Milvus 2.4 new features

The Most Advanced Vector Database, built on top of Milvus

James Luan
April 3, 2024

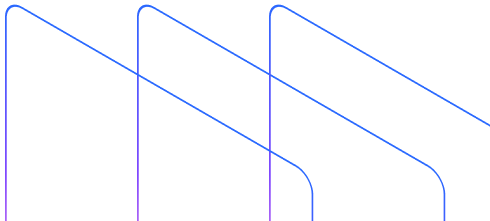
Highlight Summary

New Features

- Multi-Vector & Hybrid search
- Grouping search
- Inverted index
- Milvus CDC
- GPU index
- FP16 support
- Sparse vector(beta)
- Spark Connector

Enhancements

- Refine all SDKs
- Mmap optimization
- More collections&partitions supported
- Introduce L0 Segment to improve write/delete throughput
- Refactor bulkinsert logic
- Filtered search improvement



Multi-Vector & Hybrid Search

Before
- Only one embedding/vector field per collection

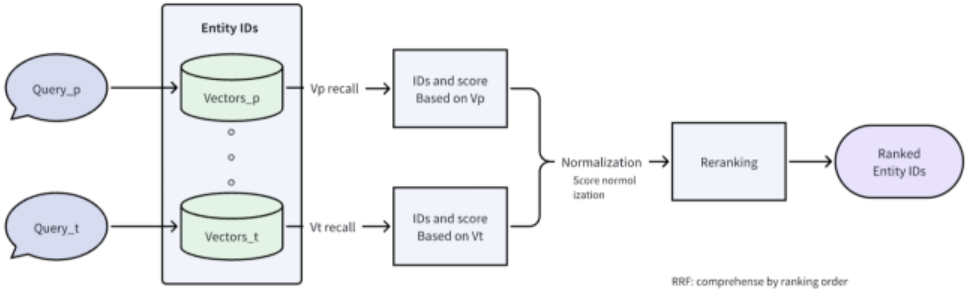
After
- Native support **multi-modal and multi-angular info**
- **Hybrid search framework** introduce more flexibility, combine two or more Vectors, Sparse + Dense etc

Multi-Vector

"Tesla Model S"

id	Vector from text	Vector from front image	Vector from side image
0	[0.13, ...]	[0.07, ...]	[0.15, ...]
1	[0.05, ...]	[0.05, ...]	[0.01, ...]
2	[0.21, ...]	[0.1, ...]	[0.25, ...]
3	[0.01, ...]	[0.05, ...]	[0.01, ...]
4	[0.09, ...]	[0.03, ...]	[0.08, ...]
5	[0.13, ...]	[0.19, ...]	[0.03, ...]

Hybrid Search



RRF: comprehense by ranking order
Weighted avg: comprehense by score

Input: Multi-Vectors Dataset: Multi-fields

Output: Consolidated ranking results

Sparse Vector (beta)

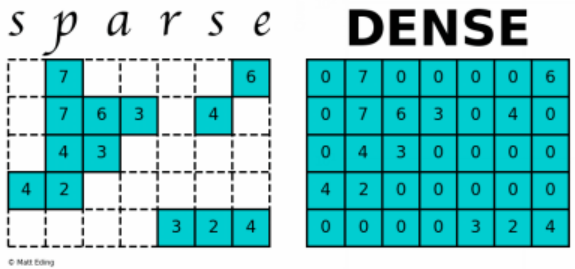
Before
- Only support dense vectors

After
- performs better in terms of out-of-domain knowledge search performance, keyword-awareness, and interpretability than dense vector models
- A hybrid search solution based on: Sparse + Dense

Usage
... dtype=DataType.**SPARSE_FLOAT_VECTOR)** ...
... "index_type": **"SPARSE_INVERTED_INDEX" | "SPARSE_WAND"...**

Current support
• Index: SPARSE_INVERTED_INDEX and SPARSE_WAND
• Metric type: IP
• Formats:
  - Sparse Matrices (sparse_embeddings = rand(num_entities, dim, density=0.005, format='csr')),
  - List of Dictionaries ({2: 0.33, 98: 0.72, ...}),
  - List of Iterables of Tuples ([(2, 0.33), (98, 0.72), ...])

Sparse explained



dense = [0.2, 0.3, 0.5, 0.7, ...] # several hundred floats
sparse = [{331: 0.5}, {14136: 0.7}...] # 10 key value pairs

Grouping Search

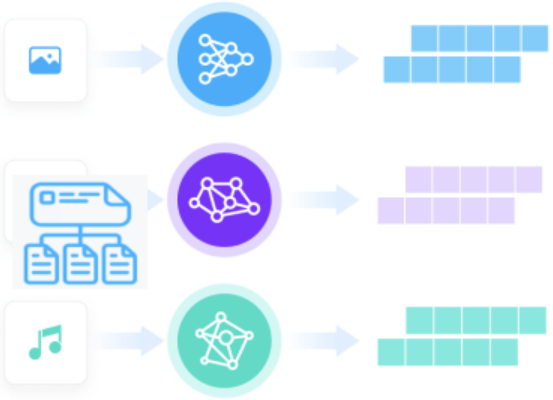
Before
- Return results based on vector level, the split chunks

After
- You can get results on **high level grouping**, e.g. from chunks to document
- Avoid over-centralization of results

Example： search books contain “Harry Potter”

ID	Vector	Chunk	Doc
1	[..., ..., ...]	Harry Potter and the Philosopher's Stone is a …	***Philosopher's Stone***
2	[..., ..., ...]	Harry Potter lives with his neglectful uncle and aunt, …	*Philosopher's Stone*
3	[..., ..., ...]	On Harry's eleventh birthday…	*Philosopher's Stone*
100	[..., ..., ...]	While Wspending the summer at the Dursleys, twelve-year-old Harry Potter is visited by…	***Chamber of Secrets***

Documents splits to multiple Chunks <- Embedding ->



Then, grouping search can help from input vector to find document

More info

```
# Get existing collection
collection = Collection(name='group_search') # Replace with the actual name of your collection

# Group search results
results = collection.search(
    # Replace with your query vector
    data=[[0.596255488017676, 0.803331289393417, …, 0.371808804119490 24, 0.508569977301849]],
    anns_field="passage_vector_field", # Query vector field name
    param={"metric_type": "L2", "params": {"nprobe": 10}}, # Search parameters
    limit=10, # Max. number of search results to return
    group_by_field="doc_id", # Group results by document ID
    output_fields=["doc_id", "passage_id"]
)
print(results)
```

Group-by param

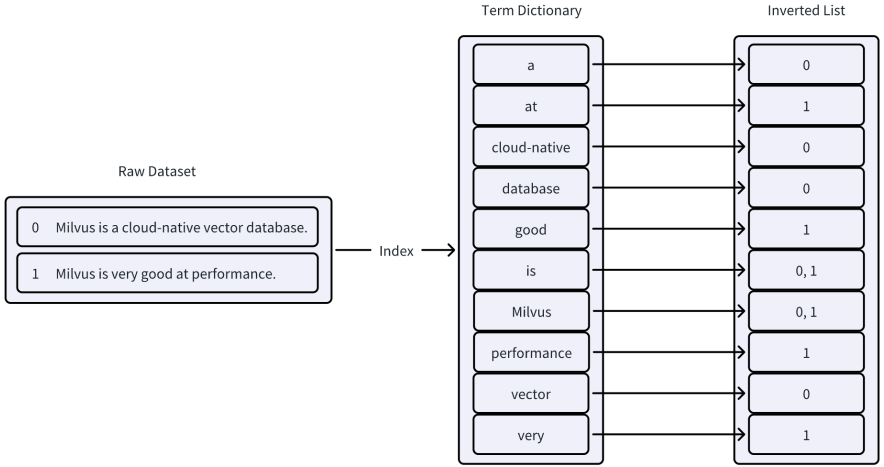
```
# Results

[
    {"id": 1, "distance": 0.0, "entity": {"doc_id": 1, "passage_id": 1}},
    {"id": 3109, "distance": 0.012906046584248543, "entity": {"doc_id": 9, "passage_id": 3109}},
    {"id": 2226, "distance": 0.01377844251692295, "entity": {"doc_id": 6, "passage_id": 2226}},
    {"id": 2105, "distance": 0.028072306886315346, "entity": {"doc_id": 5, "passage_id": 2105}},
    {"id": 187, "distance": 0.03216659277677536, "entity": {"doc_id": 7, "passage_id": 187}},
    {"id": 4890, "distance": 0.03625018149614334, "entity": {"doc_id": 10, "passage_id": 4890}},
]
```


'Like' fuzzy matching and inverted index

Before
- ‘like’ only support prefix match, e.g. column like ‘prefix%’

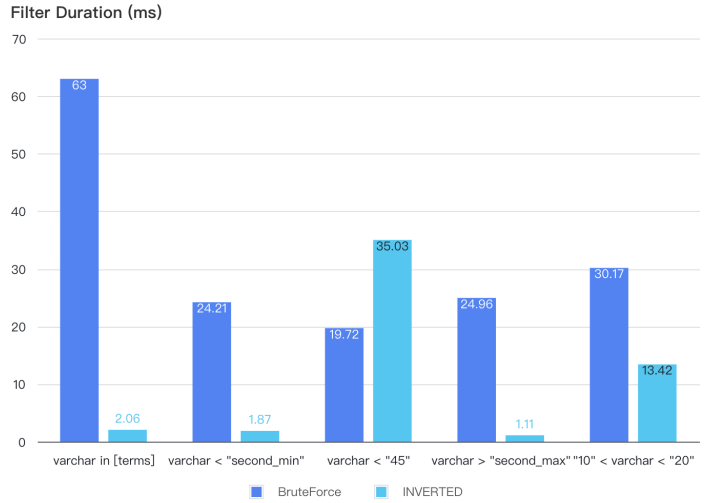
After
- Now can support **fuzzy match**, including ‘%infix%’,’%suffix’
- Inverted index **accelerate** search performance

Inverted index structure




```
collection.create_index( "release_year", {"index_type": "INVERTED"})
```

Inverted Index Performance



- Supported: INT8, INT16, INT32, INT64, FLOAT, DOUBLE, BOOL and VARCHAR.
- Not supported yet: ARRAY and JSON.

More info

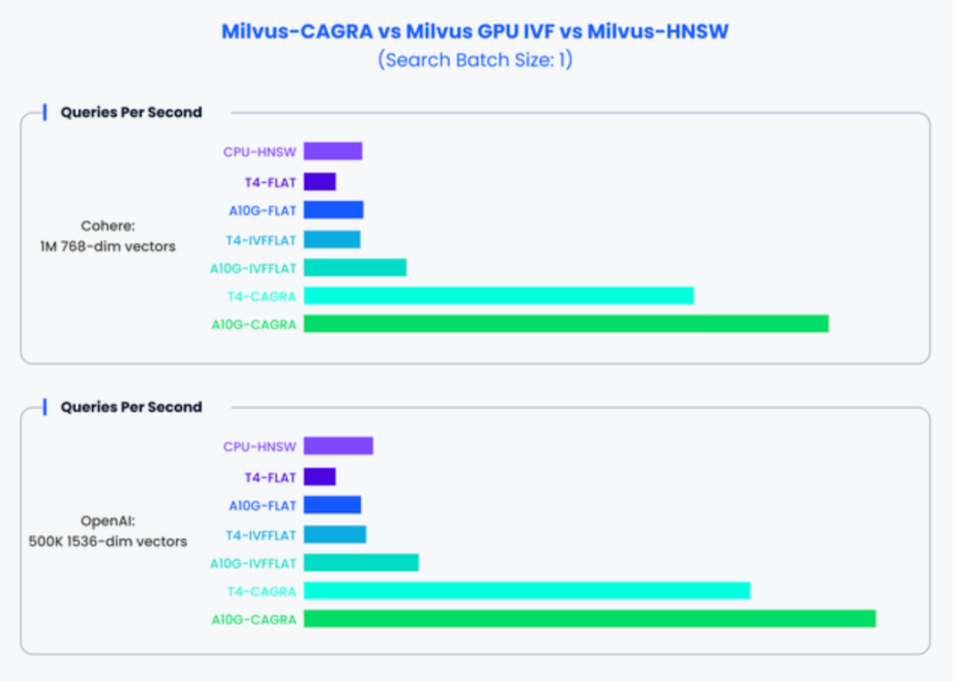
GPU index

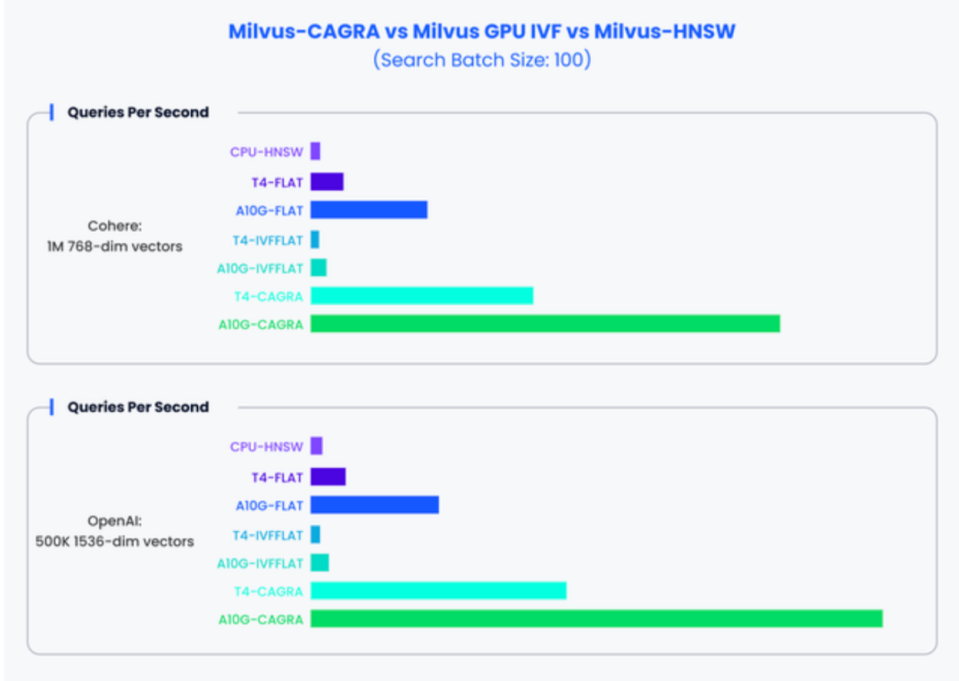
Improvement over CPU
- QPS: 100x for BruteForce, 10x for Graph, 30x for IVF
- Build Index: 10x for Graph, 100x for IVF(A100)

Scenarios
- Batch search scenarios(large NQ)
- High qps with high performance/high recall

Supported Index & Metric type
- Index Type: GPU_IVF_FLAT, GPU_IVF_PQ, GPU_CAGRA, GPU_BRUTE_FORCE. (Updated with Nvidia Raft)
- Metric Type: L2, IP

GPU index





zilliz

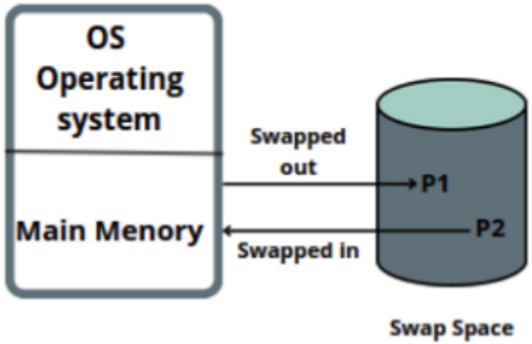
MMAP dynamic switches

Before
- Mmap enabled at cluster level and need to reboot
- Can't balance well in performance and storage-optimization

After
- Mmap can be set at collection/index level, **dynamic** settings provide **flexiblity**
- **4x size larger** only with performance decrease in 5%
- plan to be applied **on production**

```
# Set memory mapping property to True at collection level
collection.set_properties({'mmap.enabled': True})

# Set memory mapping property to True at index level
collection.alter_index( index_name="vector_index", # Replace with your vector index name
extra_params={"mmap.enabled": True} # Enable memory mapping for index )
```

Mem-mapped explained

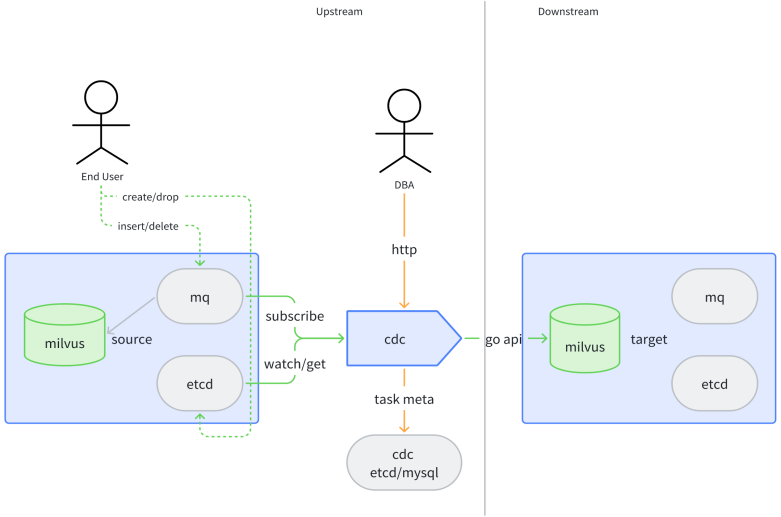


Milvus CDC – Change Data Capture

Before
- No global disaster recovery solution
- Migration by BR needs down-time

After
- Improve data **reliability**, e.g. cross region DR, SLA from 99.9 -> **99.99**
- Build data **ecosystem**, e.g. exporting data to various targets

Data flow



Definition
Milvus-CDC is a change data capture tool for Milvus. It can capture the changes of upstream Milvus collections and sink them to downstream Milvus.

Typical use cases
- Data Replication, e.g. cross-region DR solution, zero-down time migration etc
- Data Integration, e.g. kafka or s3

Supported operations
• Create/Drop Collection
• Insert/Delete
• Create/Drop Partition
• Create/Drop Index
• Load/Release/Flush
• Load/Release Partitions
• Create/Drop Database
Anything not mentioned is not supported;

More info

Roadmap of Milvus 3.0

Category	Directions	Features
Scalability and Performance *World-class scalability and performance*	***Cost Effectiveness***	Lazy load Auto Scaler
	Performance	GPU Accelaration* Binary quantization support and refine
	Storage	A new vector storage for faster retrieve and filtering speed
Ease of Use Provide flexible usabilities and maintainance	**Milvus local mode**	A simple vector search lib with same API as miles
	SDK	GA Rust and C# SDK
	More Database operations	Schema Change Null and default value support
Functionalities *Features for production-ready*	***More Data Types, Indexes***	Index on JSON and Array Date datatype and Geospatial search
	Primary Key/ Vector Deduplication	Dedup primary key and vector field while insertion
AI-augment Capability *Designed features for AI applications*	***Integrate with embedding, ranking and generation model***	Embedding integration* Ranking/Reranking modules
	More semantic search use cases	NN Filtering Multi target search

QA

zilliz