Single Store Report

Vishravars Ramasubramanian

PROBLEM INVESTIGATED

The paper investigates how to add vector search used for semantic search — into a general-purpose SQL database, without losing the performance and flexibility. The paper argues that most of the existing vector databases are either Specialized systems like Milvus or Pinecone - that are fast but don't support full SQL or transactions, or General-purpose systems like PostgreSQL with extensions (e.g., pgvector) that support SQL but are not performant.

A notable challenge is that vector search is different from traditional database queries. It involves finding the most similar vectors in high-dimensional space, which is computationally expensive and it also needs to integrate with filters, joins, or fulltext search.

The paper asks: *Can we design a system that supports fast and scalable vector search inside a full SQL database, with support for filters, joins, transactions, and analytics?*

KEY IDEA OF SINGLESTORE

SingleStore-V supports both OLTP and OLAP workloads. At the core, SingleStore organizes data in shards, which is distributed as leaf nodes in a cluster. Parts of the leaf node are:

Rowstore (in-memory): A lock-free skip list data structure which is used to store recently inserted or modified rows. The primary design theme is for fast writes and real-time data visibility.

Columnstore (disk): A Log-Structured Merge(LSM) which comprises of immutable segments, where each segment holds the compressed columnar data that is optimized for scan performance.

This hybrid storage engine enables SingleStore to balance between low-latency updates and high throughput in one single store without data duplication.

SingleStore-V integrates the vector search into this architecture with the support of VECTOR data type. It builds vector indexes on top of the immutable columnstore segments. These indexes are further classified into two types:

Per-segment vector indexes: This index is built once the segment is created, and it avoids the complexity of updation.

Cross-segment vector indexes: These are merged in the background to improve the performance so that the rowstore memory dependency is reduced. Larger write transactions are designed to bypass the in-memory rowstore to directly write to columnstore segments.

VECTOR SUPPORT DESIGN

The SingleStore-V, integrates vector search into the SingleStore database engine, so that the users can run vector queries alongside normal SQL operations like filters, joins, and fulltext search. The following are few key ideas to make this work efficiently:

New VECTOR Data Type: SingleStore-V adds a new column type called VECTOR, which stores high-dimensional vectors. The vectors are typically embeddings from text or images.

Vector Indexing on Immutable Segments: Instead of building a large and mutable index, SingleStore-V creates per-segment vector indexes on each columnstore segment (immutable). This makes the indexing fast and stable and also allows different indexes to be used for one vector..

Cross-Segment Index Merger: To improve the search across many segments, the system builds a *merged vector indexes* in the background which spans across the segments. During vector search, the larger cross-segment vector index is used to improve search efficiency.

Top() Operator: A physical operator called Top() is used by the SQL execution engine. It makes the system to activate vector search into the scan phase, making it work with other SQL features like filters (or) ORDER BY ... LIMIT.

Instead of scanning all rows and then sorting them by distance, SingleStore-V turns this into an internal operation like:

```
Top(embedding <-> @query_vector, limit=10)
```

Pluggable and Auto-Tuned Indexes: The design supports multiple index algorithms - treating them as a black box. This choice is said to automatically choose the best index based on the data and user goals. The user goal (like high_recall) is set when building the index. While the index type is fixed, query-time options like nprobe or efSearch, dynamically shifts the index choice for speed vs accuracy.

VECTOR SEARCH OPTIMISATION

SingleStore stores both structured data and vector embeddings in the same table. It builds per-segment or cross-segment vector indexes on top of the immutable segments that contain all columns of the table. Since it knows which rows match a filter before or during index access, it uses segment metadata to skip irrelevant segments and use vector indexes that only cover rows matching the filter. This helps avoid post-filtering after the ANN search. In case the segment has no index, the engine performs KNN search.

EVALUATION OF SEARCH RESULTS

The paper evaluates SingleStore-V using VectorDBBench which is an open-source benchmarking tool for vector databases. The experiments assess search speed, accuracy, scalability, and index performance on real world datasets.

Datasets Used

- SIFT (SIFT1M, 10M, 100M and 1B): 128-dimensional vectors for image descriptors.
- GIST1M: 960-dimensional vectors derived from image data.

- Cohere10M: 768-dimensional embeddings from Wikipedia passages (language model outputs).

These datasets are from the domains of image search, semantic search, and document retrieval.

Evaluation Setup

Experiments were run on r6a.8xlarge EC2 instances (32 vCPUs, 256GB RAM) with Single Instruction Multiple Data (SIMD). The Distributed scalability tests used 8-node clusters to evaluate performance with increasing data.

Comparison

The systems compared against SingleStore-V are:

- Milvus: A specialized, in-memory vector database built on Faiss.
- pgvector: A PostgreSQL extension for vector search (general-purpose).

Index Types

Two indexing algorithms were tested:

- IVF_PQFS: A quantization-based index targetted to be fast and compact.
- HNSW: A graph-based index targetted for high recall and low latency.

Metrics

- Throughput (QPS): Queries per second.
- Recall: Accuracy of top-k results compared to ground truth.
- Index Build Time: Time required to build indexes.
- Memory Usage: RAM used by vector indexes.
- Scalability: Performance trend when dataset size increase.

Results

- Throughput: SingleStore V achives 78–99% of Milvus throughput depending on data and index type. For IVF_PQFS, it was up to 110× faster than pgvector and for HNSW, 1.7–2.6× faster.
- Recall: The recall was comparable to Milvus for both IVF and HNSW indexes and shows high accuracy.
- Index Build: SingleStore-V built indexes 1.6–6× faster than Milvus. It performs better than pgvector.
- Scalability: SingleStore-V scales from 1M to 1B vectors. Query throughput degrades but they are still significantly remain high. The paper confirms its ability to handle production-scale vector workloads.
- Real-Time Search: The Experiment Figure 7a and 7v shows that if more data remains in the in-memory rowstore, throughput drops, but recall improves due to exact search on fresh rows.

RELATED WORK

The paper inludes a related work section and it discusses prior work about Vector Databases like Milvus, Faiss, Pinecone that are specialized, and pgvector, PASE, AnalyticDB-V that are generalized.

EVALUATION METHODS

Following are the pros of the evaluation methods used in the paper

(1) Usage of publically available Benchmark tool - VectorDBBench
(2) Baseline comparison with Milvus(Specialized) and pgvector(Generalized)
(3) Metrics reported such as Throughput, Recall, Index time

Some of the areas of improvement can include

(1) Autoindex evaluation needs improvement. There is no dedicated evaluation to show how well it performs.
(2) Could add resource utilization metrics like CPU and memory.
(3) There can also be negative tests like the failure point and lessons learnt.

LIMITATIONS

Limitations of the Generalized Approach: The authors mention that vector search can be dealt as just another data type inside a relational database. Its evident that Vector indexes are different from B-trees as they rely on approximation methods which does not naturally fit into traditional SQL planners. It will be good to compare and state the gap in the industry practices to see why some production systems use hybrid approach by using specialized databased alongside existing non-vector data store.

Limitations of Row + Column Storage for Vector Search: The hybrid rowstore and columnstore design in SingleStore-V said to provide both real-time access and high-performance search, it may also introduce trade-offs. The paper says that vector data in the rowstore is not indexed and relies on exact search, which can slow query performance when data accumulates. This means that under frequent writes, when vectors stay in memory before being flushed to disk, search efficiency can be affected. Hence it may require careful tuning to avoid performance degradation. The suggestion is to write in bulk as the data will be written to disk directly.

Support for Approximate Nearest Neighbour: At the time of writing, the paper states pgvector does not support ANN based indexes like HNSW. But while this report is written, postgres pgvector supports HSNW.

REFERENCES

Yunan Zhang, Shige Liu, and Jianguo Wang. 2024. Are There Fundamental Limitations in Supporting Vector Data Management in Relational Databases? A Case Study of PostgreSQL. In International Conference on Data Engineering (ICDE).