Metadata-Aware Vector Approximate Nearest Neighbor (MAVANN)

Nithin Mani (Cosdata)

July 12, 2025

Contents

1 Introduction

This document details the design and implementation of metadata-based filtering capabilities in the Cosdata vector database, specifically for HNSW dense vectors. The core innovation of MAVANN (Metadata-Aware Vector Approximate Nearest Neighbor) lies in its hierarchical routing structure that efficiently directs queries to metadata-specific subspaces. By implementing a two-phase approach with specialized pseudo-metadata nodes in upper index levels and distinct metadata node "cones" below, MAVANN enables efficient filtering while preserving the accuracy of similarity search. This architecture allows for staged distance measurement that separates metadata evaluation from base vector comparison, effectively narrowing the search space to the relevant metadata silos without sacrificing search performance.

2 Comparison with Traditional Filtering Approaches

2.1 Pre-Filtering

Pre-filtering applies constraints before the similarity search begins, reducing the candidate pool. While effective, it suffers from:

- **Reduced Search Space:** By filtering before the vector search, it artificially limits the potential matches and may miss semantically relevant results that don't exactly match the metadata criteria.

- **Performance Issues:** Maintaining separate metadata indexes alongside vector indexes can be slow, especially with high-cardinality metadata fields. Query planning overhead also adds complexity.

2.2 Post-Filtering

Post-filtering applies metadata constraints after retrieving the nearest neighbors. While preserving ANN speed, it introduces:

- **Inefficient Resource Usage:** Full vector similarity search is performed before filtering, leading to wasted computation on discarded results.
- **Result Quality Issues:** If too many top matches are filtered out, the final result set may be insufficient, requiring retrieval of more candidates.
- **Architectural Complexity:** Requires separate filtering logic, complicating caching, optimization, and ranking.

2.3 MAVANN's Novel Approach

MAVANN eliminates the inefficiencies of pre- and post-filtering by integrating metadata into the vector representation itself. By encoding metadata within the vector space, filtering becomes an inherent part of the ANN search process, avoiding unnecessary computation and improving precision.

3 Vector Structure in MAVANN

Instead of treating metadata as separate filters applied before or after the similarity search, MAVANN expands the vector space by appending metadata dimensions to the core feature vector. This means each vector has two segments:

- **Values Segment:** The primary dimensions capturing the core vector representation (e.g., an embedding from a neural network).
- **Metadata Segment:** Additional dimensions encoding metadata, such as categorical tags, numerical constraints, or temporal information.

Formally, for an original vector $\boldsymbol{v}$ with $\boldsymbol{d}$ dimensions, MAVANN extends it to $\boldsymbol{v}$ with $\boldsymbol{d} + \boldsymbol{m}$ dimensions, where $\boldsymbol{m}$ represents metadata dimensions.

4 Phased Distance Metric Computation

MAVANN enables a staged similarity computation, optimizing performance while maintaining precision. This staged approach is particularly beneficial in hierarchical search structures like HNSW and is relevant to both indexing and query processing.

4.1 Step 1: Metadata Segment Evaluation

- Compute the distance metric (e.g., cosine similarity, Euclidean distance) only on the **metadata segment**.
- If the metadata segment satisfies the required constraints (e.g., similarity threshold is met), proceed to the next step.

4.2 Step 2: Full Vector Similarity Computation

- Compute the similarity using the full vector (**values** + **metadata**).
- This ensures that filtering is integrated into the ANN search without requiring explicit pre- or post-processing.

5 Index Structure

5.1 Base Vector Representation

All vectors in the system must maintain the same dimensionality for HNSW graph construction and search. To achieve this, MAVANN extends the values segment with additional metadata segment dimensions, ensuring consistent indexing across all vectors.

5.2 Metadata Segment Encoding

The metadata segment follows a carefully designed encoding scheme that balances accuracy, efficiency, and storage requirements. Each metadata field requires a specific number of additional dimensions based on its cardinality (number of possible values).

5.2.1 Dimension Allocation

For each metadata field, the number of required dimensions is calculated by rounding up the field's cardinality to the nearest power of 2. This allocation ensures efficient binary encoding of values. Examples include:

- A field representing months (12 possible values) requires 4 dimensions (equivalent to `2^4 = 16` binary bits).
- A field for days of the week (7 values) requires 3 dimensions (equivalent to `2^3` binary bits).
- A binary field (2 values) requires 2 dimensions (because 0 implicitly represents null or field not specified.).
- A field with 100 possible values requires 7 dimensions (equivalent to `2^7` binary bits).

5.3 Metadata Schema

At the time of creating a collection, the user specifies a **metadata schema**, which includes:

- Metadata fields that require filtering.
- Unique values for each metadata field, allowing implicit lexicographical sorting for numeric mapping.
- *AND* / *OR* query support specifications to optimize the creation of appropriate metadata nodes.

The metadata schema is stored in LMDB. If an insert request contains metadata values outside the schema, an error is returned.

5.3.1 Updating Metadata Schema

If new metadata values need to be added, there may be available dimensions due to power-of-2 rounding. If no room exists, a reindexing process is required. Adding a new metadata field also necessitates reindexing due to dimensionality changes.

5.4 Hierarchical Metadata Routing Structure

MAVANN implements a novel hierarchical routing structure that efficiently directs queries to appropriate metadata-filtered subspaces. This structure consists of two primary components:

5.4.1 Pseudo-metadata Nodes

These are special static nodes that represent all possible metadata field combinations and serve as routing guides in the upper levels of the index:

- Each combination of metadata field values is represented by a pseudo-metadata node
- For fields with cardinalities $C_1, C_2, ..., C_n$, we create nodes for individual fields ($\sum_{C_i}$) and necessary combinations (product of relevant cardinalities)
- Pseudo-metadata nodes have metadata dimensions set according to their represented values, while base vector dimensions can be zero
- A deterministic number of top levels in the HNSW structure are reserved for these nodes

5.4.2 Pseudo-root Node

Alongside the main HNSW root node, a special pseudo-root node is created at the highest level with all metadata dimensions set to 1. This node serves as the primary entry point for metadata-filtered searches, directing queries to the appropriate pseudo-metadata nodes below.

5.5 Vector Extension and Indexing Process

The indexing process follows a two-phase approach:

5.5.1 Phase 1: Pseudo-metadata Structure Creation

1. Calculate the number of levels needed for the pseudo-metadata structure:
    - $N = \log_{10}(\text{total-combinations})$

- Example: For fields age (cardinality 25) and color (cardinality 3), with dimensions 2^5 and 2^2, we have $2^7 = 128$ possible combinations, requiring $N = 3$ levels

2. Reserve the top N levels of the HNSW structure:
    - If the regular HNSW has M levels (e.g., $M = 9$), levels (M-N) through (M-1) are reserved
    - Insert all pseudo-metadata nodes at level (M-N)
    - Probabilistically promote 1/10th of nodes to each higher level
    - Place the pseudo-root node at the highest level (M-1)
3. Ensure connectivity:
    - All pseudo-metadata nodes maintain children connections down to level 0
    - This creates distinct routing paths for different metadata combinations

5.5.2 Phase 2: Metadata Node Insertion

For each vector to be indexed:

1. Create necessary metadata nodes:
    - Base node for pure similarity searches
    - Individual field nodes for each metadata field (supporting *OR* queries)
    - Combined field nodes for field combinations (supporting *AND* queries)
2. Populate metadata dimensions:
    - For each metadata node, set appropriate binary values (0 or 1) in the metadata dimensions
    - Each dimension corresponds to a specific bit position in the binary representation of the metadata value
    - For a field value with binary representation $[b_1, b_2, ..., b_n]$, set the corresponding dimensions to these binary values
    - This creates a unique binary signature for each metadata combination

3. Insert these metadata nodes:
    - Metadata nodes are inserted only at levels below the pseudo-metadata structure (level (*M-N-1*) and below)
    - Follow standard HNSW probabilistic level assignment (1/10th promotion rate)
    - Ensure each node connects to its corresponding pseudo-metadata parent

This two-phase approach creates distinct "cones" or "silos" of metadata nodes beneath their corresponding pseudo-metadata nodes. There is no interconnection between nodes from different metadata criteria, ensuring clean separation of search spaces.

6 Query Processing

6.1 Pure Similarity Search

When no metadata filtering is needed, queries follow the regular HNSW search path, bypassing the metadata routing structure entirely.

6.2 Metadata Filtering Queries

For metadata-filtered searches, the query process follows these steps:

1. Construct query vector with appropriate +1/-1 values in metadata dimensions:
    - +1 for matching the desired value's position
    - -1 for other positions to prevent false matches
2. Entry and routing:
    - Search begins at the pseudo-root node
    - Traverses through relevant pseudo-metadata nodes in upper levels
    - Eventually reaches the appropriate "cone" of metadata nodes
3. Filtering logic:
    - Single field filters use the corresponding field's metadata nodes

- *OR* conditions may require multiple search paths, with results merged afterward
- *AND* conditions use the pre-computed combination metadata nodes

This approach ensures that queries only explore relevant portions of the index, significantly improving search efficiency for metadata-filtered queries.

6.3 Query Vector Encoding Details

The effectiveness of metadata filtering relies on a carefully designed query vector encoding scheme that uses +1/-1 values to ensure precise matching. This encoding scheme is fundamental to supporting both equality and inequality filters.

6.3.1 Equality Filter Encoding

When searching for vectors with a specific metadata value, the system employs a binary encoding strategy across the dimensions allocated for that field. For example, when filtering for value 1 in a field, the query vector would have:

- A positive value (+1) in the position corresponding to bit 0
- A negative value (-1) in the position corresponding to bit 1
- Similar negative values in all other bit positions for that field

This encoding ensures accurate discrimination between different values. For instance, when searching for value 1, a vector with value 3 (binary 11) will not match because the negative query value at position 1 will create a repelling force in the dot product calculation, effectively eliminating false matches.

6.3.2 Inequality Filter Encoding

For inequality filters $field \neq x$, the system inverts the encoding used in equality filters. Taking the same example of filtering for "not equal to 1":

- The positive and negative values from the equality encoding are inverted
- Position 0 becomes -1

- Position 1 becomes $+1$
- Other positions retain appropriate values to maintain filtering accuracy

During dot product calculations, these inverted values create attractive forces for all values except the one being excluded, effectively implementing the inequality constraint.

The $+1/-1$ encoding in query vectors creates substantial differences in dot product results between matching and non-matching vectors. This ensures reliable filtering even in the presence of approximate nearest neighbor search.

7 Performance Considerations

MAVANN achieves efficient ANN filtering without degrading search performance:

- **Hierarchical routing**: The pseudo-metadata structure in upper levels provides efficient initial routing
- **Isolated metadata cones**: Prevents interference between different metadata criteria
- **Progressive refinement**: As search descends levels, more metadata nodes provide increasingly accurate results
- **Balanced overhead**: The number of pseudo-metadata levels (N) provides a tunable parameter between routing efficiency and storage overhead
- **Optimized structure**: No connections between metadata nodes of different criteria eliminates irrelevant path exploration

The distinct "cones" formed by metadata nodes under each pseudo-metadata node create natural partitioning of the search space without requiring separate indexes.

8 Trade-offs and Considerations

While MAVANN provides significant advantages for metadata-filtered vector search, implementers should be aware of certain trade-offs:

- **Index Size**: The creation of multiple metadata nodes per vector increases storage requirements compared to traditional HNSW. This overhead scales with the number of metadata fields and their cardinality.
- **Indexing Complexity**: The two-phase approach requires careful implementation of node creation and connection logic, particularly for maintaining the pseudo-metadata structure.
- **Schema Evolution**: Changes to the metadata schema (adding fields or expanding cardinality) may require partial or complete reindexing, which can be resource-intensive for large collections.
- **Dimension Balance**: The effectiveness of MAVANN depends on appropriate allocation of dimensions between the base vector and metadata segments, which requires thoughtful configuration.

These considerations should be weighed against the performance benefits for applications where metadata filtering is a frequent operation. For use cases with minimal metadata filtering requirements, simpler approaches may be sufficient.

9 Conclusion

MAVANN integrates metadata filtering seamlessly into the ANN search process by expanding the vector space and encoding metadata within the vector representation. By leveraging a hierarchical metadata routing structure and staged similarity computation, MAVANN ensures efficient and accurate retrieval while preserving the high performance of HNSW-based vector search. Unlike traditional pre-filtering and post-filtering methods, MAVANN optimally balances precision, recall, and computational efficiency, making it a superior approach for metadata-aware ANN search.