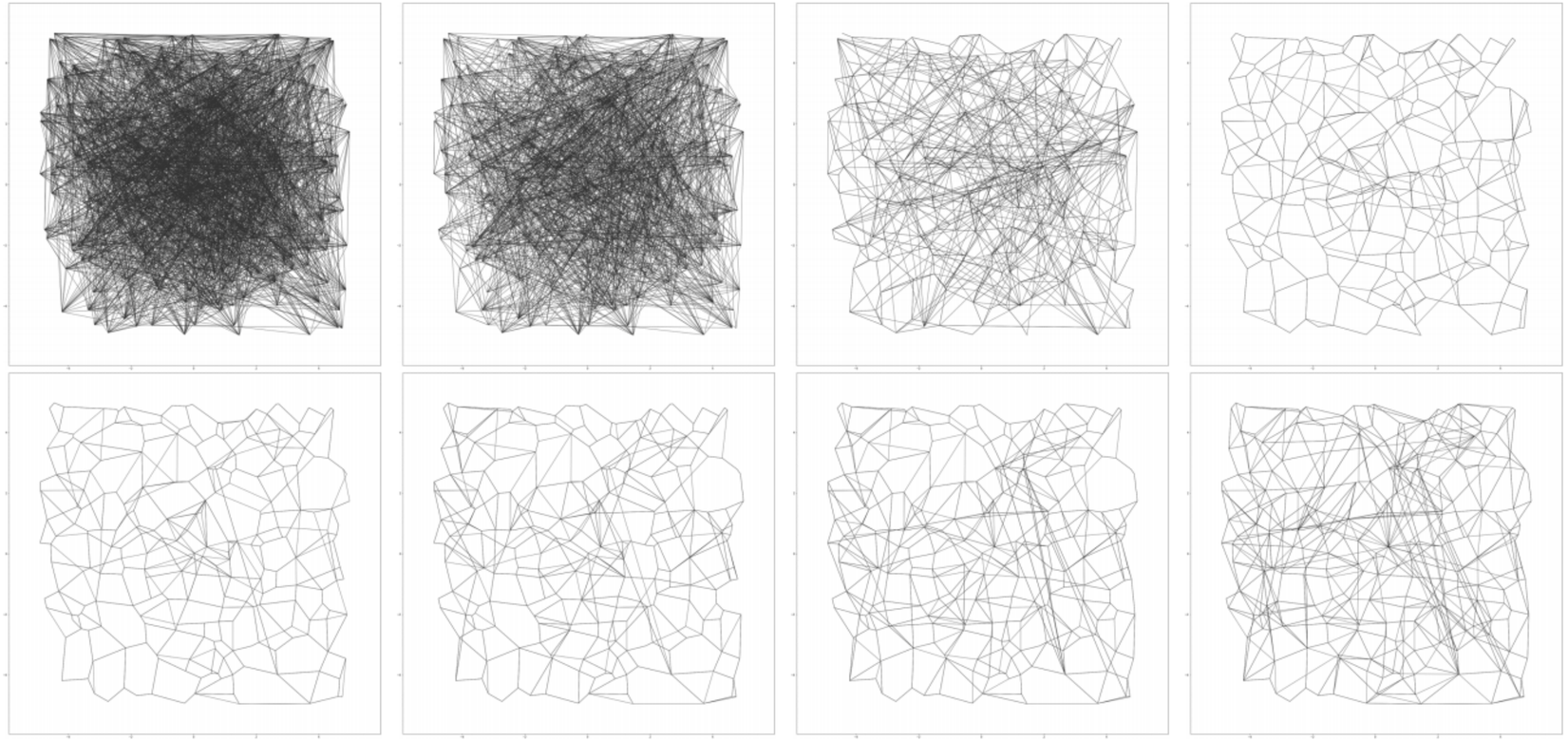
Figure 2: Structure of IVF Index

Uses two-layer clustering for the IVF index in order to balance the load of clusters.

Selects top- Nclusters and top-M sub-clusters based on centroid similarity.Set N*M to cover 5–10% of the dataset proves effective in achieving high recall with minimal latency

Binary sort on vectors according to distance to the query.

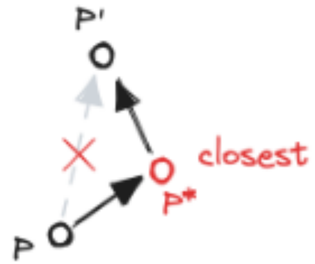
GRAPH ANN INDEX CONSTRUCTION



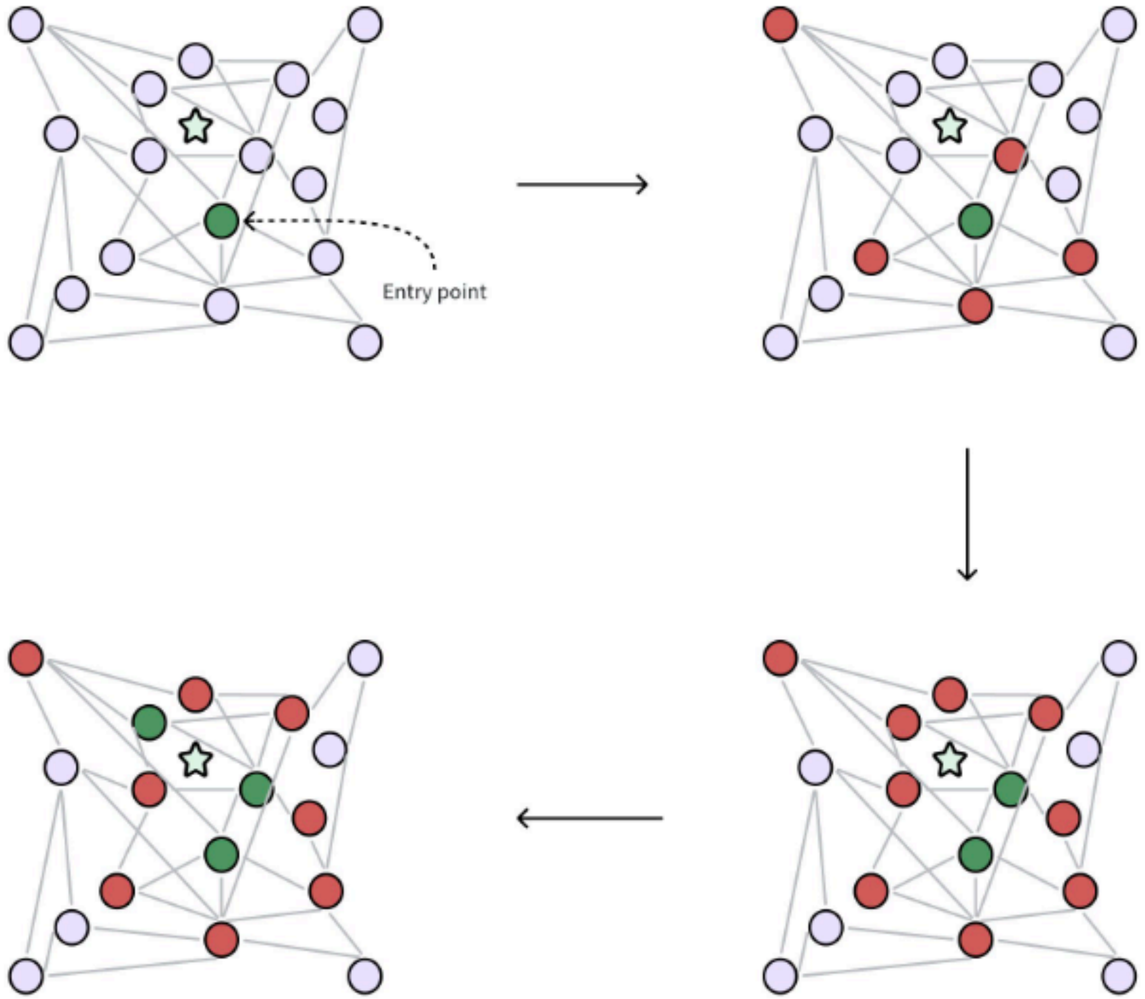
Vamana

1.Random Initialization
2.Navigation Point Selection
3.First Iteration (α=1）: For every point, an approximate nearest neighbor search is run starting from the navigation point. All nodes visited along the search path are added to the candidate set, followed by edge pruning with α=1to establish connectivity.
4.Refinement Iteration (α>1): To address the low quality of the initial random graph, the search and pruning process is repeated with a relaxed threshold (e.g., α=1.2).

$$\text{alpha} * d(p^*, p') < d(p, p')$$



GRAPH ANN INDEX

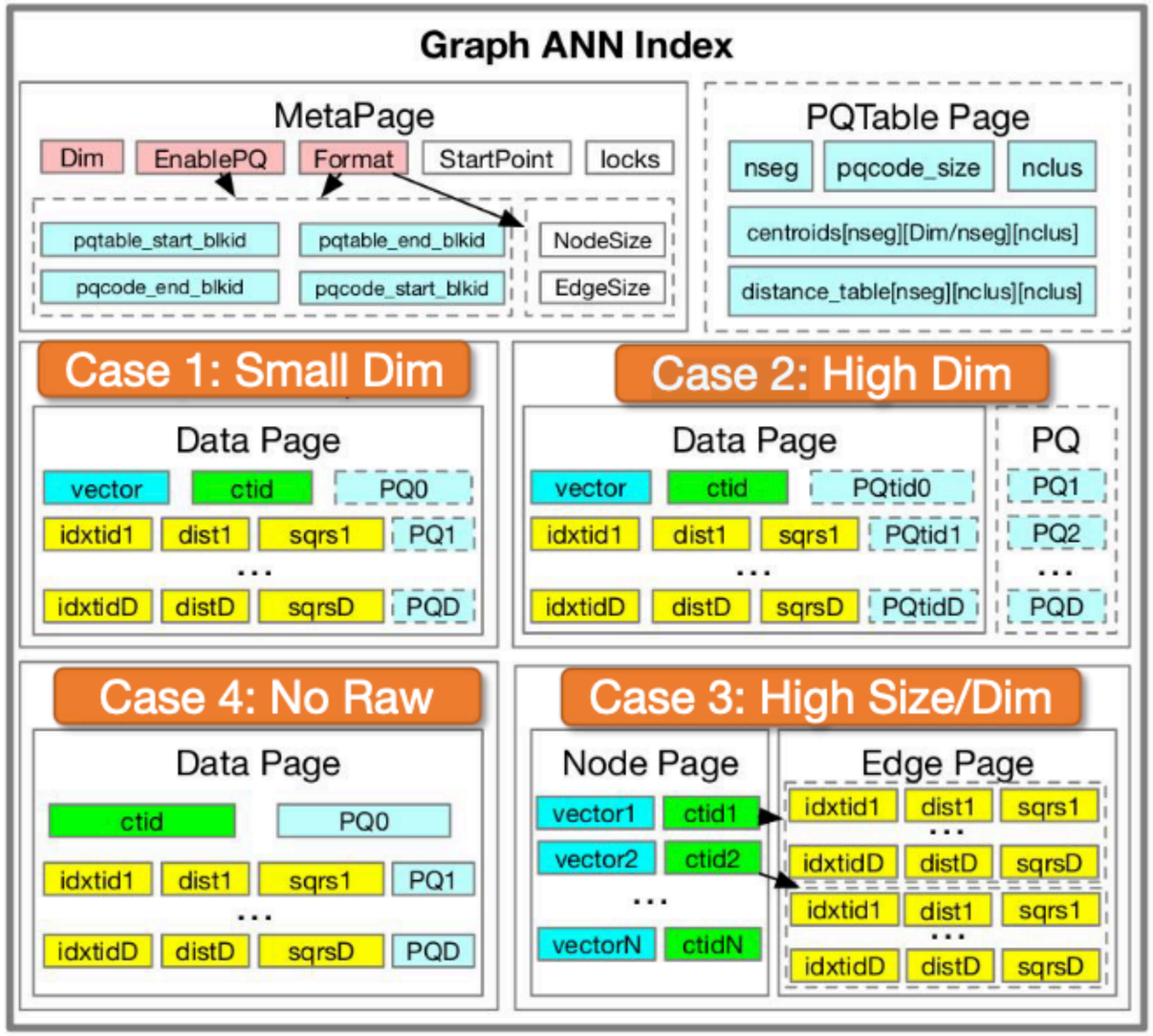


Optimization Strategies：

- Scalable Graph Construction via Overlapping Partitioning
- PQ-based Storage in memory
- Data organized into fixed-length records [raw-vector, Rneightbors]
- Entry-Region Caching
- Beam Search

STRUCTURE OF GRAPH ANN INDEX

Use Tailored Storage Structures based on dataset size/dimensions to reduce I/O



Case 1: Small Dimensionality

- Raw Vectors + PQ Codes stored together
- Each node is stored with all its neighbors on a single page to reduce I/O during neighborhood scan

Case 2: High Dimensionality

- Raw Vectors are too large and cannot be stored together with PQ Codes
- PQ Codes stored in separately pages and cached in memory

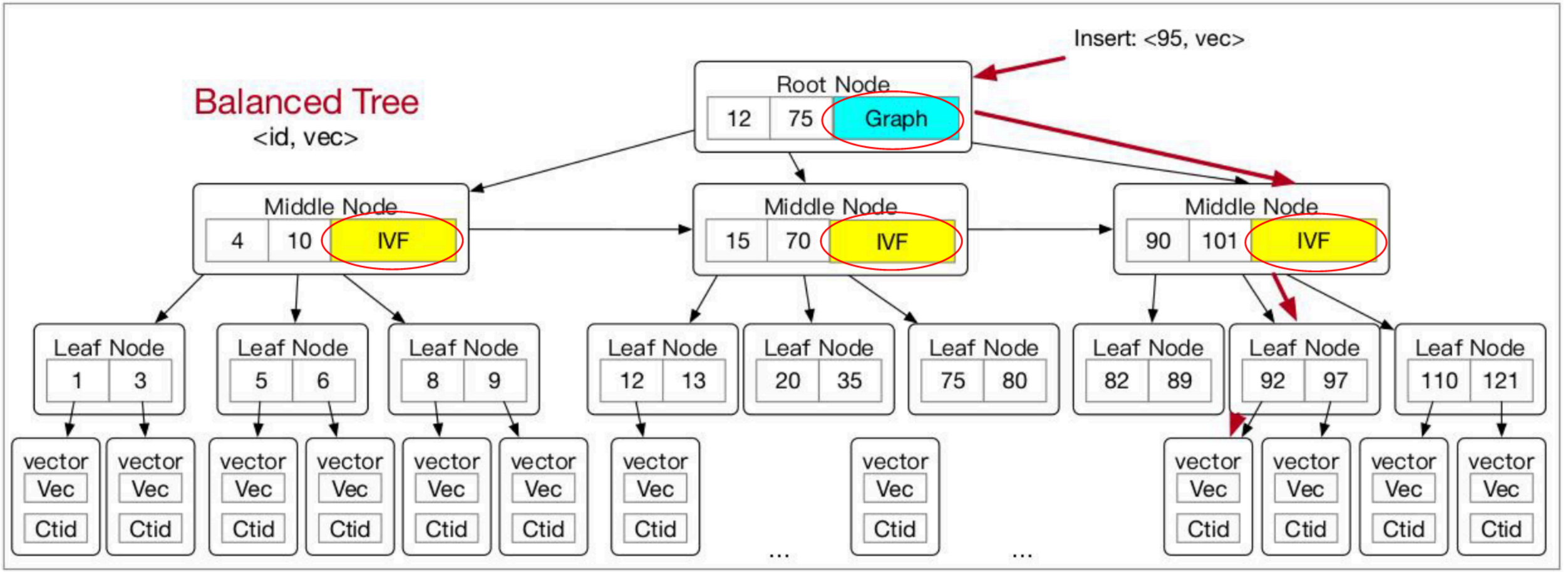
Case 3: High Size/Dimensionality

- Raw Vectors are too large and cannot be stored with PQ Codes nor Neighbors
- Store Nodes and Neighborhoods separately

Balanced Tree Hybrid Index

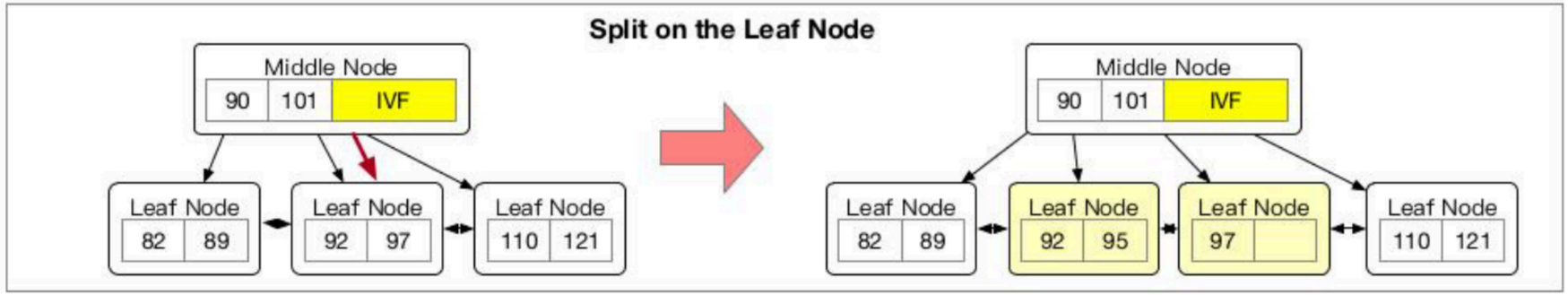
Use Balanced Tree with intermediate sub-indexes to support fast pre- / single-stage filtering

- ***High Selectivity** (few satisfying records): Sequential Scan over leaf nodes*
- ***Low Selectivity** (many satisfying records): Single-Stage Scan over sub-indexes*



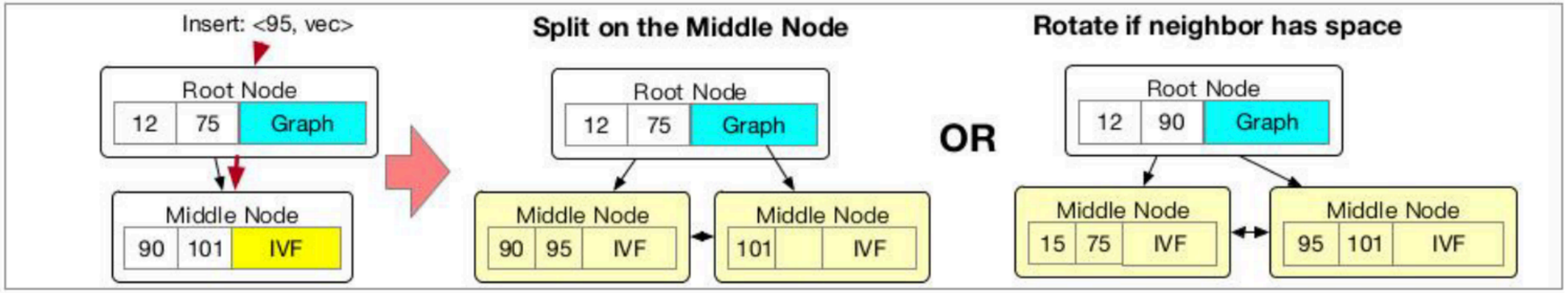
Incremental Index Updates

Balanced Tree with intermediate sub-indexes enables incremental updates for data freshness



(a) Splitting a leaf node following an insert

Leaf node split requires no index update

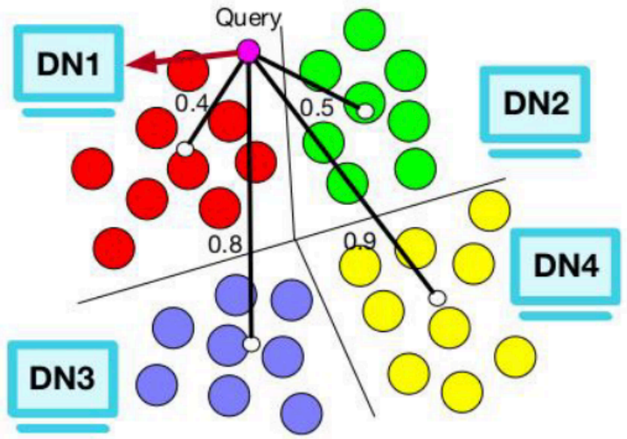


(b) Splitting an inner node following an insert

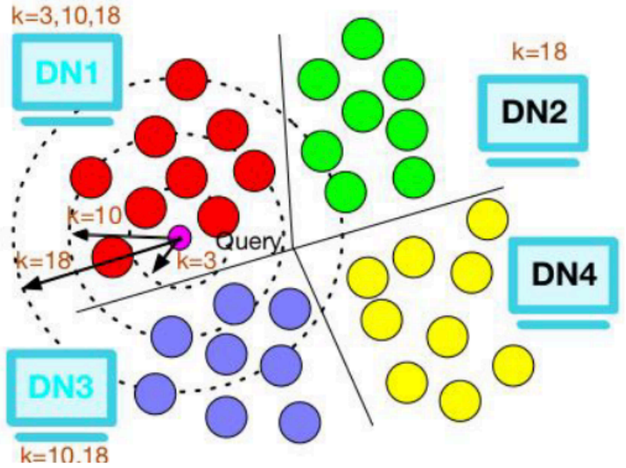
Inner node with IVF index is easy to update

Data Sharding & Distributed Search

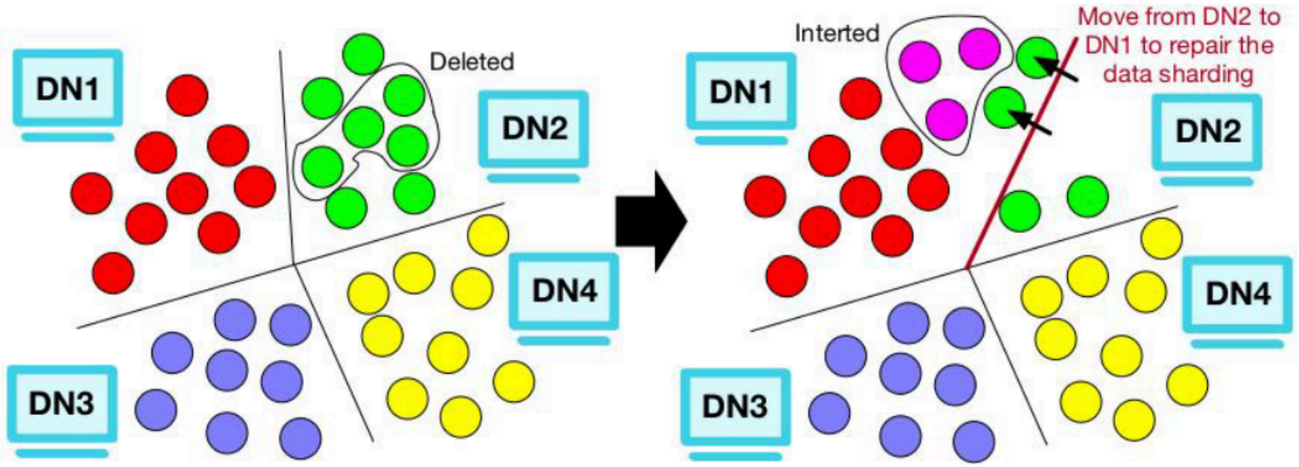
Balanced Tree with intermediate sub-indexes enables incremental updates for data freshness



(a) Insert into Nearest Shard



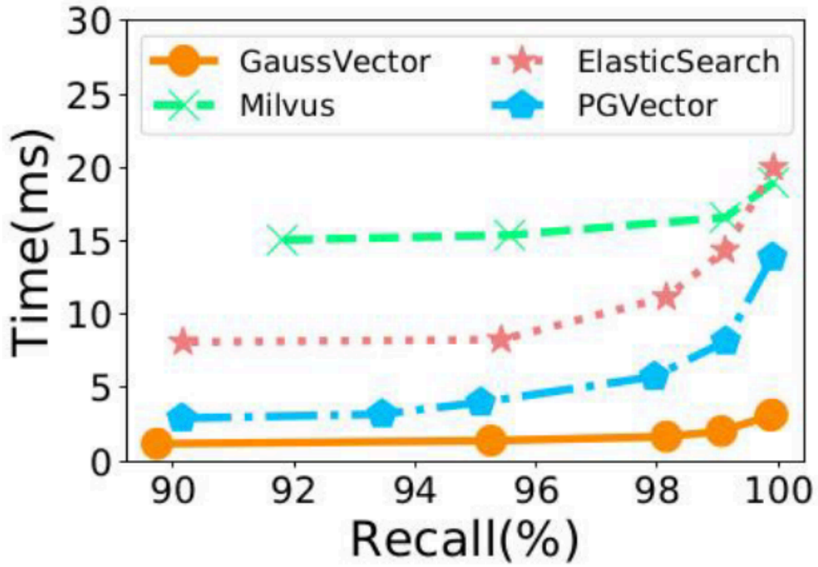
(b) Search Shards based on Estimated Cardinality



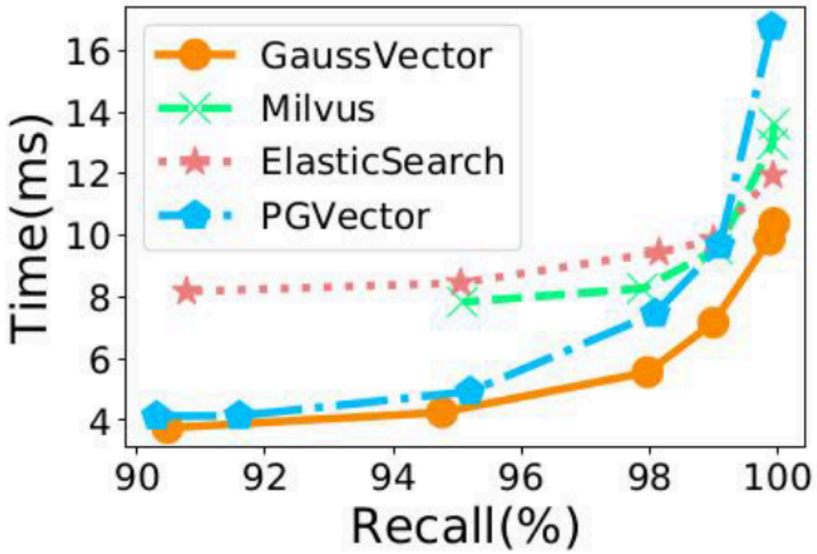
(c) Background Monitoring & Remigration

Tailored Storage Structures

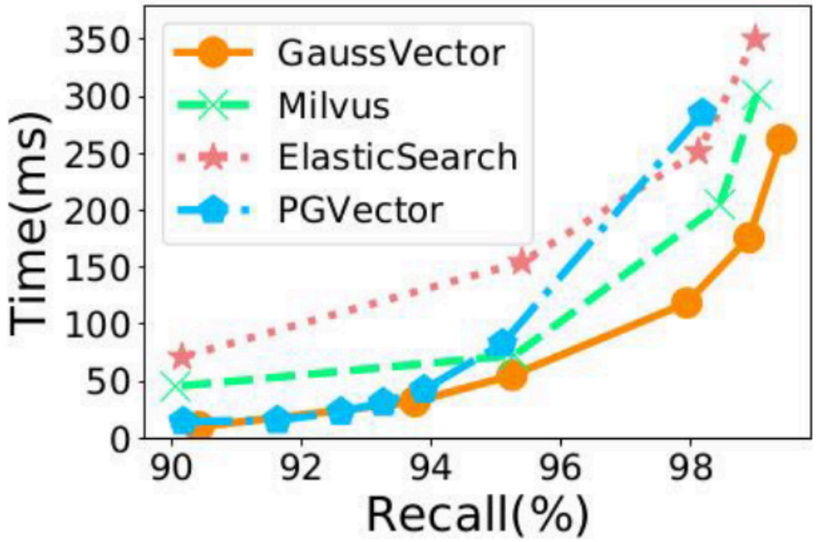
Use Tailored Storage Structures based on dataset size/dimensions to reduce I/O



(a) SIFT dataset.



(b) GIST dataset.



(c) HUAWEINet dataset.

Case 1: Small Dim

ElasticSearch & Milvus require loading multiple disk segments into memory

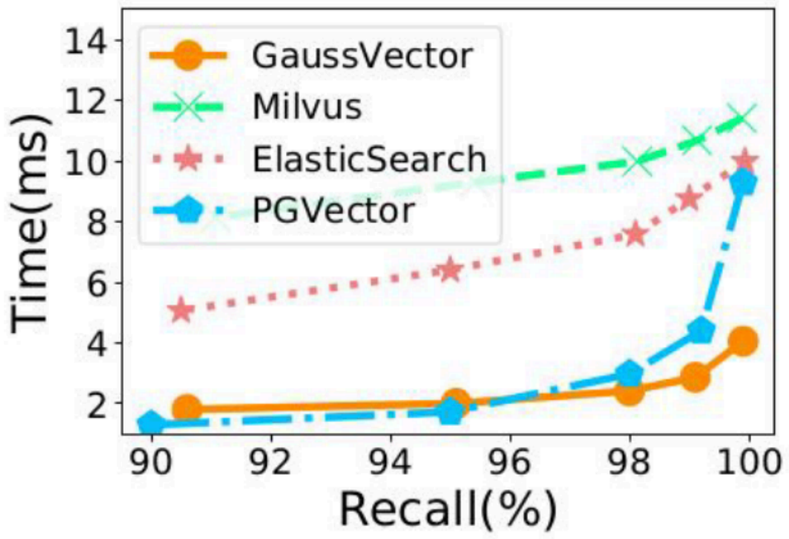
Case 2: High Dim

GaussDB-Vector adapts to high-dim scenarios to maintain high search performance

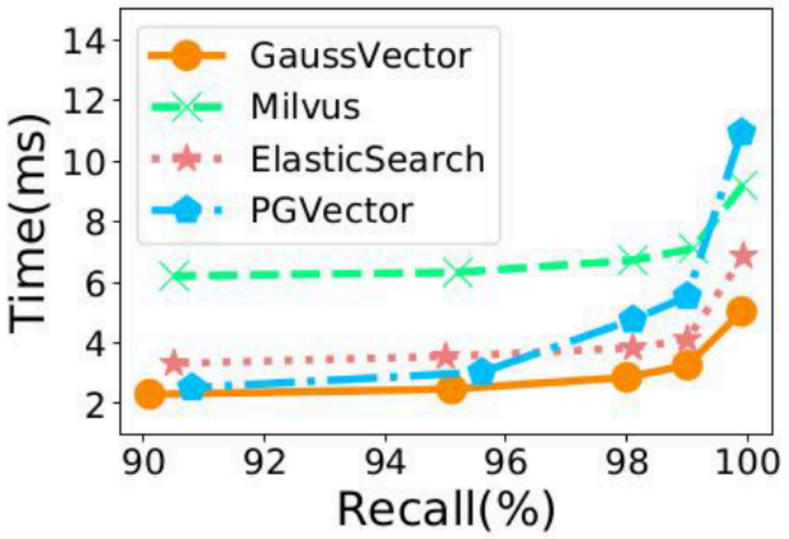
Balanced Tree Hybrid Index

Use Balanced Tree with intermediate sub-indexes to support fast pre- / single-stage filtering

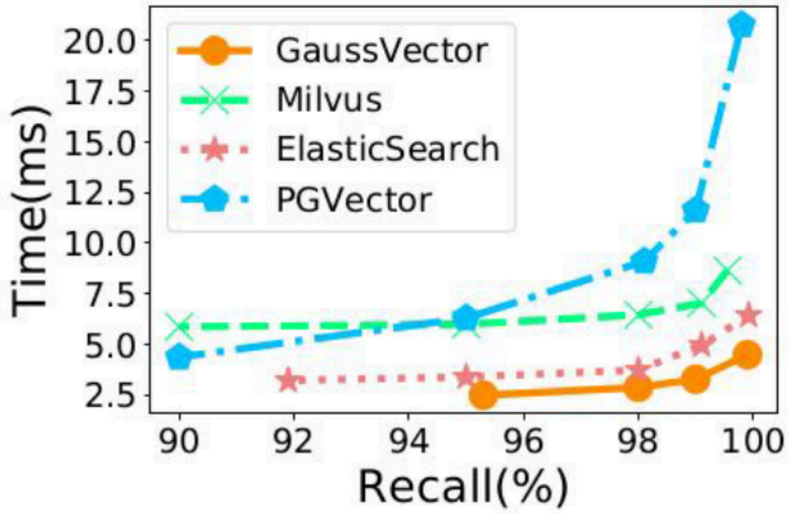
- ***High Selectivity** (few satisfying records): Sequential Scan over leaf nodes*
- ***Low Selectivity** (many satisfying records): Single-Stage Scan over sub-indexes*



(a) SIFT dataset, 90% scalar selectivity.



(b) SIFT dataset, 50% scalar selectivity.



(c) SIFT dataset, 10% scalar selectivity.

Many Satisfying Records

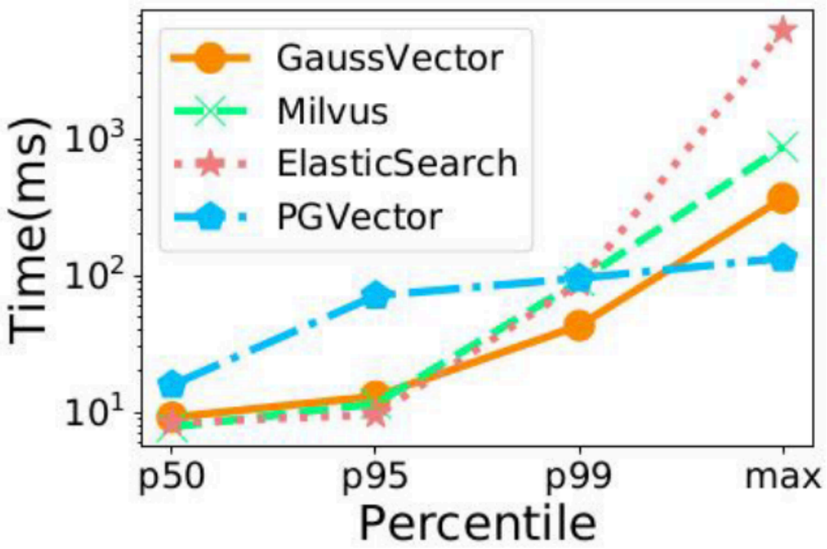
ElasticSearch & Milvus require merging across many segments

Few Satisfying Records

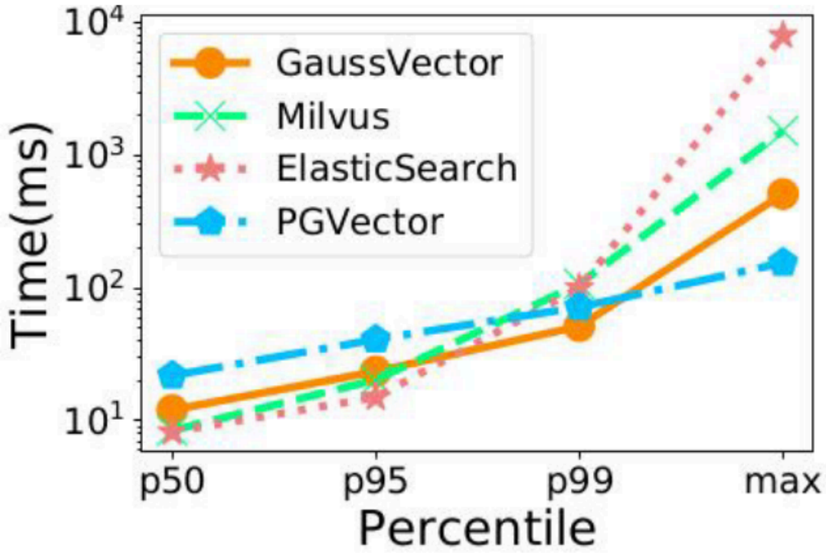
Pre-Filtering via index followed by Sequential Scan beats single-stage scan over segments

Incremental Index Updates

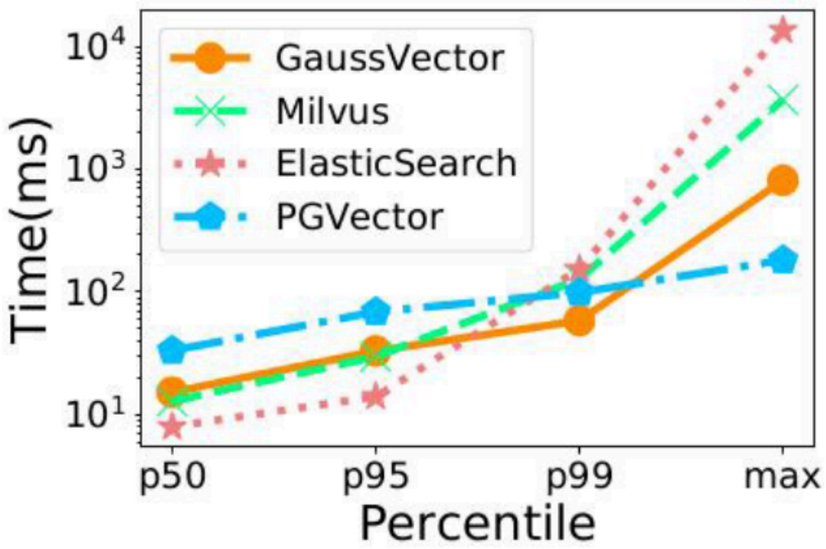
Balanced Tree with intermediate sub-indexes enables incremental updates for data freshness



(a) SIFT dataset.



(b) GIST dataset.

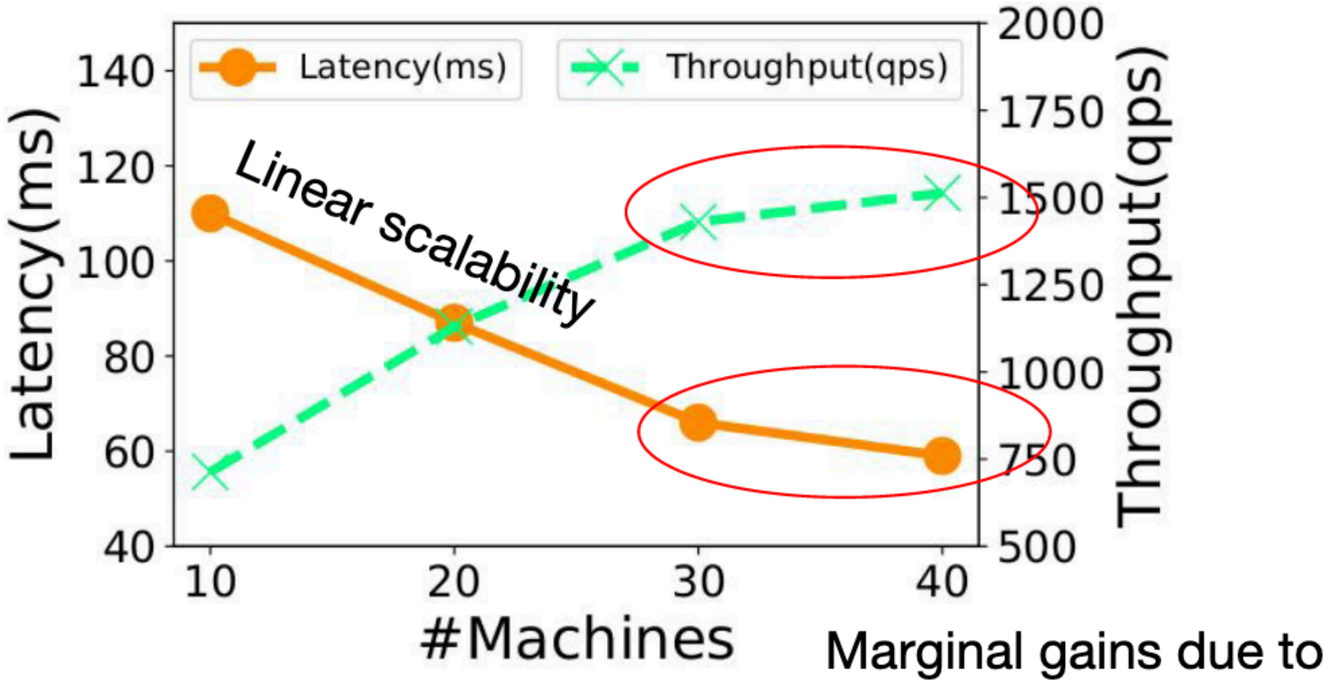


(c) HUAWEINet dataset.

- Large max latency due to inner node / root node splits but otherwise competitive with baselines
- Insert / search latency balanced by adjusting indexing granularity

Data Sharding & Distributed Search

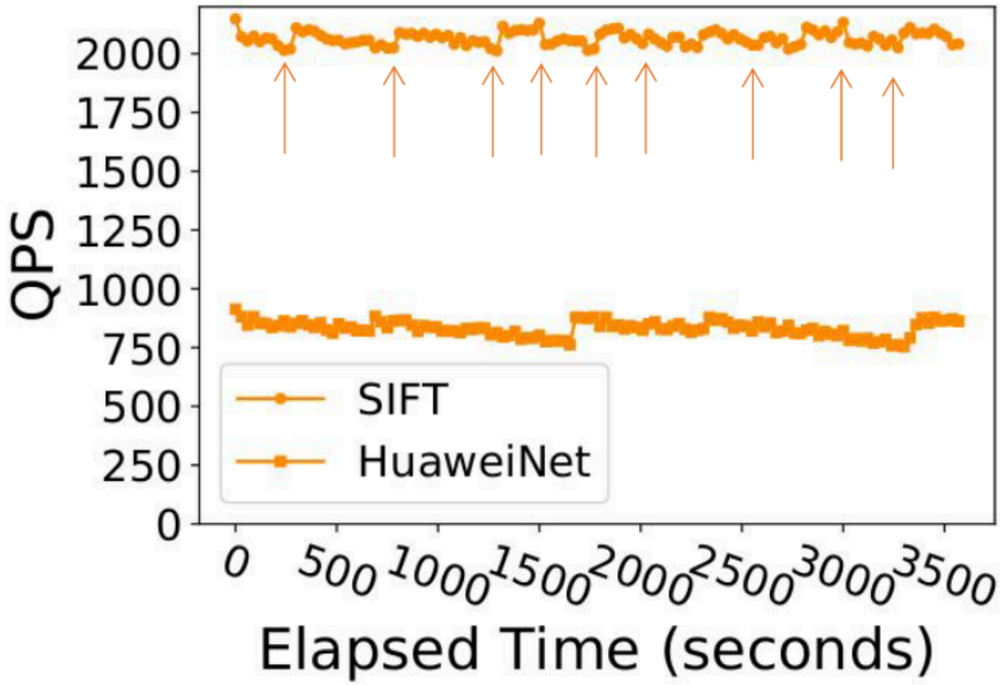
Balanced Tree with intermediate sub-indexes enables incremental updates for data freshness



Marginal gains due to higher merge costs

(a) Scalability

Rebalancing Spikes



(b) Update Throughput

Summary & Conclusion

GaussDB-Vector integrates vector search capabilities within existing data management system

	Query Processor	Storage & Indexing	Query Optimizer	Query Executor
Milvus 	• k-NN • Range NN • Predicated k-NN • Multi-vector search • Grouping Search • Full Text Search • Reranking	• Replicas • Data Partitions • Multiple index types • Blob storage (backup & persistence)	• Cost-based predicated query planning	• Scatter-gather • R/W Disagg. • CPU/Store Disagg. • Log-backed durable writes • Cache & SIMD acceleration • GPU acceleration • Tunable consistency
PostgreSQL (pgvector) 	• k-NN • Range NN • Predicated k-NN • Full Text Search • ***Relational queries***	• Page-Based Storage • Shards & Replicas • HNSW, IVF • Quantization	• Cost-based predicated query planning	• MVCC • Scatter-gather • Log-backed durable writes • etc.
GaussDB-Vector 	• k-NN • Range NN • Predicated k-NN • ***Relational queries***	• **Page-Based Storage** • **Shards & Replicas** • HNSW, IVF, **Hybrid Index** • Quantization	• Cost-based predicated query planning	• GaussDB Transaction Manager • **Scatter-gather** • Log-backed durable writes • Cache & SIMD acceleration • NPU/GPU acceleration