



CS 839 Next Generation Data Systems

Lecture 5: Vector DB Systems

Tianyu Li
litianyu@cs.wisc.edu

Reading Check-in

- You have ~10 min (until 11:15) to answer 2 questions
- No electronics

Administrivia

- If you are asked to be matched, watch out for an email in the next few days
- Project Proposals due 10/8

Recap: ANN & Vector Indexes

- Over the last 10 years, text embeddings have made semantic retrieval practical at large scale
- Next Question: how do we “operationalize” vector retrieval so we can build real applications on it?

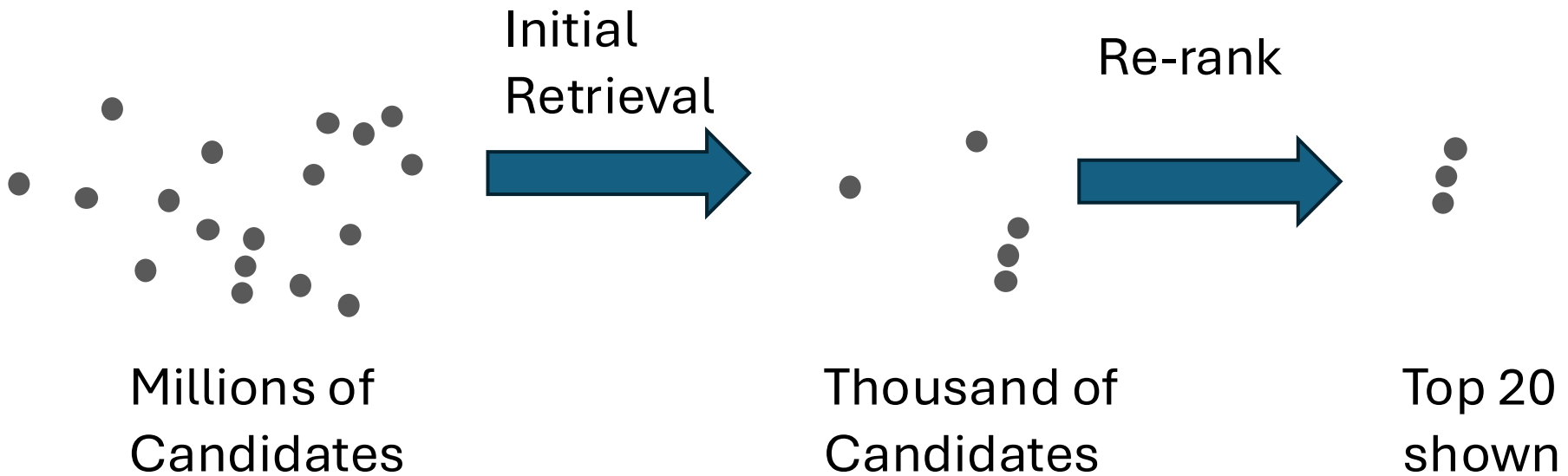
Applications with Vector DB

- Search Workloads – given query, retrieve relevant objects
 - Ranging from simple text search to multi-modal and scientific
 - Results are ranked in terms of relevance and displayed
- Recommendation – featurize users, items, music, news, video, etc. and recommend relevant content or content of interest
 - Apply personalized ranking and business rules after
- RAG – find information relevant for a generation task and populate context

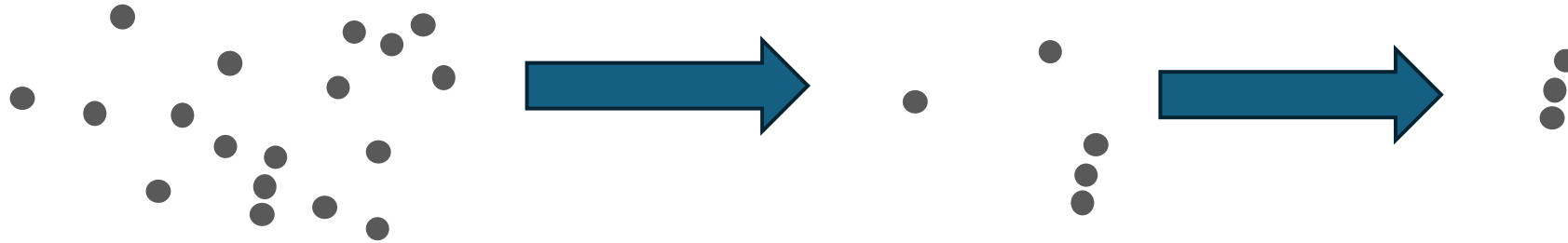
Trade-offs

- Most downstream applications for vector DB are latency & cost sensitive (e.g., interactive search should return with 100s of ms, many concurrent requests from large user base)
- Intuitively, there is a trade-off between search quality and raw performance
- Sophisticated systems navigate this by splitting retrieval into stages

Reranking



Reranking



- Production systems can have more than 2 stages. Generally:
 - Early stages are cheap per candidate and recall focused
 - Late stages can apply more expensive techniques and focus on quality
- Vector DBs are typically early in the pipeline whereas reranking can use application rules, specialized models or even LLMs

Performance Engineering

- Goal: maximize quality within latency & cost budget
- Recall loss anywhere in the pipeline is permanent
 - Measured against application metrics, recall loss can happen in the embedding phase, retrieval, or downstream processing
- Low precision may increase performance pressure downstream
 - Often must pass more candidates downstream for expensive processing to retain the same number of relevant candidates

Back of Envelope

- Search budget is often 200 – 500 ms
 - Reranking stages can take up 100s of ms
 - Typical budget left for vector DB < 100ms @1000s of QPS
- Recall target: application dependent
 - However, typically vector DBs need to narrow the candidate set down to 100s – 1000s of objects with high recall
- This places strict performance and efficiency requirements on the underlying vector DB implementation!

It Gets Worse

- New objects arrive constantly, old objects may change or disappear
 - Think YouTube / Social Media Posts
- Application may want richer query API
 - E.g., filtered search or other combinations with relational operators
- Data often will not fit onto one machine and must be distributed
 - Now that we are distributed, we should also think about fault tolerance
- Must be elastic enough to serve fluctuating load and balance between background maintenance and foreground serving

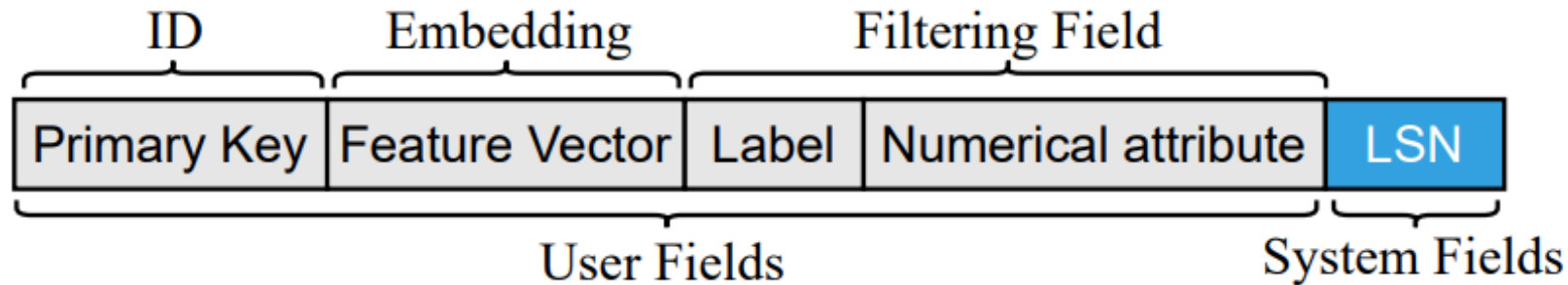
Manu: Cloud-Native Vector DB

- Research paper of the open-source system Milvus 2.0
- Three core workloads:
 - Ingestion: durably record changes promptly
 - Indexing: update & maintain vector indexes and other lookup structures
 - Query Serving: meet tight latency deadlines under high throughput
- Key Idea: disaggregation similar to prior cloud-native systems

Manu: Key Claims

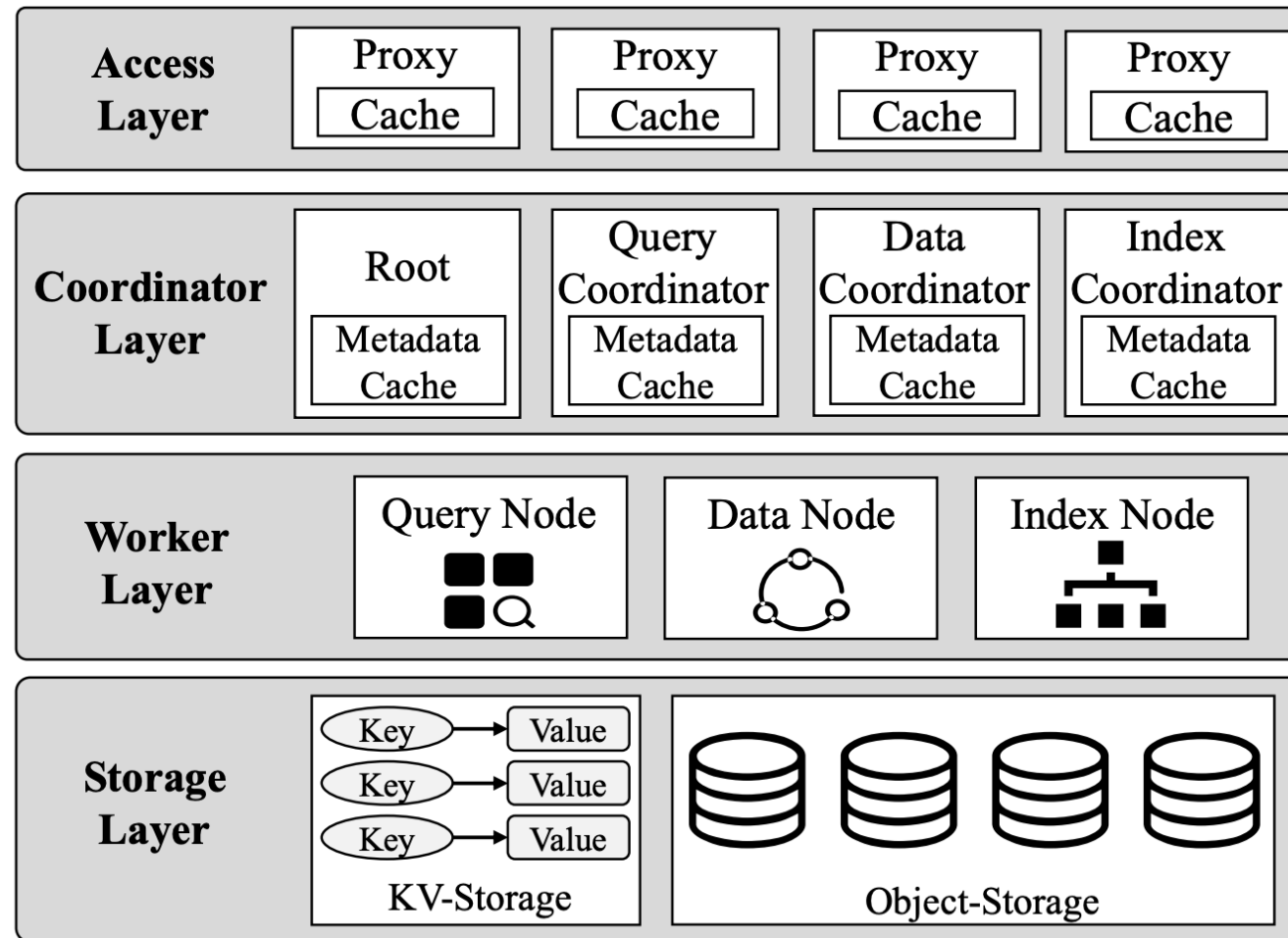
- Support for complex transactions is not necessary
- Tunable consistency is important
- Fine-grained elasticity is key

Manu: Data Model

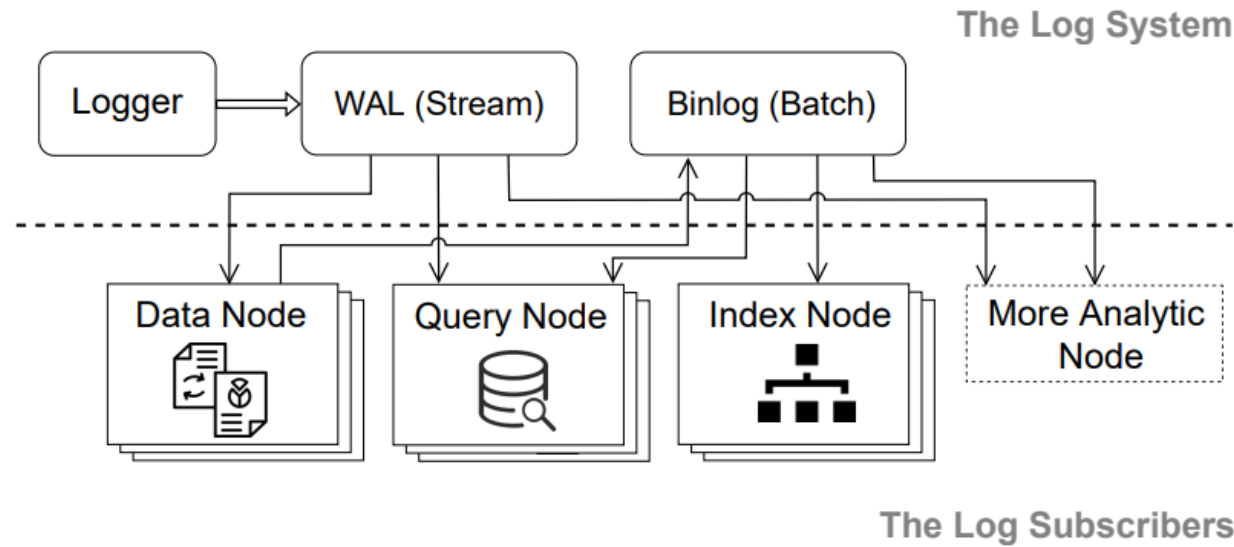


- Each row in the database is ID + vector + scalar attributes for filtering
- Collection: analogous to a table, but no joins

Manu: Architecture



Manu: Writes



- Writes are logically logged for durability. The log is then consumed like an event stream by various other parts of the system
- Ingestion is sharded by primary key, and data nodes are responsible for buffering data in-memory and writing it out in batches (i.e., segment) in columnar format (i.e. binlog) onto S3

Manu: Indexing

- LSM Tree for ID -> Segment lookup
- Each segment typically gets its own search index
 - Parallel kNN over each segment
 - Results are merged in a reduction phase
- Different types of index possible (IVF, HNSW, PQ, B-Tree, etc.)
 - Indexes are built and written to storage
 - At runtime, query nodes serve stored index and cache segments
 - For freshness, segments not yet indexed are directly scanned

Manu: Consistency

- Problem: with event-driven architecture, nodes consume updates asynchronously and indexes may be out-of-date
- Manu provides tunable consistency
 - Users specify some staleness tolerance (e.g., 10s)
 - Log stream has watermarks (e.g., it's 12:00:10) that inform consumers where they are in the stream
 - Query handling wait until the consumer has caught up to the required point
- Data is versioned (i.e., MVCC)

Manu: Filtered Search

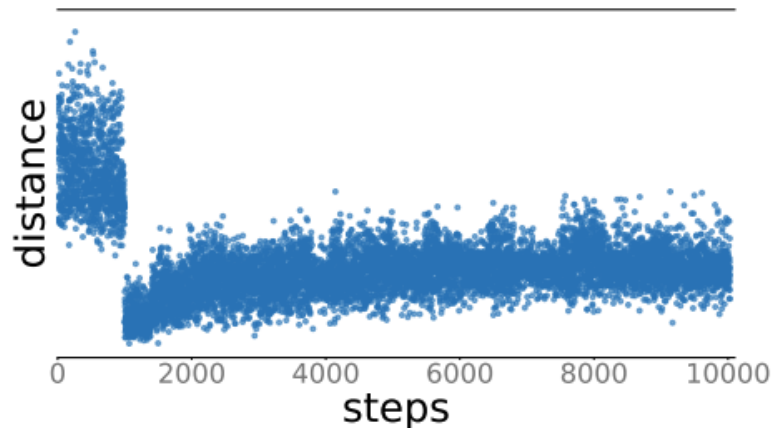
- Example: `SELECT id FROM products WHERE price < 100 ORDER BY sim(embedding, "Sick Sneakers") DESC LIMIT 100;`
- Several Strategies:
 - Filter + Simple Scan (when filter is very selective)
 - Filter + ANN with bitmap
 - ANN and then Filter
- Can use cost-based optimization to select strategy

What can go wrong?

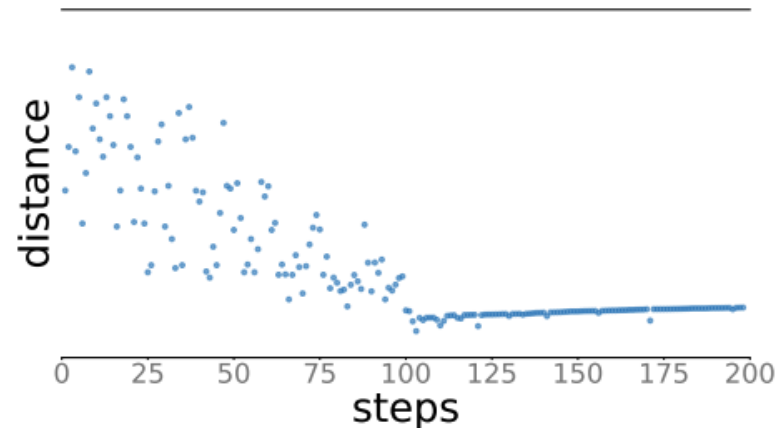
- Filter + Exact Scan
 - What if too many items pass the filter?
- Filter + ANN with bitmap
 - What if filter is expensive to evaluate upfront?
 - What if qualifying points are placed adversarially?
- ANN + Filter
 - What if we don't retrieve enough candidates that survive the filter?
 - What if we retrieve too many and waste work?

VBase: Rethinking the Interface

- Insight: mismatch between top-K interface and relational interface
 - Specifically, no “monotonicity”, as ANN search can encounter a closer vector after a farther vector



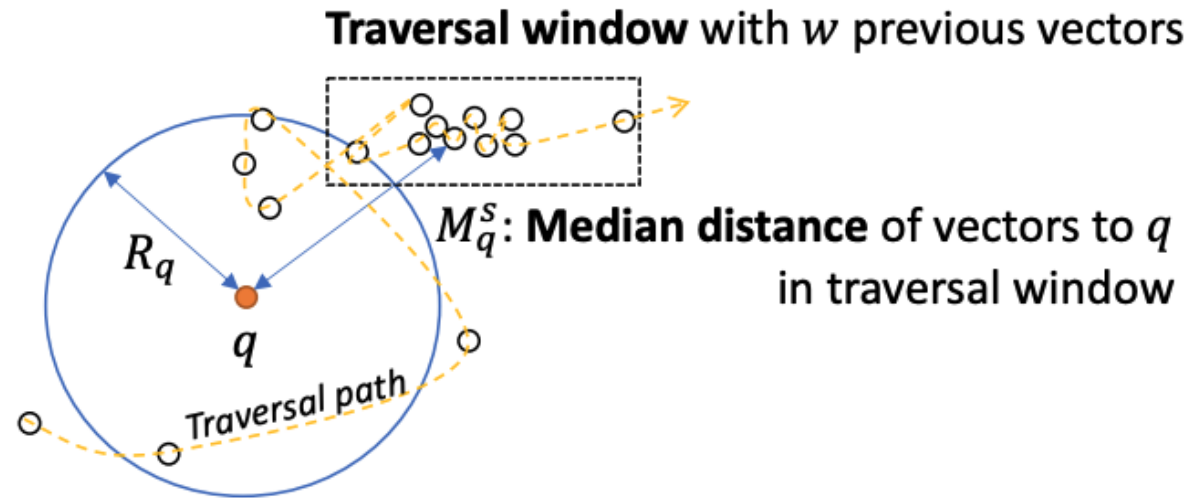
(a) FAISS IVFFlat



(b) HNSW

Relaxed Monotonicity

- ANN search has two phases – approach the query's neighborhood, and then start exploring farther away



Neighbor sphere of a target vector q with a radius R_q , which contains E nearest vectors to q .

VBase Engine

- Idea: ANN exposes the same Open()/Next()/Close() interface as DB iterators
 - Integrates naturally into the relational engine
- Use relaxed monotonicity as a termination check
 - If average distance starts to become larger, we are relatively certain we have started moving away from the neighborhood

VBase: Discussion

- VBase supports more complex features that feel more “relational”
 - Composite ranking using multiple columns
 - Distance-threshold search
 - Similarity joins
- In contrast to Manu, VBase has chosen to integrate vectors in a relational engine rather than build a new system
 - Paper shows 2 - 3x speed-up for simple retrieval / + filter, and orders of magnitude better performance for complex queries
 - Caveat: small scale compared to what Manu is optimized for

VBase: Problems

- VBase is still fundamentally a post-filter technique
 - What if many candidates close to the query vector fail the predicate?
- Can we do better in the index design if we know we will use it along with filters?
 - Many work in this space and still an active area of research!
 - Today we will quickly cover Filtered-DiskANN

Filtered DiskANN: Background

- DiskANN is a system from Microsoft that allows for efficient graph indexing using SSDs
 - High level idea: store compressed vectors in memory, store full-precision and adjacency lists on SSD, cache hot vertices
 - Uses Vamana – a single-level graph index design
 - Efficiency depends on reducing dependent I/O rounds
- DiskANN is open-source

Filtered DiskANN: Idea

- Assume we know labels to filter upfront for each vector
- When building graph, consider connectivity for subgraph where only vertices with a label is kept
 - Will add redundant edges when constructing the graph
- When searching, traverse the labeled subgraph instead of the full graph

Filtered Vamana: Idea

- Assume we know labels to filter upfront for each vector
- When building graph, consider connectivity for subgraph where only vertices with a label is kept
 - Will add redundant edges when constructing the graph
- When searching, traverse the labeled subgraph instead of the full graph

Stitched Vamana: Idea

- Again, assume we know labels to filter upfront for each vector
- Build disjoint graphs for each label
- Combine the graphs and prune to control size

Takeaways

- Vector DBs are a fascinating field that combines systems work with innovations in algorithms and theory
- Still an active area of research! Specifically, filtered search remains challenging
 - Frontier is expanding from simply choosing the right algorithm/architecture to actively modifying index design to serve filters better

Discussion

- Are embeddings just another column with an index, or do they require a different database architecture?
 - Does your thinking change from last lecture?
- How reasonable is it to assume that filter conditions are known beforehand? How to solve combinatorial blow-up of filter conditions for solutions like Filtered DiskANN?
- Modern VectorDBs are highly tunable with intricate interactions with downstream interactions. How should one approach tuning these engines? Where would performance gain actually matter?

Next Week

- We will discuss various techniques aimed at improving retrieval when vector embeddings are the bottleneck